SQL

Structured Query Language

YAYASAN PRIMA AGUS TEKNIK

Dr. Joseph Teguh Santoso, S.Kom, M.Kom

SQL

Structured Query Language

Dr. Joseph Teguh Santoso, S.Kom, M.Kom

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK

JL. Majapahit No. 605 Semarang

Telp. (024) 6723456. Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

SQL Structured Query Language

Penulis :
Dr. Joseph Teguh Santoso, S.Kom, M.Kom

ISBN : 9 786236 141755

Editor :
Muhammad Sholikan, M.Kom

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto. S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Model data logis dan sistem database muncul untuk mengatasi beberapa kekurangan dari sistem berorientasi file. Mereka bertindak sebagai struktur data terpusat yang terintegrasi yang membatasi masalah redundansi sampai tingkat tertentu. Konsep penting yang muncul dengan sistem manajemen basis data adalah Kamus Data yang memberikan makna pada bidang yang disimpan dalam basis data. Tiba-tiba semua orang memiliki akses ke kumpulan catatan yang terkontrol dan bermakna berkat kamus data yang diatur dengan baik.

Saya menulis teks pendahulu buku ini pada akhir tahun 2012 dan menaruhnya di Internet dengan harapan seseorang akan menganggapnya berguna. Ada cukup tanggapan sehingga saya terus menulis dan kami memiliki buku ini sebagai hasilnya. Ada banyak perubahan dan penambahan dari waktu ke waktu, tetapi tujuan intinya tetap sama – pengenalan singkat tentang SQL dengan asumsi tidak ada pengalaman sebelumnya di dalamnya.

Setelah membaca teks ini, pembaca harus dapat mengenali bagian dari kueri yang mereka temui dan bahkan dapat menulis sendiri pernyataan dan kueri SQL sederhana. Buku ini, bagaimanapun, tidak dimaksudkan sebagai karya referensi atau untuk administrator database penuh waktu karena tidak memiliki cakupan topik yang lengkap.

Penulis memberikan pendekatan langkah demi langkah yang terperinci untuk menerjemahkan persyaratan bahasa Inggris sederhana ke kueri SQL. Contoh-contohnya banyak dan ditujukan untuk pemula secara lengkap.

Buku ini mencoba untuk mencakup semua paket DBMS utama dan memberikan penjelasan yang bagus tentang pernyataan itu sendiri.

Semarang, 7 Agustus 2021

Dr. Joseph Teguh Santoso, S.Kom, M.Kom

DAFTAR ISI

HALAMAN JUDUL

KATA PENGANTAR i

DAFTAR ISI ii

BAB 1 PENGANTAR SQL 1

BAB 2 EXPLORASI DATA DENGAN SELECT 11

BAB 3 MEMAHAMI JENIS DATA 21

BAB 4 IMPORT DAN EXPORT DATA 35 35

BAB 5 MATEMATIKA DASAR DAN STATUS DENGAN SQL 50

BAB 6 MENGGABUNGKAN TABEL DALAM BASIS DATA RELASIONAL 65

BAB 7 MENDESAIN TABEL 81

BAB 8 MENGGALI INFORMASI DENGAN MENGELOMPOKAN JAWABA 100

BAB 9 MEMERIKSA DAN MENGUBAH DATA 115

BAB 10 FUNGSI STATISTIK DALAM SQL 137

BAB 11 KUERI TANGGAL DAN WAKTU 151

BAB 12 TEKNIK KUERI TINGKAT TINGGI 168

BAB 13 PENCARIAN TEKS UNTUK MENEMUKAN DATA LENGKAP 186

BAB 14 ANALISIS DATA PATIAL DENGAN POSTGIS 214

BAB 15 TAMPILAN WAKTU DAN MENYEDERHANAKAN FUNGSI 237

BAB 16 MENGGUNAKAN POSTGRESQL DARI COMMANDLINE 258

BAB 17 MEMELIHARA BASIS DATA 278

DAFTAR PUSTAKA 288

BAB I
PENGANTAR SQL

SQL lebih dari sekedar sarana untuk mengekstrak pengetahuan dari data. Ini juga merupakan bahasa untuk mendefinisikan struktur yang menyimpan data sehingga kita dapat mengatur hubungan dalam data. Kepala di antara struktur itu adalah tabel.

Tabel adalah kisi-kisi baris dan kolom yang menyimpan data. Setiap baris menampung kumpulan kolom, dan setiap kolom berisi data dari tipe tertentu: paling umum, angka, karakter, dan tanggal. Kami menggunakan SQL untuk mendefinisikan struktur tabel dan bagaimana setiap tabel mungkin berhubungan dengan tabel lain dalam database. Kami juga menggunakan SQL untuk mengekstrak, atau membuat kueri, data dari tabel.

Memahami tabel sangat penting untuk memahami data dalam database Anda. Setiap kali saya mulai bekerja dengan database baru, hal pertama yang saya lakukan adalah melihat tabel di dalamnya. Saya mencari petunjuk dalam nama tabel dan struktur kolomnya. Apakah tabel berisi teks, angka, atau keduanya? Berapa banyak baris dalam setiap tabel?

Selanjutnya, saya melihat berapa banyak tabel dalam database. Database paling sederhana mungkin memiliki satu tabel. Aplikasi full-bore yang menangani data pelanggan atau melacak perjalanan udara mungkin memiliki lusinan atau ratusan. Jumlah tabel tidak hanya memberi tahu saya berapa banyak data yang perlu saya analisis, tetapi juga mengisyaratkan bahwa saya harus menjelajahi hubungan di antara data di setiap tabel.

Sebelum Anda menggali SQL, mari kita lihat contoh seperti apa isi tabel. Kami akan menggunakan database hipotetis untuk mengelola pendaftaran kelas sekolah; dalam database itu ada beberapa tabel yang melacak mahasiswa dan kelas mereka. Tabel pertama, yang disebut student_enrollment, menunjukkan siswa yang mendaftar untuk setiap bagian kelas:

Tabel 1.1 Daftar Entity Database

student_id	class_id	class_section	semester
CHRISPA004	COMPSCI101	3	Fall 2017
DAVISHE010	COMPSCI101	3	Fall 2017
ABRILDA002	ENG101	40	Fall 2017
DAVISHE010	ENG101	40	Fall 2017
RILEYPH002	ENG101	40	Fall 2017

Tabel ini menunjukkan bahwa dua mahasiswa telah mendaftar ke COMPSCI101, dan tiga mahasiswa telah mendaftar ke ENG101. Tapi di mana rincian tentang masing-masing mahasiswa dan kelas? Dalam contoh ini, detail ini disimpan dalam tabel terpisah yang disebut siswa dan kelas, dan setiap tabel terkait dengan tabel ini. Di sinilah kekuatan database relasional mulai menunjukkan dirinya.
Beberapa baris pertama tabel mahasiswa meliputi:

Tabel 1.2 Daftar Entity Database 2

student_id	first_name	last_name	dob
ABRILDA002	Abril	Davis	1999-01-10
CHRISPA004	Chris	Park	1996-04-10
DAVISHE010	Davis	Hernandez	1987-09-14
RILEYPH002	Riley	Phelps	1996-06-15

Tabel siswa berisi rincian setiap mahasiswa, menggunakan nilai di kolom student_id untuk mengidentifikasi masing-masing mahasiswa. Nilai tersebut bertindak sebagai kunci unik yang menghubungkan kedua tabel, memberi Anda kemampuan untuk membuat baris seperti berikut dengan kolom class_id dari student_enrollment dan kolom first_name dan last_name dari mahasiswa:

Tabel 1.3 Data Nama Siswa

class_id	first_name	last_name
COMPSCI101	Davis	Hernandez
COMPSCI101	Chris	Park
ENG101	Abril	Davis
ENG101	Davis	Hernandez
ENG101	Riley	Phelps

Tabel kelas akan bekerja dengan cara yang sama, dengan kolom class_id dan beberapa kolom detail tentang kelas. Pembuat basis data lebih suka mengatur data menggunakan tabel terpisah untuk setiap entitas utama yang dikelola basis data untuk mengurangi data yang berlebihan. Dalam contoh, kami menyimpan nama dan tanggal lahir setiap siswa hanya sekali. Bahkan jika mahasiswa mendaftar untuk beberapa kelas — seperti yang dilakukan Davis Hernandez — kami tidak menyia-nyiakan ruang database dengan memasukkan namanya di sebelah setiap kelas di tabel student_enrollment. Kami hanya menyertakan ID mahasiswanya.

Mengingat bahwa tabel adalah blok bangunan inti dari setiap database, dalam bab ini Anda akan memulai petualangan pengkodean SQL dengan membuat tabel di dalam database baru. Kemudian Anda akan memuat data ke dalam tabel dan melihat tabel yang sudah selesai.

Membuat Basis Data

Program PostgreSQL yang Anda unduh di Pendahuluan adalah sistem manajemen basis data, paket perangkat lunak yang memungkinkan Anda mendefinisikan, mengelola, dan membuat kueri basis data. Saat Anda menginstal PostgreSQL, itu membuat server database — sebuah instance dari aplikasi yang berjalan di komputer Anda — yang menyertakan database default yang disebut postgres. Basis data adalah kumpulan objek yang mencakup tabel, fungsi, peran pengguna, dan banyak lagi. Menurut dokumentasi PostgreSQL, database default "dimaksudkan untuk digunakan oleh pengguna, utilitas, dan aplikasi pihak ketiga" (lihat https://www.postgresql.org/docs/current/static/app-initdb.html). Dalam latihan di bab ini, kita akan membiarkan default apa adanya dan sebagai gantinya membuat yang baru. Kami akan melakukan ini untuk menjaga objek yang terkait dengan topik atau aplikasi tertentu terorganisir bersama.

Untuk membuat database, Anda hanya menggunakan satu baris SQL, yang ditunjukkan pada Gambar1.1 Kode ini, bersama dengan semua contoh dalam buku ini, tersedia untuk diunduh melalui sumber di https://www.nostarch.com/practicalSQL/

```
CREATE DATABASE analysis;
```

***Gambar 1.1**: Membuat database bernama analysis*

Pernyataan ini membuat database di server Anda bernama analisis menggunakan pengaturan default PostgreSQL. Perhatikan bahwa kode terdiri dari dua kata kunci — CREATE dan DATABASE — diikuti dengan nama database baru. Pernyataan diakhiri dengan titik koma, yang menandakan akhir dari perintah. Titik koma mengakhiri semua pernyataan PostgreSQL dan merupakan bagian dari standar ANSI SQL. Terkadang Anda dapat menghilangkan titik koma, tetapi tidak selalu, dan khususnya tidak saat menjalankan beberapa pernyataan di admin. Jadi, menggunakan titik koma adalah kebiasaan yang baik untuk dibentuk.

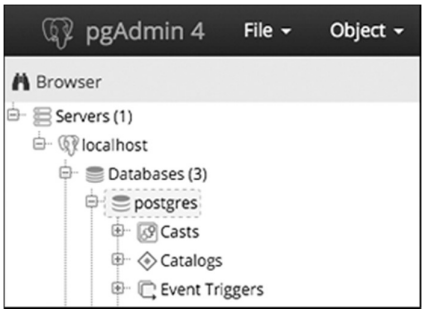
Menjalankan SQL di pgAdmin

Sebagai bagian dari Pengantar buku ini, Anda juga menginstal alat administrasi grafis pgAdmin (jika belum, lanjutkan dan lakukan sekarang). Untuk sebagian besar pekerjaan kami, Anda akan menggunakan pgAdmin untuk menjalankan (atau mengeksekusi) pernyataan SQL yang kami tulis. Kemudian dalam buku di Bab selanjutnya, penulis akan menunjukkan kepada Anda cara menjalankan pernyataan SQL di jendela terminal menggunakan program baris perintah PostgreSQL psql, tetapi memulai sedikit lebih mudah dengan antarmuka grafis.

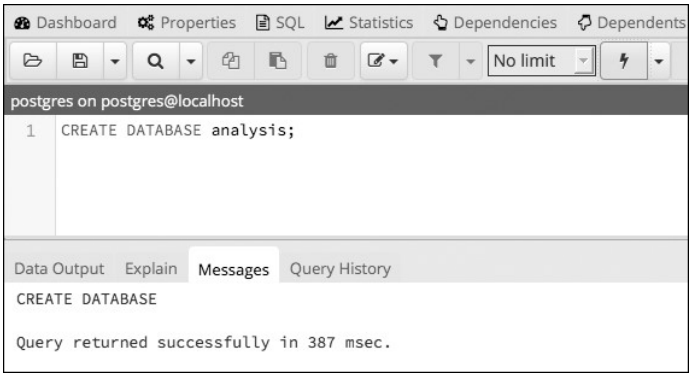
Kami akan menggunakan pgAdmin untuk menjalankan pernyataan SQL di Tabel 1.1 yang membuat database. Kemudian, kita akan terhubung ke database baru dan membuat tabel.

Ikuti langkah ini:

1. Jalankan PostgreSQL. Jika Anda menggunakan Windows, penginstal mengatur PostgreSQL untuk diluncurkan setiap kali Anda boot. Di macOS, Anda harus mengklik dua kali Postgres.app di folder Aplikasi Anda.
2. Luncurkan pgAdmin. Seperti yang Anda lakukan di Pendahuluan, di panel vertikal kiri (browser objek) perluas tanda plus di sebelah kiri node Server untuk menampilkan server default. Bergantung pada cara Anda menginstal PostgreSQL, server default mungkin bernama localhost atau PostgreSQL x, di mana x adalah versi aplikasi.
3. Klik dua kali nama server. Jika Anda memberikan kata sandi selama instalasi, masukkan saat diminta. Anda akan melihat pesan singkat bahwa pgAdmin sedang membangun koneksi.
4. Di browser objek pgAdmin, perluas Databases dan klik sekali pada database postgres untuk menyorotnya, seperti yang ditunjukkan pada Gambar 1.2.
5. Buka Alat Kueri dengan memilih Alat > Alat Kueri.
6. Di panel SQL Editor (panel horizontal atas), ketik atau salin kode dari Tabel 1.1.
7. Klik ikon petir untuk menjalankan pernyataan. PostgreSQL membuat basis data, dan di panel Output di Alat Kueri di bawah Pesan, Anda akan melihat pemberitahuan yang menunjukkan kueri berhasil dikembalikan, seperti yang ditunjukkan pada Gambar 1.3.

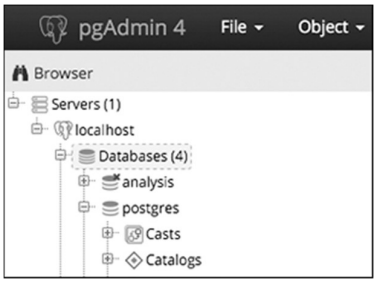


Gambar 1.2: Menghubungkan ke database postgres default



Gambar 1.3: Membuat database analisis

8. Untuk melihat database baru Anda, klik kanan Databases di browser objek. Dari menu pop-up, pilih Refresh, dan database analisis akan muncul dalam daftar, seperti yang ditunjukkan pada Gambar 1.4.



Gambar 1.4: Database analisis ditampilkan di browser objek

Kerja bagus! Anda sekarang memiliki database yang disebut analisis, yang dapat Anda gunakan untuk sebagian besar latihan dalam buku ini. Dalam pekerjaan Anda sendiri, biasanya merupakan praktik terbaik untuk membuat database baru untuk setiap proyek untuk menyimpan tabel dengan data terkait bersama-sama.

Menghubungkan ke Database Analisis

Sebelum Anda membuat tabel, Anda harus memastikan bahwa pgAdmin terhubung ke database analisis daripada ke database default postgres.

Untuk melakukannya, ikuti langkah-langkah berikut:

1. Tutup Alat Kueri dengan mengklik X di kanan atas alat. Anda tidak perlu menyimpan file saat diminta.
2. Di browser objek, klik sekali pada database analisis.
3. Buka kembali Alat Kueri dengan memilih Alat Alat Kueri.
4. Anda sekarang akan melihat analisis label pada postgres@localhost di bagian atas jendela Alat Kueri. (Sekali lagi, alih-alih localhost, versi Anda mungkin menampilkan PostgreSQL.)

Sekarang, kode apa pun yang Anda jalankan akan diterapkan ke database analisis.

Membuat Tabel

Seperti yang saya sebutkan sebelumnya, tabel adalah tempat data hidup dan hubungannya didefinisikan. Saat Anda membuat tabel, Anda menetapkan nama untuk setiap kolom (kadang-kadang disebut sebagai bidang atau atribut) dan menetapkannya sebagai tipe data. Ini adalah nilai yang akan diterima kolom—seperti teks, bilangan bulat, desimal, dan tanggal—dan definisi tipe data adalah salah satu cara SQL menegakkan integritas data. Misalnya, kolom yang didefinisikan sebagai tanggal akan mengambil data dalam salah satu dari beberapa format standar, seperti YYYY-MM-DD. Jika Anda mencoba memasukkan karakter yang tidak dalam format tanggal, misalnya, kata persik, Anda akan menerima pesan kesalahan.

Data yang disimpan dalam tabel dapat diakses dan dianalisis, atau ditanyakan, dengan pernyataan SQL. Anda dapat mengurutkan, mengedit, dan melihat data, dan dengan mudah mengubah tabel nanti jika kebutuhan Anda berubah. Mari kita buat tabel di database analisis.

Pernyataan CREATE TABLE

Untuk latihan ini, kami akan menggunakan data yang sering dibahas: gaji dosen. Kode dibawah ini menunjukkan pernyataan SQL untuk membuat tabel yang disebut dosen:

```
CREATE TABLE teachers (
      id bigserial,
      first_name varchar(25),
      last_name varchar(50),
      school varchar(50),
      hire_date date,
      salary numeric
);
```

Definisi tabel ini jauh dari komprehensif. Misalnya, tidak ada beberapa batasan yang akan memastikan bahwa kolom yang harus diisi memang memiliki data atau bahwa kita tidak secara tidak sengaja memasukkan nilai duplikat. Saya membahas kendala secara rinci di Bab 7, tetapi di bab-bab awal ini saya menghilangkannya untuk fokus agar Anda mulai menjelajahi data.

Kode dimulai dengan dua kata kunci SQL **1** CREATE dan TABLE yang, bersama dengan nama pengajar, memberi sinyal kepada PostgreSQL bahwa bit kode berikutnya menjelaskan tabel untuk ditambahkan ke database. Setelah kurung buka, pernyataan tersebut menyertakan daftar nama kolom yang dipisahkan koma beserta tipe datanya. Untuk tujuan gaya, setiap baris kode baru berada pada barisnya sendiri dan menjorok empat spasi, yang tidak diperlukan, tetapi membuat kode lebih mudah dibaca.

Setiap nama kolom mewakili satu elemen data diskrit yang ditentukan oleh tipe data. Kolom id adalah **2** tipe data bigserial, tipe integer khusus yang bertambah secara otomatis setiap kali Anda menambahkan baris ke tabel. Baris pertama menerima nilai **1** di kolom id, baris kedua **2**, dan seterusnya. Tipe data bigserial dan tipe serial lainnya adalah implementasi khusus PostgreSQL, tetapi kebanyakan sistem database memiliki fitur serupa.

Selanjutnya, kita membuat kolom untuk nama depan dan belakang guru, dan sekolah tempat mereka mengajar **3**. Masing-masing dari tipe data varchar, kolom teks dengan panjang maksimum yang ditentukan oleh nomor dalam tanda kurung. Kami berasumsi bahwa tidak ada seorang pun di database yang memiliki nama belakang lebih dari 50 karakter. Meskipun ini adalah asumsi yang aman, Anda akan menemukan dari waktu ke waktu bahwa pengecualian akan selalu mengejutkan Anda.

Tanggal_pekerjaan guru **4** diatur ke tanggal tipe data, dan kolom gaji **5** adalah numerik. Saya akan membahas tipe data lebih menyeluruh di Bab 3, tetapi tabel ini menunjukkan beberapa contoh umum tipe data. Blok kode membungkus **6** dengan tanda kurung tutup dan titik koma. Sekarang setelah Anda mengetahui tampilan SQL, mari jalankan kode ini di pgAdmin.

Membuat Tabel Guru

Anda memiliki kode dan terhubung ke database, sehingga Anda dapat membuat tabel menggunakan langkah yang sama seperti yang kita lakukan saat membuat database:

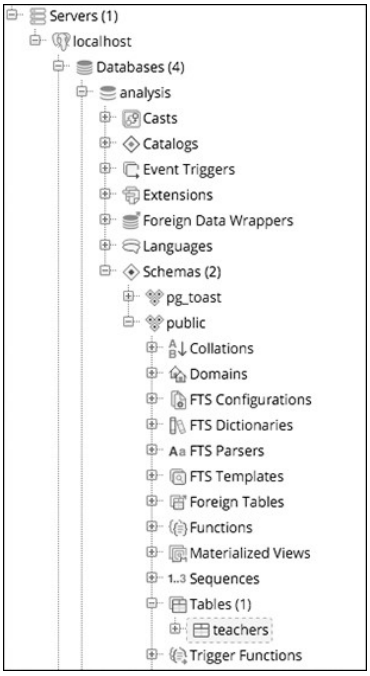
1. Buka Alat Kueri pgAdmin (jika belum terbuka, klik sekali pada database analisis di browser objek pgAdmin, lalu pilih Alat Alat Kueri).
2. Salin skrip CREATE TABLE dari Tabel 1.2 ke SQL Editor.
3. Jalankan skrip dengan mengklik ikon petir.

Jika semuanya berjalan dengan baik, Anda akan melihat pesan di panel keluaran bawah Alat Kueri pgAdmin yang berbunyi, Kueri berhasil dikembalikan tanpa hasil dalam 84 mdtk. Tentu saja, jumlah milidetik akan bervariasi tergantung pada sistem Anda. Sekarang, temukan tabel yang Anda buat.

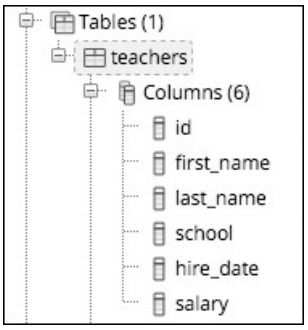
Kembali ke jendela pgAdmin utama dan, di browser objek, klik kanan database analisis dan pilih **Refresh**. Pilih **Skema → public → Tabel** untuk melihat tabel baru Anda, seperti yang ditunjukkan pada Gambar 1-4.

Perluas node tabel guru dengan mengklik tanda plus di sebelah kiri namanya. Ini mengungkapkan rincian lebih lanjut tentang tabel, termasuk nama kolom, seperti yang ditunjukkan pada Gambar 1.6. Informasi lain juga muncul, seperti indeks, pemicu, dan batasan, tetapi saya akan membahasnya di bab selanjutnya. Mengklik nama tabel dan kemudian memilih menu SQL di ruang kerja pgAdmin akan menampilkan pernyataan SQL yang digunakan untuk membuat tabel guru.

Selamat! Sejauh ini, Anda telah membangun database dan menambahkan tabel ke dalamnya. Langkah selanjutnya adalah menambahkan data ke tabel sehingga Anda bisa menulis kueri pertama Anda.



Gambar 1.6: Tabel guru di browser objek



Gambar 1.7: Detail tabel untuk guru

Memasukkan Baris ke dalam Tabel

Anda dapat menambahkan data ke tabel PostgreSQL dengan beberapa cara. Seringkali, Anda akan bekerja dengan banyak baris, jadi metode termudah adalah mengimpor data dari file teks atau database lain langsung ke tabel. Tetapi untuk memulai, kami akan menambahkan beberapa baris menggunakan pernyataan INSERT INTO ... VALUES yang menentukan kolom target dan nilai data. Kemudian kita akan melihat data di rumah barunya.

Pernyataan INSERT

Untuk memasukkan beberapa data ke dalam tabel, Anda harus terlebih dahulu menghapus pernyataan CREATE TABLE yang baru saja Anda jalankan. Kemudian, dengan mengikuti langkah yang sama seperti yang Anda lakukan untuk membuat database dan tabel, salin kode di Tabel 1.3 ke Alat Kueri pgAdmin Anda:

```
INSERT INTO teachers (first_name, last_name, school, hire_date, salary)
VALUES ('Janet', 'Smith', 'F.D. Roosevelt HS', '2011-10-30', 36200),
```

```
('Lee', 'Reynold', 'F.D. Roosevelt HS', 1993-05-22', 65000)
('Samuel', 'Cole', 'Myers Middle School', 2005-08-01', 43500)
('Samantha', 'Bush', 'Myers Middle School', 2011-10-30, 36200)
('Betty', 'Diaz', 'Myres Midlle School', '2005-08-30', 43500)
('Kathleen', 'Roush', 'F.D. Roosevelt HS', '2010-10-22', 38500)
```

Blok kode ini menyisipkan nama dan data enam guru. Di sini, sintaks PostgreSQL mengikuti standar ANSI SQL: setelah kata kunci INSERT INTO adalah nama tabel, dan dalam tanda kurung adalah kolom yang harus diisi. Di baris berikutnya adalah kata kunci VALUES dan data yang akan dimasukkan ke setiap kolom di setiap baris. Anda perlu menyertakan data untuk setiap baris dalam kumpulan tanda kurung, dan di dalam setiap kumpulan tanda kurung, gunakan koma untuk memisahkan setiap nilai kolom. Urutan nilai juga harus sesuai dengan urutan kolom yang ditentukan setelah nama tabel. Setiap baris data diakhiri dengan koma, dan baris terakhir mengakhiri seluruh pernyataan dengan titik koma.

Perhatikan bahwa nilai-nilai tertentu yang kita sisipkan diapit oleh tanda kutip tunggal, tetapi ada juga yang tidak. Ini adalah persyaratan SQL standar. Teks dan tanggal memerlukan kutipan; angka, termasuk bilangan bulat dan desimal, tidak memerlukan tanda kutip. Saya akan menyoroti persyaratan ini seperti yang muncul dalam contoh. Juga, perhatikan format tanggal yang kami gunakan: tahun empat digit diikuti oleh bulan dan tanggal, dan setiap bagian digabungkan dengan tanda hubung. Ini adalah standar internasional untuk format tanggal; menggunakannya akan membantu Anda menghindari kebingungan. (Mengapa sebaiknya menggunakan format YYYY-MM-DD? Lihat https://xkcd.com/1179/ untuk melihat komik hebat tentangnya.) PostgreSQL mendukung banyak format tanggal tambahan, dan saya akan menggunakan beberapa di contoh.

Anda mungkin bertanya-tanya tentang kolom id, yang merupakan kolom pertama dalam tabel. Saat Anda membuat tabel, skrip Anda menetapkan kolom itu sebagai tipe data serial besar. Jadi saat PostgreSQL menyisipkan setiap baris, secara otomatis mengisi kolom id dengan bilangan bulat yang bertambah secara otomatis. Saya akan membahasnya secara rinci di Bab 3 ketika saya membahas tipe data.

Sekarang, jalankan kodenya. Kali ini pesan di Alat Kueri harus menyertakan kata Kueri berhasil dikembalikan: 6 baris terpengaruh.

Melihat Data

Anda dapat melihat sekilas data yang baru saja Anda muat ke tabel pengajar menggunakan pgAdmin. Di browser objek, cari tabel dan klik kanan. Di menu pop-up, pilih Lihat/Edit Data Semua Baris. Seperti yang ditunjukkan Gambar 1.9, Anda akan melihat enam baris data dalam tabel dengan setiap kolom diisi oleh nilai-nilai dalam pernyataan SQL.

Data Output | Explain | Messages | Query History

	id bigint	first_name character vary	last_name character vary	school character varying	hire_date date	salary numeric
1	1	Janet	Smith	F.D. Roosevelt ...	2011-10-30	36200
2	2	Lee	Reynolds	F.D. Roosevelt ...	1993-05-22	65000
3	3	Samuel	Cole	Myers Middle S...	2005-08-01	43500
4	4	Samantha	Bush	Myers Middle S...	2011-10-30	36200
5	5	Betty	Diaz	Myers Middle S...	2005-08-30	43500
6	6	Kathleen	Roush	F.D. Roosevelt ...	2010-10-22	38500

Gambar 1.8: Melihat data tabel secara langsung di pgAdmin

Perhatikan bahwa meskipun Anda tidak memasukkan nilai untuk kolom id, setiap guru memiliki nomor ID yang ditetapkan.

Anda dapat melihat data menggunakan antarmuka pgAdmin dalam beberapa cara, tetapi kami akan fokus pada penulisan SQL untuk menangani tugas-tugas tersebut.

Ketika Kode Menjadi Buruk

Mungkin ada alam semesta tempat kode selalu berfungsi, tetapi sayangnya, kami belum menemukan mesin yang mampu membawa kami ke sana. Terjadi kesalahan. Apakah Anda membuat kesalahan ketik atau mencampuradukkan urutan operasi, bahasa komputer tidak kenal ampun tentang sintaks. Misalnya, jika Anda lupa koma dalam kode PostgreSQL membalas kesalahan:

```
ERROR: syntax error at or near "("
LINE 5: ('Samuel', 'Cole', 'Myers Middle School', '2005-08-01', 43...
           ^
********** Error **********
```

Untungnya, pesan kesalahan mengisyaratkan apa yang salah dan di mana: kesalahan sintaksis berada di dekat tanda kurung terbuka pada baris 5. Namun terkadang pesan kesalahan bisa lebih kabur. Dalam hal ini, Anda melakukan apa yang dilakukan pembuat kode terbaik: pencarian internet cepat untuk pesan kesalahan. Kemungkinan besar, orang lain pernah mengalami masalah yang sama dan mungkin tahu jawabannya.

Memformat SQL untuk Keterbacaan

SQL tidak memerlukan pemformatan khusus untuk dijalankan, jadi Anda bebas menggunakan gaya psikedelik Anda sendiri dari huruf besar, huruf kecil, dan lekukan acak. Tapi itu tidak akan membuat Anda mendapatkan teman ketika orang lain perlu bekerja dengan kode Anda (dan cepat atau lambat seseorang akan melakukannya). Demi keterbacaan dan menjadi pembuat kode yang baik, yang terbaik adalah mengikuti konvensi ini:

- Kata kunci SQL huruf besar, seperti SELECT. Beberapa pembuat kode SQL juga menggunakan huruf besar untuk nama tipe data, seperti TEXT dan INTEGER. Saya menggunakan karakter huruf kecil untuk tipe data dalam buku ini untuk memisahkannya dalam pikiran Anda dari kata kunci, tetapi Anda dapat menggunakan huruf besar jika diinginkan.

- Hindari huruf besar unta dan sebagai gantinya gunakan huruf kecil_dan_garis bawah untuk nama objek, seperti nama tabel dan kolom.
- Indentasi klausa dan blok kode agar mudah dibaca menggunakan dua atau empat spasi. Beberapa pembuat kode lebih memilih tab daripada spasi; gunakan mana yang paling cocok untuk Anda atau organisasi Anda.

Kami akan menjelajahi konvensi pengkodean SQL lainnya saat kami membaca buku ini, tetapi ini adalah dasar-dasarnya.

Latihan Soal

Berikut adalah dua latihan untuk membantu Anda menjelajahi konsep yang terkait dengan database, tabel, dan hubungan data:

1. Bayangkan Anda sedang membangun database untuk membuat katalog semua hewan di kebun binatang lokal Anda. Anda ingin satu tabel untuk melacak jenis hewan dalam koleksi dan tabel lain untuk melacak secara spesifik pada setiap hewan. Tulis pernyataan CREATE TABLE untuk setiap tabel yang menyertakan beberapa kolom yang Anda butuhkan. Mengapa Anda menyertakan kolom yang Anda pilih?
2. Sekarang buat pernyataan INSERT untuk memuat data sampel ke dalam tabel. Bagaimana Anda bisa melihat data melalui alat pgAdmin? Buat pernyataan INSERT tambahan untuk salah satu tabel Anda. Sengaja menghilangkan salah satu koma yang diperlukan untuk memisahkan entri dalam klausa VALUES kueri. Apa pesan kesalahannya? Apakah ini akan membantu Anda menemukan kesalahan dalam kode?

BAB II
EKSPLORASI DATA DENGAN SELECT

Bagi saya, bagian terbaik dari menggali data bukanlah prasyarat untuk mengumpulkan, memuat, atau membersihkan data, tetapi ketika saya benar-benar dapat mengidentifikasi data. Itulah saat-saat ketika saya menemukan apakah data itu bersih atau kotor, apakah itu lengkap, dan yang terpenting, cerita apa yang bisa diceritakan oleh data tersebut. Pikirkan memilih data sebagai proses yang mirip dengan mewawancarai seseorang yang melamar pekerjaan. Anda ingin mengajukan pertanyaan yang mengungkapkan apakah realitas keahlian mereka cocok dengan resume mereka.

Wawancara itu mengasyikkan karena Anda menemukan kebenaran. Misalnya, Anda mungkin menemukan bahwa separuh responden lupa mengisi kolom email di kuesioner, atau walikota belum membayar pajak properti selama lima tahun terakhir. Atau Anda mungkin mengetahui bahwa data Anda kotor: nama dieja tidak konsisten, tanggal salah, atau angka tidak sesuai dengan harapan Anda. Temuan Anda menjadi bagian dari cerita data.

Dalam SQL, mewawancarai data dimulai dengan kata kunci SELECT, yang mengambil baris dan kolom dari satu atau lebih tabel dalam database.

Pernyataan SELECT bisa sederhana, mengambil semuanya dalam satu tabel, atau bisa cukup rumit untuk menautkan lusinan tabel sambil menangani beberapa perhitungan dan memfilter berdasarkan kriteria yang tepat. Kami akan mulai dengan pernyataan SELECT sederhana.

Sintaks SELECT Dasar
Berikut adalah pernyataan SELECT yang mengambil setiap baris dan kolom dalam tabel yang disebut tabel_saya:

```
SELECT * FROM my_table;
```

Satu baris kode ini menunjukkan bentuk paling dasar dari kueri SQL. Tanda bintang setelah kata kunci SELECT adalah wildcard. Wildcard seperti stand-in untuk suatu nilai: ia tidak mewakili sesuatu yang khusus dan sebaliknya mewakili semua nilai yang mungkin ada. Di sini, ini adalah singkatan untuk "pilih semua kolom." Jika Anda telah memberikan nama kolom alih-alih wildcard, perintah ini akan memilih nilai di kolom itu. Kata kunci FROM menunjukkan bahwa Anda ingin kueri mengembalikan data dari tabel tertentu. Titik koma setelah nama tabel memberi tahu PostgreSQL bahwa ini adalah akhir dari pernyataan kueri.

Mari gunakan pernyataan SELECT ini dengan wildcard asterisk pada tabel pengajar yang Anda buat di Bab 1. Sekali lagi, buka pgAdmin, pilih database analisis, dan buka Alat Kueri. Kemudian jalankan pernyataan yang ditunjukkan pada:

```
SELECT * FROM teachers;
```

Hasil yang disetel di panel keluaran Alat Kueri berisi semua baris dan kolom yang Anda sisipkan ke tabel pengajar di Bab 1. Baris mungkin tidak selalu muncul dalam urutan ini, tapi tidak apa-apa.

Tabel 2.1 Tabel Kueri Karyawan

id	First_name	Last_name	school	hire_date	salary
1	Janet	Smith	F.D. Roosevelt HS	2011-10-30	36200
2	Lee	Reynolds	F.D. Roosevelt HS	1993-05-22	65000
3	Samuel	Cole	Myers Middle School	2005-08-01	43500
4	Samantha	Bush	Myers Middle School	2011-10-30	36200
5	Betty	Diaz	Myers Middle School	2005-08-30	43500
6	Kathleen	Roush	F.D. Roosevelt HS	2010-10-22	38500

Perhatikan bahwa kolom id (bertipe bigserial) secara otomatis terisi dengan bilangan bulat berurutan, meskipun Anda tidak memasukkannya secara eksplisit. Sangat berguna. Integer yang bertambah otomatis ini bertindak sebagai pengidentifikasi unik, atau kunci, yang tidak hanya memastikan setiap baris dalam tabel itu unik, tetapi juga nantinya akan memberi kita cara untuk menghubungkan tabel ini ke tabel lain dalam database. Mari kita lanjutkan untuk menyempurnakan kueri ini.

Menanyakan Subset Kolom

Menggunakan wildcard asterisk sangat membantu untuk menemukan seluruh isi tabel. Namun seringkali lebih praktis untuk membatasi kolom yang diambil kueri, terutama dengan database besar. Anda dapat melakukannya dengan menamai kolom, dipisahkan dengan koma, tepat setelah kata kunci SELECT. Sebagai contoh:

```
SELECT some_column, another_column, amazing_column FROM table_name;
```

Dengan sintaks itu, kueri akan mengambil semua baris hanya dari tiga kolom itu.

Mari kita terapkan ini ke tabel guru. Mungkin dalam analisis Anda, Anda ingin fokus pada nama dan gaji guru, bukan sekolah tempat mereka bekerja atau saat mereka dipekerjakan. Dalam hal ini, Anda mungkin memilih hanya beberapa kolom dari tabel alih-alih menggunakan wildcard asterisk. Masukkan pernyataan yang ditunjukkan pada Daftar 2-2. Perhatikan bahwa urutan kolom dalam kueri berbeda dari urutan dalam tabel: Anda dapat mengambil kolom dalam urutan apa pun yang Anda inginkan.

```
SELECT last_name, first_name, salary FROM teachers;
```

Sekarang, di kumpulan hasil, Anda telah membatasi kolom menjadi tiga:

```
last_name   first_name  salary
---------   ----------  ------
Smith       Janet       36200
Reynold     Lee         65000
Cole        Samuel      43500
Bush        Samantha    36200
Diaz        Betty       43500
Roush       Kathleen    38500
```

Meskipun contoh-contoh ini adalah dasar, mereka menggambarkan strategi yang baik untuk memulai wawancara Anda tentang kumpulan data. Umumnya, adalah bijaksana untuk memulai analisis Anda dengan memeriksa apakah data Anda ada dan dalam format yang Anda harapkan. Apakah tanggal dalam format bulan-tanggal-tahun yang lengkap, atau apakah mereka dimasukkan (seperti yang pernah saya amati dengan sedih) sebagai teks dengan bulan dan tahun saja? Apakah setiap baris memiliki nilai? Apakah secara misterius tidak ada nama belakang yang dimulai dengan huruf di luar "M"? Semua masalah ini menunjukkan potensi bahaya mulai dari data yang hilang hingga pencatatan yang buruk di suatu tempat dalam alur kerja.

Kami hanya bekerja dengan tabel enam baris, tetapi ketika Anda menghadapi tabel ribuan atau bahkan jutaan baris, penting untuk membaca cepat kualitas data Anda dan rentang nilai yang dikandungnya. Untuk melakukan ini, mari kita gali lebih dalam dan tambahkan beberapa kata kunci SQL.

Dalam sebuah tabel, bukan hal yang aneh jika kolom berisi baris dengan nilai duplikat. Dalam tabel guru, misalnya, kolom sekolah mencantumkan nama sekolah yang sama beberapa kali karena setiap sekolah mempekerjakan banyak guru. Untuk memahami rentang nilai dalam kolom, kita dapat menggunakan kata kunci DISTINCT sebagai bagian dari kueri yang menghilangkan duplikat dan hanya menampilkan nilai unik. Gunakan kata kunci DISTINCT segera setelah SELECT

```
SELECT DISTINCT school
FROM teachers;
```

Hasilnya adalah sebagai berikut:

```
School
---------------
F.D. Roosevelt HS
Myers Middle School
```

Meskipun enam baris ada di tabel, output hanya menunjukkan dua nama sekolah yang unik di kolom sekolah. Ini adalah langkah pertama yang bermanfaat untuk menilai kualitas data. Misalnya, jika nama sekolah dieja lebih dari satu cara, variasi ejaan tersebut akan mudah dikenali dan diperbaiki. Saat Anda bekerja dengan tanggal atau angka, DISTINCT akan membantu menyoroti pemformatan yang tidak konsisten atau rusak. Misalnya, Anda mungkin mewarisi kumpulan data di mana tanggal dimasukkan dalam kolom yang diformat dengan tipe data teks. Praktik itu (yang harus Anda hindari) memungkinkan tanggal yang salah format ada:

```
Date
----------
5/30/2019
6//2019
6/1/2019
6/2/2019
```

Kata kunci DISTINCT juga berfungsi di lebih dari satu kolom sekaligus. Jika kita menambahkan kolom, kueri mengembalikan setiap pasangan nilai yang unik.

```
SELECT DISTINCT school, salary
FROM teachers;
```

Sekarang kueri mengembalikan setiap gaji unik (atau berbeda) yang diperoleh di setiap sekolah. Karena dua guru di Myers Middle School memperoleh Rp 652.500.000, pasangan itu terdaftar hanya dalam satu baris, dan kueri mengembalikan lima baris, bukan keenamnya dalam tabel:

school	salary
Myers Middle School	43500
Myers Middle School	36200
F.D. Roosevelt HS	65000
F.D. Roosevelt HS	38500
F.D. Roosevelt HS	36200

Teknik ini memberi kita kemampuan untuk bertanya, “Untuk setiap x dalam tabel, berapa semua nilai y?” Untuk setiap pabrik, apa semua bahan kimia yang dihasilkannya? Untuk setiap distrik pemilihan, siapa saja calon yang mencalonkan diri? Untuk setiap gedung konser, siapa artis yang tampil bulan ini?

SQL menawarkan teknik yang lebih canggih dengan fungsi agregat yang memungkinkan kita menghitung, menjumlahkan, dan menemukan nilai minimum dan maksimum. Saya akan membahasnya secara rinci di Bab 5 dan Bab 8.

Menyortir Data dengan ORDER BY

Data dapat lebih masuk akal, dan dapat mengungkapkan pola dengan lebih mudah, jika diatur secara berurutan daripada dicampuradukkan secara acak.

Dalam SQL, kami mengurutkan hasil kueri menggunakan klausa yang berisi kata kunci ORDER BY diikuti dengan nama kolom atau kolom yang akan diurutkan. Menerapkan klausa ini tidak mengubah tabel asli, hanya hasil kueri.

```
SELECT first_name, last_name, salary
FROM teachers
ORDER BY salary DESC;
```

Secara default, ORDER BY mengurutkan nilai dalam urutan menaik, tetapi di sini saya mengurutkan dalam urutan menurun dengan menambahkan kata kunci DESC. (Kata kunci ASC opsional menentukan pengurutan dalam urutan menaik.) Sekarang, dengan mengurutkan kolom gaji dari tertinggi ke terendah, saya dapat menentukan guru mana yang berpenghasilan paling banyak:

first_name	last_name	salary
Lee	Reynolds	65000
Samuel	Cole	43500
Betty	Diaz	43500
Kathleen	Roush	38500
Janet	Smith	36200
Samantha	Bush	36200

Menyortir Teks

Menyortir kolom angka di PostgreSQL menghasilkan apa yang mungkin Anda harapkan: peringkat data dari nilai terbesar ke terkecil atau sebaliknya tergantung pada apakah Anda menggunakan kata kunci DESC atau tidak. Tetapi menyortir kolom dengan huruf atau karakter lain mungkin memberikan hasil yang mengejutkan, terutama jika kolom tersebut memiliki campuran karakter huruf besar dan kecil, tanda baca, atau angka yang diperlakukan sebagai teks.

Selama instalasi PostgreSQL, server diberi lokasi tertentu untuk pemeriksaan, atau pengurutan teks, serta kumpulan karakter. Keduanya didasarkan pada pengaturan di sistem operasi komputer atau opsi khusus yang disediakan selama penginstalan. (Anda dapat membaca lebih lanjut tentang collation di dokumentasi PostgreSQL resmi di
https://www.postgresql.org/docs/current/static/ collation.html.)

Misalnya, di Mac saya, instalasi PostgreSQL saya diatur ke lokal en_US, atau AS Inggris, dan set karakter UTF-8. Anda dapat melihat pengaturan susunan server Anda dengan menjalankan pernyataan SHOW ALL; dan melihat nilai parameter lc_collate.

Dalam kumpulan karakter, setiap karakter mendapatkan nilai numerik, dan urutan pengurutannya bergantung pada urutan nilai tersebut.

Berdasarkan UTF-8, PostgreSQL mengurutkan karakter dalam urutan ini:

1. Tanda baca, termasuk tanda kutip, tanda kurung, dan operator matematika
2. Angka 0 sampai 9
3. Tanda baca tambahan, termasuk tanda tanya
4. Huruf kapital dari A sampai Z
5. Lebih banyak tanda baca, termasuk tanda kurung dan garis bawah
6. Huruf kecil a sampai z
7. Tanda baca tambahan, karakter khusus, dan alfabet tambahan

Biasanya, urutan pengurutan tidak akan menjadi masalah karena kolom karakter biasanya hanya berisi nama, tempat, deskripsi, dan teks langsung lainnya. Tetapi jika Anda bertanya-tanya mengapa kata Ladybug muncul sebelum ladybug dalam jenis Anda, Anda sekarang memiliki penjelasan.

Kemampuan untuk mengurutkan dalam kueri kami memberi kami fleksibilitas besar dalam cara kami melihat dan menyajikan data. Misalnya, kami tidak terbatas pada pengurutan hanya pada satu kolom.

```
SELECT last_name, school, hire_date
FROM teachers
ORDER BY salary ASC, hire-date DESC;
```

Dalam hal ini, kami mengambil nama belakang guru, sekolah mereka, dan tanggal mereka dipekerjakan. Dengan mengurutkan kolom sekolah dalam urutan menaik dan tanggal_pekerjaan dalam urutan menurun, kami membuat daftar guru yang dikelompokkan berdasarkan sekolah dengan guru yang baru direkrut dicantumkan terlebih dahulu. Ini menunjukkan kepada kita siapa guru terbaru di setiap sekolah. Kumpulan hasil akan terlihat seperti ini:

```
last_name     school                hire_date
Smith         F.D. Roosevelt HS     2011-10-30
Roush         F.D. Roosevelt HS     2010-10-22
Reynolds      F.D. Roosevelt HS     1993-05-22
Bush          Myers Middle School   2011-10-30
Diaz          Myers Middle School   2005-08-30
Cole          Myers Middle School   2005-08-01
```

Anda dapat menggunakan ORDER BY di lebih dari dua kolom, tetapi Anda akan segera mencapai titik pengembalian yang semakin berkurang di mana efeknya hampir tidak terlihat.

Bayangkan jika Anda menambahkan kolom tentang gelar perguruan tinggi tertinggi yang dicapai guru, tingkat kelas yang diajarkan, dan tanggal lahir ke klausa ORDER BY. Akan sulit untuk memahami berbagai arah pengurutan dalam output sekaligus, apalagi mengomunikasikannya kepada orang lain. Mencerna data terjadi paling mudah ketika hasilnya berfokus pada menjawab pertanyaan tertentu; oleh karena itu, strategi yang lebih baik adalah membatasi jumlah kolom dalam kueri Anda hanya pada yang paling penting, dan kemudian menjalankan beberapa kueri untuk menjawab setiap pertanyaan yang Anda miliki.

Memfilter Baris dengan WHERE

Terkadang, Anda ingin membatasi baris yang dikembalikan kueri hanya pada baris yang satu atau beberapa kolomnya memenuhi kriteria tertentu. Dengan menggunakan guru sebagai contoh, Anda mungkin ingin menemukan semua guru yang direkrut sebelum tahun tertentu atau semua guru yang berpenghasilan lebih dari Rp 1.125.000.000 di sekolah dasar. Untuk tugas ini, kami menggunakan klausa WHERE.

Kata kunci WHERE memungkinkan Anda menemukan baris yang cocok dengan nilai tertentu, rentang nilai, atau beberapa nilai berdasarkan kriteria yang diberikan melalui operator. Anda juga dapat mengecualikan baris berdasarkan kriteria.

Listing dibawah ini menunjukkan contoh dasar. Perhatikan bahwa dalam sintaks SQL standar, klausa WHERE mengikuti kata kunci FROM dan nama tabel atau tabel yang ditanyakan:

```
SELECT last_name, school, hire_date
FROM teachers
WHERE school = Myers Middle School;
```

Kumpulan hasil hanya menunjukkan guru yang ditugaskan ke Myers Middle School:

```
last_name     school                hire_date
Cole          Myers Middle School   2005-08-01
Bush          Myers Middle School   2011-10-30
Diaz          Myers Middle School   2005-08-30
```

Di sini, saya menggunakan operator perbandingan yang sama untuk menemukan baris yang sama persis dengan nilai, tetapi tentu saja Anda dapat menggunakan operator lain dengan WHERE untuk menyesuaikan kriteria filter Anda. Tabel 2-1 memberikan ringkasan operator

pembanding yang paling umum digunakan. Tergantung pada sistem database Anda, mungkin lebih banyak lagi yang tersedia.

Tabel 2.1: Operator Perbandingan dan Pencocokan di PostgreSQL

Operator	Function	Example
=	Equal to	WHERE school = 'Baker Middle'
<> or !=	Not equal to*	WHERE school <> 'Baker Middle'
>	Greater than	WHERE salary > 20000
<	Less than	WHERE salary < 60500
>=	Greater than or equal to	WHERE salary >= 20000
<=	Less than or equal to	WHERE salary <= 60500
BETWEEN	Within a range	WHERE salary BETWEEN 20000 AND 40000
IN	Match one of a set of values	WHERE last_name IN ('Bush', 'Roush')
LIKE	Match a pattern (case sensitive)	WHERE first_name LIKE 'Sam%'
ILIKE	Match a pattern (case insensitive)	WHERE first_name ILIKE 'sam%'
NOT	Negates a condition	WHERE first_name NOT ILIKE 'sam%'

*The! = Operator bukan bagian dari ANSI SQL standar tetapi tersedia di PostgreSQL dan beberapa sistem database lainnya.

Contoh berikut menunjukkan operator perbandingan beraksi. Pertama, kami menggunakan operator equals untuk menemukan guru yang nama depannya adalah Janet:

```
SELECT first_name, last_name, school
FROM teachers
WHERE first_name = ‘Janet’;
```

Selanjutnya, kami mencantumkan semua nama sekolah dalam tabel tetapi mengecualikan F.D. Roosevelt HS menggunakan operator yang tidak sama:

```
SELECT school
FROM teachers
WHERE school != ‘F.D. Roosevelt HS’;
```

Di sini kami menggunakan operator less than untuk mencantumkan guru yang direkrut sebelum 1 Januari 2000 (menggunakan format tanggal YYYY-MM-DD):

```
SELECT first_name, last_name, hire_date
FROM teachers
WHERE hire_date < ‘2000-01-01’;
```

Kemudian kami menemukan guru yang berpenghasilan Rp 652.500.000 atau lebih menggunakan operator > =:

```
SELECT first_name, last_name, salary
FROM teachers
WHERE salary < ‘43500’;
```

Kueri berikutnya menggunakan operator ANTARA untuk menemukan guru yang berpenghasilan antara Rp 600.000.000 dan Rp 975.000.000. Perhatikan bahwa BETWEEN

bersifat inklusif, artinya hasilnya akan menyertakan nilai yang cocok dengan rentang awal dan akhir yang ditentukan.

```
SELECT first_name, last_name, school, salary
FROM teachers
WHERE salary BETWEEN 40000 AND 650000;
```

Kami akan kembali ke operator ini sepanjang buku ini, karena mereka akan memainkan peran kunci dalam membantu kami menemukan data dan jawaban yang ingin kami temukan.

Menggunakan LIKE dan ILIKE dengan WHERE

Operator perbandingan cukup mudah, tetapi LIKE dan ILIKE layak mendapat penjelasan tambahan. Pertama, keduanya memungkinkan Anda mencari pola dalam string dengan menggunakan dua karakter khusus:

Tanda persen (%) Sebuah wildcard yang cocok dengan satu atau lebih karakter

Garis bawah (_) Sebuah wildcard yang cocok dengan hanya satu karakter

Misalnya, jika Anda mencoba mencari kata baker, berikut LIKE pola akan cocok dengannya:

```
LIKE 'b%'
LIKE '%ak%'
LIKE '_aker'
LIKE 'ba_er'
```

Perbedaan? Operator LIKE, yang merupakan bagian dari standar SQL ANSI, peka huruf besar/kecil. Operator ILIKE, yang merupakan implementasi khusus PostgreSQL, tidak peka huruf besar/kecil. Daftar 2-8 menunjukkan bagaimana dua kata kunci memberikan hasil yang berbeda. Klausa WHERE pertama menggunakan LIKE untuk menemukan nama yang dimulai dengan karakter sam, dan karena peka huruf besar/kecil, maka akan mengembalikan hasil nol. Yang kedua, menggunakan ILIKE yang tidak peka huruf besar-kecil, akan mengembalikan Samuel dan Samantha dari tabel:

```
SELECT first_name
FROM teachers
WHERE first_name LIKE 'sam%';

SELECT first_name
FROM teachers
WHERE first_name ILIKE 'sam%';
```

Selama bertahun-tahun, saya tertarik menggunakan ILIKE dan operator wildcard dalam pencarian untuk memastikan saya tidak secara tidak sengaja mengecualikan hasil dari pencarian. Saya tidak berasumsi bahwa siapa pun yang mengetik nama orang, tempat, produk, atau kata benda lainnya selalu ingat untuk menggunakan huruf kapital. Dan jika salah satu tujuan dari mewawancarai data adalah untuk memahami kualitasnya, menggunakan pencarian case-insensitive akan membantu Anda menemukan variasi.

Karena LIKE dan ILIKE mencari pola, kinerja pada basis data besar bisa menjadi lambat. Kami dapat meningkatkan kinerja menggunakan indeks, yang akan saya bahas di “Mempercepat Kueri dengan Indeks” di halaman 108.

Menggabungkan Operator dengan AND dan OR

Operator perbandingan menjadi lebih berguna saat kita menggabungkannya. Untuk melakukan ini, kami menghubungkannya menggunakan kata kunci AND dan OR bersama dengan, jika perlu, tanda kurung.

Pernyataan dalam Daftar 2-9 menunjukkan tiga contoh yang menggabungkan operator dengan cara ini:

```
SELECT *
FROM teachers
WHERE school = 'Myers Middle School'
    AND salary < 40000;

SELECT *
FROM teachers
WHERE last_name = 'Cole'
    OR last_name = 'Bush';

SELECT *
FROM teachers
WHERE school = 'F.D. Roosevelt HS'
    AND (salary < 38000 OR salary > 40000);
```

Kueri pertama menggunakan AND dalam klausa WHERE untuk menemukan guru yang bekerja di Myers Middle School dan memiliki gaji kurang dari Rp 600.000.000. Karena kami menghubungkan dua kondisi menggunakan AND, keduanya harus benar untuk satu baris untuk memenuhi kriteria dalam klausa WHERE dan dikembalikan dalam hasil kueri.

Contoh kedua menggunakan OR untuk mencari guru yang nama belakangnya cocok dengan Cole atau Bush. Saat kita menghubungkan kondisi menggunakan OR, hanya satu kondisi yang harus benar untuk baris yang memenuhi kriteria klausa WHERE.

Contoh terakhir mencari guru di Roosevelt yang gajinya kurang dari Rp 570.000.000 atau lebih dari Rp 600.000.000 . Ketika kita menempatkan pernyataan di dalam tanda kurung, pernyataan tersebut dievaluasi sebagai sebuah kelompok sebelum digabungkan dengan kriteria lain. Dalam hal ini, nama sekolah harus persis F.D. Roosevelt HS dan gaji harus kurang atau lebih tinggi dari yang ditentukan untuk satu baris untuk memenuhi kriteria klausa WHERE.

Menyatukan Semuanya

Anda dapat mulai melihat bagaimana bahkan kueri sederhana sebelumnya memungkinkan kami mempelajari data kami dengan fleksibilitas dan presisi untuk menemukan apa yang kami cari.

Anda dapat menggabungkan pernyataan operator perbandingan menggunakan kata kunci AND dan OR untuk memberikan beberapa kriteria pemfilteran, dan Anda dapat menyertakan klausa ORDER BY untuk menentukan peringkat hasil.

Dengan mengingat informasi sebelumnya, mari gabungkan konsep-konsep dalam bab ini menjadi satu pernyataan untuk menunjukkan bagaimana mereka cocok bersama. SQL khusus tentang urutan kata kunci, jadi ikuti konvensi ini:

```
SELECT column_names
FROM table_name
WHERE criteria
ORDER BY column_names;
```

Listing diatas menunjukkan kueri terhadap tabel guru yang mencakup semua bagian yang disebutkan di atas:

```
SELECT first_name, last_name, school, hire_date, salary
FROM teachers
WHERE school LIKE '%Roos%'
ORDER BY hire_date DESC;
```

Daftar ini mengembalikan guru di Roosevelt High School, diurutkan dari karyawan terbaru hingga paling awal. Kita dapat melihat korelasi yang jelas antara tanggal pengangkatan guru di sekolah dan tingkat gajinya saat ini:

```
first_name  last_name   school              hire_date    salary
Janet       Smith       F.D. Roosevelt HS   2011-10-30   36200
Kathleen    Roush       F.D. Roosevelt HS   2010-10-22   38500
Lee         Reynolds    F.D. Roosevelt HS   1993-05-22   65000
```

Sekarang setelah Anda mempelajari struktur dasar dari beberapa kueri SQL yang berbeda, Anda telah memperoleh dasar untuk banyak keterampilan tambahan yang akan saya bahas di bab selanjutnya. Menyortir, memfilter, dan memilih hanya kolom yang paling penting dari sebuah tabel dapat menghasilkan sejumlah informasi mengejutkan dari data Anda dan membantu Anda menemukan cerita yang diceritakannya. Di bab berikutnya, Anda akan belajar tentang aspek dasar SQL lainnya: tipe data.

Latihan Soal

Jelajahi kueri dasar dengan latihan berikut:

1. Pengawas distrik sekolah meminta daftar guru di setiap sekolah. Tulis kueri yang mencantumkan sekolah dalam urutan abjad bersama dengan guru yang diurutkan dengan nama belakang A – Z.
2. Tulis kueri yang menemukan satu guru yang nama depannya dimulai dengan huruf S dan yang berpenghasilan lebih dari Rp 600.000.000.
3. Peringkat guru yang direkrut sejak 1 Januari 2010, diurutkan berdasarkan bayaran tertinggi hingga terendah.

BAB III
MEMAHAMI JENIS DATA

Setiap kali saya menggali database baru, saya memeriksa tipe data yang ditentukan untuk setiap kolom di setiap tabel. Jika saya beruntung, saya bisa mendapatkan kamus data: dokumen yang mencantumkan setiap kolom; menentukan apakah itu angka, karakter, atau tipe lainnya; dan menjelaskan nilai kolom. Sayangnya, banyak organisasi tidak membuat dan memelihara dokumentasi yang baik, sehingga tidak jarang terdengar, "Kami tidak memiliki kamus data." Dalam hal ini, saya mencoba belajar dengan memeriksa struktur tabel di pgAdmin.

Sangat penting untuk memahami tipe data karena menyimpan data dalam format yang sesuai adalah dasar untuk membangun database yang dapat digunakan dan melakukan analisis yang akurat. Selain itu, tipe data adalah konsep pemrograman yang berlaku untuk lebih dari sekedar SQL. Konsep yang akan Anda jelajahi dalam bab ini akan ditransfer dengan baik ke bahasa tambahan yang mungkin ingin Anda pelajari.

Dalam database SQL, setiap kolom dalam tabel dapat menampung satu dan hanya satu tipe data, yang didefinisikan dalam pernyataan CREATE TABLE. Anda mendeklarasikan tipe data setelah memberi nama kolom. Berikut adalah contoh sederhana yang mencakup dua kolom, satu berupa tanggal dan yang lainnya bilangan bulat:

```
CREATE TABLE eagle_watch (
    observed_date date,
    eagle_seen integer
);
```

Dalam tabel ini bernama eagle_watch (untuk inventaris tahunan elang botak), kolom tanggal_pengamatan dinyatakan memiliki nilai tanggal dengan menambahkan deklarasi tipe tanggal setelah namanya. Demikian pula, eagles_seen diatur untuk menampung bilangan bulat dengan deklarasi tipe integer.

Tipe data ini termasuk di antara tiga kategori yang paling sering Anda temui:

Karakter	Setiap karakter atau simbol
Bilangan	Termasuk bilangan bulat dan pecahan
Tanggal dan waktu	Jenis yang menyimpan informasi temporal

Mari kita lihat setiap tipe data secara mendalam; Saya akan mencatat apakah mereka bagian dari standar ANSI SQL atau khusus untuk PostgreSQL.

KARAKTER

Tipe string karakter adalah tipe tujuan umum yang cocok untuk kombinasi teks, angka, dan simbol apa pun. Jenis karakter meliputi:

char(n)

Kolom dengan panjang tetap di mana panjang karakter ditentukan oleh n. Kolom yang ditetapkan pada char (20) menyimpan 20 karakter per baris terlepas dari berapa banyak karakter yang Anda masukkan. Jika Anda memasukkan kurang dari 20 karakter di baris mana pun, PostgreSQL mengisi kolom lainnya dengan spasi. Tipe ini, yang merupakan bagian dari SQL standar, juga dapat ditentukan dengan karakter nama yang lebih panjang (n). Saat ini, char(n) jarang digunakan dan sebagian besar merupakan sisa dari sistem komputer lama.

varchar(n)

Kolom panjang variabel di mana panjang maksimum ditentukan oleh n. Jika Anda memasukkan lebih sedikit karakter dari jumlah maksimum, PostgreSQL tidak akan menyimpan spasi tambahan. Misalnya, string biru akan mengambil empat spasi, sedangkan string 123 akan mengambil tiga. Dalam database besar, praktik ini menghemat banyak ruang. Tipe ini, termasuk dalam SQL standar, juga dapat ditentukan menggunakan karakter nama yang lebih panjang yang bervariasi (n).

teks

Kolom panjang variabel dengan panjang tak terbatas. (Menurut dokumentasi PostgreSQL, string karakter terpanjang yang dapat Anda simpan adalah sekitar 1 gigabyte.) Jenis teks bukan bagian dari standar SQL, tetapi Anda akan menemukan implementasi serupa di sistem database lain, termasuk Microsoft SQL Server dan MySQL . Menurut dokumentasi PostgreSQL di
https://www.postgresql.org/docs/current/static/datatype-character.html,
tidak ada perbedaan substansial dalam kinerja di antara ketiga jenis tersebut. Itu mungkin berbeda jika Anda menggunakan pengelola basis data lain, jadi sebaiknya periksa dokumen. Fleksibilitas dan penghematan ruang potensial dari varchar dan teks tampaknya memberi mereka keuntungan. Tetapi jika Anda mencari diskusi online, beberapa pengguna menyarankan bahwa mendefinisikan kolom yang akan selalu memiliki jumlah karakter yang sama dengan char adalah cara yang baik untuk menandakan data apa yang seharusnya ada di dalamnya. Misalnya, Anda dapat menggunakan char(2) untuk singkatan pos negara bagian AS.

Untuk melihat ketiga tipe karakter ini beraksi, jalankan skrip di Daftar 3-1. Skrip ini akan membuat dan memuat tabel sederhana dan kemudian mengekspor data ke file teks di komputer Anda.

```
CREATE TABLE char_data_types (
      varchar_column varchar(10),
      char_column char(10),
      text_column text
);
INSERT INTO char_data_types
VALUES
      (‘abc’, ‘abc’, ‘abc’),
      (‘defghi’, ‘defghi’, ‘defghi’);

COPY char_data_types TO C:\YourDirectory\typetest.txt’
WITH (FORMAT CSV, HEADER, DELIMITER ‘|’);
```

Script mendefinisikan tiga kolom karakter dari jenis yang berbeda dan memasukkan dua baris string yang sama ke dalam masing-masing . Berbeda dengan pernyataan INSERT INTO yang Anda pelajari di Bab 1, di sini kami tidak menentukan nama kolom. Jika pernyataan VALUES cocok dengan jumlah kolom dalam tabel, database akan menganggap Anda memasukkan nilai sesuai urutan definisi kolom yang ditentukan dalam tabel.

Selanjutnya, skrip menggunakan kata kunci PostgreSQL COPY untuk mengekspor data ke file teks bernama typetest.txt di direktori yang Anda tentukan. Anda harus mengganti C:\YourDirectory\ dengan path lengkap ke direktori di komputer tempat Anda ingin menyimpan file. Contoh dalam buku ini menggunakan format Windows dan jalur ke direktori bernama YourDirectory pada drive C:. Jalur file Linux dan macOS memiliki format yang berbeda. Di Mac saya, jalur ke file di desktop adalah /Users/anthony/Desktop/. Di Linux, desktop saya terletak di /home/anthony/Desktop/. Direktori harus sudah ada; PostgreSQL tidak akan membuatnya untuk Anda.

Di PostgreSQL, COPY table_name FROM adalah fungsi impor dan COPY table_name TO adalah fungsi ekspor. Saya akan membahasnya secara mendalam di Bab 4; untuk saat ini, yang perlu Anda ketahui adalah bahwa opsi kata kunci WITH akan memformat data dalam file dengan setiap kolom dipisahkan oleh karakter pipa (|). Dengan begitu, Anda dapat dengan mudah melihat di mana spasi mengisi bagian kolom char yang tidak digunakan.

Untuk melihat hasilnya, buka typetest.txt menggunakan editor teks biasa (bukan Word atau Excel, atau aplikasi spreadsheet lainnya). Isinya akan terlihat seperti ini:

```
varchar_column|char_column|text_column
abc|abc       |abc
defghi|defghi    |defghi
```

Meskipun Anda menentukan 10 karakter untuk kolom varchar dan char, hanya kolom char yang menghasilkan 10 karakter setiap kali, mengisi karakter yang tidak digunakan dengan spasi. Kolom varchar dan teks hanya menyimpan karakter yang Anda sisipkan.
Sekali lagi, tidak ada perbedaan kinerja yang nyata di antara ketiga tipe tersebut, meskipun contoh ini menunjukkan bahwa char berpotensi menghabiskan lebih banyak ruang penyimpanan daripada yang dibutuhkan. Beberapa ruang yang tidak terpakai di setiap kolom mungkin tampak tidak berarti, tetapi kalikan itu dengan jutaan baris dalam lusinan tabel dan Anda akan segera berharap Anda lebih hemat.

Biasanya, menggunakan varchar dengan nilai n yang cukup untuk menangani outlier adalah strategi yang solid.

BILANGAN

Kolom angka menampung berbagai jenis angka (Anda dapat menebaknya), tetapi tidak hanya itu: kolom ini juga memungkinkan Anda melakukan penghitungan pada angka tersebut. Itu perbedaan penting dari angka yang Anda simpan sebagai string dalam kolom karakter, yang tidak dapat ditambahkan, dikalikan, dibagi, atau melakukan operasi matematika lainnya. Juga,

seperti yang saya bahas di Bab 2, angka yang disimpan sebagai karakter diurutkan secara berbeda dari angka yang disimpan sebagai angka, disusun dalam teks daripada urutan numerik. Jadi, jika Anda mengerjakan matematika atau urutan numerik itu penting, gunakan tipe angka.

Jenis nomor SQL meliputi:

Integer	Bilangan bulat, positif dan negatif
Scale dan floating-point	Dua format pecahan bilangan bulat

Integer

Tipe data integer adalah tipe angka paling umum yang akan Anda temukan saat menjelajahi data dalam database SQL. Pikirkan semua tempat bilangan bulat muncul dalam hidup: nomor jalan atau apartemen Anda, nomor seri di lemari es Anda, nomor pada tiket undian. Ini adalah bilangan bulat, baik positif maupun negatif, termasuk nol.

Standar SQL menyediakan tiga tipe integer: smallint, integer, dan bigint. Perbedaan antara ketiga jenis ini adalah ukuran maksimum angka yang dapat mereka pegang. Tabel 3-1 menunjukkan batas atas dan batas bawah masing-masing, serta berapa banyak penyimpanan yang dibutuhkan masing-masing dalam byte.

Tabel 3.1: Tipe Data Integer

Data type	Storage size	Range
`smallint`	2 bytes	–32768 to +32767
`integer`	4 bytes	–2147483648 to +2147483647
`bigint`	8 bytes	–9223372036854775808 to +9223372036854775807

Meskipun memakan paling banyak penyimpanan, bigint akan mencakup hampir semua persyaratan yang pernah Anda miliki dengan kolom angka. Penggunaannya adalah suatu keharusan jika Anda bekerja dengan angka yang lebih besar dari sekitar 2,1 miliar, tetapi Anda dapat dengan mudah menjadikannya sebagai default dan tidak perlu khawatir. Di sisi lain, jika Anda yakin angka akan tetap berada dalam batas bilangan bulat, jenis itu adalah pilihan yang baik karena tidak menghabiskan ruang sebanyak bigint (kekhawatiran saat menangani jutaan baris data).

Ketika nilai data akan tetap dibatasi, smallint masuk akal: hari dalam sebulan atau tahun adalah contoh yang baik. Tipe smallint akan menggunakan setengah penyimpanan sebagai bilangan bulat, jadi ini adalah keputusan desain database yang cerdas jika nilai kolom akan selalu sesuai dengan jangkauannya.

Jika Anda mencoba memasukkan angka ke salah satu kolom ini yang berada di luar jangkauannya, database akan menghentikan operasi dan mengembalikan kesalahan di luar jangkauan.

Bilangan Bulat Bertambah Otomatis

Pada Bab 1, ketika Anda membuat tabel teacher, Anda membuat kolom id dengan deklarasi bigserial: this dan saudaranya smallserial dan serial bukan tipe data yang benar sebagai implementasi khusus dari smallint, integer, dan jenis bigint. Saat Anda menambahkan kolom dengan tipe serial, PostgreSQL akan otomatis menaikkan nilai di kolom setiap kali Anda memasukkan baris, dimulai dengan 1, hingga maksimum setiap tipe integer.

Tipe serial adalah implementasi standar ANSI SQL untuk kolom identitas bernomor otomatis. Setiap manajer database mengimplementasikan ini dengan caranya sendiri. Misalnya, Microsoft SQL Server menggunakan kata kunci IDENTITY untuk menyetel kolom ke peningkatan otomatis.

Untuk menggunakan tipe serial pada kolom, nyatakan dalam pernyataan CREATE TABLE seperti yang Anda lakukan pada tipe integer. Misalnya, Anda bisa membuat tabel bernama orang yang memiliki kolom id di setiap baris:

```
CREATE TABLE people (
      Id serial,
      Person_name varchar(100)
);
```

Setiap kali person_name baru ditambahkan ke tabel, kolom id akan bertambah 1. Tabel 3.2 menunjukkan jenis serial dan rentang yang dicakupnya.

Tabel 3.2: Tipe Data Serial

Data type	Storage size	Range
smallserial	2 bytes	1 to 32767
serial	4 bytes	1 to 2147483647
bigserial	8 bytes	1 to 9223372036854775807

Seperti contoh di Bab 1, pembuat database sering menggunakan tipe serial untuk membuat nomor ID unik, juga dikenal sebagai kunci, untuk setiap baris dalam tabel. Setiap baris kemudian memiliki ID sendiri yang dapat dirujuk oleh tabel lain dalam database. Saya akan membahas konsep terkait tabel ini di Bab 6. Karena kolom bertambah secara otomatis, Anda tidak perlu memasukkan angka ke dalam kolom itu saat menambahkan data; PostgreSQL menanganinya untuk Anda.

Catatan: Meskipun kolom dengan tipe serial otomatis bertambah setiap kali baris ditambahkan, beberapa skenario akan membuat celah dalam urutan angka di kolom. Jika sebuah baris dihapus, misalnya, nilai dalam baris tersebut tidak akan pernah diganti. Atau, jika sisipan baris dibatalkan, urutan kolom akan tetap bertambah.

Bilangan Desimal

Berbeda dengan bilangan bulat, desimal mewakili bilangan bulat ditambah sebagian kecil dari bilangan bulat; pecahan diwakili oleh angka yang mengikuti titik desimal. Dalam database SQL, mereka ditangani oleh tipe data titik tetap dan titik mengambang.

Misalnya, jarak dari rumah saya ke toko terdekat adalah 6,7 mil; Saya dapat memasukkan 6.7 ke dalam kolom fixed-point atau floating-point tanpa keluhan dari PostgreSQL. Satu-satunya perbedaan adalah bagaimana komputer menyimpan data. Sebentar lagi, Anda akan melihat bahwa itu memiliki implikasi penting.

Real

Tipe titik tetap, juga disebut tipe presisi arbitrer, adalah numerik (presisi, skala). Anda memberikan presisi argumen sebagai jumlah digit maksimum di kiri dan kanan titik desimal, dan skala argumen sebagai jumlah digit yang diizinkan di sebelah kanan titik desimal. Sebagai alternatif, Anda dapat menentukan jenis ini menggunakan decimal (presisi,skala). Keduanya adalah bagian dari standar ANSI SQL.

Jika Anda menghilangkan menentukan nilai skala, skala akan diatur ke nol; pada dasarnya, yang menciptakan bilangan bulat. Jika Anda menghilangkan menentukan presisi dan skala, database akan menyimpan nilai presisi dan skala hingga maksimum yang diizinkan. (Itu hingga 131.072 digit sebelum titik desimal dan 16.383 digit setelah titik desimal, menurut dokumentasiPostgreSQLdi https://www.postgresql.org/docs/current/static/datatype-numeric.html.)
Misalnya, katakanlah Anda mengumpulkan total curah hujan dari beberapa bandara lokal—bukan tugas analisis data yang tidak mungkin. Layanan Cuaca Nasional A.S. menyediakan data ini dengan curah hujan yang biasanya diukur hingga dua tempat desimal. (Dan, jika Anda seperti saya, Anda memiliki ingatan jauh tentang guru matematika kelas tiga Anda yang menjelaskan bahwa dua digit setelah desimal adalah tempat keseratus.)

Untuk mencatat curah hujan dalam database menggunakan total lima digit (presisi) dan maksimum dua digit di sebelah kanan desimal (skala), Anda akan menentukannya sebagai numerik (5,2). Basis data akan selalu mengembalikan dua digit di sebelah kanan titik desimal, bahkan jika Anda tidak memasukkan angka yang berisi dua digit. Misalnya, 1,47, 1,00, dan 121,50.

Tipe floating-point

Dua tipe floating-point adalah presisi nyata dan ganda. Perbedaan antara keduanya adalah berapa banyak data yang mereka simpan. Tipe nyata memungkinkan presisi hingga enam digit desimal, dan presisi ganda hingga 15 titik desimal presisi, keduanya mencakup jumlah digit di kedua sisi titik.

Tipe floating-point ini juga disebut tipe presisi variabel. Basis data menyimpan nomor dalam bagian yang mewakili digit dan eksponen—lokasi di mana titik desimal berada. Jadi, tidak seperti numerik, di mana kita menentukan presisi dan skala tetap, titik desimal dalam kolom tertentu dapat "mengambang" tergantung pada nomornya.

Menggunakan Tipe *Real* dan *Tipe floating-point*

Setiap jenis memiliki batasan yang berbeda pada jumlah digit total, atau presisi yang dapat ditampungnya, seperti yang ditunjukkan pada Tabel 3.3.

Tabel 3.3: Tipe Real dan Tipe floating-point

Tipe data	Ukuran penyimpanan	Jenis penyimpanan	Jarak
numerik, desimal	variabel	Titik pasti	Hingga 131072 digit sebelum titik desimal; hingga 16383 digit setelah titik desimal
Real	4 byte	Titik-mengambang	6 digit desimal presisi
Double	8 byte	Titik-mengambang	15 digit desimal presisi

Untuk melihat bagaimana masing-masing dari tiga tipe data menangani angka yang sama, buat tabel kecil dan sisipkan berbagai kasus uji, seperti yang ditunjukkan dibawah ini

```
CREATE TABLE number_data_types (
      numeric_column numeric(20,5),
      real-column real,
      double_column double precision
);

INSERT INTO number_data_types
VALUES
      (.7, .7, .7),
      (2.13579, 2.13579, 2.13579),
      (2.1357987654, 2.1357987654, 2.1357987654,);

SELECT * FROM number_data_types;
```

Kami telah membuat tabel dengan satu kolom untuk masing-masing tipe data pecahan dan memuat tiga baris ke dalam tabel . Setiap baris mengulangi angka yang sama di ketiga kolom. Ketika baris terakhir skrip berjalan dan kami memilih semuanya dari tabel, kami mendapatkan yang berikut:

```
numeric_column    real_column    double_column
--------------    -----------    -------------
       0.70000            0.7              0.7
       2.13579        2.13579          2.13579
       2.13580         2.1358    2.1357987654
```

Perhatikan apa yang terjadi. Kolom numerik, diatur dengan skala lima, menyimpan lima digit setelah titik desimal terlepas dari apakah Anda memasukkan sebanyak itu atau tidak. Jika kurang dari lima, sisanya diisi dengan nol. Jika lebih dari lima, maka akan dibulatkan— seperti angka baris ketiga dengan 10 digit setelah desimal.

Kolom presisi nyata dan ganda hanya menyimpan jumlah digit yang ada tanpa bantalan. Sekali lagi pada baris ketiga, angka dibulatkan ketika dimasukkan ke dalam kolom nyata karena jenis itu memiliki presisi maksimum enam digit. Kolom presisi ganda dapat menampung hingga 15 digit, sehingga menyimpan seluruh nomor.

Masalah dengan floating-point

Jika Anda berpikir, "Nah, angka yang disimpan sebagai floating-point terlihat seperti angka yang disimpan sebagai tetap," melangkahlah dengan hati-hati. Cara komputer menyimpan angka floating-point dapat menyebabkan kesalahan matematika yang tidak diinginkan. Lihatlah apa yang terjadi ketika kita melakukan beberapa perhitungan pada angka-angka ini. Jalankan skrip di bawah ini:

```
SELECT
     numeric_column * 10000000 AS “Fixed”,
     real_column * 100000000 AS “Float”
FROM number_data_types
WHERE numeric_column = .7;
```

Di sini, kita mengalikan numeric_column dan real_column dengan 10 juta dan menggunakan klausa WHERE untuk memfilter hanya baris pertama . Kita harus mendapatkan hasil yang sama untuk kedua perhitungan, bukan? Inilah yang dikembalikan oleh kueri:

```
 Fixed              Float
 7000000.00000      6999999.88079071
```

Tidak heran tipe floating-point disebut sebagai "tidak tepat." Untung saya tidak menggunakan matematika ini untuk meluncurkan misi ke Mars atau menghitung defisit anggaran federal.

Alasan matematika floating-point menghasilkan kesalahan seperti itu adalah bahwa komputer mencoba untuk memeras banyak informasi ke dalam jumlah bit yang terbatas. Topiknya adalah subjek dari banyak tulisan dan berada di luar cakupan buku ini, tetapi jika Anda tertarik, Anda akan menemukan tautan ke sinopsis yang bagus di https://www.nostarch.com/practicalSQL/.

Penyimpanan yang dibutuhkan oleh tipe data numerik adalah variabel, dan tergantung pada presisi dan skala yang ditentukan, numerik dapat mengkonsumsi lebih banyak ruang daripada tipe floating-point. Jika Anda bekerja dengan jutaan baris, ada baiknya mempertimbangkan apakah Anda dapat hidup dengan matematika floating-point yang relatif tidak tepat.

Memilih Tipe Data Nomor Anda

Untuk saat ini, berikut adalah tiga panduan yang perlu dipertimbangkan saat Anda berurusan dengan tipe data angka:

1. Gunakan bilangan bulat jika memungkinkan. Kecuali jika data Anda menggunakan desimal, tetap gunakan tipe integer.
2. Jika Anda bekerja dengan data desimal dan membutuhkan perhitungan yang tepat (berurusan dengan uang, misalnya), pilih numerik atau yang setara, desimal. Jenis float akan menghemat ruang, tetapi ketidaktepatan matematika floating-point tidak akan berhasil di banyak aplikasi. Gunakan hanya ketika ketepatan tidak begitu penting.
3. Pilih jenis angka yang cukup besar. Kecuali jika Anda mendesain database untuk menampung jutaan baris, kesalahan di sisi yang lebih besar. Saat menggunakan angka atau desimal, atur presisi yang cukup besar untuk mengakomodasi jumlah digit di kedua sisi titik

desimal. Dengan bilangan bulat, gunakan bigint kecuali Anda benar-benar yakin nilai kolom akan dibatasi agar sesuai dengan tipe integer atau smallint yang lebih kecil.

Tanggal dan Waktu

Setiap kali Anda memasukkan tanggal ke dalam pencarian, Anda mengambil manfaat dari database yang memiliki kesadaran waktu saat ini (diterima dari server) ditambah kemampuan untuk menangani format tanggal, waktu, dan nuansa kalender, seperti sebagai tahun kabisat dan zona waktu. Ini penting untuk bercerita dengan data, karena isu tentang kapan sesuatu terjadi biasanya sama berharganya dengan pertanyaan siapa, apa, atau berapa banyak yang terlibat. Format tanggal dan waktu PostgreSQL mencakup empat tipe data utama yang ditunjukkan pada Tabel 3.4.

Tabel 3.4: Tipe Data Tanggal dan Waktu

Tipe data	**Ukuran penyimpanan**	**Deskripsi**	**Jarak**
Timestamp	8 byte	Tanggal dan waktu	4713 SM hingga 294276 M
tanggal	4 byte	Tanggal (tidak ada waktu)	4713 SM sampai 5874897 M
waktu	8 byte	Waktu (tanpa tanggal)	00:00:00 hingga 24:00:00
interval	16 byte	Jarak waktu	+/− 178.000.000 tahun

Berikut adalah tipe data untuk waktu dan tanggal di PostgreSQL:

Timestamp

Mencatat tanggal dan waktu, yang berguna untuk berbagai situasi yang mungkin Anda lacak: keberangkatan dan kedatangan penerbangan penumpang, jadwal pertandingan Major League Baseball, atau insiden di sepanjang garis waktu. Biasanya, Anda ingin menambahkan kata kunci dengan zona waktu untuk memastikan bahwa waktu yang direkam untuk suatu peristiwa mencakup zona waktu tempat peristiwa itu terjadi. Jika tidak, waktu yang tercatat di berbagai tempat di seluruh dunia menjadi tidak mungkin untuk dibandingkan. Stempel waktu format dengan zona waktu adalah bagian dari standar SQL; dengan PostgreSQL Anda dapat menentukan tipe data yang sama menggunakan timestamptz.

Tanggal Catat tanggalnya saja.

Waktu Merekam waktu saja. Sekali lagi, Anda ingin menambahkan kata kunci dengan zona waktu.

interval Memegang nilai yang mewakili satuan waktu yang dinyatakan dalam satuan format kuantitas. Itu tidak mencatat awal atau akhir periode waktu, hanya panjangnya. Contohnya termasuk 12 hari atau 8 jam.

(Dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/datatype-datetime.html mencantumkan nilai satuan mulai dari mikrodetik hingga milenium.) Anda biasanya akan menggunakan jenis ini untuk penghitungan atau pemfilteran pada kolom tanggal dan waktu lainnya.

Mari kita fokus pada stempel waktu dengan zona waktu dan tipe interval. Untuk melihat ini beraksi, jalankan skrip di bawah ini:

```
CREATE TABLE date_time_type (
timestamp_column timestamp with time zone,
   interval_column interval
);

INSERT INTO date_time_types
VALUES
('2018-12-31 01:00 EST','2 days'),
('2018-12-31 01:00 -8','1 month'),
('2018-12-31 01:00 Australia/Melbourne','1 century'),
(now(), '1 week';

SELECT * FROM date_time_types;
```

Di sini, kami membuat tabel dengan kolom untuk kedua jenis dan menyisipkan empat baris. Untuk tiga baris pertama, sisipan kami untuk timestamp_column menggunakan tanggal dan waktu yang sama (31 Desember 2018 pukul 1 pagi) menggunakan format International Organization for Standardization (ISO) untuk tanggal dan waktu: YYYY-MM-DD HH:MM: SS. SQL mendukung format tanggal tambahan (seperti MM/DD/YYYY), tetapi ISO direkomendasikan untuk portabilitas di seluruh dunia.

Mengikuti waktu, kami menentukan zona waktu tetapi menggunakan format yang berbeda di masing-masing dari tiga baris pertama: di baris pertama, kami menggunakan singkatan EST, yang merupakan Waktu Standar Timur di Amerika Serikat.

Di baris kedua, kami mengatur zona waktu dengan nilai -8. Itu menunjukkan jumlah perbedaan jam, atau offset, dari Coordinated Universal Time (UTC). UTC mengacu pada standar waktu dunia secara keseluruhan serta nilai UTC +/− 00:00, zona waktu yang mencakup Inggris Raya dan Afrika Barat.(Untuk peta zona waktu UTC, lihat https://en.wikipedia.org/wiki/Coordinated_Universal_Time#/media/File:Standard_World_Time_Zones.png

Menggunakan nilai -8 menentukan zona waktu delapan jam di belakang UTC, yang merupakan zona waktu Pasifik di Amerika Serikat dan Kanada.

Untuk baris ketiga, kami menentukan zona waktu menggunakan nama area dan lokasi: Australia/Melbourne. Format tersebut menggunakan nilai yang ditemukan dalam database zona waktu standar yang sering digunakan dalam pemrograman komputer. Anda dapat mempelajari lebih lanjut tentang basis data zona waktu di https://en.wikipedia.org/wiki/Tz_database.

Di baris keempat, alih-alih menentukan tanggal, waktu, dan zona waktu, skrip menggunakan fungsi Now() PostgreSQL , yang menangkap waktu transaksi saat ini dari perangkat keras Anda. Setelah skrip berjalan, hasilnya akan terlihat seperti (tetapi tidak persis seperti) ini:

```
timestamp_column             interval_column
-----------------------      -----------------
2018-12-31 01:00:00-05       2 days
2018-12-31 01:00:00-05       1 month
2018-12-31 01:00:00-05       100 years
2018-01-25 21:31:15.716063-05 7 days
```

fdMeskipun kami memberikan tanggal dan waktu yang sama di tiga baris pertama pada timestamp_column, output setiap baris berbeda. Alasannya adalah pgAdmin melaporkan tanggal dan waktu relatif terhadap zona waktu saya, yang dalam hasil yang ditunjukkan ditunjukkan oleh offset UTC -05 di akhir setiap cap waktu. Offset UTC -05 berarti lima jam di belakang waktu UTC, setara dengan zona waktu Timur AS, tempat saya tinggal. Jika Anda tinggal di zona waktu yang berbeda, Anda mungkin akan melihat offset yang berbeda; waktu dan tanggal juga mungkin berbeda dari yang ditampilkan di sini. Kita dapat mengubah cara PostgreSQL melaporkan nilai stempel waktu ini, dan saya akan membahas cara melakukannya ditambah tip lain untuk tanggal dan waktu di Bab selanjutnya

Terakhir, interval_column menunjukkan nilai yang Anda masukkan. PostgreSQL mengubah 1 abad menjadi 100 tahun dan 1 minggu menjadi 7 hari karena pengaturan default yang disukai untuk tampilan interval. Baca bagian "Input Interval" dari dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/datatype-datetime.html untuk mempelajari lebih lanjut tentang opsi yang terkait dengan interval.

Menggunakan Interval Tipe Data dalam Perhitungan

Tipe data interval berguna untuk perhitungan yang mudah dipahami pada data tanggal dan waktu. Misalnya, katakanlah Anda memiliki kolom yang berisi tanggal klien menandatangani kontrak. Dengan menggunakan data interval, Anda dapat menambahkan 90 hari ke setiap tanggal kontrak untuk menentukan kapan harus menindaklanjuti dengan klien.

Untuk melihat cara kerja tipe data interval, kita akan menggunakan tabel date_time_types yang baru saja kita buat, seperti yang ditunjukkan pada Listing dibawah ini:

```
SELECT
timestamp_column,
interval_column,
timestamp_column - interval_column AS new_date
FROM date_time_types;
```

Ini adalah pernyataan SELECT yang khas kecuali kita akan menghitung kolom bernama new_date yang berisi hasil timestamp_column dikurangi interval_column. (Kolom yang dihitung disebut ekspresi; kami akan sering menggunakan teknik ini.) Di setiap baris, kami mengurangi unit waktu yang ditunjukkan oleh tipe data interval dari tanggal. Ini menghasilkan hasil berikut:

```
timestamp_column             interval_column   new_date
----------------------------------------------------------------
2018-12-31 01:00:00-05       2 days            2018-12-29 01:00:00-05
```

```
2018-12-31 01:00:00-05        1 month                 2018-11-30 04:00:00-05
2018-12-31 01:00:00-05        100 years               2018-12-30 09:00:00-05
2018-01-25 21:31:15.716063-05 7 days                  2018-01-18 21:31:15.716063-05
```

Perhatikan bahwa kolom new_date secara default diformat sebagai stempel waktu tipe dengan zona waktu, memungkinkan tampilan nilai waktu serta tanggal jika nilai interval menggunakannya. Sekali lagi, output Anda mungkin berbeda berdasarkan zona waktu Anda.

Jenis Lain-lain

Jenis karakter, angka, dan tanggal/waktu yang telah Anda pelajari sejauh ini kemungkinan akan menjadi bagian terbesar dari pekerjaan yang Anda lakukan dengan SQL. Tetapi PostgreSQL mendukung banyak jenis tambahan, termasuk tetapi tidak terbatas pada:

- Tipe Boolean yang menyimpan nilai true atau false
- Jenis geometris yang meliputi titik, garis, lingkaran, dan objek dua dimensi lainnya
- Jenis alamat jaringan, seperti alamat IP atau MAC
- Tipe Universally Unique Identifier (UUID), terkadang digunakan sebagai nilai kunci unik dalam tabel
- Tipe data XML dan JSON yang menyimpan informasi dalam format terstruktur tersebut. Saya akan membahas tipe ini seperti yang diperlukan di seluruh buku ini.

Mengubah Nilai dari Satu Jenis ke Jenis Lainnya dengan CAST

Kadang-kadang, Anda mungkin perlu mengubah nilai dari tipe data yang disimpan ke tipe lain; misalnya, ketika Anda mengambil nomor sebagai karakter sehingga Anda dapat menggabungkannya dengan teks atau ketika Anda harus memperlakukan tanggal yang disimpan sebagai karakter sebagai tipe tanggal yang sebenarnya sehingga Anda dapat mengurutkannya dalam urutan tanggal atau melakukan perhitungan interval. Anda dapat melakukan konversi ini menggunakan fungsi CAST().

Fungsi CAST() hanya berhasil bila tipe data target dapat mengakomodasi nilai aslinya. Casting integer sebagai teks dimungkinkan, karena tipe karakter dapat menyertakan angka. Casting teks dengan huruf alfabet sebagai nomor tidak.

Listing dibawah ini memiliki tiga contoh menggunakan tiga tabel tipe data yang baru saja kita buat. Dua contoh pertama berfungsi, tetapi contoh ketiga akan mencoba melakukan konversi jenis yang tidak valid sehingga Anda dapat melihat seperti apa kesalahan pengecoran jenis.

```
SELECT timestamp_column, CAST(timestamp_column AS varchar(10))
FROM date_time_types;

SELECT numeric_column,
   CAST(numeric_column AS integer),
   CAST(numeric_column AS varchar(6)),
FROM number_date_types;

SELECT CAST(char_columnAS integer) FROM char_data_types;
```

Pernyataan SELECT pertama mengembalikan nilai timestamp_column sebagai varchar, yang akan Anda ingat adalah kolom karakter dengan panjang variabel. Dalam hal ini, saya telah mengatur panjang karakter menjadi 10, yang berarti ketika dikonversi ke string karakter, hanya 10 karakter pertama yang disimpan. Itu berguna dalam kasus ini, karena itu hanya memberi kita segmen tanggal kolom dan mengecualikan waktu. Tentu saja, ada cara yang lebih baik untuk menghapus waktu dari stempel waktu, dan saya akan membahasnya di "Mengekstrak Komponen Nilai stempel waktu" di bab selanjutnya.

Pernyataan SELECT kedua mengembalikan numeric_column tiga kali: dalam bentuk aslinya dan kemudian sebagai bilangan bulat dan sebagai karakter. Setelah konversi ke bilangan bulat, PostgreSQL membulatkan nilainya menjadi bilangan bulat. Tetapi dengan konversi varchar, tidak ada pembulatan yang terjadi: nilainya hanya diiris pada karakter keenam.

SELECT terakhir tidak berfungsi: SELECT mengembalikan kesalahan sintaks input yang tidak valid untuk bilangan bulat karena huruf tidak dapat menjadi bilangan bulat!

Notasi Pintasan CAST

Itu selalu yang terbaik untuk menulis SQL yang dapat dibaca oleh orang lain yang mungkin mengambilnya nanti, dan cara CAST() ditulis membuat apa yang Anda maksudkan saat Anda menggunakannya cukup jelas. Namun, PostgreSQL juga menawarkan notasi pintasan yang kurang jelas yang membutuhkan lebih sedikit ruang: titik dua ganda.

Sisipkan titik dua di antara nama kolom dan tipe data yang ingin Anda ubah. Misalnya, dua pernyataan ini menampilkan timestamp_column sebagai varchar:

```
SELECT timestamp_column, CAST(timestamp_column AS varchar(10))
FROM date_time_types;

SELECT timestamp_column::varchar(10)
FROM date_time_types;
```

Gunakan mana saja yang cocok untuk Anda, tetapi perlu diketahui bahwa titik dua ganda adalah implementasi khusus PostgreSQL yang tidak ditemukan di varian SQL lainnya.

Anda sekarang diperlengkapi untuk lebih memahami nuansa format data yang Anda temui saat menggali database. Jika Anda menemukan nilai moneter yang disimpan sebagai angka floating-point, Anda pasti akan mengubahnya menjadi desimal sebelum melakukan matematika apa pun. Dan Anda akan tahu cara menggunakan jenis kolom teks yang tepat agar database Anda tidak tumbuh terlalu besar.

Selanjutnya, saya akan melanjutkan dengan dasar-dasar SQL dan menunjukkan kepada Anda cara mengimpor data eksternal ke dalam database Anda.

Cobalah sendiri

Lanjutkan menjelajahi tipe data dengan latihan berikut:

1. Perusahaan Anda mengirimkan buah dan sayuran ke toko bahan makanan lokal, dan Anda perlu melacak jarak tempuh yang ditempuh oleh setiap pengemudi setiap hari

hingga sepersepuluh mil. Dengan asumsi tidak ada pengemudi yang akan melakukan perjalanan lebih dari 999 mil dalam sehari, apa tipe data yang sesuai untuk kolom jarak tempuh di tabel Anda? Mengapa?

2. Pada tabel yang mencantumkan setiap pengemudi di perusahaan Anda, tipe data apa yang sesuai untuk nama depan dan belakang pengemudi? Mengapa merupakan ide yang baik untuk memisahkan nama depan dan belakang menjadi dua kolom daripada memiliki satu kolom nama yang lebih besar?
3. Asumsikan Anda memiliki kolom teks yang menyertakan string yang diformat sebagai tanggal. Salah satu string ditulis sebagai '4//2017'. Apa yang akan terjadi ketika Anda mencoba mengonversi string itu ke tipe data stempel waktu?

BAB IV
IMPOR DAN EKSPOR DATA

Sejauh ini, Anda telah mempelajari cara menambahkan beberapa baris ke tabel menggunakan pernyataan SQL INSERT. Sisipan baris demi baris berguna untuk membuat tabel uji cepat atau menambahkan beberapa baris ke tabel yang ada. Tetapi kemungkinan besar Anda harus memuat ratusan, ribuan, atau bahkan jutaan baris, dan tidak ada yang ingin menulis pernyataan INSERT terpisah dalam situasi tersebut. Untungnya, Anda tidak perlu melakukannya.

Jika data Anda ada dalam file teks yang dibatasi (dengan satu baris tabel per baris teks dan setiap nilai kolom dipisahkan dengan koma atau karakter lain) PostgreSQL dapat mengimpor data secara massal melalui perintah COPY-nya. Perintah ini adalah implementasi khusus PostgreSQL dengan opsi untuk menyertakan atau mengecualikan kolom dan menangani berbagai jenis teks yang dibatasi.

Sebaliknya, COPY juga akan mengekspor data dari tabel PostgreSQL atau dari hasil kueri ke file teks yang dibatasi. Teknik ini berguna saat Anda ingin berbagi data dengan rekan kerja atau memindahkannya ke format lain, seperti file Excel.

Saya secara singkat menyentuh SALIN untuk ekspor di "Karakter" di halaman 24, tapi dalam bab ini saya akan membahas impor dan ekspor secara lebih mendalam. Untuk mengimpor, saya akan mulai dengan memperkenalkan Anda ke salah satu kumpulan data favorit saya: penghitungan populasi Sensus AS Tahunan menurut wilayah.

Tiga langkah membentuk garis besar sebagian besar impor yang akan Anda lakukan:
1. Siapkan data sumber dalam bentuk file teks yang dibatasi.
2. Buat tabel untuk menyimpan data.
3. Tulis skrip SALIN untuk melakukan impor.

Setelah impor selesai, kami akan memeriksa data dan melihat opsi tambahan untuk mengimpor dan mengekspor.

File teks yang dibatasi adalah format file paling umum yang portabel di seluruh sistem berpemilik dan sumber terbuka, jadi kami akan fokus pada jenis file tersebut. Jika Anda ingin mentransfer data dari format kepemilikan program database lain langsung ke PostgreSQL, seperti Microsoft Access atau MySQL, Anda harus menggunakan alat pihak ketiga. Periksa wiki PostgreSQL di https://wiki.postgresql.org/wiki/ dan cari "Mengonversi dari Database lain ke PostgreSQL" untuk daftar alat.
Jika Anda menggunakan SQL dengan pengelola database lain, periksa dokumentasi database lain untuk mengetahui cara menangani impor massal. Database MySQL, misalnya, memiliki pernyataan LOAD DATA INFILE, dan Microsoft SQL Server memiliki perintah BULK INSERT sendiri.

Bekerja dengan File Teks Dibatasi

Banyak aplikasi perangkat lunak menyimpan data dalam format yang unik, dan menerjemahkan satu format data ke format data lainnya semudah mencoba membaca alfabet Cyrillic jika mereka hanya mengerti bahasa Inggris. Untungnya, sebagian besar perangkat lunak dapat mengimpor dari dan mengekspor ke file teks terbatas, yang merupakan format data umum yang berfungsi sebagai jalan tengah.

File teks yang dipisahkan berisi baris data, dan setiap baris mewakili satu baris dalam tabel. Di setiap baris, karakter memisahkan, atau membatasi, setiap kolom data. Saya telah melihat semua jenis karakter yang digunakan sebagai pembatas, dari ampersand hingga pipa, tetapi koma paling sering digunakan; maka nama jenis file yang akan sering Anda lihat: nilai yang dipisahkan koma (CSV). Istilah CSV dan dipisahkan koma dapat dipertukarkan.

Berikut adalah baris data umum yang mungkin Anda lihat dalam file yang dipisahkan koma:

```
John, Doe, 123 Main St., Hyde Park, NY, 845-555-1212
```

Perhatikan bahwa koma memisahkan setiap bagian data—nama depan, nama belakang, jalan, kota, negara bagian, dan telepon—tanpa spasi. Koma memberitahu perangkat lunak untuk memperlakukan setiap item sebagai kolom terpisah, baik pada saat impor atau ekspor. Cukup sederhana.

Mengutip Kolom yang Mengandung Pembatas

Menggunakan koma sebagai pembatas kolom menyebabkan potensi dilema: bagaimana jika nilai dalam kolom menyertakan koma? Misalnya, terkadang orang menggabungkan nomor apartemen dengan alamat jalan, seperti di 123 Main St., Apartment 200. Kecuali jika sistem untuk membatasi akun untuk koma tambahan itu, selama impor baris akan tampak memiliki kolom tambahan dan menyebabkan impor gagal.

Untuk menangani kasus seperti itu, file delimited membungkus kolom yang berisi karakter pembatas dengan karakter arbitrer yang disebut kualifikasi teks yang memberitahu SQL untuk mengabaikan karakter pembatas yang ada di dalamnya. Sebagian besar waktu dalam file yang dibatasi koma, kualifikasi teks yang digunakan adalah tanda kutip ganda. Berikut contoh baris data lagi, tetapi dengan nama jalan yang diapit oleh tanda kutip ganda:

```
John, Doe, “123 Main St., Apartement 200”, Hyde Park, NY,845-555-1212
```

Saat impor, database akan mengenali bahwa tanda kutip ganda menandakan satu kolom terlepas dari apakah ia menemukan pembatas di dalam tanda kutip. Saat mengimpor file CSV, PostgreSQL secara default mengabaikan pembatas di dalam kolom kutip ganda, tetapi Anda dapat menentukan qualifier teks yang berbeda jika impor Anda memerlukannya. (Dan, mengingat terkadang pilihan aneh yang dibuat oleh para profesional TI, Anda mungkin memang perlu menggunakan karakter yang berbeda.)

Menangani Baris Header

Fitur lain yang sering Anda temukan di dalam file teks yang dibatasi adalah baris header. Seperti namanya, ini adalah satu baris di bagian atas, atau kepala, file yang mencantumkan nama setiap bidang data. Biasanya, header dibuat selama ekspor data dari database. Berikut ini contoh dengan baris terbatas yang saya gunakan:

```
FIRSTNAME,LASTNAME,STREET,CITY,STATE,PHONE
John, Doe, “123 Main St., Apartement 200”, Hyde Park, NY,845-555-1212
```

Baris header memiliki beberapa tujuan. Pertama, nilai di baris header mengidentifikasi data di setiap kolom, yang sangat berguna saat Anda menguraikan konten file. Kedua, beberapa manajer database (walaupun bukan PostgreSQL) menggunakan baris header untuk memetakan kolom dalam file yang dibatasi ke kolom yang benar di tabel impor. Karena PostgreSQL tidak menggunakan baris header, kami tidak ingin baris tersebut diimpor ke tabel, jadi kami akan menggunakan opsi HEADER dalam perintah COPY untuk mengecualikannya. Saya akan membahas ini dengan semua opsi SALIN di bagian selanjutnya.

Menggunakan COPY untuk Mengimpor Data

Untuk mengimpor data dari file eksternal ke database kita, pertama kita perlu memeriksa file CSV sumber dan membuat tabel di PostgreSQL untuk menyimpan data. Setelah itu, pernyataan SQL untuk impor relatif sederhana. Yang Anda butuhkan hanyalah tiga baris kode .

```
COPY table_name
FROM ‘C:\YourDirectory\your_file.csv’
WITH (FORMAT CSV, HEADER);
```

Blok kode dimulai dengan kata kunci COPY diikuti dengan nama tabel target, yang harus sudah ada di database Anda. Pikirkan sintaks ini sebagai makna, "Salin data ke tabel saya yang disebut table_name."

Kata kunci FROM mengidentifikasi path lengkap ke file sumber, termasuk namanya. Cara Anda menentukan jalur tergantung pada sistem operasi Anda. Untuk Windows, mulai dengan huruf drive, titik dua, garis miring terbalik, dan nama direktori. Misalnya, untuk mengimpor file yang terletak di desktop Windows saya, baris FROM akan berbunyi:

```
FROM ‘C:\Users\Anthony\Desktop\my_file.csv’
```

Di macOS atau Linux, mulai dari direktori root sistem dengan garis miring dan lanjutkan dari sana. Inilah yang mungkin terlihat seperti garis FROM saat mengimpor file yang terletak di desktop Mac saya:

```
FROM ‘\Users\Anthony\Desktop\my_file.csv’
```

Perhatikan bahwa dalam kedua kasus, path lengkap dan nama file diapit oleh tanda kutip tunggal. Untuk contoh dalam buku ini, saya menggunakan jalur gaya Windows C:\YourDirectory\ sebagai pengganti. Ganti itu dengan jalur tempat Anda menyimpan file.

Kata kunci WITH memungkinkan Anda menentukan opsi, dikelilingi oleh tanda kurung, yang dapat Anda sesuaikan dengan file input atau output Anda. Di sini kita menentukan bahwa file eksternal harus dibatasi koma, dan kita harus mengecualikan baris header file dalam impor. Sebaiknya periksa semua opsi dalam dokumentasi PostgreSQL resmi di https://www.postgresql.org/docs/current/ static/sql-copy.html, tetapi berikut adalah daftar opsi yang biasa Anda gunakan:

Format file input dan output

Gunakan opsi FORMAT format_name untuk menentukan jenis file yang sedang Anda baca atau tulis. Nama formatnya adalah CSV, TEXT, atau BINARY. Kecuali Anda mendalami membangun sistem teknis, Anda akan jarang menemukan kebutuhan untuk bekerja dengan BINARY, di mana data disimpan sebagai urutan byte. Lebih sering, Anda akan bekerja dengan file CSV standar. Dalam format TEXT, karakter tab adalah pembatas secara default (walaupun Anda dapat menentukan karakter lain) dan karakter garis miring terbalik seperti \r dikenali sebagai padanan ASCII dalam hal ini, carriage return. Format TEXT digunakan terutama oleh program pencadangan bawaan PostgreSQL.

HEADER

Saat mengimpor, gunakan HEADER untuk menentukan bahwa file sumber memiliki baris header. Anda juga dapat menentukannya secara langsung sebagai HEADER ON, yang memberi tahu database untuk mulai mengimpor dengan baris kedua file, mencegah impor header yang tidak diinginkan. Anda tidak ingin nama kolom di header menjadi bagian dari data dalam tabel. Pada ekspor, menggunakan HEADER memberitahu database untuk memasukkan nama kolom sebagai baris header dalam file output, yang biasanya berguna untuk dilakukan.

Pembatas/DELIMITER

Opsi 'karakter' DELIMITER memungkinkan Anda menentukan karakter mana yang digunakan file impor atau ekspor sebagai pembatas. Pembatas harus berupa karakter tunggal dan tidak boleh berupa carriage return. Jika Anda menggunakan FORMAT CSV, pembatas yang diasumsikan adalah koma. Saya menyertakan DELIMITER di sini untuk menunjukkan bahwa Anda memiliki opsi untuk menentukan pembatas yang berbeda jika begitulah cara data Anda tiba. Misalnya, jika Anda menerima data yang dibatasi pipa, Anda akan memperlakukan opsi seperti ini: DELIMITER '|'.

Kutipan Karakter/Quote

Sebelumnya, Anda mengetahui bahwa dalam CSV, koma di dalam satu nilai kolom akan mengacaukan impor Anda kecuali jika nilai kolom dikelilingi oleh karakter yang berfungsi sebagai kualifikasi teks, memberi tahu database untuk menangani nilai di dalam sebagai satu kolom. Secara default, PostgreSQL menggunakan tanda kutip ganda, tetapi jika CSV yang Anda impor menggunakan karakter yang berbeda, Anda dapat menentukannya dengan opsi QUOTE 'quote_character'. Sekarang setelah Anda lebih memahami file yang dibatasi, Anda siap untuk mengimpornya.

Mengimpor Data Sensus yang Menggambarkan Kabupaten

Kumpulan data yang akan Anda gunakan dalam latihan impor ini jauh lebih besar daripada tabel guru yang Anda buat di Bab 1. Ini berisi data sensus tentang setiap daerah di Amerika Serikat dan memiliki kedalaman 3.143 baris dan lebar 91 kolom.

Untuk memahami data, ada baiknya mengetahui sedikit tentang Sensus AS. Setiap 10 tahun, pemerintah melakukan penghitungan penduduk secara penuh—salah satu dari beberapa program yang sedang berlangsung oleh Biro Sensus untuk mengumpulkan data demografis. Setiap rumah tangga di Amerika menerima kuesioner tentang setiap orang di dalamnya—usia, jenis kelamin, ras, dan apakah mereka orang Hispanik atau bukan. Konstitusi AS mengamanatkan penghitungan untuk menentukan berapa banyak anggota dari setiap negara bagian yang membentuk Dewan Perwakilan Rakyat AS. Berdasarkan Sensus 2010, misalnya, Texas memperoleh empat kursi di DPR sementara New York dan Ohio masing-masing kehilangan dua kursi. Meskipun pembagian kursi DPR adalah tujuan utama penghitungan, data ini juga merupakan keuntungan bagi pelacak tren yang mempelajari populasi. Sebuah sinopsis yang baik dari temuan hitungan 2010 tersedia di https://www.census.gov/prod/cen2010/briefs/c2010br-01.pdf.

Biro Sensus melaporkan total populasi keseluruhan dan jumlah berdasarkan ras dan etnis untuk berbagai geografi termasuk negara bagian, kabupaten, kota, tempat, dan distrik sekolah. Untuk latihan ini, saya menyusun kumpulan kolom terpilih untuk penghitungan tingkat kabupaten Sensus 2010 ke dalam file bernama us_counties_2010.csv. Unduh file us_counties_2010.csv dari https://www.nostarch.com/practicalSQL/ dan simpan ke folder di komputer Anda.

Buka file dengan editor teks biasa. Anda akan melihat baris header yang dimulai dengan kolom berikut:

```
NAME,STUSAB,SUMLEV,REGION,DIVISION,STATE,COUNTY --snip--
```

Mari kita jelajahi beberapa kolom dengan memeriksa kode untuk membuat tabel impor.

Membuat Tabel us_counties_2010

Kode dalam Daftar 4-2 hanya menampilkan versi singkat dari skrip CREATE TABLE; banyak kolom telah dihilangkan. Versi lengkap tersedia (dan diberi keterangan) bersama dengan semua contoh kode di sumber buku. Untuk mengimpornya dengan benar, Anda harus mengunduh definisi tabel lengkap.

```
CREATE TABLE us_counties_2000 (
geo_name varchar(90),
state_us_abbrevitation varchar(2),
summary_level varchar(3),
region smallint,
division smallint,
state_fips varchar(2),
country_fips varchar(3),
   area_land bigint,
   area_water bigint,
   population_count_100_percent integer,
   housing_unit_count_100_percent integer,
   internal_point_lat numeric(10,7),
```

```
internal_point_lon numeric(10,7),
p0010001 integer,
p0010002 integer,
p0010003 integer,
p0010004 integer,
p0010005 integer,
--snip--
p0040049 integer,
p0040065 integer,
p0040072 integer,
p0010001 integer,
p0010002 integer,
p0010003 integer,
);
```

Untuk membuat tabel, di pgAdmin klik database analisis yang Anda buat di Bab 1. (Sebaiknya simpan data dalam buku ini dalam analisis karena kami akan menggunakannya kembali di bab selanjutnya.) Dari bilah menu pgAdmin, pilih Alat Alat Kueri. Rekatkan skrip ke jendela dan jalankan.
Kembali ke jendela pgAdmin utama, dan di browser objek, klik kanan dan segarkan database analisis. Pilih Skema > publik > Tabel untuk melihat tabel baru. Meskipun kosong, Anda dapat melihat strukturnya dengan menjalankan kueri SELECT dasar di Alat Kueri pgAdmin:

```
SELECT * from us_counties_2010;
```

Saat Anda menjalankan kueri SELECT, Anda akan melihat kolom dalam tabel yang Anda buat. Belum ada baris data.

Kolom Sensus dan Tipe Data

Sebelum kita mengimpor file CSV ke dalam tabel,. Sebagai panduan saya, saya menggunakan kamus data sensus resmi untuk kumpulan data ini yang ditemukan di http://www.census.gov/prod/cen2010/doc/pl94-171.pdf meskipun saya memberikan beberapa kolom nama yang lebih mudah dibaca dalam tabel definisi. Mengandalkan kamus data bila memungkinkan adalah praktik yang baik, karena membantu Anda menghindari kesalahan konfigurasi kolom atau potensi kehilangan data. Selalu tanyakan apakah ada yang tersedia, atau lakukan pencarian online jika datanya bersifat publik.

Dalam kumpulan data sensus ini, dan dengan demikian tabel yang baru saja Anda buat, setiap baris menggambarkan demografi satu kabupaten, dimulai dengan geo_name dan singkatan negara bagian dua karakternya, state_us_abbreviation ❷. Karena keduanya adalah teks, kami menyimpannya sebagai varchar. Kamus data menunjukkan bahwa panjang maksimum bidang geo_name adalah 90 karakter, tetapi karena sebagian besar nama lebih pendek, menggunakan varchar akan menghemat ruang jika kita mengisi bidang dengan nama yang lebih pendek, seperti Lee County, sementara memungkinkan kita untuk menentukan maksimum 90 karakter.

Geografi, atau tingkat ringkasan, yang diwakili oleh setiap baris dijelaskan oleh tingkat_ringkasan. Kami hanya bekerja dengan data tingkat kabupaten, jadi kodenya sama untuk setiap baris: 050. Meskipun kode itu menyerupai angka, kami memperlakukannya sebagai teks dengan menggunakan varchar lagi. Jika kita menggunakan tipe integer, awalan 0 akan dihapus saat impor, menyisakan 50. Kita tidak ingin melakukannya karena 050 adalah

kode level ringkasan lengkap, dan kita akan mengubah arti data jika 0 terkemuka hilang. Juga, kami tidak akan melakukan matematika dengan nilai ini.

Angka dari 0 sampai 9 di region dan divisi mewakili lokasi sebuah county di Amerika Serikat, seperti Northeast, Midwest, atau South Atlantic. Tidak ada angka yang lebih tinggi dari 9, jadi kami mendefinisikan kolom dengan tipe smallint. Kami kembali menggunakan varchar untuk state_fips dan county_fips, yang merupakan kode federal standar untuk entitas tersebut, karena kode tersebut berisi nol di depan yang tidak boleh dihilangkan. Selalu penting untuk membedakan kode dari angka; nilai negara bagian dan kabupaten ini sebenarnya adalah label yang bertentangan dengan angka yang digunakan untuk matematika.
Jumlah meter persegi untuk tanah dan air di kabupaten dicatat di area_tanah dan area_air, masing-masing. Di tempat-tempat tertentu 'seperti Alaska' di mana ada banyak tanah yang harus dilalui dengan semua salju itu beberapa nilai dengan mudah melampaui nilai maksimum tipe integer 2.147.483.648. Untuk alasan itu, kami menggunakan bigint, yang akan menangani 376.855.656.455 meter persegi di Area Sensus Yukon-Koyukuk dengan ruang kosong.

Selanjutnya, population_count_100_percent dan housing_unit_count_100_percent adalah jumlah total populasi dan unit perumahan dalam geografi. Pada tahun 2010, Amerika Serikat memiliki 308,7 juta orang dan 131,7 juta unit rumah. Populasi dan unit perumahan untuk daerah mana pun sangat cocok dengan batas tipe data bilangan bulat, jadi kami menggunakannya untuk keduanya.
Lintang dan bujur suatu titik di dekat pusat county, yang disebut titik internal, ditentukan dalam internal_point_lat dan internal_point_lon, masing-masing. Biro Sensus—bersama dengan banyak sistem pemetaan—mengungkapkan koordinat lintang dan bujur menggunakan sistem derajat desimal. Lintang mewakili posisi utara dan selatan di dunia, dengan khatulistiwa pada 0 derajat, Kutub Utara pada 90 derajat, dan Kutub Selatan pada 90 derajat.

Bujur mewakili lokasi timur dan barat, dengan Meridian Utama yang melewati Greenwich di London pada 0 derajat bujur. Dari sana, bujur meningkat baik timur dan barat (angka positif ke timur dan negatif ke barat) sampai mereka bertemu pada 180 derajat di sisi berlawanan dari dunia. Lokasi di sana, yang dikenal sebagai antimeridian, digunakan sebagai dasar untuk Garis Tanggal Internasional.

Saat melaporkan titik interior, Biro Sensus menggunakan hingga tujuh tempat desimal. Dengan nilai hingga 180 di sebelah kiri desimal, kita perlu menghitung total maksimum 10 digit. Jadi, kami menggunakan numerik dengan presisi 10 dan skala 7.

Catatan: PostgreSQL, melalui ekstensi PostGIS, dapat menyimpan data geometris, yang mencakup titik-titik yang mewakili garis lintang dan garis bujur dalam satu kolom. Kami akan menjelajahi data geometris saat kami membahas kueri geografis di Bab 14.

Akhirnya, kami mencapai serangkaian kolom yang berisi iterasi jumlah populasi menurut ras dan etnis untuk county serta jumlah unit perumahan. Kumpulan lengkap data Sensus 2010 berisi 291 kolom ini. Saya telah menguranginya menjadi 78 untuk latihan ini, menghilangkan banyak kolom untuk membuat kumpulan data lebih ringkas untuk latihan ini.

Saya tidak akan membahas semua kolom sekarang, tetapi Tabel 4.1 menunjukkan contoh kecil.

Tabel 4.1: Kolom Penghitungan Penduduk Sensus

Nama kolom	Deskripsi
p0010001	Jumlah penduduk
p0010002	Populasi satu ras
p0010003	Populasi satu ras: Putih saja
p0010004	Populasi satu ras: Hitam atau Afrika Amerika saja
p0010005	Populasi satu ras: Indian Amerika dan Penduduk Asli Alaska saja
p0010006	Populasi satu ras: Asia saja
p0010007	Populasi satu ras: Penduduk Asli Hawaii dan Penduduk Kepulauan Pasifik Lainnya saja
p0010008	Populasi satu ras: Beberapa Ras Lain saja

Anda akan menjelajahi data ini lebih lanjut di bab berikutnya ketika kita melihat matematika dengan SQL. Untuk saat ini, mari kita jalankan impor.

Melakukan Sensus Impor dengan COPY

Sekarang Anda siap untuk membawa data sensus ke dalam tabel. Jalankan kode di Listing dibawah ini, ingat untuk mengubah path ke file agar sesuai dengan lokasi data di komputer Anda:

```
COPY us_counties_2010
FROM 'C:\YourDirectory\us_countiess_2010.csv'
WITH (FORMAT CSV. HEADER);
```

Saat kode dijalankan, Anda akan melihat pesan berikut di pgAdmin:

```
Query returned successfully: 3143 rows affected
```

Itu kabar baik: CSV impor memiliki jumlah baris yang sama. Jika Anda memiliki masalah dengan CSV sumber atau pernyataan impor Anda, database akan menampilkan kesalahan. Misalnya, jika salah satu baris di CSV memiliki lebih banyak kolom daripada di tabel target, Anda akan melihat pesan kesalahan yang memberikan petunjuk tentang cara memperbaikinya:

```
ERROR: Extra data after last expected column
SQL state: 22P04
Context:      COPY      us_counties_2010,      line      2:      Autauga
County,AL,050,3,5,6,01..."
```

Meskipun tidak ada kesalahan yang dilaporkan, ada baiknya untuk memindai secara visual data yang baru saja Anda impor untuk memastikan semuanya terlihat seperti yang diharapkan. Mulailah dengan kueri SELECT dari semua kolom dan baris:

```
SELECT * FROM us_counties_2010;
```

Seharusnya ada 3.143 baris yang ditampilkan di pgAdmin, dan saat Anda menggulir ke kiri dan kanan melalui kumpulan hasil, setiap bidang harus memiliki nilai yang diharapkan. Mari kita

tinjau beberapa kolom yang sangat hati-hati kami definisikan dengan tipe data yang sesuai. Misalnya, jalankan kueri berikut untuk memperlihatkan kabupaten dengan nilai area_land terbesar. Kami akan menggunakan klausa LIMIT, yang akan menyebabkan kueri hanya mengembalikan jumlah baris yang kami inginkan; di sini, kami akan meminta tiga:

```
SELECT geo_name, state_us_abbrevitation, area_land
FROM us_counties_2010
ORDER BY area_land DESC
LIMIT 3;
```

Kueri ini memberi peringkat geografi tingkat kabupaten dari luas daratan terbesar hingga terkecil dalam meter persegi. Kami mendefinisikan area_land sebagai bigint karena nilai terbesar di lapangan lebih besar dari kisaran atas yang disediakan oleh bilangan bulat biasa. Seperti yang Anda duga, geografi besar Alaska berada di urutan teratas:

```
geo_name             state_us_abbreviation   area_land
-----------------------  ---------------------   ------------
Yukom-Koyoku Census Area  AK                  376855656455
North Slope Borough       AK                  229720054439
Bethel Census Area        AK                  105075822708
```

Selanjutnya, periksa kolom lintang dan bujur dari internal_point_lat dan internal_point_lon, yang kita definisikan dengan numerik (10,7). Kode ini mengurutkan kabupaten berdasarkan garis bujur dari nilai terbesar hingga terkecil. Kali ini, kami akan menggunakan LIMIT untuk mengambil lima baris:

```
SELECT geo_name, state_us_abbreviation, internal_point_lon
FROM us_counties_2010
ORDER BY internal_point_lon DESC
LIMIT 5;
```

Garis bujur mengukur lokasi dari timur ke barat, dengan lokasi di sebelah barat Meridian Utama di Inggris direpresentasikan sebagai angka negatif yang dimulai dengan 1, 2, 3, dan seterusnya semakin jauh ke barat Anda pergi. Kami mengurutkan dalam urutan menurun, jadi kami berharap kabupaten paling timur di Amerika Serikat muncul di bagian atas hasil kueri. Ada satu-satunya geografi Alaska di bagian atas:

```
geo_name             state_us_abbreviation   internal_point_lon
-------------------------  ---------------------   ------------------
Aleutians West Census Area AK                      178.3388130
Washington County          ME                      -67.6093542
Hancock County             ME                      -68.3707034
Aroostook County           ME                      -68.6494098
Penobscot County           ME                      -68.6574869
```

Inilah alasannya: Kepulauan Aleutian Alaska membentang begitu jauh ke barat (lebih jauh ke barat dari Hawaii) sehingga mereka melintasi antimeridian pada 180 derajat bujur dengan kurang dari 2 derajat. Setelah melewati antimeridian, garis bujur berubah menjadi positif, menghitung mundur ke 0. Untungnya, data tersebut tidak salah; namun, itu adalah fakta yang dapat Anda simpan untuk kompetisi tim trivia berikutnya.

Selamat! Anda memiliki kumpulan data demografis pemerintah yang sah di database Anda. Saya akan menggunakannya untuk mendemonstrasikan mengekspor data dengan COPY nanti

di bab ini, dan kemudian Anda akan menggunakannya untuk mempelajari fungsi matematika di Bab 5. Sebelum kita beralih ke mengekspor data, mari kita periksa beberapa teknik pengimporan tambahan.

Mengimpor Subset Kolom dengan COPY

Jika file CSV tidak memiliki data untuk semua kolom di tabel database target, Anda masih dapat mengimpor data yang Anda miliki dengan menentukan kolom mana yang ada dalam data. Pertimbangkan skenario ini: Anda sedang meneliti gaji semua pengawas kota di negara bagian Anda sehingga Anda dapat menganalisis tren pengeluaran pemerintah berdasarkan geografi. Untuk memulai, Anda membuat tabel bernama supervisor_salaries dengan kode di Listing bawah ini:

```
CREATE TABLE supervisor_salaries (
   town varchar(30),
   county varchar(30),
   supervisor varchar(30),
   start_date date,
   salary money,
   benefits money
);
```

Anda ingin kolom untuk kota dan kabupaten, nama supervisor, tanggal dia mulai, dan gaji dan tunjangan (dengan asumsi Anda hanya peduli dengan level saat ini). Namun, petugas county pertama yang Anda hubungi mengatakan, “Maaf, kami hanya memiliki kota, supervisor, dan gaji. Anda harus mendapatkan sisanya dari tempat lain. ” Anda tetap memberi tahu mereka untuk mengirim CSV. Anda akan mengimpor apa yang Anda bisa.

Saya telah menyertakan contoh CSV yang dapat Anda unduh di sumber buku di https://www.nostarch.com/practicalSQL/, yang disebut supervisor_salaries.csv. Anda dapat mencoba mengimpornya menggunakan sintaks COPY dasar ini:

```
COPY supervisor_salaries
FROM 'C:\YourDirectory\supervisor_salaries.csv'
WITH (FORMAT CSV, HEADER);
```

Tetapi jika Anda melakukannya, PostgreSQL akan mengembalikan kesalahan:

```
****************Error****************
ERROR: missing data for column "start_date"
SQL state: 22P04
Context: COPY supervisor_salaries, line 2: "Anytown,Jones,27000"
```

Basis data mengeluh bahwa ketika sampai ke kolom keempat tabel, start_date, tidak dapat menemukan data apa pun di CSV. Solusi untuk situasi ini adalah memberi tahu database kolom mana dalam tabel yang ada di CSV, seperti yang ditunjukkan pada dibawah ini

```
COPY supervisor_salaries (town, supervisor, salary)
FROM 'C:\YourDirectory\supervisor_salaries.csv'
WITH (FORMAT CSV, HEADER);
```

Dengan mencatat dalam tanda kurung tiga kolom yang ada setelah nama tabel, kami memberi tahu PostgreSQL untuk hanya mencari data untuk mengisi kolom tersebut saat membaca CSV.

Sekarang, jika Anda memilih beberapa baris pertama dari tabel, Anda hanya akan melihat kolom yang terisi:

```
Town            county      supervisor  start_date  salary      benefits
------------    ------      ----------  ----------  ----------  --------
Anytown                     Jones                   $27,000.00
Bumblyburg                  Baker                   $24,999.00
```

Menambahkan Nilai Default ke Kolom Selama Impor

Bagaimana jika Anda ingin mengisi kolom county selama impor, meskipun nilainya hilang dari file CSV? Anda dapat melakukannya dengan menggunakan tabel sementara. Tabel sementara hanya ada sampai Anda mengakhiri sesi database Anda. Saat Anda membuka kembali database (atau kehilangan koneksi), tabel tersebut akan hilang. Mereka berguna untuk melakukan operasi perantara pada data sebagai bagian dari alur pemrosesan Anda; kami akan menggunakannya untuk menambahkan nama county ke tabel supervisor_salaries saat kami mengimpor CSV.

Mulailah dengan menghapus data yang sudah Anda impor ke supervisor_salaries menggunakan kueri DELETE:

```
DELETE FROM supervisor_salaries;
```

Saat kueri itu selesai, jalankan kode Listing ini

```
CREATE TEMPORARY TABLE supervisor_salaries_temp (LIKE supervisor_salaries);

COPY supervisor_salaries_temp (town, supervisor, salary)
FROM 'C:\YourDirectory\supervisor_salaries.csv'
WITH (FORMAT CSV, HEADER);

INSERT INTO supervisor_salaries (town, county, supervisor, salary)
SELECT town, 'Some County', supervisor, salary
FROM supervisor_salaries_temp:

DROP TABLE supervisor_salaries_temp;
```

Script ini melakukan empat tugas. Pertama, kita membuat tabel sementara bernama supervisor_salaries_temp berdasarkan tabel supervisor_salaries asli dengan meneruskan kata kunci LIKE sebagai argumen (tercakup dalam "Menggunakan LIKE dan ILIKE dengan WHERE" di halaman 19) diikuti oleh tabel induk untuk disalin. Kemudian kita mengimpor file supervisor_salaries.csv ke dalam tabel sementara menggunakan sintaks COPY yang sekarang sudah dikenal.

Selanjutnya, kita menggunakan pernyataan INSERT untuk mengisi tabel gaji. Alih-alih menentukan nilai, kami menggunakan pernyataan SELECT untuk menanyakan tabel sementara. Kueri itu menentukan nilai untuk kolom kedua, bukan sebagai nama kolom, tetapi sebagai string di dalam tanda kutip tunggal.

Akhirnya, kami menggunakan DROP TABLE untuk menghapus tabel sementara. Tabel sementara akan otomatis hilang saat Anda memutuskan sambungan dari sesi PostgreSQL, tetapi ini menghapusnya sekarang jika kita ingin menjalankan kueri lagi terhadap CSV lain.

Setelah Anda menjalankan kueri, jalankan pernyataan SELECT pada beberapa baris pertama untuk melihat efeknya:

```
Town            county      supervisor  start_date  salary      benefits
------------    ----------- ----------  ----------  ----------  --------
Anytown    some County      Jones                   $27,000.00
Bumblyburg some County      Baker                   $24,999.00
```

Sekarang Anda telah mengisi bidang county dengan nilai. Jalur ke impor ini mungkin tampak sulit, tetapi penting untuk melihat bagaimana pemrosesan data dapat memerlukan beberapa langkah untuk mendapatkan hasil yang diinginkan. Kabar baiknya adalah bahwa demo tabel sementara ini merupakan indikator yang tepat dari fleksibilitas yang ditawarkan SQL untuk mengontrol penanganan data.

Menggunakan COPY untuk Mengekspor Data

Perbedaan utama antara mengekspor dan mengimpor data dengan COPY adalah daripada menggunakan FROM untuk mengidentifikasi sumber data, Anda menggunakan TO untuk jalur dan nama file output. Anda mengontrol berapa banyak data yang akan diekspor diseluruh tabel, hanya beberapa kolom, atau untuk menyempurnakannya lebih jauh lagi, hasil kueri.
Mari kita lihat tiga contoh singkat.

Mengekspor Semua Data

Ekspor paling sederhana mengirimkan semua yang ada di tabel ke file. Sebelumnya, Anda membuat tabel us_counties_2010 dengan 91 kolom dan 3.143 baris data sensus. Pernyataan SQL dibawah ini untuk mengekspor semua data ke file teks bernama us_counties_export.txt. Opsi kata kunci WITH memberi tahu PostgreSQL untuk menyertakan baris header dan menggunakan simbol pipa alih-alih koma untuk pembatas. Saya telah menggunakan ekstensi file .txt di sini karena dua alasan. Pertama, ini menunjukkan bahwa Anda dapat mengekspor ke format file teks apa pun; kedua, kami menggunakan pipa untuk pembatas, bukan koma. Saya suka menghindari panggilan file .csv kecuali jika mereka benar-benar memiliki koma sebagai pemisah.

Ingatlah untuk mengubah direktori keluaran ke lokasi pilihan Anda.

```
COPY us_counties_2010
TO 'C:\YourDirectory\us_counties_export.txt'
WITH (FORMAT CSV, HEADER, DELIMITER '|');
```

Mengekspor Kolom Tertentu

Anda tidak selalu perlu (atau ingin) mengekspor semua data Anda: Anda mungkin memiliki informasi sensitif, seperti nomor Jaminan Sosial atau tanggal lahir, yang perlu dirahasiakan. Atau, dalam kasus data daerah sensus, mungkin Anda bekerja dengan program pemetaan dan hanya memerlukan nama daerah dan koordinat geografisnya untuk memplot lokasi. Kita hanya dapat mengekspor ketiga kolom ini dengan mencantumkannya dalam tanda kurung setelah nama tabel, seperti yang ditunjukkan pada Listing dibawah ini. Tentu saja, Anda harus memasukkan nama kolom ini persis seperti yang tercantum dalam data agar PostgreSQL dapat mengenalinya.

```
COPY us_counties_2010 (geo_name, internal_point_lat, internal_point _lon
TO 'C:\YourDirectory\us_counties_latlon_export.txt'
WITH (FORMAT CSV, HEADER, DELIMITER '|');
```

Mengekspor Hasil Kueri

Selain itu, Anda dapat menambahkan kueri ke COPY untuk menyempurnakan output Anda. Dalam listing dibawah ini kami mengekspor nama dan singkatan negara bagian dari hanya kabupaten yang namanya mengandung huruf besar atau kecil dengan menggunakan ILIKE case-insensitive dan karakter % wildcard yang kami bahas di "Menggunakan LIKE dan ILIKE dengan WHERE"

```
COPY (
   SELECT geo_name, state_us_abbreviation
   FROM us_counties_2010
   WHERE geo_name ILIKE '%mill%'
)
TO 'C:\YourDirectory\us_counties_mill_export.txt'
WITH (FORMAT CSV, HEADER, DELIMITER '|');
```

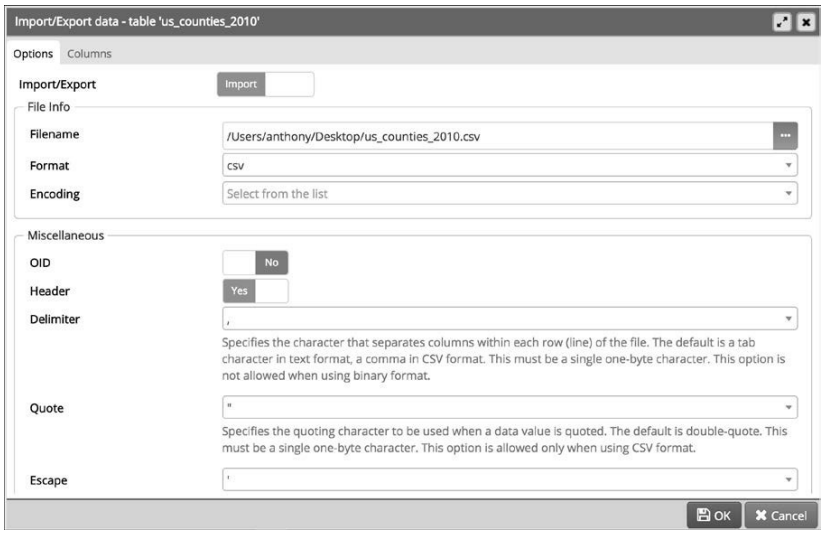
Setelah menjalankan kode, file keluaran Anda harus memiliki sembilan baris dengan nama daerah termasuk Miller, Roger Mills, dan Vermillion.

Mengimpor dan Mengekspor Melalui pgAdmin

Terkadang, perintah SQL COPY tidak akan dapat menangani impor dan ekspor tertentu, biasanya saat Anda terhubung ke instance PostgreSQL berjalan di komputer selain komputer Anda, mungkin di tempat lain di jaringan. Ketika itu terjadi, Anda mungkin tidak memiliki akses ke sistem file komputer itu, yang membuat pengaturan jalur dalam klausa FROM atau TO menjadi sulit.

Salah satu solusinya adalah dengan menggunakan wizard impor/ekspor bawaan pgAdmin. Di browser objek pgAdmin (panel vertikal kiri), cari daftar tabel di database analisis Anda dengan memilih Databases ➔ analysis ➔ Schemas ➔ public ➔Tables.

Selanjutnya, klik kanan pada tabel yang ingin Anda impor atau ekspor, dan pilih Impor/Ekspor. Sebuah dialog muncul yang memungkinkan Anda memilih untuk mengimpor atau mengekspor dari tabel itu, seperti yang ditunjukkan pada Gambar 4.1.



Gambar 4.1: Dialog Impor/Ekspor pgAdmin

Untuk mengimpor, pindahkan penggeser Impor/Ekspor ke Impor. Kemudian klik tiga titik di sebelah kanan kotak Filename untuk menemukan file CSV Anda. Dari daftar drop-down Format, pilih csv. Kemudian sesuaikan header, delimiter, quoting, dan opsi lainnya sesuai kebutuhan. Klik OK untuk mengimpor data. Untuk mengekspor, gunakan dialog yang sama dan ikuti langkah serupa.

Sekarang setelah Anda mempelajari cara memasukkan data eksternal ke dalam database, Anda dapat mulai menggali berbagai kumpulan data, apakah Anda ingin menjelajahi salah satu dari ribuan kumpulan data yang tersedia untuk umum, atau data yang terkait dengan karier atau studi Anda sendiri. Banyak data tersedia dalam format CSV atau format yang mudah dikonversi ke CSV. Cari kamus data untuk membantu Anda memahami data dan memilih tipe data yang tepat untuk setiap bidang.

Data sensus yang Anda impor sebagai bagian dari latihan bab ini akan memainkan peran utama dalam bab berikutnya di mana kita mengeksplorasi fungsi matematika dengan SQL.

Latihan Soal

Lanjutkan penjelajahan Anda tentang impor dan ekspor data dengan latihan ini. Ingatlah untuk membaca dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/sql-copy.html untuk petunjuk:

1. Tulis pernyataan WITH untuk disertakan dengan COPY untuk menangani impor file teks imajiner yang beberapa baris pertamanya terlihat seperti ini:

```
Id:movie:actor
50:#Mission: Impossible#: Tom Cruise
```

2. Menggunakan tabel us_counties_2010 yang Anda buat dan isi di bab ini, ekspor ke file CSV 20 kabupaten di Amerika Serikat yang memiliki unit rumah paling banyak. Pastikan Anda hanya mengekspor nama, negara bagian, dan jumlah unit rumah setiap daerah. (Petunjuk: Unit perumahan dijumlahkan untuk setiap county di kolom housing_unit_count_100_percent.)
3. Bayangkan Anda mengimpor file yang berisi kolom dengan nilai berikut:

```
17519.668
20084.461
18976.335
```

4. Akankah kolom di tabel target Anda dengan tipe data numerik (3,8) berfungsi untuk nilai-nilai ini? Mengapa atau mengapa tidak?

BAB V
MATEMATIKA DASAR DAN STATUS DENGAN SQL

Jika data Anda mencakup salah satu tipe data angka yang kita jelajahi di Bab 3— bilangan bulat, desimal, atau titik mengambang cepat atau lambat analisis Anda akan mencakup beberapa perhitungan. Misalnya, Anda mungkin ingin mengetahui rata-rata semua nilai dolar dalam satu kolom, atau menambahkan nilai dalam dua kolom untuk menghasilkan total untuk setiap baris. SQL menangani perhitungan mulai dari matematika dasar hingga statistik lanjutan.

Dalam bab ini, saya akan mulai dengan dasar-dasar dan kemajuan ke fungsi matematika dan statistik awal. Saya juga akan membahas perhitungan yang terkait dengan persentase dan perubahan persen. Untuk beberapa latihan, kami akan menggunakan data Sensus Tahunan 2010 yang Anda impor di Bab 4.

Operator Matematika

Mari kita mulai dengan matematika dasar yang Anda pelajari di sekolah dasar (dan semuanya dimaafkan jika Anda lupa sebagian). Tabel 5.1 menunjukkan sembilan operator matematika yang paling sering Anda gunakan dalam perhitungan Anda. Empat yang pertama (penambahan, pengurangan, perkalian, dan pembagian) adalah bagian dari standar ANSI SQL yang diimplementasikan di semua sistem basis data. Yang lainnya adalah operator khusus PostgreSQL, meskipun jika Anda menggunakan database lain, kemungkinan memiliki fungsi atau operator untuk melakukan operasi tersebut. Misalnya, operator modulo (%) bekerja di Microsoft SQL Server dan MySQL serta dengan PostgreSQL. Jika Anda menggunakan sistem database lain, periksa dokumentasinya.

Tabel 5.1: Operator Matematika Dasar

Operator	Deskripsi
+	Tambahan
-	Pengurangan
*	Perkalian
/	Divisi (mengembalikan hasil bagi saja, tidak ada sisa)
%	Modulo (mengembalikan sisanya saja)
^	Eksponensial
|	Akar kuadrat
|| /	Akar kubus
!	Faktorial

Kami akan melangkah melalui masing-masing operator ini dengan mengeksekusi kueri SQL sederhana pada angka biasa daripada beroperasi pada tabel atau objek basis data lainnya. Anda dapat memasukkan pernyataan secara terpisah ke dalam alat kueri pgAdmin dan menjalankannya satu per satu, atau jika Anda menyalin kode untuk bab ini dari sumber di https://www.nostarch.com/practicalSQL/ , Anda dapat menyorot masing-masing baris sebelum menjalankannya.

Matematika dan Tipe Data

Saat Anda mengerjakan contoh, perhatikan tipe data dari setiap hasil, yang tercantum di bawah setiap nama kolom di kisi hasil pgAdmin. Tipe yang dikembalikan untuk perhitungan akan bervariasi tergantung pada operasi dan tipe data dari nomor input.

Dalam perhitungan dengan operator antara dua angka (penambahan, pengurangan, perkalian, dan pembagian) tipe data yang dikembalikan mengikuti pola ini:

- Dua bilangan bulat mengembalikan bilangan bulat.
- Angka di kedua sisi operator mengembalikan angka.
- Apa pun dengan nomor floating-point mengembalikan nomor floating-point tipe presisi ganda.

Namun, fungsi eksponensial, akar, dan faktorial berbeda. Masing-masing mengambil satu angka baik sebelum atau sesudah operator dan mengembalikan tipe numerik dan titik-mengambang, bahkan ketika inputnya adalah bilangan bulat.

Terkadang tipe data hasil akan sesuai dengan kebutuhan Anda; di lain waktu, Anda mungkin perlu menggunakan CAST untuk mengubah tipe data, seperti yang disebutkan dalam "Mengubah Nilai dari Satu Jenis ke Jenis Lainnya dengan CAST", seperti jika Anda perlu memasukkan hasilnya ke dalam fungsi yang mengambil jenis tertentu. Saya akan mencatat saat-saat itu saat kami mengerjakan buku ini.

Menambah, Mengurangi, dan Mengalikan

Mari kita mulai dengan penjumlahan, pengurangan, dan perkalian bilangan bulat sederhana. Listing dibawah ini menunjukkan tiga contoh, masing-masing dengan kata kunci SELECT diikuti dengan rumus matematika. Dalam Bab 2, kami telah menggunakan SELECT untuk tujuan utamanya: untuk mengambil data dari sebuah tabel. Tetapi dengan PostgreSQL, Microsoft SQL Server, MySQL, dan beberapa sistem manajemen basis data lainnya, nama tabel untuk operasi matematika dan string dapat dihilangkan saat pengujian, seperti yang kita lakukan di sini. Demi keterbacaan, saya sarankan Anda menggunakan satu spasi sebelum dan sesudah operator matematika; meskipun menggunakan spasi tidak sepenuhnya diperlukan agar kode Anda berfungsi, ini adalah praktik yang baik.

```
SELECT 2 + 2;
SELECT 9 - 1;
SELECT 3 * 4;
```

Tak satu pun dari pernyataan ini adalah ilmu roket, jadi Anda tidak perlu terkejut bahwa menjalankan SELECT 2 + 2; di alat kueri menunjukkan hasil 4. Demikian pula, contoh pengurangan dan perkalian menghasilkan apa yang Anda harapkan: 8 dan 12. Output ditampilkan dalam kolom, seperti hasil kueri apa pun. Tetapi karena kami tidak mengkueri tabel dan menentukan kolom, hasilnya muncul di bawah ?Kolom? nama, menandakan kolom yang tidak dikenal:

```
?column?
--------
       4
```

Tidak apa-apa. Kami tidak memengaruhi data apa pun dalam tabel, hanya menampilkan hasilnya.

Divisi dan Modulo

Pembagian dengan SQL menjadi sedikit lebih rumit karena perbedaan antara matematika dengan bilangan bulat dan matematika dengan desimal, yang telah disebutkan sebelumnya. Tambahkan modulo, operator yang mengembalikan hanya sisanya dalam operasi pembagian, dan hasilnya bisa membingungkan. Contoh:

```
SELECT 11 / 6;
SELECT 11 % 6;

SELECT 11.0 / 6;
SELECT CAST (11 AS numeric(3,1) / 6;
```

Pernyataan pertama menggunakan operator / untuk membagi bilangan bulat 11 dengan bilangan bulat lain, 6. Jika Anda melakukan matematika itu di kepala Anda, Anda tahu jawabannya adalah 1 dengan sisa 5. Namun, menjalankan kueri ini menghasilkan 1, yaitu bagaimana SQL menangani pembagian satu bilangan bulat dengan bilangan bulat lainnya dengan hanya melaporkan hasil bagi bilangan bulat. Jika Anda ingin mengambil sisanya sebagai bilangan bulat, Anda harus melakukan perhitungan yang sama menggunakan operator modulo %, seperti pada . Pernyataan itu hanya mengembalikan sisanya, dalam hal ini 5. Tidak ada operasi tunggal yang akan memberi Anda hasil bagi dan sisanya sebagai bilangan bulat.

Modulo berguna untuk lebih dari sekadar mengambil sisa: Anda juga dapat menggunakannya sebagai kondisi pengujian. Misalnya, untuk memeriksa apakah suatu bilangan genap, Anda dapat mengujinya menggunakan operasi % 2. Jika hasilnya 0 tanpa sisa, maka bilangan tersebut genap.

Jika Anda ingin membagi dua angka dan mendapatkan hasilnya kembali sebagai tipe numerik, Anda dapat melakukannya dengan dua cara: pertama, jika salah satu atau kedua angka adalah angka, hasilnya secara default akan dinyatakan sebagai angka. . Itulah yang terjadi ketika saya membagi 11.0 dengan 6. Jalankan kueri itu, dan hasilnya adalah 1,83333. Jumlah digit desimal yang ditampilkan dapat bervariasi sesuai dengan PostgreSQL dan pengaturan sistem Anda.

Kedua, jika Anda bekerja dengan data yang disimpan hanya sebagai bilangan bulat dan perlu memaksakan pembagian desimal, Anda dapat CAST salah satu bilangan bulat ke tipe numerik . Menjalankan ini lagi mengembalikan 1,83333.

Eksponen, Akar, dan Faktorial

Di luar dasar-dasar, SQL rasa PostgreSQL juga menyediakan operator untuk kuadrat, kubus, atau menaikkan angka dasar ke eksponen, serta menemukan akar atau faktorial dari suatu angka. List program dibawah ini menunjukkan operasi ini dalam tindakan:

```
SELECT 3 ^ 4;
SELECT |/10;
SELECT sqrt(10);
SELECT ||/10;
SELECT 4 !;
```

Operator eksponensial (^) memungkinkan Anda untuk menaikkan angka dasar yang diberikan ke eksponen, seperti pada , di mana 3 ^ 4 (bahasa sehari-hari, kami menyebutnya tiga pangkat keempat) mengembalikan 81.

Anda dapat menemukan akar kuadrat dari suatu bilangan dengan dua cara: menggunakan |/ operator atau fungsi kuadrat (n). Untuk akar pangkat tiga, gunakan operator ||/ . Keduanya adalah operator awalan, dinamai karena mereka datang sebelum nilai tunggal.

Untuk mencari faktorial suatu bilangan, gunakan! operator. Ini adalah operator sufiks, yang muncul setelah satu nilai. Anda akan menggunakan faktorial di banyak tempat dalam matematika, tetapi mungkin yang paling umum adalah menentukan berapa banyak cara sejumlah item dapat dipesan. Katakanlah Anda memiliki empat foto. Berapa banyak cara Anda dapat memesannya di samping satu sama lain di dinding? Untuk menemukan jawabannya, Anda akan menghitung faktorial dengan memulai dengan jumlah item dan mengalikan semua bilangan bulat positif yang lebih kecil. Jadi, dipernyataan faktorial dari 4! setara dengan 4 × 3 × 2 × 1.

Sekali lagi, operator ini khusus untuk PostgreSQL; mereka bukan bagian dari standar SQL. Jika Anda menggunakan aplikasi database lain, periksa dokumentasinya untuk mengetahui bagaimana aplikasi tersebut mengimplementasikan operasi ini.

Memperhatikan Urutan Operasi

Dapatkah Anda mengingat dari pelajaran matematika awal Anda apa urutan operasi, atau prioritas operator, pada ekspresi matematika? Ketika Anda merangkai beberapa angka dan operator, perhitungan manakah yang dieksekusi SQL terlebih dahulu? Tidak mengherankan, SQL mengikuti standar matematika yang ditetapkan. Untuk operator PostgreSQL yang dibahas sejauh ini, urutannya adalah:

1. Eksponen dan akar
2. Perkalian, pembagian, modul
3. Penambahan dan pengurangan

Mengingat aturan ini, Anda harus membungkus operasi dalam tanda kurung jika Anda ingin menghitungnya dalam urutan yang berbeda. Misalnya, dua ekspresi berikut menghasilkan hasil yang berbeda:

```
SELECT 7 + 8 * 9;
SELECT (7 + 8) * 9:
```

Ekspresi pertama mengembalikan 79 karena operasi perkalian menerima prioritas dan diproses sebelum penambahan. Yang kedua mengembalikan 135 karena tanda kurung memaksa operasi penambahan terjadi terlebih dahulu.

Berikut adalah contoh kedua menggunakan eksponen:

```
SELECT 3 ^ 3 – 1;
SELECT 3 ^ (3 – 1);
```

Operasi eksponen lebih diutamakan daripada pengurangan, jadi tanpa tanda kurung seluruh ekspresi dievaluasi dari kiri ke kanan dan operasi untuk menemukan 3 pangkat 3 terjadi terlebih dahulu. Kemudian 1 dikurangkan, menghasilkan 26. Pada contoh kedua, tanda kurung

memaksa pengurangan terjadi terlebih dahulu, sehingga operasi menghasilkan 9, yaitu 3 pangkat 2.
Ingatlah prioritas operator untuk menghindari keharusan mengoreksi analisis Anda nanti!

Mengerjakan Matematika di Seluruh Kolom Tabel Sensus

Mari kita coba menggunakan operator matematika SQL yang paling sering digunakan pada data nyata dengan menggali tabel populasi Sensus Tahunan 2010, us_counties_2010, yang Anda impor di Bab 4. Alih-alih menggunakan angka dalam kueri, kita akan menggunakan nama kolom yang berisi angka-angka. Saat kita mengeksekusi query, perhitungan akan terjadi pada setiap baris tabel.

Untuk menyegarkan ingatan Anda tentang data, jalankan skrip dbawah ini. Ini harus mengembalikan 3.143 baris yang menunjukkan nama dan negara bagian setiap county di Amerika Serikat, dan jumlah orang yang diidentifikasi dengan salah satu dari enam kategori ras atau kombinasi dari dua atau lebih ras.
Formulir Sensus 2010 yang diterima oleh setiap rumah tangga — yang disebut “formulir singkat” —memungkinkan orang untuk mencentang hanya satu atau beberapa kotak di bawah pertanyaan ras. (Anda dapat meninjau formulir di https://www.census.gov/2010census/pdf/2010_Questionnaire_Info.pdf.)

Orang yang mencentang satu kotak dihitung dalam kategori seperti “Putih Sendiri” atau “Hitam atau Sendirian Afrika-Amerika.” Responden yang memilih lebih dari satu kotak ditabulasi dalam kategori keseluruhan “Dua Ras atau Lebih”, dan kumpulan data sensus merincinya secara rinci.

```
SELECT geo_name,
State_us_abbreviation AS “st”,
   p0010001 AS “Total Population”,
   p0010003 AS “White Alone”,
   p0010004 AS “Black or African American Alone”,
   p0010005 AS “Am Indian/Alaska Native Alone”,
   p0010006 AS “Asian Alone”.
   p0010007 AS “Native Hawaiian and Other Pacific Islander Alone”,
   p0010008 AS “Some Other Race Alone”,
   p0010009 AS “Two or More Races”
FROM us_counties_2010;
```

Di us_counties_2010, setiap kolom data ras dan rumah tangga berisi kode sensus. Misalnya, kolom “Asian Alone” dilaporkan sebagai p0010006. Meskipun kode-kode tersebut mungkin ekonomis dan kompak, kode-kode tersebut membuat sulit untuk memahami kolom mana yang ketika kueri kembali hanya dengan kode itu. Dalam Daftar 5-4, saya menggunakan sedikit trik untuk memperjelas output dengan menggunakan kata kunci AS untuk memberikan alias yang lebih mudah dibaca pada setiap kolom dalam kumpulan hasil. Kita dapat mengganti nama semua kolom saat diimpor, tetapi dengan sensus, sebaiknya gunakan kode untuk merujuk ke nama kolom yang sama dalam dokumentasi jika diperlukan.

Menambah dan Mengurangi Kolom

Sekarang, mari kita coba kalkulasi sederhana pada dua kolom Ras, dengan menambahkan jumlah orang yang diidentifikasi sebagai orang kulit putih saja atau orang kulit hitam saja di setiap daerah.

```
SELECT geo_name,
state_us_abbrevliation AS “st”,
   p0010003 AS “White Alone”,
   p0010004 AS “Black Alone”,
   p0010003 + p0010004 “Total White and Black”
FROM us_counties_2010;
```

Menyediakan p0010003 + p0010004 sebagai salah satu kolom dalam pernyataan SELECT menangani perhitungan. Sekali lagi, saya menggunakan kata kunci AS untuk memberikan alias yang dapat dibaca untuk kolom. Jika Anda tidak memberikan alias, PostgreSQL menggunakan label? Kolom ?, yang kurang membantu. Jalankan kueri untuk melihat hasilnya. Beberapa baris pertama harus menyerupai output ini:

geo_name	st	White Alone	Black Alone	Total White and Black
Autauga County	AL	42855	9643	52498
Baldwin County	AL	156153	17105	173258
Barbour County	AL	13180	12875	26055

Pemeriksaan cepat dengan kalkulator atau pensil dan kertas memastikan bahwa total kolom sama dengan jumlah kolom yang Anda tambahkan. Bagus sekali!

Sekarang, mari kita membangun ini untuk menguji data kita dan memvalidasi bahwa kita mengimpor kolom dengan benar. Enam kolom ras "Sendiri" ditambah kolom "Dua Balapan atau Lebih" harus berjumlah hingga jumlah yang sama dengan total populasi. Kode di listprogram harus menunjukkan bahwa itu:

```
SELECT geo_name,
state_us_abbrevliation AS “st”,
p0010001 AS “Total”,
p0010003 + p0010004 + p0010005 + p0010006 + p0010007
      + p0010008 + p0010009 AS “All Races”,
(p0010003 + p0010004 + p0010005 + p0010006 + p0010007
      + p0010008 + p0010009) – p0010001 AS ‘Difference”
FROM us_counties_2010
ORDER BY “Difference” DESC;
```

Kueri ini mencakup total populasi, diikuti dengan penghitungan yang menambahkan tujuh kolom ras sebagai Semua Ras. Total populasi dan total ras harus identik, tetapi daripada memeriksa secara manual, kami juga menambahkan kolom yang mengurangi kolom total populasi dari jumlah kolom ras . Kolom itu, bernama Selisih, harus berisi nol di setiap baris jika semua data berada di tempat yang tepat. Untuk menghindari keharusan memindai semua 3.143 baris, kami menambahkan klausa ORDER BY pada kolom bernama. Setiap baris menunjukkan perbedaan akan muncul di bagian atas atau bawah hasil kueri. Jalankan kueri; beberapa baris pertama harus memberikan hasil ini:

```
geo_name   st    Total       All Race    Difference
--------------  ---   ------              --------    ----------
Autauga   County      AL    54571       54571       0
Baldwin County  AL    182265              182265            0
Barbour County  AL    27457       27457       0
```

Dengan kolom Selisih menunjukkan nol, kita dapat yakin bahwa impor kita bersih. Setiap kali saya menemukan atau mengimpor kumpulan data baru, saya suka melakukan tes kecil seperti ini. Mereka membantu saya lebih memahami data dan mencegah potensi masalah sebelum saya menggali analisis.

Menemukan Persentase Keseluruhan

Mari kita menggali lebih dalam data sensus untuk menemukan perbedaan yang berarti dalam demografi populasi kabupaten. Salah satu cara untuk melakukan ini (dengan kumpulan data apa pun, sebenarnya) adalah dengan menghitung persentase keseluruhan yang diwakili oleh variabel tertentu. Dengan data sensus, kita dapat belajar banyak dengan membandingkan persentase dari kabupaten ke kabupaten dan juga dengan memeriksa bagaimana persentase bervariasi dari waktu ke waktu.

Untuk mengetahui persentase dari keseluruhan, bagilah angka yang dimaksud dengan totalnya. Misalnya, jika Anda memiliki sekeranjang 12 apel dan menggunakan 9 dalam satu pai, itu akan menjadi 9/12 atau 0,75 — biasanya dinyatakan sebagai 75 persen.

Untuk mencoba ini pada data kabupaten sensus, gunakan kode dibawah ini, yang menghitung persentase penduduk yang melaporkan ras mereka sebagai orang Asia untuk setiap kabupaten:

```
SELECT geo_name,
state_us_abbrevliation AS “st”,
(CAST(p0010006 AS numeric(8,1)) / p0010001 * 100 AS “pct_asian”
FROM us_counties_2010
ORDER BY “pct_asian” DESC;
```

Bagian kunci dari kueri ini membagi p0010006, kolom dengan jumlah orang Asia saja, dengan p001001, kolom untuk total populasi .

Jika kita menggunakan data sebagai tipe integer aslinya, kita tidak akan mendapatkan hasil pecahan yang kita butuhkan: setiap baris akan menampilkan hasil 0, hasil bagi. Sebagai gantinya, kami memaksa pembagian desimal dengan menggunakan CAST pada salah satu bilangan bulat. Bagian terakhir mengalikan hasilnya dengan 100 untuk menyajikan hasilnya sebagai pecahan dari 100, cara kebanyakan orang memahami persentase.

Dengan mengurutkan dari persentase tertinggi ke terendah, bagian atas output adalah sebagai berikut:

```
geo_name              st    pct_asian
--------------              ---   ------
Honolulu County       HI    43.89497769109962474000
Aleutians East Borough              AK    35.97580388411333970100
San Francisco County        CA    33.27165361664607226500
Santa Clara County          CA    32.02237037519322063600
Kauai County                HI    31.32461880132953749400
```

```
Aleutians West Census Area  AK    28.87969789606185937800
```

Melacak Perubahan Persen

Indikator kunci lain dalam analisis data adalah perubahan persen: seberapa besar, atau lebih kecil, satu angka dari yang lain? Penghitungan persentase perubahan sering digunakan saat menganalisis perubahan dari waktu ke waktu, dan sangat berguna untuk membandingkan perubahan di antara item serupa.

Beberapa contoh termasuk:

- Perubahan dari tahun ke tahun dalam jumlah kendaraan yang dijual oleh masing-masing pembuat mobil.
- Perubahan bulanan dalam langganan untuk setiap daftar email yang dimiliki oleh perusahaan pemasaran.
- Peningkatan atau penurunan tahunan dalam pendaftaran di sekolah-sekolah di seluruh negeri.

Rumus untuk menghitung persen perubahan dapat dinyatakan seperti ini:

```
(nomor baru - nomor lama) / nomor lama
```

Jadi, jika Anda memiliki kios limun dan menjual 73 gelas limun hari ini dan 59 gelas kemarin, Anda akan menghitung persentase perubahan harian seperti ini:

```
(73 – 59) / 59 = .237 = 23.7%
```

Mari kita coba ini dengan kumpulan kecil data uji yang terkait dengan pengeluaran di departemen pemerintah daerah hipotetis. Cmenghitung departemen mana yang memiliki persentase kenaikan dan kerugian terbesar:

```
CREATE TABLE percent_change (
      department varchar(20),
      spend_2014 numeric(10,2),
      spend_2017 numeric(10,2),
);

INSERT INTO percent_change
VALUES
      ('Building', 250000, 289000),
      ('Assesor', 178556, 179500),
      ('Library', 87777, 900001),
      ('Clerk', 451980, 650000),
      ('Police', 250000, 223000),
      ('Recreation', 199000, 195000);

SELECT department,
      spend_2014,
      spend_2017,

      round( (spend_2017 – spend_2014) /
           spend_2014 * 100, 1) AS "pct_change"

FROM percent_change;
```

Listprogam diatas adalah untuk membuat tabel kecil yang disebut persen_perubahan dan menyisipkan enam baris dengan data pengeluaran departemen untuk tahun 2014 dan 2017. Rumus perubahan persen mengurangi pengeluaran_2014 dari pengeluaran_2017 lalu dibagi dengan pengeluaran_2014. Kami mengalikan dengan 100 untuk menyatakan hasilnya sebagai bagian dari 100.

Untuk menyederhanakan output, kali ini saya telah menambahkan fungsi round() untuk menghapus semua kecuali satu tempat desimal. Fungsi ini membutuhkan dua argumen: kolom atau ekspresi yang akan dibulatkan, dan jumlah tempat desimal yang akan ditampilkan. Karena kedua angka tersebut bertipe numerik, maka hasilnya juga akan berupa angka, hasil yang akan muncul dalam program tersebut adalah sebagai berikuti:

```
department spend_2014  spend_2017  pct_change
---------- ----------  ----------  ----------
Building   250000.00   289000.00   15.6
Assessor   178556.00   179500.00   0.5
Library          87777.00    90001.00    2.5
Clerk            451980.00   650000.00   43.8
Police           250000.00   223000.00   -10.8
Recreation 199000.00   195000.00   -2.0
```

Sekarang, tinggal mencari tahu mengapa pengeluaran departemen Clerk melebihi pengeluaran lainnya di kota.

Fungsi Agregat untuk Rata-rata dan Jumlah

Sejauh ini, kami telah melakukan operasi matematika di seluruh kolom di setiap baris tabel. SQL juga memungkinkan Anda menghitung hasil dari nilai dalam kolom yang sama menggunakan fungsi agregat. Anda dapat melihat daftar lengkap dari agregat PostgreSQL, yang menghitung satu hasil dari beberapa masukan, di
https://www.postgresql.org/docs/current/static/functions-aggregate.html
Dua dari fungsi agregat yang paling banyak digunakan dalam analisis data adalah avg() dan sum().

Kembali ke tabel sensus us_counties_2010, wajar jika ingin menghitung jumlah penduduk semua kabupaten ditambah rata-rata penduduk semua kabupaten. Menggunakan avg() dan sum() pada kolom p0010001 (total populasi) membuatnya mudah, seperti yang ditunjukkan pada dilist program bawah ini. Sekali lagi, kami menggunakan fungsi bulat () untuk menghapus angka setelah titik desimal dalam perhitungan rata-rata.

```
 SELECT sum(p0010001) AS “County Sum”,
     Round(avg(p0010001), 0) AS “County Average”
FROM us_counties_2010;
```

Perhitungan ini menghasilkan hasil sebagai berikut:

```
County SumCounty Average
------------------------
308745538 98233
```

Populasi untuk semua kabupaten di Amerika Serikat pada tahun 2010 bertambah menjadi sekitar 308,7 juta, dan rata-rata penduduk kabupaten adalah 98.233.

Menemukan Median

Nilai median dalam serangkaian angka sama pentingnya dengan indikator, jika tidak lebih, daripada rata-rata. Inilah perbedaan antara median dan rata-rata, dan mengapa median penting:

Rata-rata Jumlah semua nilai dibagi dengan jumlah nilai

Median Nilai "tengah" dalam kumpulan nilai yang terurut

Mengapa median penting untuk analisis data? Pertimbangkan contoh ini: katakanlah enam anak, usia 10, 11, 10, 9, 13, dan 12, melakukan karyawisata. Sangat mudah untuk menambahkan usia dan membagi enam untuk mendapatkan usia rata-rata grup:

(10 + 11 + 10 + 9 + 13 + 12) / 6 = 10.8

Karena usia berada dalam rentang yang sempit, rata-rata 10,8 adalah representasi yang baik dari kelompok tersebut. Tetapi rata-rata kurang membantu ketika nilai-nilainya dikelompokkan, atau condong, ke salah satu ujung distribusi, atau jika grup tersebut menyertakan outlier.

Misalnya, bagaimana jika pendamping yang lebih tua bergabung dengan karyawisata? Dengan usia 10, 11, 10, 9, 13, 12, dan 46, usia rata-rata meningkat pesat:

(10 + 11 + 10 + 9 + 13 + 12 + 46) / 7 = 15,9

Sekarang rata-rata tidak mewakili grup dengan baik karena outlier mencondongkannya, menjadikannya indikator yang tidak dapat diandalkan.

Di sinilah median bersinar. Median adalah titik tengah dalam daftar nilai yang diurutkan, titik di mana setengah nilainya lebih dan setengahnya lebih sedikit. Dengan menggunakan karyawisata, kami mengurutkan usia peserta dari terendah ke tertinggi:

9, 10, 10, 11, 12, 13, 46

Nilai tengah (median) adalah 11. Setengah nilainya lebih tinggi, dan setengahnya lebih rendah. Mengingat kelompok ini, median 11 adalah gambaran yang lebih baik tentang usia tipikal daripada rata-rata 15,9. Jika himpunan nilai adalah bilangan genap, rata-ratakan dua angka tengah untuk menemukan median. Mari tambahkan siswa lain (usia 12) ke karyawisata:

9, 10, 10, 11, 12, 12, 13, 46

Sekarang, dua nilai tengahnya adalah 11 dan 12. Untuk mencari median, kita rata-ratakan: 11.5.

Median sering dilaporkan dalam berita keuangan. Laporan harga perumahan sering menggunakan median karena beberapa penjualan McMansions dalam Kode Pos yang sederhana dapat membuat rata-rata tidak berguna. Hal yang sama berlaku untuk gaji pemain olahraga: satu atau dua superstar dapat mengubah rata-rata tim.

Tes yang baik adalah menghitung rata-rata dan median untuk sekelompok nilai. Jika mereka dekat, grup tersebut mungkin terdistribusi secara normal (kurva lonceng yang sudah dikenal), dan rata-ratanya berguna. Jika mereka berjauhan, nilainya tidak terdistribusi normal dan median adalah representasi yang lebih baik.

Mencari Median dengan Fungsi Persentile

PostgreSQL (seperti kebanyakan database relasional) tidak memiliki fungsi median () bawaan, mirip dengan yang Anda temukan di Excel atau program spreadsheet lainnya. Itu juga tidak termasuk dalam standar ANSI SQL. Tetapi kita dapat menggunakan fungsi persentil SQL untuk menemukan median serta kuantil atau titik potong lainnya, yang merupakan titik yang membagi sekelompok angka menjadi ukuran yang sama. Fungsi persentil adalah bagian dari standar ANSI SQL.

Dalam statistik, persentile menunjukkan titik dalam kumpulan data terurut di mana persentase tertentu dari data ditemukan. Misalnya, dokter mungkin memberi tahu Anda bahwa tinggi badan Anda menempatkan Anda di persentil ke-60 untuk orang dewasa dalam kelompok usia Anda. Itu berarti 60 persen orang memiliki tinggi badan Anda atau lebih pendek.

Median setara dengan persentil ke-50, setengah nilainya di bawah dan setengah di atas. Fungsi persentil SQL memungkinkan kita menghitungnya dengan mudah, meskipun kita harus memperhatikan perbedaan dalam cara kedua versi fungsi — persentile_kont (n) dan persentile_disc (n). Kedua fungsi tersebut merupakan bagian dari standar ANSI SQL dan hadir di PostgreSQL, Microsoft SQL Server, dan database lainnya.

Fungsi persentile_cont (n) menghitung persentil sebagai nilai kontinu. Artinya, hasilnya tidak harus berupa salah satu angka dalam kumpulan data tetapi dapat berupa nilai desimal di antara dua angka. Ini mengikuti metodologi untuk menghitung median pada sejumlah nilai genap, di mana median adalah rata-rata dari dua angka tengah. Di sisi lain, persentile_disc (n) hanya mengembalikan nilai diskrit. Artinya, hasil yang dikembalikan akan dibulatkan ke salah satu angka dalam himpunan. Untuk memperjelas perbedaan ini, mari gunakan listing program dibawah ini untuk membuat tabel pengujian dan mengisi enam angka.

```
CREATE TABLE percentile_test (
   number integer
);

INSERT INTO percentile_test (numbers) VALUES
   (1), (2), (3), (4), (5), (6);

SELECT
   percentile_cont(.5)
   WITHIN GROUP (ORDER BY numbers),
   percentile_disc(.5)

      WITHIN GROUP (ORDER BY numbers),
```

```
FROM percentile_test;
```

Dalam fungsi persentil dan diskrit kontinu, kita memasukkan .5 untuk mewakili persentil ke-50, yang setara dengan median. Menjalankan kode mengembalikan yang berikut:

```
percentile_cont      percentile_disc
---------------      ---------------
       3.5                    3
```

Fungsi persentil_cont () mengembalikan apa yang kita harapkan median menjadi: 3.5. Tetapi karena persentil_disc () menghitung nilai diskrit, ia melaporkan 3, nilai terakhir dalam 50 persen pertama angka. Karena metode penghitungan median yang diterima adalah dengan merata-ratakan dua nilai tengah dalam himpunan bernomor genap, gunakan persentil_cont (.5) untuk menemukan median.

Median dan Persentile dengan Data Sensus

Data sensus kami dapat menunjukkan bagaimana median menceritakan kisah yang berbeda dari rata-rata. Dalam listing program dibawah ini dengan menambahkan persentil_cont () di samping jumlah () dan rata-rata () agregat yang telah kami gunakan sejauh ini:

```
SELECT sum(p0010001) ASv"County Sum",
   round(avg(p0010001), 0) AS "County Average",
   percentile_cont(.5)
      WITHIN GROUP (ORDER BY p0010001) AS "County Median"
FROM us_counties_2010;
```

Hasil Anda harus sama dengan yang berikut:

```
County Sum     County Average          County Median
----------     --------------          -------------
308745538      98233             25857
```

Median dan rata-rata berjauhan, yang menunjukkan bahwa rata-rata dapat menyesatkan. Pada 2010, separuh kabupaten di Amerika memiliki kurang dari 25.857 orang, sedangkan separuhnya memiliki lebih banyak. Jika Anda memberikan presentasi tentang A.S. demografis dan mengatakan kepada hadirin bahwa "rata-rata county di Amerika memiliki 98.200 orang," mereka akan pergi dengan gambaran realitas yang miring. Hampir 40 kabupaten memiliki satu juta orang atau lebih pada Sensus Sepuluh Tahun 2010, dan Los Angeles County memiliki hampir 10 juta. Itu mendorong rata-rata lebih tinggi.

Menemukan Kuantil Lain dengan Fungsi Persentile

Anda juga dapat mengiris data menjadi kelompok-kelompok kecil yang sama. Paling umum adalah kuartil (empat kelompok yang sama), kuintil (lima kelompok), dan desil (10 kelompok). Untuk menemukan nilai individual, Anda cukup menghubungkannya ke fungsi persentil. Misalnya, untuk menemukan nilai yang menandai kuartil pertama, atau 25 persen data terendah, Anda akan menggunakan nilai .25:

```
percentile_cont(.25)
```

Namun, memasukkan nilai satu per satu sulit jika Anda ingin menghasilkan banyak titik potong. Sebagai gantinya, Anda bisa meneruskan nilai ke persentil_cont() menggunakan larik, tipe data SQL yang berisi daftar item. Listing dibawah ini menunjukkan cara menghitung keempat kuartil sekaligus:

```
SELECT percentilr_cont(array[.25,.5,.75])
      WITHIN GROUP (ORDER BY p0010001) AS “quartiles”
FROM us_counties_2010;
```

Dalam contoh ini, kami membuat array titik potong dengan menyertakan nilai dalam konstruktor yang disebut array []. Di dalam tanda kurung siku, kami memberikan nilai yang dipisahkan koma yang mewakili tiga titik yang akan dipotong untuk membuat empat kuartil. Jalankan kueri, dan Anda akan melihat output ini:

```
quartiles
---------------------
{11104.5,25857,66699}
```

Karena kita melewati sebuah array, PostgreSQL mengembalikan sebuah array, dilambangkan dengan kurung kurawal. Setiap kuartil dipisahkan dengan koma. Kuartil pertama adalah 11.104,5, yang berarti 25 persen kabupaten memiliki populasi yang sama atau lebih rendah dari nilai ini. Kuartil kedua sama dengan median: 25.857. Kuartil ketiga adalah 66.699, artinya 25 persen kabupaten terbesar memiliki setidaknya populasi sebesar ini.

Array datang dengan sejumlah fungsi (tercatat untuk PostgreSQL di https:// www.postgresql.org/docs/current/static/functions-array.html) yang memungkinkan Anda melakukan tugas seperti menambah atau menghapus nilai atau menghitung elemen.

Fungsi praktis untuk bekerja dengan hasil yang dikembalikan pada listing program dibawah ini adalah unnest (), yang membuat larik lebih mudah dibaca dengan mengubahnya menjadi baris. Lihat listing program dibawah ini:

```
SELECT unnest(
    percentile_cont(array[.25,.5,.75])
    WITHIN GROUP (ORDER BY p0010001)
    ) AS “quartiles”
FROM us_counties_2010;
```

Sekarang output harus dalam baris:

```
quartiles
---------
11104.5
25857
66699
```

Jika kita menghitung desil, menariknya dari larik yang dihasilkan dan menampilkannya dalam baris akan sangat membantu.

Membuat fungsi median ()

Meskipun PostgreSQL tidak memiliki fungsi agregat median () bawaan, jika Anda suka bertualang, wiki PostgreSQL di http://wiki.postgresql.org/wiki/Aggregate_Median menyediakan skrip untuk membuatnya. Lihat contoh listing dibawah ini:

```
CREATE OR REPLACE FUNCTION _final_median(anyarray)
    RETURN float8 AS
$$
    WITH q AS
    (
       SELECT val
       FROM unnest($1) val
       WHERE VAL IS NOT NULL
       ORDER BY 1
     ),
     cnt AS
     (
       SELECT COUNT(*) AS c FROM q
     )
       SELECT AVG(val)::float8
     FROM
     (
       SELECT val FROM q
       LIMIT 2 - MOD((SELECT c FROM cnt), 2)
       OFFSET GREATEST(CEIL((SELECT c FROM cnt) / 2.0) - 1,0)
       ) q2;
$$
LANGUAGE sql IMMUTABLE;

CREATE AGGREGATE median(anyelement) (
       SFUNG=array_append,
       STYPE=anyarray,
       FINALFUNC=final_median,
       INITCOND='{}'
);
```

Mengingat apa yang telah Anda pelajari sejauh ini, kode untuk membuat fungsi agregat median () mungkin terlihat tidak dapat dipahami. Saya akan membahas fungsi secara lebih mendalam nanti di buku ini, tetapi untuk saat ini perhatikan bahwa kode tersebut berisi dua blok utama: satu untuk membuat fungsi yang disebut _final_median yang mengurutkan nilai dalam kolom dan menemukan titik tengahnya, dan detik yang berfungsi sebagai median fungsi agregat yang dapat dipanggil () dan meneruskan nilai ke _final_median. Untuk saat ini, Anda dapat melewati peninjauan skrip baris demi baris dan cukup jalankan kodenya.

Mari tambahkan fungsi median () ke kueri sensus dan coba di sebelah persentil_cont (), seperti yang ditunjukkan pada:

```
SELECT sum(p0010001) AS “County Sum”,
    round(AVG(p0010001), 0) AS “County Average”;
    median(p0010001) AS “County Median”,
    percentile_cont(.5)
    WITHIN GROUP (ORDER BY p0010001) AS “50th Percentile”
FROM us_counties_2010;
```

Hasil kueri menunjukkan bahwa fungsi median dan fungsi persentil mengembalikan nilai yang sama:

```
County Sum      County Average    County Median     50th Percentile
------------    --------------    -------------     ---------------
308745538       98233             25857             25857
```

Jadi kapan Anda harus menggunakan median() alih-alih fungsi persentil? Tidak ada jawaban yang sederhana. Sintaks median() lebih mudah diingat, meskipun merupakan tugas yang harus disiapkan untuk setiap database, dan khusus untuk PostgreSQL. Juga, dalam praktiknya, median() dieksekusi lebih lambat dan mungkin berkinerja buruk pada kumpulan data besar atau mesin lambat. Di sisi lain, persentil_cont() portabel di beberapa manajer basis data SQL, termasuk Microsoft SQL Server, dan memungkinkan Anda menemukan persentil apa pun dari 0 hingga 100. Pada akhirnya, Anda dapat mencoba keduanya dan memutuskan.

Menemukan Modus

Selain itu, kita dapat menemukan mode, nilai yang paling sering muncul, menggunakan fungsi mode PostgreSQL (). Fungsi ini bukan bagian dari SQL standar dan memiliki sintaks yang mirip dengan fungsi persentil. Listing dibawah ini menunjukkan perhitungan mode () pada p001001, kolom total populasi:

```
SELECT mode() WITHIN GROUP (ORDER BY p0010001)
FROM us_counties_2010;
```

Hasilnya adalah 21720, jumlah populasi yang dibagi oleh kabupaten di Mississippi, Oregon, dan Virginia Barat.

Bekerja dengan angka adalah langkah kunci dalam memperoleh makna dari data Anda, dan dengan keterampilan matematika yang tercakup dalam bab ini, Anda siap untuk menangani dasar-dasar analisis numerik dengan SQL. Nanti di buku ini, Anda akan belajar tentang konsep statistik yang lebih dalam termasuk regresi dan korelasi. Pada titik ini, Anda memiliki dasar-dasar penjumlahan, rata-rata, dan persentil. Anda juga telah mempelajari bagaimana median dapat menjadi penilaian yang lebih adil dari sekelompok nilai daripada rata-rata. Itu saja dapat membantu Anda menghindari kesimpulan yang tidak akurat.

Di bab berikutnya, saya akan memperkenalkan Anda pada kekuatan menggabungkan data dalam dua atau lebih tabel untuk meningkatkan pilihan Anda untuk analisis data. Kami akan menggunakan data Sensus 2010 yang telah Anda muat ke dalam database analisis dan menjelajahi kumpulan data tambahan.

Latihan Soal

Berikut adalah tiga latihan untuk menguji kemampuan matematika SQL Anda:

1. Tulis pernyataan SQL untuk menghitung luas lingkaran yang jari-jarinya 5 inci. (Jika Anda tidak ingat rumusnya, ini adalah pencarian web yang mudah.) Apakah Anda memerlukan tanda kurung dalam perhitungan Anda? Mengapa atau mengapa tidak?
2. Dengan menggunakan data daerah Sensus 2010, cari tahu daerah negara bagian New York mana yang memiliki persentase populasi tertinggi yang diidentifikasi sebagai “Orang Indian Amerika/Alaska Native Alone.” Apa yang dapat Anda pelajari tentang county itu dari penelitian online yang menjelaskan proporsi populasi Indian Amerika yang relatif besar dibandingkan dengan county New York lainnya?
3. Apakah populasi median county 2010 lebih tinggi di California atau New York?

BAB VI
MENGGABUNGKAN TABEL DALAM BASIS DATA RELASIONAL

Di Bab 1, saya memperkenalkan konsep database relasional, sebuah aplikasi yang mendukung data yang disimpan di beberapa, terkait tabel. Dalam model relasional, setiap tabel biasanya menyimpan data pada satu entitas seperti siswa, mobil, pembelian, rumah dan setiap baris dalam tabel menggambarkan salah satu entitas tersebut. Proses yang dikenal sebagai table join memungkinkan kita untuk menghubungkan baris dalam satu tabel ke baris di tabel lain.

Konsep database relasional berasal dari ilmuwan komputer Inggris Edgar F. Codd. Saat bekerja untuk IBM pada tahun 1970, ia menerbitkan sebuah makalah berjudul "A Relational Model of Data for Large Shared Data Banks."

Ide-idenya merevolusi desain database dan mengarah pada pengembangan SQL. Dengan menggunakan model relasional, Anda dapat membuat tabel yang menghilangkan data duplikat, lebih mudah dirawat, dan memberikan peningkatan fleksibilitas dalam menulis kueri untuk mendapatkan data yang Anda inginkan saja.

Menautkan Tabel Menggunakan JOIN

Untuk menghubungkan tabel dalam kueri, kami menggunakan pernyataan JOIN ... ON (atau salah satu varian JOIN lainnya yang akan saya bahas dalam bab ini). Pernyataan JOIN menautkan satu tabel ke tabel lain dalam database selama kueri, menggunakan nilai yang cocok dalam kolom yang kami tentukan di kedua tabel. Sintaksnya mengambil bentuk ini:

```
SELECT *
FROM table_a JOIN table_b
ON table_a.key_column = table_b.foreign_key_column
```

Ini mirip dengan sintaks SELECT dasar yang telah Anda pelajari, tetapi alih-alih menamai satu tabel dalam klausa FROM, kami memberi nama tabel, memberikan kata kunci JOIN, dan kemudian memberi nama tabel kedua. Kata kunci ON mengikuti, di mana kita menentukan kolom yang ingin kita gunakan untuk mencocokkan nilai. Saat kueri berjalan, kueri memeriksa kedua tabel lalu mengembalikan kolom dari kedua tabel yang nilainya cocok dengan kolom yang ditentukan dalam klausa AKTIF.

Pencocokan berdasarkan kesetaraan antara nilai adalah penggunaan paling umum dari klausa ON, tetapi Anda dapat menggunakan ekspresi apa pun yang mengevaluasi hasil Boolean benar atau salah. Misalnya, Anda dapat mencocokkan di mana nilai dari satu kolom lebih besar dari atau sama dengan nilai di kolom lainnya:

```
ON table_a.key_column >= table_b.foreign_key_column
```

Itu jarang terjadi, tetapi ini merupakan opsi jika analisis Anda membutuhkannya.

Menghubungkan Tabel dengan Kolom Kunci

Pertimbangkan contoh tabel yang berhubungan dengan kolom kunci ini: bayangkan Anda adalah seorang analis data dengan tugas memeriksa pengeluaran penggajian badan publik berdasarkan departemen. Anda mengajukan permintaan Freedom of Information Act untuk data gaji agensi tersebut, mengharapkan untuk menerima spreadsheet sederhana yang mencantumkan setiap karyawan dan gaji mereka, diatur seperti ini:

```
dept  location    first_name  last_name   salary
----  --------    ----------  ---------   ------
Tax   Atlanta     Nancy       Jones       62500
Tax   Atalanta    Lee         Smith       59300
IT    Boston      Soo         Nguyen      83000
IT    Boston      Janet       King        95000
```

Tapi bukan itu yang datang. Sebagai gantinya, agensi mengirimi Anda dump data dari sistem penggajiannya: selusin file CSV, masing-masing mewakili satu tabel dalam databasenya. Anda membaca dokumen yang menjelaskan tata letak data (pastikan untuk selalu menanyakannya!) Dan mulailah memahami kolom di setiap tabel. Dua tabel menonjol: satu bernama karyawan dan satu lagi bernama departemen.

Menggunakan listing dibawah ini, mari buat versi tabel ini, sisipkan baris, dan periksa cara menggabungkan data di kedua tabel. Menggunakan database analisis yang telah Anda buat untuk latihan ini, jalankan semua kode, lalu lihat datanya baik dengan menggunakan pernyataan SELECT dasar atau mengklik nama tabel di pgAdmin dan memilih Lihat/Edit Data Semua Baris.

```
CREATE TABLE departements (
   dept_id bigserial,
   dept varchar(100),
   city varchar(100),
   CONTRAINT dept_key PRIMARY KEY (dept_id),
   CONTRAINT dept_city_unique UNIQUE (dept, city),
);

CREATE TABLE employee (
   emp_id bigserial,
   first_name varchar(100),
   last_name varchar(100),
   salary integer,
   dept_id integer REFERENCES departements (dept_id),
   CONSTRAINT emp_key PRIMARY KEY (emp_id),
   CONSTRAINT emp_dept_unique UNIQUE (emp_id, dept_id),
);

INSERT INTO departements (dept, city),
VALUES
   ('Tax', 'Atlanta'),
   ('IT', 'Boston');

INSERT INTO employees (first_name, last_name, salary, dept_id)
VALUES
   ('Nancy', 'Jones', 62500, 1),
   ('Lee', 'Smith', 59300, 1),
   ('Soo', 'Nguyen', 83000, 2),
   ('Janet', 'King', 95000, 2);
```

Kedua tabel mengikuti model relasional Codd di mana masing-masing menggambarkan atribut tentang satu entitas, dalam hal ini departemen dan karyawan agensi. Di tabel departemen, Anda akan melihat konten berikut:

```
dept_id   dept  city
-------   ----  --------
      1   Tax   Atlanta
      2   IT    Boston
```

Kolom dept_id adalah kunci utama tabel. Kunci utama adalah kolom atau kumpulan kolom yang nilainya secara unik mengidentifikasi setiap baris dalam sebuah tabel. Kolom kunci utama yang valid memberlakukan batasan tertentu:

- Kolom atau kumpulan kolom harus memiliki nilai unik untuk setiap baris.
- Kolom atau kumpulan kolom tidak boleh memiliki nilai yang hilang.

Anda mendefinisikan kunci utama untuk departemen dan karyawan menggunakan kata kunci CONSTRAINT, yang akan saya bahas secara mendalam dengan jenis batasan tambahan di Bab 7. Kolom dept_id secara unik mengidentifikasi departemen, dan meskipun contoh ini hanya berisi nama departemen dan kota, tabel seperti itu kemungkinan akan menyertakan informasi tambahan, seperti alamat atau informasi kontak. Tabel karyawan harus memiliki konten berikut:

```
emp_id          first_name  last_name   salary      dept_id
-------         ----------  ---------   ------      -------
      1         Nancy       Jones       62500       1
      2         Lee         Smith       59300       1
      3         Soo         Nguyen      83000       2
      4         Janet       King        95000       2
```

Kolom emp_id secara unik mengidentifikasi setiap baris dalam tabel karyawan.
Agar Anda mengetahui di departemen mana setiap karyawan bekerja, tabel menyertakan kolom dept_id. Nilai di kolom ini mengacu pada nilai di kunci utama tabel departemen. Kami menyebutnya kunci asing, yang Anda tambahkan sebagai batasan saat membuat tabel. Batasan kunci asing memerlukan nilai yang dimasukkan dalam kolom agar sudah ada di kunci utama tabel yang dirujuknya. Jadi, nilai di dept_id di tabel karyawan harus ada di dept_id di tabel departemen; jika tidak, Anda tidak dapat menambahkannya. Tidak seperti kunci utama, kolom kunci asing bisa kosong, dan bisa berisi nilai duplikat.

Dalam contoh ini, dept_id yang terkait dengan karyawan Nancy Jones adalah 1; ini mengacu pada nilai 1 di kunci utama tabel departemen, dept_id. Itu memberitahu kita bahwa Nancy Jones adalah bagian dari departemen Pajak yang berlokasi di Atlanta.

Catatan Nilai kunci utama hanya perlu unik di dalam tabel. Itulah mengapa tabel karyawan dan tabel departemen boleh saja memiliki nilai kunci utama menggunakan angka yang sama.

Kedua tabel tersebut juga menyertakan sebuah UNIQUE constraint, yang juga akan saya bahas secara lebih mendalam di “The UNIQUE Constraint” pada halaman 105. Secara singkat, ini menjamin bahwa nilai dalam kolom, atau kombinasi nilai dalam lebih banyak dari satu kolom, adalah unik. Di departemen, itu mengharuskan setiap baris memiliki pasangan nilai unik untuk

dept dan kota . Pada karyawan, setiap baris harus memiliki pasangan unik emp_id dan dept_id. Anda menambahkan batasan ini untuk menghindari duplikat data. Misalnya, Anda tidak dapat memiliki dua departemen pajak di Atlanta.

Anda mungkin bertanya: apa keuntungan memecah data menjadi komponen-komponen seperti ini? Nah, pertimbangkan seperti apa contoh data ini jika Anda menerimanya seperti yang Anda pikirkan pada awalnya, semuanya dalam satu tabel:

```
dept location    first_name  last_name   salary
---- --------    ----------  ---------   ------
Tax  Atlanta          Nancy       Jones       62500
Tax  Atlanta          Lee         Smith       59300
IT   Boston           Soo         Nguyen      83000
IT   Boston           Janet       King        95000
```

Pertama, ketika Anda menggabungkan data dari berbagai entitas dalam satu tabel, mau tidak mau Anda harus mengulang informasi. Ini terjadi di sini: nama departemen dan lokasi dijabarkan untuk setiap karyawan. Ini bagus ketika tabel terdiri dari empat baris seperti ini, atau bahkan 4.000. Tetapi ketika sebuah tabel menampung jutaan baris, pengulangan string yang panjang adalah mubazir dan membuang-buang ruang yang berharga.

Kedua, menjejalkan data yang tidak terkait ke dalam satu tabel membuat pengelolaan data menjadi sulit. Bagaimana jika departemen Marketing berganti nama menjadi Brand Marketing? Setiap baris dalam tabel akan membutuhkan pembaruan. Lebih mudah untuk menyimpan nama dan lokasi departemen hanya dalam satu tabel dan memperbaruinya hanya sekali.

Sekarang setelah Anda mengetahui dasar-dasar bagaimana tabel dapat berhubungan, mari kita lihat cara menggabungkannya dalam kueri.

Membuat Kueri Beberapa Tabel Menggunakan JOIN

Saat Anda menggabungkan tabel dalam kueri, database menghubungkan baris di kedua tabel di mana kolom yang Anda tentukan untuk bergabung memiliki nilai yang cocok. Hasil kueri kemudian menyertakan kolom dari kedua tabel jika Anda memintanya sebagai bagian dari kueri. Anda juga dapat menggunakan kolom dari tabel gabungan untuk memfilter hasil menggunakan klausa WHERE.

Query yang menggabungkan tabel memiliki sintaks yang mirip dengan pernyataan SELECT dasar.

Perbedaannya adalah bahwa kueri juga menentukan hal berikut:

- Tabel dan kolom untuk digabungkan, menggunakan pernyataan SQL JOIN ... ON
- Jenis join yang akan dilakukan menggunakan variasi kata kunci JOIN

Mari kita lihat sintaks JOIN ... ON secara keseluruhan terlebih dahulu lalu jelajahi berbagai jenis join. Untuk bergabung dengan contoh tabel karyawan dan departemen dan melihat semua data terkait dari keduanya, mulailah dengan menulis kueri seperti yang ada di listing dibawah ini:

```
SELECT *
FROM employees
JOIN departements ON employees.dept_id;
```

Dalam contoh, Anda menyertakan wildcard asterisk dengan pernyataan SELECT untuk memilih semua kolom dari kedua tabel . Selanjutnya, kata kunci JOIN berada di antara dua tabel yang Anda inginkan datanya. Terakhir, Anda menentukan kolom untuk bergabung dengan tabel menggunakan kata kunci ON . Untuk setiap tabel, Anda memberikan nama tabel, titik, dan kolom yang berisi nilai kunci. Tanda sama dengan di antara dua nama tabel dan kolom.

Saat Anda menjalankan kueri, hasilnya menyertakan semua nilai dari kedua tabel di mana nilai di kolom dept_id cocok. Bahkan, bidang dept_id muncul dua kali karena Anda memilih semua kolom dari kedua tabel:

```
emp_id    first_name  last_name   salary      dept_id     dept  city
------    ----------  ---------   ------      -------     ----  -------
1         Nancy       Jones       62500       1           Tax   Atlanta
2         Lee         Smith       59300       1           Tax   Atlanta
3         Soo         Nguyen      83000       2           IT    Boston
4         Janet       King        95000       2           IT    Boston
```

Jadi, meskipun data berada dalam dua tabel, masing-masing dengan kumpulan kolom yang terfokus, Anda dapat membuat kueri tabel tersebut untuk menarik kembali data yang relevan. Dalam “Memilih Kolom Tertentu dalam Gabung” di halaman 85, saya akan menunjukkan kepada Anda cara mengambil hanya kolom yang Anda inginkan dari kedua tabel.

Jenis JOIN

Ada lebih dari satu cara untuk menggabungkan tabel di SQL, dan jenis gabungan yang akan Anda gunakan bergantung pada cara Anda ingin mengambil data. Daftar berikut menjelaskan berbagai jenis gabungan. Saat meninjau masing-masing, ada baiknya untuk memikirkan dua tabel berdampingan, satu di sebelah kiri kata kunci JOIN dan yang lainnya di sebelah kanan.

Contoh berdasarkan data dari setiap gabungan mengikuti daftar:

- **JOIN** Mengembalikan baris dari kedua tabel di mana nilai yang cocok ditemukan di kolom gabungan dari kedua tabel. Sintaks alternatifnya adalah INNER JOIN.
- **LEFT JOIN** Mengembalikan setiap baris dari tabel kiri ditambah baris yang cocok dengan nilai dalam kolom gabungan dari tabel kanan. Ketika baris tabel kiri tidak memiliki kecocokan di tabel kanan, hasilnya tidak menunjukkan nilai dari tabel kanan.
- **RIGHT JOIN** Mengembalikan setiap baris dari tabel kanan ditambah baris yang cocok dengan nilai kunci pada kolom kunci dari tabel kiri. Ketika baris tabel kanan tidak memiliki kecocokan di tabel kiri, hasilnya tidak menunjukkan nilai dari tabel kiri.
- **FULL OUTER JOIN** Mengembalikan setiap baris dari kedua tabel dan mencocokkan baris; kemudian bergabung dengan baris di mana nilai dalam kolom yang digabungkan cocok. Jika tidak ada nilai yang cocok di tabel kiri atau kanan, hasil kueri berisi baris kosong untuk tabel lainnya.
- **CROSS JOIN** Mengembalikan setiap kemungkinan kombinasi baris dari kedua tabel.

Jenis gabungan ini paling baik diilustrasikan dengan data. Katakanlah Anda memiliki dua tabel sederhana yang memuat nama sekolah. Untuk memvisualisasikan tipe gabungan dengan lebih baik, sebut saja tabel school_left danschool_right. Ada empat baris di school_left:

```
id  left_school
--  ------------------------
1   Oak Street School
2   Roosevelt High School
5   Washington Middle School
6   Jefferson High School
```

Ada lima baris di school_right:

```
id  right_school
--  ------------------------
1   Oak Street School
2   Roosevelt High School
3   Morrison Elementary
4   Chase Magnet Academy
6   Jefferson High School
```

Perhatikan bahwa hanya sekolah dengan id 1, 2, dan 6 yang cocok di kedua tabel. Bekerja dengan dua tabel data serupa adalah skenario umum untuk analis data, dan tugas umum adalah mengidentifikasi sekolah mana yang ada di kedua tabel. Menggunakan gabungan yang berbeda dapat membantu Anda menemukan sekolah tersebut, ditambah detail lainnya.

Sekali lagi menggunakan database analisis Anda, jalankan kode di dibawah ini untuk mengisi dua tabel ini:

```
CREATE TABLE schools_left (
      id integer CONSTRAINT left_id_key PRIMARY KEY,
      left_school varchar(30)
);

CREATE TABLE schools_right (
         id integer CONSTRAINT right_id_key PRIMARY KEY,
      right_school varchar(30)
);

INSERT INTO schools_left (id, left_schools) VALUES
      (1, 'Oak Street School'),
      (2, 'Roosevelt High School'),
      (5, 'Washington Middle School'),
      (6, 'Jefferson High School');

INSERT INTO schools_right (id, right_schools) VALUES
      (1, 'Oak Street School'),
      (2, 'Roosevelt High School'),
      (3, 'Morrison Elementary'),
      (4, 'Chase Magnet Academy),
      (6, 'Jefferson High School');
```

Kami membuat dan mengisi dua tabel: deklarasi untuk ini seharusnya sudah terlihat familier, tetapi ada satu elemen baru: kami menambahkan kunci utama ke setiap tabel. Setelah deklarasi untuk kolom id sekolah_kiri dan kolom id sekolah_kanan, kata kunci CONSTRAINT key_name PRIMARY KEY menunjukkan bahwa kolom tersebut akan berfungsi sebagai kunci utama untuk tabel mereka.

Artinya untuk setiap baris di kedua tabel, kolom id harus diisi dan berisi nilai yang unik untuk setiap baris dalam tabel tersebut. Akhirnya, kami menggunakan pernyataan INSERT yang sudah dikenal untuk menambahkan data ke tabel.

JOIN

Kami menggunakan JOIN, atau INNER JOIN, ketika kami ingin mengembalikan baris yang cocok dengan kolom yang kami gunakan untuk bergabung. Untuk melihat contohnya, jalankan kode ini, yang menggabungkan dua tabel yang baru saja Anda buat:

```
SELECT *
FROM schools_left JOIN schools_right
ON schools_left.id = schools_right.id;
```

Mirip dengan metode yang kami gunakan di list diatas, kami menentukan dua tabel untuk digabungkan di sekitar kata kunci JOIN. Kemudian kita tentukan kolom mana yang akan kita gabung, dalam hal ini kolom id dari kedua tabel. Tiga ID sekolah cocok di kedua tabel, jadi JOIN hanya mengembalikan tiga baris ID yang cocok. Sekolah yang hanya ada di salah satu dari dua tabel tidak muncul di hasil. Perhatikan juga bahwa kolom dari tabel kiri ditampilkan di sebelah kiri tabel hasil:

```
id  left_school            id   right_school
--  --------------------       --   ---------------
1   Oak Street School      1    Oak Street School
2   Roosevelt High School  2    Roosevelt High School
6   Jefferson High School  6    Jefferson High School
```

Kapan sebaiknya Anda menggunakan JOIN? Biasanya, saat Anda bekerja dengan kumpulan data yang terstruktur dengan baik, terpelihara dengan baik, dan hanya perlu menemukan baris yang ada di semua tabel yang Anda ikuti. Karena GABUNG tidak menyediakan baris yang hanya ada di salah satu tabel, jika Anda ingin melihat semua data dalam satu atau beberapa tabel, gunakan salah satu tipe gabungan lainnya.

LEFT JOIN dan RIGHT JOIN

Berbeda dengan JOIN, kata kunci LEFT JOIN dan RIGHT JOIN masing-masing mengembalikan semua baris dari satu tabel dan menampilkan baris kosong dari tabel lainnya jika tidak ditemukan nilai yang cocok pada kolom yang digabungkan. Mari kita lihat aksi LEFT JOIN terlebih dahulu. Jalankan kode ini:

```
SELECT *
FROM schools_left LEFT JOIN schools_right
ON schools_left.id = schools_right.id;
```

Hasil kueri menunjukkan keempat baris dari school_left serta tiga baris di school_right tempat bidang id cocok. Karena school_right tidak berisi nilai 5 di kolom right_id-nya, tidak ada kecocokan, jadi LEFT JOIN menampilkan baris kosong di sebelah kanan daripada menghilangkan seluruh baris dari tabel kiri seperti pada JOIN. Baris dari school_right yang tidak cocok dengan nilai apa pun di school_left dihilangkan dari hasil:

```
id  left_schools               id   right_schools
--  -------------------------      --   ---------------
1   Oak Street School          1    Oak Street School
2   Roosevelt High School      2    Roosevelt High School
```

```
5   Washington Middle School
6   Jefferson High School        6     Jefferson High School
```

Kami melihat perilaku serupa tetapi berlawanan dengan menjalankan RIGHT JOIN, seperti pada listing dibawah ini:

```
SELECT *
FROM schools_left RIGHT JOIN schools_right
ON schools_left.id = schools_right.id;
```

Kali ini, kueri mengembalikan semua baris dari sekolah_kanan ditambah baris dari sekolah_kiri di mana kolom id memiliki nilai yang cocok, tetapi kueri tidak mengembalikan baris sekolah_kiri yang tidak cocok dengan sekolah_kanan:

```
id  left_schools                 id    right_schools
--  -------------------------    --    ---------------
1   Oak Street School            1     Oak Street School
2   Roosevelt High School        2     Roosevelt High School
                                 3     Morrison Elementary
                                 4     Chase Magnet Academy
6   Jefferson High School        6     Jefferson High School
```

Anda akan menggunakan salah satu dari jenis gabungan ini dalam beberapa keadaan:

- Anda ingin hasil kueri Anda berisi semua baris dari salah satu tabel.
- Anda ingin mencari nilai yang hilang di salah satu tabel; misalnya, saat Anda membandingkan data tentang entitas yang mewakili dua periode waktu yang berbeda.
- Saat Anda mengetahui beberapa baris dalam tabel yang digabungkan tidak akan memiliki nilai yang cocok.

FULL OUTER JOIN

Saat Anda ingin melihat semua baris dari kedua tabel dalam gabungan, terlepas dari apakah ada kecocokan, gunakan opsi FULL OUTER JOIN. Untuk melihatnya beraksi.

```
SELECT *
FROM schools_left FULL OUTER JOIN schools_right
ON schools_left.id = schools_right.id
```

Hasilnya memberikan setiap baris dari tabel kiri, termasuk baris yang cocok dan kosong untuk baris yang hilang dari tabel kanan, diikuti oleh baris yang hilang dari tabel kanan:

```
id  left_schools                 id    right_schools
--  -------------------------    --    ---------------
1   Oak Street School            1     Oak Street School
2   Roosevelt High School        2     Roosevelt High School
5   Washington Middle School
6   Jefferson High School        6     Jefferson High School
                                 4     Chase Magnet Academy
                                 3     Morrison Elementary
```

Gabungan luar penuh diakui kurang berguna dan lebih jarang digunakan daripada gabungan dalam dan kiri atau kanan. Namun, Anda dapat menggunakannya untuk beberapa tugas: untuk menggabungkan dua sumber data yang sebagian tumpang tindih atau untuk memvisualisasikan sejauh mana tabel berbagi nilai yang cocok.

CROSS JOIN

Dalam kueri CROSS JOIN, hasilnya (juga dikenal sebagai produk Cartesian) mengurutkan setiap baris di tabel kiri dengan setiap baris di tabel kanan untuk menampilkan semua kemungkinan kombinasi baris. Daftar 6-8 menunjukkan sintaks CROSS JOIN; karena join tidak perlu mencari kecocokan antar field kunci, tidak perlu memberikan klausa menggunakan kata kunci ON.

```
SELECT *
FROM schools_left CROSS JOIN schools_right;
```

Hasilnya memiliki 20 baris — produk dari empat baris di tabel kiri dikalikan lima baris di kanan:

```
id  left_schools                id  right_schools
--  -------------------------   --  ---------------
1   Oak Street School           1   Oak Street School
1   Oak Street School           2   Roosevelt High School
1   Oak Street School           3   Morrison Elementary
1   Oak Street School           4   Chase Magnet Academy
1   Oak Street School           6   Jefferson High School
2   Roosevelt High School       1   Oak Street School
2   Roosevelt High School       2   Roosevelt High School
2   Roosevelt High School       3   Morrison Elementary
2   Roosevelt High School       4   Chase Magnet Academy
2   Roosevelt High School       6   Jefferson High School
5   Washington Middle School    1   Oak Street School
5   Washington Middle School    2   Roosevelt High School
5   Washington Middle School    3   Morrison Elementary
5   Washington Middle School    4   Chase Magnet Academy
5   Washington Middle School    6   Jefferson High School
6   Jefferson High School       1   Oak Street School
6   Jefferson High School       2   Roosevelt High School
6   Jefferson High School       3   Morrison Elementary
6   Jefferson High School       4   Chase Magnet Academy
6   Jefferson High School       6   Jefferson High School
```

Kecuali jika Anda ingin rehat kopi ekstra lama, saya sarankan untuk menghindari kueri CROSS JOIN di tabel besar. Dua tabel dengan 250.000 catatan masing-masing akan menghasilkan kumpulan hasil 62,5 miliar baris dan membebani server yang paling keras sekalipun. Penggunaan yang lebih praktis akan menghasilkan data untuk membuat daftar periksa, seperti semua warna yang ingin Anda tawarkan untuk setiap gaya kemeja di gudang.

Menggunakan NULL untuk Menemukan Baris dengan Nilai yang Hilang

Mampu mengungkapkan data yang hilang dari salah satu tabel sangat berharga saat Anda menggali data. Setiap kali Anda bergabung dengan tabel, sebaiknya periksa kualitas data dan pahami lebih baik dengan menemukan apakah semua nilai kunci dalam satu tabel muncul di tabel lain. Ada banyak alasan mengapa perbedaan mungkin ada, seperti kesalahan klerikal, output yang tidak lengkap dari database, atau beberapa perubahan data dari waktu ke waktu. Semua informasi ini adalah konteks penting untuk membuat kesimpulan yang benar tentang data.

Bila Anda hanya memiliki beberapa baris, mengamati data adalah cara mudah untuk mencari baris dengan data yang hilang. Untuk tabel besar, Anda memerlukan strategi yang lebih baik: pemfilteran untuk menampilkan semua baris tanpa kecocokan. Untuk melakukan ini, kami menggunakan kata kunci NULL.

Dalam SQL, NULL adalah nilai khusus yang mewakili kondisi di mana tidak ada data yang ada atau di mana data tidak diketahui karena tidak disertakan. Misalnya, jika seseorang mengisi formulir alamat melewati bidang "Inisial Tengah", daripada menyimpan string kosong dalam database, kami akan menggunakan NULL untuk mewakili nilai yang tidak diketahui. Penting untuk diingat bahwa NULL berbeda dari 0 atau string kosong yang Anda tempatkan di bidang karakter menggunakan dua tanda kutip (""). Kedua nilai tersebut dapat memiliki beberapa makna yang tidak diinginkan yang terbuka untuk salah tafsir, jadi Anda menggunakan NULL untuk menunjukkan bahwa nilainya tidak diketahui. Dan tidak seperti 0 atau string kosong, Anda dapat menggunakan NULL di seluruh tipe data.

Ketika gabungan SQL mengembalikan baris kosong di salah satu tabel, kolom tersebut tidak kembali kosong melainkan kembali dengan nilai NULL. Listing program dibawah ini kita akan menemukan baris tersebut dengan menambahkan klausa WHERE untuk menyaring NULL dengan menggunakan frase IS NULL pada kolom right_id. Jika kita ingin mencari kolom dengan data, kita akan menggunakan IS NOT NULL.

```
SELECT *
  FROM Schools_left LEFT JOIN schools_right
ON schools_left.id = schools_right.id
WHERE schools_right.id IS NULL;
```

Sekarang hasil gabungan hanya menunjukkan satu baris dari tabel kiri yang tidak memiliki kecocokan di sisi kanan.

```
 id   left_schools                 id    right_schools
 --   -------------------------    --    ---------------
 5    Washington Middle School
```

Tiga Jenis Hubungan Tabel

Bagian dari ilmu (atau seni, beberapa orang mungkin mengatakan) menggabungkan tabel melibatkan pemahaman bagaimana desainer database bermaksud untuk menghubungkan tabel, juga dikenal sebagai model relasional database. Tiga jenis hubungan tabel adalah satu ke satu, satu ke banyak, dan banyak ke banyak.

One-to-One Relationship

Dalam contoh JOIN kami pada pembahasa sebelumnya hanya ada satu kecocokan untuk id di masing-masing dari dua tabel. Selain itu, tidak ada nilai id duplikat di kedua tabel: hanya ada satu baris di tabel kiri dengan id 1, dan hanya satu baris di tabel kanan yang memiliki id 1. Dalam bahasa database, ini disebut hubungan satu-ke-satu. Pertimbangkan contoh lain: menggabungkan dua tabel dengan data sensus negara bagian demi negara bagian. Satu tabel mungkin berisi data pendapatan rumah tangga dan data lainnya tentang pencapaian pendidikan. Kedua tabel akan memiliki 51 baris (satu untuk setiap negara bagian ditambah Washington, DC), dan jika kami ingin menggabungkannya pada kunci seperti nama negara bagian, singkatan negara bagian, atau kode geografi standar, kami hanya memiliki satu kecocokan untuk masing-masing baris. nilai kunci di setiap tabel.

One-to-Many Relationship

Dalam hubungan satu ke banyak, nilai kunci pada tabel pertama akan memiliki beberapa nilai pencocokan di kolom gabungan tabel kedua. Pertimbangkan database yang melacak mobil. Satu meja akan menyimpan data pabrikan mobil, dengan masing-masing satu baris untuk Ford, Honda, Kia, dan seterusnya. Tabel kedua dengan nama model, seperti Focus, Civic, Sedona, dan Accord, akan memiliki beberapa baris yang cocok dengan setiap baris di tabel pabrikan.

Many-to-Many Relationship

Dalam hubungan banyak ke banyak, beberapa baris di tabel pertama akan memiliki beberapa baris yang cocok di tabel kedua. Sebagai contoh, tabel pemain bisbol dapat digabungkan ke tabel posisi lapangan. Setiap pemain dapat ditugaskan ke beberapa posisi, dan setiap posisi dapat dimainkan oleh banyak orang. Memahami hubungan ini sangat penting karena membantu kita membedakan apakah hasil kueri secara akurat mencerminkan struktur database.

Memilih Kolom Tertentu dalam Join

Sejauh ini, kami telah menggunakan wildcard asterisk untuk memilih semua kolom dari kedua tabel. Tidak apa-apa untuk pemeriksaan data cepat, tetapi lebih sering Anda ingin menentukan subset kolom. Anda bisa fokus hanya pada data yang Anda inginkan dan menghindari perubahan hasil kueri secara tidak sengaja jika seseorang menambahkan kolom baru ke tabel.

Seperti yang Anda pelajari dalam kueri tabel tunggal, untuk memilih kolom tertentu Anda menggunakan kata kunci SELECT diikuti dengan nama kolom yang diinginkan. Saat menggabungkan tabel, sintaksnya sedikit berubah: Anda harus menyertakan kolom serta nama tabelnya. Alasannya adalah bahwa lebih dari satu tabel dapat berisi kolom dengan nama yang sama, yang tentu saja berlaku untuk tabel yang kita gabungkan selama ini.

Pertimbangkan kueri berikut, yang mencoba mengambil kolom id tanpa memberi nama tabel:

```
SELECT id
FROM schools-left LEFT JOIN schools_right
ON schools_left.id = schools_right.id;
```

Karena id ada di sekolah_kiri dan sekolah_kanan, server memunculkan kesalahan yang muncul di panel hasil pgAdmin: referensi kolom "id" ambigu. Tidak jelas milik id tabel mana.

Untuk memperbaiki kesalahan, kita perlu menambahkan nama tabel di depan setiap kolom yang kita kueri, seperti yang kita lakukan di klausa ON. Liting dibawah ini menunjukkan sintaks, yang menyatakan bahwa kita menginginkan kolom id dari school_left. Kami juga mengambil nama sekolah dari kedua tabel.

```
SELECT schools_left.id,
    schools_left.left_school,
    schools_right.right_school
FROM schools_left LEFT JOIN schools_right
ON schools_left.id = schools_right.id;
```

Kami hanya mengawali setiap nama kolom dengan tabel asalnya, dan sintaks kueri lainnya sama. Hasilnya mengembalikan kolom yang diminta dari setiap tabel:

```
Id  left_schools                 right_schools
--  -------------------------    --------------------
1   Oak Street School Oak        Street School
2   Roosevelt High School        Roosevelt High School
5   Washington Middle School
6   Jefferson High School        Jefferson High School
```

Kita juga dapat menambahkan kata kunci AS yang kita gunakan sebelumnya dengan data sensus untuk memperjelas pada hasil bahwa kolom id berasal dari school_left. Syntax akan terlihat seperti ini:

```
SELECT schools_left.id AS left.id, ...
```

Ini akan menampilkan nama kolom id sekolah_kiri sebagai id_kiri. Kita dapat melakukan ini untuk semua kolom lain yang kita pilih menggunakan sintaks yang sama, tetapi bagian selanjutnya menjelaskan metode lain yang lebih baik yang dapat kita gunakan untuk mengganti nama beberapa kolom.

Menyederhanakan JOIN Syntax dengan Tabel Alias

Memberi nama tabel untuk sebuah kolom cukup mudah, tetapi melakukannya untuk banyak kolom akan mengacaukan kode Anda. Salah satu cara terbaik untuk melayani rekan kerja Anda adalah dengan menulis kode yang dapat dibaca, yang biasanya tidak melibatkan membuat mereka mengarungi nama tabel yang diulang untuk 25 kolom! Cara menulis kode yang lebih ringkas adalah dengan menggunakan pendekatan singkatan yang disebut tabel alias.

Untuk membuat alias tabel, kita menempatkan satu atau dua karakter setelah nama tabel saat kita mendeklarasikannya dalam klausa FROM. (Anda dapat menggunakan lebih dari beberapa karakter untuk sebuah alias, tetapi jika tujuannya adalah untuk menyederhanakan kode, jangan berlebihan.) Karakter tersebut kemudian berfungsi sebagai alias yang dapat kita gunakan sebagai ganti nama tabel lengkap di mana pun kita referensi tabel dalam kode. Kode ini menunjukkan cara kerjanya:

```
SELECT lt.id,
      lt.left_school,
      rt.right_school
FROM schools_left AS lt LEFT JOIN school_right AS rt
ON lt.id = rt.id;
```

Dalam klausa FROM, kita mendeklarasikan alias lt untuk mewakili school_left dan alias rt untuk mewakili school_right menggunakan kata kunci AS. Setelah itu ada, kita dapat menggunakan alias alih-alih nama tabel lengkap di tempat lain dalam kode. Segera, SQL kami terlihat lebih ringkas, dan itu ideal.

JOIN untuk Beberapa Tabel

Tentu saja, gabungan SQL tidak terbatas pada dua tabel. Kami dapat terus menambahkan tabel ke kueri selama kami memiliki kolom dengan nilai yang cocok untuk bergabung. Katakanlah kita mendapatkan dua tabel terkait sekolah lagi dan ingin menggabungkannya ke school_left dalam gabungan tiga tabel. Berikut tabelnya: school_enrollment memiliki jumlah mahasiswa per sekolah:

```
Id  enrollment
--  ----------
1   360
2   1001
5   450
6   927
```

Tabel school_grades berisi tingkat kelas yang ditempatkan di setiap gedung:

```
Id  grades
--  ------
1   K-3
2   9-12
5   6-8
6   9-12
```

Untuk menulis kueri, kami akan menggunakan listing dibawah ini untuk membuat tabel dan memuat data:

```
CREATE TABLE schools_enrollment (
    id integer,
    enrollment integer
);

CREATE TABLE schools_grades (
    id integer,
    grade varchar(10)
);

INSERT INTO school_enrollment (id, enrollment)
VALUES
    (1, 360),
    (2, 1001),
    (5, 450),
    (6, 927);

INSERT INTO schools_grades (id,grades)
VALUES
    (1, 'K-3'),
    (2, '9-12'),
    (5, '6-8'),
    (6, '9-12');

SELECT lt.id, lt.left_school, en.enrollment, gr.grades
FROM schools_left AS lt LEFT JOIN schools_enrollment AS en
    ON lt.id = en.id
LEFT JOIN schools_grades AS gr
    ON lt.id = gr.id;
```

Setelah kita menjalankan bagian CREATE TABLE dan INSERT dari skrip, hasilnya terdiri dari tabel school_enrollment dan school_grades, masing-masing dengan record yang berhubungan dengan school_left dari bab sebelumnya. Kami kemudian menghubungkan ketiga tabel.

Dalam kueri SELECT, kita menggabungkan school_left ke school_enrollment menggunakan bidang tabel 'id. Kami juga mendeklarasikan alias tabel untuk menjaga agar kode tetap kompak. Selanjutnya, kueri menggabungkan school_left ke school_grades lagi pada bidang id. Hasil kami sekarang mencakup kolom dari ketiga tabel:

```
Id  left_schools                enrollment  grades
--  -------------------------   ----------  ------
1   Oak Street School Oak       360         K-3
2   Roosevelt High School       1001        9-12
5   Washington Middle School    450         6-8
6   Jefferson High School       927         9-12
```

Jika perlu, Anda dapat menambahkan lebih banyak tabel ke kueri menggunakan gabungan tambahan. Anda juga dapat bergabung di kolom yang berbeda, tergantung pada hubungan tabel. Meskipun tidak ada batasan ketat dalam SQL untuk jumlah tabel yang dapat Anda gabungkan dalam satu kueri, beberapa sistem database mungkin memberlakukannya. Periksa dokumentasi.

Perhitungan Matematika pada Kolom Tabel yang Digabung

Fungsi matematika yang kita jelajahi di Bab 5 sama bergunanya saat bekerja dengan tabel gabungan. Kita hanya perlu memasukkan nama tabel saat mereferensikan kolom dalam operasi, seperti yang kita lakukan saat memilih kolom tabel. Jika Anda bekerja dengan data apa pun yang memiliki rilis baru secara berkala, Anda akan menemukan konsep ini berguna untuk menggabungkan tabel yang baru dirilis ke tabel yang lebih lama dan menjelajahi bagaimana nilai telah berubah.

Itulah yang saya dan banyak jurnalis lakukan setiap kali data sensus baru dirilis. Kami akan memuat data baru dan mencoba menemukan pola pertumbuhan atau penurunan populasi, pendapatan, pendidikan, dan indikator lainnya. Mari kita lihat bagaimana melakukan ini dengan meninjau kembali tabel us_counties_2010 yang kita buat di Bab 4 dan memuat data daerah serupa dari Sensus Tahunan sebelumnya, pada tahun 2000, ke tabel baru. Jalankan kode ini, pastikan Anda telah menyimpan file CSV di suatu tempat terlebih dahulu:

```
CREATE TABLE us_counties_2000
    geo_name varchar(90),
    state_us_abbreviation varchar(2),
    state_fips varchar(2),
county_fips varchar(3),
    p0010001 integer,
    p0010002 integer,
       p0010003 integer,
       p0010004 integer,
       p0010005 integer,
       p0010006 integer,
       p0010007 integer,
       p0010008 integer,
       p0010009 integer,
       p0010010 integer,
       p0020002 integer,
       p0020003 integer,
);
```

```
COPY us_counties_2000
FROM 'C:\YourDirectory\us_conties_2000.cvs'
WITH (FORMAT CSV, HEADER);

SELECT c2010.geo_name,
     c2010.state_us_abbreviation AS state,
     c2010.p0010001 AS pop_2010,
     c2000.p0010001 AS pop_2000,
     c2010.p0010001 - c2000.p0010001 AS raw_change,
     round( ( CAST(c2010.p0010001 AS numeric(8,1)) - c2000.p0010001)
           / c200.p0010001 * 100 1 ) AS pct_change
     FROM us_counties_2010 c2010 INNER JOIN us_counties_2000 c2000
     ON c2010.state_fips - c2000.state_fips
           AND c2010.county_fips = c2000.county_fips
           AND c2010.p0010001 <> c2000.p0010001
     ORDER BY pct_change DECS;
```

Dalam kode ini, kami membangun fondasi sebelumnya. Kami memiliki pernyataan CREATE TABLE yang sudah dikenal , yang untuk latihan ini mencakup kode negara bagian dan county, kolom geo_name dengan nama lengkap negara bagian dan county, dan sembilan kolom dengan jumlah populasi termasuk total populasi dan jumlah berdasarkan ras. Pernyataan COPY mengimpor file CSV dengan data sensus; Anda dapat menemukan us_counties_2000.csv bersama dengan semua sumber daya buku di https://www.nostarch.com/practicalSQL/. Setelah Anda mengunduh file, Anda harus mengubah jalur file ke lokasi tempat Anda menyimpannya.

Setelah selesai mengimpor, Anda harus memiliki tabel bernama us_counties_2000 dengan 3.141 baris. Seperti halnya data 2010, tabel ini memiliki kolom bernama p001001 yang berisi jumlah penduduk untuk setiap county di Amerika Serikat. Karena kedua tabel memiliki kolom yang sama, masuk akal untuk menghitung persentase perubahan populasi untuk setiap kabupaten antara tahun 2000 dan 2010. Kabupaten mana yang memimpin pertumbuhan negara? Manakah yang mengalami penurunan populasi?

Kami akan menggunakan perhitungan persentase perubahan yang kami gunakan di Bab 5 untuk mendapatkan jawabannya. Pernyataan SELECT menyertakan nama county dan singkatan negara bagian dari tabel 2010, yang disamakan dengan c2010. Berikutnya adalah kolom populasi total p0010001 dari tabel 2010 dan 2000, keduanya diganti namanya dengan nama unik menggunakan AS untuk membedakannya dalam hasil. Untuk mendapatkan perubahan mentah dalam populasi, kita kurangi 2000 populasi dari hitungan 2010, dan untuk menemukan perubahan persen, kita menggunakan rumus dan membulatkan hasilnya ke satu titik desimal.

Kami bergabung dengan mencocokkan nilai dalam dua kolom di kedua tabel: state_fips dan county_fips. Alasan untuk menggabungkan dua kolom dan bukan satu adalah karena di kedua tabel, kita memerlukan kombinasi kode negara bagian dan kode wilayah untuk menemukan wilayah yang unik. Saya telah menambahkan kondisi ketiga untuk mengilustrasikan penggunaan pertidaksamaan. Ini membatasi gabungan ke kabupaten di mana kolom populasi p0010001 memiliki nilai yang berbeda. Kami menggabungkan ketiga kondisi menggunakan kata kunci AND. Dengan menggunakan sintaks itu, gabungan terjadi ketika ketiga kondisi

terpenuhi. Terakhir, hasil diurutkan dalam urutan menurun berdasarkan perubahan persen sehingga kita dapat melihat penanam tercepat di atas.

Itu banyak pekerjaan, tapi itu sepadan. Inilah yang ditunjukkan oleh lima baris pertama dari hasil:

```
name             state pop_2010    pop_2000    raw_change  pct_change
----             ----- ---------   --------    ----------  ----------
Kendall County   IL    114736      54544       60192       110.4
Pinal County     AZ    374770      179727      196043      109.1
Flagler County   FL    95696       49832       45864       92.0
Lincoln County   SD    44828       24131       20697       85.8
Loudoun County   VA    312311      169599      142712      84.1
```

Dua kabupaten, Kendall di Illinois dan Pinal di Arizona, lebih dari dua kali lipat populasi mereka dalam 10 tahun, dengan kabupaten di Florida, South Dakota, dan Virginia tidak jauh di belakang. Itu adalah kisah berharga yang kami ambil dari analisis ini dan titik awal untuk memahami tren populasi nasional. Jika Anda menggali data lebih jauh, Anda mungkin menemukan bahwa banyak kabupaten dengan pertumbuhan terbesar dari tahun 2000 hingga 2010 adalah komunitas kamar tidur pinggiran kota yang diuntungkan dari ledakan perumahan dekade ini, dan tren yang lebih baru melihat orang Amerika meninggalkan daerah pedesaan untuk pindah ke kota. Itu bisa menjadi analisis yang menarik setelah Sensus Desennial 2020.

Mengingat bahwa hubungan tabel adalah dasar untuk arsitektur database, belajar menggabungkan tabel dalam kueri memungkinkan Anda menangani banyak kumpulan data yang lebih kompleks yang akan Anda temui. Bereksperimen dengan berbagai jenis gabungan pada tabel dapat memberi tahu Anda banyak hal tentang bagaimana data dikumpulkan dan diungkapkan ketika ada masalah kualitas. Jadikan mencoba berbagai gabungan sebagai bagian rutin dari eksplorasi kumpulan data baru Anda.

Ke depan, kami akan terus membangun konsep yang lebih besar ini saat kami menelusuri lebih dalam untuk menemukan informasi dalam kumpulan data dan bekerja dengan nuansa penanganan tipe data yang lebih baik dan memastikan kami memiliki data yang berkualitas. Tapi pertama-tama, kita akan melihat satu elemen dasar lagi: menerapkan praktik terbaik untuk membangun database yang andal dan cepat dengan SQL.

Latihan Soal

Lanjutkan eksplorasi gabungan Anda dengan latihan berikut:

1. Tabel us_counties_2010 berisi 3.143 baris, dan us_counties_2000 memiliki 3.141. Itu mencerminkan penyesuaian yang sedang berlangsung pada geografi tingkat kabupaten yang biasanya dihasilkan dari pengambilan keputusan pemerintah. Dengan menggunakan gabungan yang sesuai dan nilai NULL, identifikasi negara mana yang tidak ada di kedua tabel. Untuk bersenang-senang, cari online untuk mencari tahu mengapa mereka hilang.
2. Dengan menggunakan fungsi median() atau persentil_cont() di Bab 5, tentukan median dari persentase perubahan dalam populasi county.
3. Kabupaten mana yang memiliki persentase kehilangan penduduk terbesar antara tahun 2000 dan 2010? Apakah Anda tahu mengapa? (Petunjuk: Peristiwa cuaca besar terjadi pada tahun 2005.)

BAB VII
MENDESAIN TABEL

Obsesi dengan detail bisa menjadi hal yang baik. Saat Anda berlari keluar pintu, itu meyakinkan untuk mengetahui kunci Anda akan tergantung di pengait di mana Anda selalu meninggalkannya. Hal yang sama berlaku untuk desain database. Saat Anda perlu menggali bongkahan informasi dari lusinan tabel dan jutaan baris, Anda akan menghargai dosis obsesi detail yang sama. Saat Anda mengatur data ke dalam kumpulan tabel yang disetel dengan baik dan diberi nama yang cerdas, pengalaman analisis menjadi lebih mudah dikelola.

Dalam bab ini, saya akan membangun Bab 6 dengan memperkenalkan praktik terbaik untuk mengatur dan menyetel database SQL, apakah itu milik Anda atau yang Anda warisi untuk analisis. Anda sudah tahu cara membuat tabel dasar dan menambahkan kolom dengan tipe data dan kunci utama yang sesuai. Sekarang, kita akan menggali lebih dalam desain tabel dengan menjelajahi aturan dan konvensi penamaan, cara menjaga integritas data Anda, dan cara menambahkan indeks ke tabel untuk mempercepat kueri.

Memberi Nama Tabel, Kolom, dan Pengidentifikasi Lainnya Other

Pengembang cenderung mengikuti pola gaya SQL yang berbeda saat memberi nama tabel, kolom, dan objek lain (disebut pengidentifikasi). Beberapa lebih suka menggunakan kotak unta, seperti dalam berrySmoothie, di mana kata-kata dirangkai dan huruf pertama dari setiap kata dikapitalisasi kecuali untuk kata pertama. Huruf Pascal, seperti dalam BerrySmoothie, mengikuti pola yang sama tetapi menggunakan huruf kapital untuk huruf pertama dari kata pertama juga. Dengan kasus ular, seperti dalam berry_smoothie, semua kata adalah huruf kecil dan dipisahkan oleh garis bawah. Sejauh ini, saya telah menggunakan kasus ular di sebagian besar contoh, seperti di tabel us_counties_2010.

Anda akan menemukan pendukung yang bersemangat dari setiap konvensi penamaan, dan beberapa preferensi terkait dengan aplikasi database atau bahasa pemrograman individual. Misalnya, Microsoft merekomendasikan kasus Pascal untuk pengguna SQL Server-nya. Konvensi mana pun yang Anda sukai, yang paling penting adalah memilih gaya dan menerapkannya secara konsisten. Pastikan untuk memeriksa apakah organisasi Anda memiliki panduan gaya atau tawaran untuk berkolaborasi dalam satu, dan kemudian ikuti secara religius.

Mencampur gaya atau mengikuti tidak ada umumnya mengarah ke kekacauan. Akan sulit untuk mengetahui tabel mana yang terbaru, yang merupakan cadangan, atau perbedaan antara dua tabel yang memiliki nama serupa. Misalnya, bayangkan menghubungkan ke database dan menemukan kumpulan tabel berikut:

```
Customers
customers
custBackup
customer_analysis
customer_test2
customer_testMarch2012
customeranalysis
```

Selain itu, bekerja tanpa skema penamaan yang konsisten menyulitkan orang lain untuk menyelami data Anda dan menyulitkan Anda untuk melanjutkan dari bagian terakhir yang Anda tinggalkan.
Mari kita telusuri pertimbangan terkait dengan pengidentifikasi penamaan dan saran untuk praktik terbaik.

Menggunakan Kutipan di Sekitar Pengidentifikasi untuk Mengaktifkan Kasus Campuran
SQL ANSI standar dan banyak varian khusus database dari SQL memperlakukan pengenal sebagai case-insensitive kecuali Anda memberikan pembatas di sekitarnya, biasanya tanda kutip ganda. Pertimbangkan dua pernyataan CREATE TABLE hipotetis ini untuk PostgreSQL:

```
 CREATE TABLE customers (
      customer_id serial,
      -- snip –
);

CREATE TABLE Customers (
      customer_id serial,
      -- snip –
);
```

Ketika Anda menjalankan pernyataan ini secara berurutan, perintah CREATE TABLE pertama membuat tabel yang disebut pelanggan. Tetapi daripada membuat tabel kedua yang disebut Pelanggan, pernyataan kedua akan menimbulkan kesalahan: relasi "pelanggan" sudah ada. Karena Anda tidak mengutip pengidentifikasi, PostgreSQL memperlakukan pelanggan dan Pelanggan sebagai pengidentifikasi yang sama, dengan mengabaikan kasusnya. Jika Anda ingin mempertahankan huruf besar dan membuat tabel terpisah bernama Pelanggan, Anda harus mengapit pengenal dengan tanda kutip, seperti ini:

```
CREATE TABLE “Customers” (
      customer_id serial,
      -- snip –
);
```

Sekarang, PostgreSQL mempertahankan huruf besar C dan membuat Pelanggan serta pelanggan. Kemudian, untuk menanyakan Pelanggan daripada pelanggan, Anda harus mengutip namanya dalam pernyataan SELECT:

```
SELECT * FROM “Customers”;
```

Tentu saja, Anda tidak ingin dua tabel dengan nama yang mirip karena risiko kesalahan yang tinggi. Contoh ini hanya menggambarkan perilaku SQL di PostgreSQL.

Kesalahan dengan Mengutip Pengidentifikasi
Menggunakan tanda kutip juga mengizinkan karakter yang tidak diizinkan dalam pengidentifikasi, termasuk spasi. Tetapi waspadalah terhadap hal-hal negatif dari penggunaan metode ini: misalnya, Anda mungkin ingin memberikan tanda kutip di sekitar "pohon yang ditanam" dan menggunakannya sebagai nama kolom dalam database reboisasi, tetapi kemudian semua pengguna harus memberikan tanda kutip pada setiap referensi berikutnya ke kolom itu.

Abaikan tanda kutip dan database akan merespons dengan kesalahan, mengidentifikasi pohon dan ditanam sebagai kolom terpisah tanpa koma di antaranya. Opsi yang lebih mudah dibaca dan andal adalah menggunakan kasing ular, seperti di tree_planted.

Kelemahan lain dari mengutip adalah memungkinkan Anda menggunakan kata kunci yang dicadangkan SQL, seperti TABLE, WHERE, atau SELECT, sebagai pengenal. Kata kunci yang dicadangkan adalah kata-kata yang ditetapkan SQL sebagai memiliki arti khusus dalam bahasa tersebut. Sebagian besar pengembang basis data tidak suka menggunakan kata kunci yang dicadangkan sebagai pengidentifikasi. Minimal membingungkan, dan paling buruk mengabaikan atau lupa mengutip kata kunci itu nanti akan menghasilkan kesalahan karena database akan menafsirkan kata sebagai perintah, bukan pengenal.

Catatan : Untuk PostgreSQL, Anda dapat menemukan daftar kata kunci yang didokumentasikan di https://www.postgresql.org/docs/current/static/sql-keywords-appendix.html. Selain itu, banyak editor kode dan alat database, termasuk pgAdmin, akan secara otomatis menyorot kata kunci dalam warna tertentu.

Pedoman untuk Penamaan Identifier

Mengingat beban ekstra dalam mengutip dan potensi masalah, yang terbaik adalah menjaga nama pengenal Anda tetap sederhana, tidak dikutip, dan konsisten. Berikut adalah rekomendasi saya:

- Gunakan kotak ular. Kasing ular dapat dibaca dan dapat diandalkan, seperti yang ditunjukkan pada contoh tree_planted sebelumnya. Ini digunakan di seluruh dokumentasi PostgreSQL resmi dan membantu membuat nama multikata mudah dipahami: video_on_demand sekilas lebih masuk akal daripada videoondemand.
- Buatlah nama yang mudah dimengerti dan hindari singkatan yang samar. Jika Anda sedang membangun database yang terkait dengan perjalanan, waktu_datang adalah pengingat konten yang lebih baik sebagai nama kolom daripada arv_tm.
- Untuk nama tabel, gunakan bentuk jamak. Tabel menyimpan baris, dan setiap baris mewakili satu instance dari suatu entitas. Jadi, gunakan nama jamak untuk tabel, seperti guru, kendaraan, atau departemen.
- Pikirkan panjangnya. Jumlah karakter maksimum yang diizinkan untuk nama pengenal bervariasi menurut aplikasi basis data: standar SQL adalah 128 karakter, tetapi PostgreSQL membatasi Anda hingga 63, dan sistem Oracle maksimum adalah 30. Jika Anda menulis kode yang dapat digunakan kembali di basis data lain sistem, condong ke nama pengidentifikasi yang lebih pendek.
- Saat membuat salinan tabel, gunakan nama yang akan membantu Anda mengelolanya nanti. Salah satu metodenya adalah menambahkan tanggal YYYY_MM_DD ke nama tabel saat Anda membuatnya, seperti tire_sizes_2017_10_20. Manfaat tambahan adalah bahwa nama tabel akan mengurutkan dalam urutan tanggal.

Mengontrol Nilai Kolom dengan Batasan

Tipe data kolom sudah secara luas mendefinisikan jenis data yang akan diterimanya: bilangan bulat versus karakter, misalnya. Tetapi SQL menyediakan beberapa batasan tambahan yang memungkinkan kita menentukan lebih lanjut nilai yang dapat diterima untuk kolom

berdasarkan aturan dan pengujian logika. Dengan kendala, kita dapat menghindari fenomena "sampah masuk, sampah keluar", yang terjadi ketika data berkualitas buruk menghasilkan analisis yang tidak akurat atau tidak lengkap. Batasan membantu menjaga kualitas data dan memastikan integritas hubungan antar tabel.

Di Bab 6, Anda mempelajari tentang kunci primer dan kunci asing, yang merupakan dua batasan yang paling umum digunakan. Mari kita tinjau mereka serta jenis kendala tambahan berikut:
CHECK Mengevaluasi apakah data termasuk dalam nilai yang kita tentukan.
UNIQUE Memastikan bahwa nilai dalam kolom atau grup kolom adalah unik di setiap baris dalam tabel.
NOT NULL Mencegah nilai NULL dalam kolom.

Kita dapat menambahkan batasan dalam dua cara: sebagai batasan kolom atau sebagai batasan tabel. Batasan kolom hanya berlaku untuk kolom itu. Ini dideklarasikan dengan nama kolom dan tipe data dalam pernyataan CREATE TABLE, dan akan diperiksa setiap kali ada perubahan pada kolom. Dengan batasan tabel, kami dapat menyediakan kriteria yang berlaku untuk satu atau lebih kolom. Kami mendeklarasikannya dalam pernyataan CREATE TABLE segera setelah mendefinisikan semua kolom tabel, dan itu akan diperiksa setiap kali ada perubahan pada baris dalam tabel.

Kunci Utama: Alami vs. Pengganti
Di Bab 6, Anda belajar tentang memberi tabel sebuah kunci utama: sebuah kolom atau kumpulan kolom yang nilainya secara unik mengidentifikasi setiap baris dalam sebuah tabel. Kunci utama adalah kendala, dan memberlakukan dua aturan pada kolom atau kolom yang membentuk kunci:
1. Setiap kolom dalam kunci harus memiliki nilai unik untuk setiap baris.
2. Tidak ada kolom dalam kunci yang memiliki nilai yang hilang.

Kunci utama juga menyediakan sarana untuk menghubungkan tabel satu sama lain dan mempertahankan integritas referensial, yang memastikan bahwa baris dalam tabel terkait memiliki nilai yang cocok saat kita mengharapkannya. Contoh kunci utama sederhana dalam "Menghubungkan Tabel dengan Kolom Kunci" pada halaman 74 memiliki bidang ID tunggal yang menggunakan bilangan bulat yang dimasukkan oleh kami, pengguna. Namun, seperti kebanyakan area SQL, Anda dapat mengimplementasikan kunci utama dalam beberapa cara. Seringkali, data akan menyarankan jalur terbaik. Tetapi pertama-tama kita harus menilai apakah akan menggunakan kunci alami atau kunci pengganti sebagai kunci utama.

> ***Menggunakan Kolom yang Ada untuk Kunci Alami***
> Anda menerapkan kunci alami dengan menggunakan satu atau beberapa kolom tabel yang ada daripada membuat kolom dan mengisinya dengan nilai buatan untuk bertindak sebagai kunci. Jika nilai kolom mematuhi batasan kunci utama/unik untuk setiap baris dan tidak pernah kosong, ini dapat digunakan sebagai kunci alami. Nilai dalam kolom dapat berubah selama nilai baru tidak menyebabkan pelanggaran batasan.

Contoh kunci alami adalah nomor identifikasi SIM yang dikeluarkan oleh Departemen Kendaraan Bermotor setempat. Dalam yurisdiksi pemerintah, seperti negara bagian di Amerika Serikat, kami cukup berharap bahwa semua pengemudi akan menerima ID unik pada lisensi mereka. Tetapi jika kami menyusun database SIM nasional, kami mungkin tidak dapat membuat asumsi itu; beberapa negara bagian dapat secara independen mengeluarkan kode ID yang sama. Dalam hal ini, kolom driver_id mungkin tidak memiliki nilai unik dan tidak dapat digunakan sebagai kunci alami kecuali jika digabungkan dengan satu atau beberapa kolom tambahan. Terlepas dari itu, saat Anda membuat tabel, Anda akan menemukan banyak nilai yang cocok untuk kunci alami: nomor bagian, nomor seri, atau ISBN buku adalah contoh yang baik.

Memperkenalkan Kolom untuk Kunci Pengganti

Alih-alih mengandalkan data yang ada, kunci pengganti biasanya terdiri dari satu kolom yang Anda isi dengan nilai buatan. Ini mungkin nomor urut yang dibuat secara otomatis oleh database; misalnya, menggunakan tipe data serial (tercakup dalam "Bilangan Bertambah Otomatis" di halaman 27). Beberapa pengembang suka menggunakan Pengidentifikasi Unik Universal (UUID), yang merupakan kode yang terdiri dari 32 digit heksadesimal yang mengidentifikasi perangkat keras atau perangkat lunak komputer. Berikut ini contohnya:

```
2911d8a8-6dea-4a46-af23-d64175a08237
```

Pro dan Kontra dari Jenis Kunci

Seperti kebanyakan debat SQL, ada argumen untuk menggunakan salah satu jenis kunci utama. Alasan yang dikutip untuk menggunakan kunci alami sering kali mencakup hal-hal berikut:

- Data sudah ada di tabel, dan Anda tidak perlu menambahkan kolom untuk membuat kunci.
- Karena data kunci alami memiliki arti, dapat mengurangi kebutuhan untuk menggabungkan tabel saat mencari.

Sebagai alternatif, pendukung kunci pengganti menyoroti poin-poin berikut ini:

- Karena kunci pengganti tidak memiliki arti dalam dirinya sendiri dan nilainya tidak tergantung pada data dalam tabel, jika nanti data Anda berubah, Anda tidak dibatasi oleh struktur kunci.
- Kunci alami cenderung menggunakan lebih banyak penyimpanan daripada bilangan bulat yang biasanya digunakan untuk kunci pengganti.

Tabel yang dirancang dengan baik harus memiliki satu atau lebih kolom yang dapat berfungsi sebagai kunci alami. Contohnya adalah tabel produk dengan kode produk yang unik. Namun dalam tabel karyawan, mungkin sulit untuk menemukan satu kolom, atau bahkan beberapa kolom, yang unik berdasarkan baris demi baris untuk dijadikan sebagai kunci utama. Dalam hal ini, Anda dapat membuat kunci pengganti, tetapi Anda mungkin harus mempertimbangkan kembali struktur tabel.

Sintaks Kunci Utama

Dalam "Jenis JOIN", Anda membuat kunci utama pada tabel kampus_kiri dan kanan_kampus untuk mencoba tipe JOIN. Sebenarnya, ini adalah kunci pengganti: di kedua tabel, Anda

membuat kolom yang disebut id untuk digunakan sebagai kunci dan menggunakan kata kunci CONSTRAINT key_name PRIMARY KEY untuk mendeklarasikannya sebagai kunci utama. Mari kita bahas beberapa contoh kunci utama lainnya.

Di kode dibawah ini, kami mendeklarasikan kunci utama menggunakan metode batasan kolom dan batasan tabel pada tabel yang mirip dengan contoh SIM yang disebutkan sebelumnya. Karena kami berharap ID SIM selalu unik, kami akan menggunakan kolom itu sebagai kunci alami.

```
CREATE TABLE natural_key_example (
      lisence_id varchar(10) CONSTRAIT lisence_key PRIMARY KEY,
      first_name varchar(50),
      last_name varchar(50)
);

DROP TABLE natural_key_example;

CREATE TABLE natural_key_example (
      licence_id varchar(10),
      first_name varchar(50),
      last_name varchat(50),
      CONSTRAINT license_key PRIMARY KEY (license_id)
);
```

Pertama-tama kita menggunakan sintaks batasan kolom untuk mendeklarasikan license_id sebagai kunci utama dengan menambahkan kata kunci CONSTRAINT diikuti dengan nama untuk kunci tersebut dan kemudian kata kunci PRIMARY KEY. Keuntungan menggunakan sintaks ini adalah mudah untuk memahami secara sekilas kolom mana yang ditetapkan sebagai kunci utama. Perhatikan bahwa dalam sintaks kendala kolom Anda dapat menghilangkan kata kunci dan nama CONSTRAINT untuk kunci tersebut, dan cukup gunakan KUNCI UTAMA.

Selanjutnya, kami menghapus tabel dari database dengan menggunakan perintah DROP TABLE untuk mempersiapkan contoh batasan tabel.

Untuk menambahkan kunci utama yang sama menggunakan sintaks batasan tabel, kami mendeklarasikan CONSTRAINT setelah mencantumkan kolom terakhir dengan kolom yang ingin kami gunakan sebagai kunci dalam tanda kurung. Dalam contoh ini, kita berakhir dengan kolom yang sama untuk kunci utama seperti yang kita lakukan dengan sintaks kendala kolom. Namun, Anda harus menggunakan sintaks batasan tabel saat Anda ingin membuat kunci utama menggunakan lebih dari satu kolom. Dalam hal ini, Anda akan mencantumkan kolom dalam tanda kurung, dipisahkan dengan koma. Kami akan menyelidikinya sebentar lagi.

Pertama, mari kita lihat bagaimana memiliki kunci utama melindungi Anda dari merusak integritas data Anda. Kode dibawah ini berisi dua pernyataan INSERT:

```
INSERT INTO natural_key_example (license_id, first_name, last_name)
VALUES ('T229901', 'Lynn', 'Malero');

INSERT INTO natural_key_example (license_id, first_name, last_name)
VALUES ('T229901', 'Sam', 'Tracy');
```

Saat Anda menjalankan pernyataan INSERT pertama sendiri, server memuat baris ke tabel natural_key_example tanpa masalah apa pun. Saat Anda mencoba menjalankan yang kedua, server membalas dengan kesalahan:

```
ERROR: duplicate key value violates unique constrait “license_key”
DETAIL: Key (lisence_id)=(T229901) already exists.
```

Sebelum menambahkan baris, server memeriksa apakah license_id dari T229901 sudah ada di tabel. Karena itu, dan karena kunci utama menurut definisi harus unik untuk setiap baris, server menolak operasi. Aturan dari DMV fiktif menyatakan bahwa tidak ada dua pengemudi yang dapat memiliki ID lisensi yang sama, jadi memeriksa dan menolak data duplikat adalah salah satu cara database untuk menegakkan aturan itu.

Membuat PRIMARY KEY Gabungan

Jika kita ingin membuat kunci alami tetapi satu kolom dalam tabel tidak cukup untuk memenuhi persyaratan kunci utama untuk keunikan, kita mungkin dapat membuat kunci yang sesuai dari kombinasi kolom, yang disebut komposit. kunci utama.’

Sebagai contoh hipotetis, mari gunakan tabel yang melacak kehadiran siswa di sekolah. Kombinasi kolom ID siswa dan kolom tanggal akan memberi kita data unik untuk setiap baris, melacak apakah siswa tersebut bersekolah atau tidak setiap hari selama tahun ajaran.

Untuk membuat kunci utama komposit dari dua atau lebih kolom, Anda harus mendeklarasikannya menggunakan sintaks batasan tabel yang disebutkan sebelumnya. Listing program yang akan kita buat untuk membuat tabel contoh untuk skenario kehadiran mahasiswa. Basis data sekolah akan mencatat setiap student_id hanya sekali per school_day, menciptakan nilai unik untuk baris tersebut. Kolom saat ini dari tipe data boolean menunjukkan apakah mahasiswa ada di sana pada hari itu.

```
CREATE TABLE natural_key_composite_example (
    student_id varchar(10),
    school_day date,
    present Boolean,
    CONSTRAIT student_key PRIMARY KEY (student_id, school_day)
);
```

Sintaks tersebut diatas mengikuti format batasan tabel yang sama untuk menambahkan kunci utama untuk satu kolom, tetapi kita melewatkan dua (atau lebih) kolom sebagai argumen, bukan satu. Sekali lagi, kita dapat mensimulasikan pelanggaran kunci dengan mencoba menyisipkan baris di mana kombinasi nilai dalam dua kolom KEY (student_id dan school_day) tidak unik untuk tabel. Jalankan kode ini:

```
INSERT INTO natural_key_composite_example (student_id, school_day, present)
VALUES(775, ‘1/22/2017’, ‘Y’);

INSERT INTO natural_key_composite_example (student_id, school_day, present)
VALUES(775, ‘1/23/2017’, ‘Y’);

 INSERT INTO natural_key_composite_example (student_id, school_day, present)
VALUES(775, ‘1/23/2017’, ‘N’);
```

Dua pernyataan INSERT pertama dijalankan dengan baik karena tidak ada duplikasi nilai dalam kombinasi kolom kunci. Tetapi pernyataan ketiga menyebabkan kesalahan karena nilai student_id dan school_day yang dikandungnya cocok dengan kombinasi yang sudah ada di tabel:

```
ERROR: duplicate key value violates unique constrait “student_key”
DETAIL: Key (student_id, school_day)=(775, 2017-01-23) already exists.
```

Anda dapat membuat kunci komposit dengan lebih dari dua kolom. Basis data khusus yang Anda gunakan membatasi jumlah kolom yang dapat Anda gunakan.

Membuat Kunci Pengganti Peningkatan Otomatis

Jika tabel yang Anda buat tidak memiliki kolom yang cocok untuk kunci utama alami, Anda mungkin memiliki masalah integritas data; dalam hal ini, yang terbaik adalah mempertimbangkan kembali bagaimana Anda menyusun database. Jika Anda mewarisi data untuk analisis atau sangat ingin menggunakan kunci pengganti, Anda dapat membuat kolom dan mengisinya dengan nilai unik. Sebelumnya, saya menyebutkan bahwa beberapa pengembang menggunakan UUID untuk ini; yang lain mengandalkan perangkat lunak untuk menghasilkan kode unik. Untuk tujuan kita, cara mudah untuk membuat kunci primer pengganti adalah dengan bilangan bulat yang bertambah otomatis menggunakan salah satu tipe data serial yang dibahas dalam “Bilangan Bertambah Otomatis”

Ingat tiga jenis serial: smallserial, serial, dan bigserial. Mereka sesuai dengan tipe integer smallint, integer, dan bigint dalam hal rentang nilai yang mereka tangani dan jumlah penyimpanan disk yang mereka konsumsi. Untuk kunci utama, mungkin tergoda untuk mencoba menghemat ruang disk dengan menggunakan serial, yang menangani angka sebesar 2.147.483.647. Tetapi banyak pengembang basis data telah menerima panggilan larut malam dari pengguna yang panik untuk mengetahui mengapa aplikasi mereka rusak, hanya untuk menemukan bahwa basis data mencoba menghasilkan nomor satu yang lebih besar dari maksimum tipe data. Untuk alasan ini, dengan PostgreSQL, umumnya bijaksana untuk menggunakan bigserial, yang menerima angka setinggi 9,2 triliun.

Anda dapat mengaturnya dan melupakannya, seperti yang ditunjukkan pada kolom pertama yang ditentukan dalam listing dibawah ini:

```
CREATE TABLE surrogate_key_example (
      order_number bigserial,
      product_name varchar(50),
      order_date date,
      CONSTRAIT order_key PRIMARY KEY (order_number)
);


INSERT INTO surrogate_key_example (product_name, order_date)
VALUES (‘Beachball Polish’, ‘2015-03-17’),
      (‘Wringkle De-Atomizer’, ‘2017-05-22’),
      (‘Flux Capacitor’, ‘1985-10-26’);

SELECT * FROM surrogate_key_example;
```

List diatas menunjukkan bagaimana mendeklarasikan tipe data bigserial untuk kolom order_number dan mengatur kolom tersebut sebagai primary key . Saat Anda memasukkan data ke dalam tabel , Anda dapat menghilangkan kolom order_number. Dengan order_number diatur ke bigserial, database akan membuat nilai baru untuk kolom tersebut pada setiap sisipan. Nilai baru akan lebih besar dari nilai terbesar yang sudah dibuat untuk kolom.

Jalankan SELECT * FROM surrogate_key_example; untuk melihat bagaimana kolom terisi secara otomatis:

```
order_number      product_name           order_date
------------      --------------------   ----------
           1      Beachball Polish       2015-03-17
           2      Wrinkle De-Atomizer    2017-05-22
           3      Flux Capacitor         1985-10-26
```

Basis data akan menambahkan satu ke order_number setiap kali baris baru dimasukkan. Tapi itu tidak akan mengisi celah apa pun dalam urutan yang dibuat setelah baris dihapus.

Kunci Asing

Dengan batasan kunci asing, SQL sangat membantu menyediakan cara untuk memastikan data dalam tabel terkait tidak berakhir tidak terkait, atau menjadi yatim piatu. Kunci asing adalah satu atau lebih kolom dalam tabel yang cocok dengan kunci utama tabel lain. Tetapi kunci asing juga memberlakukan batasan: nilai yang dimasukkan harus sudah ada di kunci utama atau kunci unik lainnya dari tabel yang dirujuknya. Jika tidak, nilainya ditolak. Batasan ini memastikan bahwa kita tidak berakhir dengan baris dalam satu tabel yang tidak memiliki hubungan dengan baris di tabel lain tempat kita dapat menggabungkannya.

Sebagai gambaran, kode program dibawah ini menunjukkan dua tabel dari database hipotetis yang melacak aktivitas kendaraan bermotor:

```
CREATE TABLE license (
      license_id varchar(10),
      first_name varchar(50),
      last_name varchar(50),
      CONSTRAINT licenses_key PRIMARY KEY (license_id)
);

CREATE TABLE registrations (
      registration_id varchar(10),
      registration_date date,
      license_id varchar(10) REFERENCES licenses (license_id),
      CONSTRAINT registration_key PRIMARY KEY (registration_id, license_id)
  );

INSERT INTO licenses (license_id, first_name, last_name)
VALUES ('T229901', 'Lynn', 'Malero');

INSERT INTO registrations (registration_id, registration_date, license_id)
VALUES ('A203391', '3/17/2017', 'T229901');

INSERT INTO registrations (registration_id, registration_date, license_id)
VALUES ('A75772', '3/17/2017', 'T000001');
```

Tabel pertama, licenses, mirip dengan tabel natural_key_example yang kita buat sebelumnya dan menggunakan driver's unique license_id sebagai kunci primer alami. Tabel kedua, registrasi, adalah untuk melacak registrasi kendaraan. Sebuah ID lisensi tunggal mungkin terhubung ke beberapa registrasi kendaraan, karena setiap pengemudi berlisensi dapat mendaftarkan beberapa kendaraan selama beberapa tahun. Juga, satu kendaraan dapat didaftarkan ke beberapa pemegang lisensi, membangun, seperti yang Anda pelajari di Bab 6, hubungan banyak-ke-banyak.

Berikut bagaimana hubungan tersebut diekspresikan melalui SQL: di tabel pendaftaran, kami menetapkan kolom license_id sebagai kunci asing dengan menambahkan kata kunci REFERENCES, diikuti dengan nama tabel dan kolom untuk referensi.

Sekarang, ketika kita menyisipkan baris ke dalam pendaftaran, database akan menguji apakah nilai yang dimasukkan ke dalam license_id sudah ada di kolom kunci utama license_id dari tabel lisensi. Jika tidak, database mengembalikan kesalahan, yang penting. Jika ada baris dalam pendaftaran yang tidak sesuai dengan baris dalam lisensi, kami tidak dapat menulis kueri untuk menemukan orang yang mendaftarkan kendaraan tersebut.

Untuk melihat batasan ini beraksi, buat dua tabel dan jalankan pernyataan INSERT satu per satu. Yang pertama menambahkan baris ke licenses yang menyertakan nilai T229901 untuk license_id. Yang kedua menambahkan baris ke pendaftaran di mana kunci asing berisi nilai yang sama. Sejauh ini, sangat bagus, karena nilainya ada di kedua tabel.

Tetapi kami menemukan kesalahan dengan sisipan ketiga, yang mencoba menambahkan baris ke pendaftaran dengan nilai untuk license_id yang tidak ada dalam lisensi:

```
ERROR: insert or update on table “registrations” violates foreign key
constraint “registrations_license_id_fkey”
DETAIL: Key (lisence_id)=(T0000001) is not present in table “licenses”.
```

Kesalahan yang dihasilkan baik karena menunjukkan database menjaga data tetap bersih. Tetapi ini juga menunjukkan beberapa implikasi praktis: pertama, ini mempengaruhi urutan kita memasukkan data. Kami tidak dapat menambahkan data ke tabel yang berisi kunci asing sebelum tabel lain yang direferensikan oleh kunci tersebut memiliki catatan terkait, atau kami akan mendapatkan kesalahan. Dalam contoh ini, kita harus membuat catatan SIM sebelum memasukkan catatan pendaftaran terkait (jika Anda memikirkannya, itulah yang mungkin dilakukan oleh departemen kendaraan bermotor setempat).

Kedua, berlaku sebaliknya ketika kita menghapus data. Untuk menjaga integritas referensial, batasan kunci asing mencegah kami menghapus baris dari lisensi sebelum menghapus baris terkait apa pun dalam pendaftaran, karena hal itu akan meninggalkan rekaman yatim piatu. Kami harus menghapus baris terkait dalam pendaftaran terlebih dahulu, lalu menghapus baris dalam lisensi. Namun, ANSI SQL menyediakan cara untuk menangani urutan operasi ini secara otomatis menggunakan kata kunci ON DELETE CASCADE, yang akan saya bahas selanjutnya.

Secara Otomatis Menghapus Catatan Terkait dengan CASCADE

Untuk menghapus baris dalam lisensi dan membuat tindakan itu secara otomatis menghapus baris terkait dalam pendaftaran, kita dapat menentukan perilaku itu dengan menambahkan ON DELETE CASCADE saat mendefinisikan batasan kunci asing.

Saat kita membuat tabel registrasi, kata kunci akan berada di akhir definisi kolom license_id, seperti ini:

```
CREATE TABLE registrations (
    registration_id varchar(10),
    registration_date date,
    license_id varchar(10) REFERENCES licenses ON DELETE CASCADE,
    CONSTRAIT registration_key PRIMARY KEY (registration_id, License_id)
);
```

Sekarang, menghapus baris dalam lisensi juga harus menghapus semua baris terkait dalam pendaftaran. Ini memungkinkan kami untuk menghapus SIM tanpa terlebih dahulu harus menghapus registrasi secara manual. Itu juga menjaga integritas data dengan memastikan penghapusan lisensi tidak meninggalkan baris yatim piatu dalam pendaftaran.

Kendala CHECK

Batasan CHECK mengevaluasi apakah data yang ditambahkan ke kolom memenuhi kriteria yang diharapkan, yang kami tentukan dengan pengujian logika. Jika kriteria tidak terpenuhi, database mengembalikan kesalahan. Batasan CHECK sangat berharga karena dapat mencegah kolom dimuat dengan data yang tidak masuk akal. Misalnya, tanggal lahir karyawan baru mungkin tidak boleh lebih dari 120 tahun yang lalu, jadi Anda dapat menetapkan batas tanggal lahir. Atau, di sebagian besar sekolah yang saya tahu, Z bukan nilai huruf yang valid untuk suatu mata pelajaran (walaupun nilai aljabar saya hampir tidak terasa seperti itu), jadi kami mungkin memasukkan batasan yang hanya menerima nilai A–F.

Seperti pada kunci utama, kita dapat menerapkan batasan CHECK sebagai batasan kolom atau batasan tabel. Untuk batasan kolom, nyatakan dalam pernyataan CREATE TABLE setelah nama kolom dan tipe data: CHECK (ekspresi logis). Sebagai batasan tabel, gunakan sintaks CONSTRAINT constraint_name CHECK (ekspresi logis) setelah semua kolom didefinisikan.

Daftar 7-7 menunjukkan batasan CHECK yang diterapkan pada dua kolom dalam tabel yang mungkin kita gunakan untuk melacak peran pengguna dan gaji karyawan dalam sebuah organisasi. Ini menggunakan sintaks batasan tabel untuk kunci utama dan batasan CHECK.

```
CREATE TABLE check_constraint_example (
    user_id bigserial,
    user_role varchar(50),
    salary integer,
    CONSTRAINT user_id_key PRIMARY KEY (user_id),
    CONSTRAINT check_role_in_list CHECK (user_role IN('Admin', 'Staff')),
    CONSTRAINT check_salary_not_zero CHECK (salary > 0)
);
```

Kami membuat tabel dan mengatur kolom user_id sebagai kunci primer pengganti yang bertambah secara otomatis. PERIKSA pertama menguji apakah nilai yang dimasukkan ke dalam

kolom user_role cocok dengan salah satu dari dua string yang telah ditentukan sebelumnya, Admin atau Staf, dengan menggunakan operator SQL IN. PERIKSA kedua menguji apakah nilai yang dimasukkan dalam kolom gaji lebih besar dari 0, karena tidak seorang pun boleh mendapatkan jumlah negatif . Kedua tes adalah contoh lain dari ekspresi Boolean, sebuah pernyataan yang dievaluasi sebagai benar atau salah. Jika nilai yang diuji oleh kendala dievaluasi sebagai benar, pemeriksaan lolos.

Catatan : Pengembang dapat memperdebatkan apakah logika cek termasuk dalam database, dalam aplikasi di depan database, seperti sistem sumber daya manusia, atau keduanya. Salah satu keuntungan dari pemeriksaan dalam database adalah bahwa database akan menjaga integritas data jika terjadi perubahan pada aplikasi, bahkan jika sistem baru dibangun atau pengguna diberikan cara alternatif untuk menambahkan data.

Saat nilai dimasukkan atau diperbarui, database memeriksanya dengan batasan. Jika nilai di salah satu kolom melanggar batasan atau dalam hal ini, jika batasan kunci utama dilanggardatabase akan menolak perubahan.
Jika kita menggunakan sintaks batasan tabel, kita juga dapat menggabungkan lebih dari satu pengujian dalam satu pernyataan CHECK. Katakanlah kita memiliki tabel yang berhubungan dengan prestasi siswa. Kita bisa menambahkan yang berikut ini:

```
CONSTRAINT grad_check CHECK (credits >= 120 AND tuition = 'Paid')
```

Perhatikan bahwa kita menggabungkan dua pengujian logika dengan melampirkannya dalam tanda kurung dan menghubungkannya dengan AND. Di sini, kedua ekspresi Boolean harus dievaluasi sebagai true agar seluruh pemeriksaan dapat lulus. Anda juga dapat menguji nilai di seluruh kolom, seperti dalam contoh berikut di mana kami ingin memastikan harga jual item adalah diskon dari aslinya, dengan asumsi kami memiliki kolom untuk kedua nilai:

```
CONSTRAINT sale_check CHECK (sale_prince < retail_price)
```

Di dalam tanda kurung, ekspresi logis memeriksa bahwa harga jual kurang dari harga eceran.

Batasan UNIQUE/UNIK

Kami juga dapat memastikan bahwa kolom memiliki nilai unik di setiap baris dengan menggunakan batasan UNIQUE. Jika memastikan nilai unik terdengar mirip dengan tujuan dari kunci utama, itu benar. Tetapi UNIQUE memiliki satu perbedaan penting. Dalam kunci utama, tidak ada nilai yang bisa NULL, tetapi batasan UNIK memungkinkan beberapa nilai NULL dalam satu kolom.

Untuk menunjukkan kegunaan UNIQUE, lihat kode di bawah ini, yang merupakan tabel untuk melacak info kontak:

```
CREATE TABLE unique_constraint_example (
    contact_id higserial CONSTRAINT contact_id_key PRAMARY KEY,

       first_name varchar(50),
       last_name varchar(50),
       email varchar(200),
```

```
        CONSTRAINT email_unique UNIQUE (email)
);

INSERT INTO unique_constraint_example (first_name, last_name, email)
VALUES ('Samantha', 'Lee', 'slee@example.org');

INSERT INTO unique_constraint_example (first_name, last_name, email)
VALUES ('Betty', 'Diaz', 'bdiaz@example.org');

INSERT INTO unique_constraint_example (first_name, last_name, email)
VALUES ('Sasha', 'Lee', 'slee@example.org');
```

Dalam tabel ini, contact_id berfungsi sebagai kunci primer pengganti, yang secara unik mengidentifikasi setiap baris. Tetapi kami juga memiliki kolom email, titik kontak utama dengan setiap orang. Kami berharap kolom ini hanya berisi alamat email unik, tetapi alamat tersebut mungkin berubah seiring waktu. Jadi, kami menggunakan UNIQUE untuk memastikan bahwa setiap kali kami menambahkan atau memperbarui email kontak, kami tidak memberikan email yang sudah ada. Jika kami mencoba memasukkan email yang sudah ada, database akan mengembalikan kesalahan:

```
ERROR: duplicate key value violates unique constraint "email_unique"
DETAIL: Key (email)=(slee@example.org) already exists.
```

Sekali lagi, kesalahan menunjukkan database berfungsi untuk kami.

Batasan NOT NULL

Di Bab 6, Anda mempelajari tentang NULL, nilai khusus dalam SQL yang mewakili kondisi di mana tidak ada data yang ada di baris dalam kolom atau nilainya tidak diketahui. Anda juga telah mempelajari bahwa nilai NULL tidak diperbolehkan dalam kunci utama, karena kunci utama perlu mengidentifikasi setiap baris dalam tabel secara unik. Tetapi akan ada kolom lain selain kunci utama di mana Anda tidak ingin mengizinkan nilai kosong. Misalnya, dalam tabel yang mencantumkan setiap siswa di sekolah, kolom yang berisi nama depan dan belakang harus diisi untuk setiap baris. Untuk meminta nilai dalam kolom, SQL menyediakan batasan NOT NULL, yang hanya mencegah kolom menerima nilai kosong.

```
CREATE TABLE not_null_example (
      student_id bigserial,
      first_name varchar(50) NOT NULL,
      last_name varchar(50) NOT NULL,
      CONSTRAINT student_id_key PRIMARY KEY (student_id)
);
```

Di sini, kami mendeklarasikan NOT NULL untuk kolom first_name dan last_name karena kemungkinan kami memerlukan informasi tersebut dalam tabel yang melacak informasi siswa. Jika kami mencoba INSERT pada tabel dan tidak menyertakan nilai untuk kolom tersebut, database akan memberi tahu kami tentang pelanggaran tersebut.

Menghapus Batasan atau Menambahkannya

Sejauh ini, kami telah menempatkan batasan pada tabel pada saat pembuatan. Anda juga dapat menghapus batasan atau menambahkan batasan ke tabel yang sudah ada menggunakan ALTER TABLE, perintah SQL yang membuat perubahan pada tabel dan kolom. Kami akan

bekerja dengan ALTER TABLE lebih banyak di Bab 9, tetapi untuk saat ini kami akan meninjau sintaks untuk menambahkan dan menghapus batasan.

Untuk menghapus kunci utama, kunci asing, atau batasan UNIK, Anda akan menulis pernyataan ALTER TABLE dalam format ini:

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

Untuk menghapus batasan NOT NULL, pernyataan beroperasi pada kolom, jadi Anda harus menggunakan kata kunci ALTER COLUMN tambahan, seperti:

```
ALTER TABLE table_name  ALTER COLUMN column_name DROP NOT NULL;
```

Mari kita gunakan pernyataan ini untuk memodifikasi tabel not_null_example yang baru saja Anda buat, seperti yang ditunjukkan pada kode ini:

```
ALTER TABLE not_null_example DROP CONSTRAINT student_id_key;
ALTER TABLE not_null_example ADD CONSTRAINT student_id_key PRIMARY KEY (student_id);
ALTER TABLE not_null_example ALTER COLUMN first_name DROP NOT NULL;
ALTER TABLE not_null_example ALTER COLUMN first_name SET NOT NULL;
```

Jalankan pernyataan satu per satu untuk membuat perubahan pada tabel. Setiap kali, Anda dapat melihat perubahan definisi tabel di pgAdmin dengan mengklik nama tabel satu kali, lalu mengklik tab SQL di atas jendela kueri. Dengan pernyataan ALTER TABLE pertama, kami menggunakan DROP CONSTRAINT untuk menghapus kunci utama bernama student_id_key. Kami kemudian menambahkan kunci utama kembali menggunakan ADD CONSTRAINT. Kami akan menggunakan sintaks yang sama untuk menambahkan batasan ke tabel yang ada.

Catatan : Anda hanya dapat menambahkan batasan ke tabel yang ada jika data di kolom target mematuhi batasan batasan. Misalnya, Anda tidak dapat menempatkan batasan kunci utama pada kolom yang memiliki nilai duplikat atau kosong.

Dalam pernyataan ketiga, ALTER COLUMN dan DROP NOT NULL menghapus batasan NOT NULL dari kolom first_name. Terakhir, SET NOT NULL menambahkan batasan.

Mempercepat Kueri dengan Indeks

Dengan cara yang sama seperti indeks buku membantu Anda menemukan informasi lebih cepat, Anda dapat mempercepat kueri dengan menambahkan indeks ke satu atau beberapa kolom. Basis data menggunakan indeks sebagai jalan pintas daripada memindai setiap baris untuk menemukan data. Itu memang gambaran sederhana tentang apa, dalam database SQL, adalah topik yang tidak sepele. Saya dapat menulis beberapa bab tentang indeks SQL dan menyetel basis data untuk kinerja, tetapi sebaliknya saya akan menawarkan panduan umum tentang penggunaan indeks dan contoh khusus PostgreSQL yang menunjukkan manfaatnya.

B-Tree: Indeks Default PostgreSQL

Saat mengikuti buku ini, Anda telah membuat beberapa indeks, mungkin tanpa mengetahuinya. Setiap kali Anda menambahkan kunci utama atau batasan UNIK ke tabel, PostgreSQL (serta sebagian besar sistem basis data) menempatkan indeks pada kolom. Indeks

disimpan secara terpisah dari data tabel, namun diakses secara otomatis saat Anda menjalankan kueri dan diperbarui setiap kali baris ditambahkan atau dihapus dari tabel.
Di PostgreSQL, tipe indeks default adalah indeks B-Tree. Itu dibuat secara otomatis pada kolom yang ditunjuk untuk kunci utama atau batasan UNIK, dan itu juga tipe yang dibuat secara default saat Anda menjalankan pernyataan CREATE INDEX. B-Tree, kependekan dari balanced tree, dinamakan demikian karena strukturnya mengatur data sedemikian rupa sehingga ketika Anda mencari suatu nilai, nilai itu terlihat dari atas pohon ke bawah melalui cabang-cabang hingga menemukan data yang Anda inginkan. (Tentu saja, prosesnya jauh lebih rumit dari itu. Awal yang baik untuk memahami lebih banyak tentang B-Tree adalah entri Wikipedia B-Tree.) Indeks B-Tree berguna untuk data yang dapat dipesan dan dicari menggunakan operator kesetaraan dan rentang, seperti <, <=, =, >=, >, dan BETWEEN.

PostgreSQL menggabungkan tipe indeks tambahan, termasuk Generalized Inverted Index (GIN) dan Generalized Search Tree (GiST). Masing-masing memiliki kegunaan yang berbeda, dan saya akan menggabungkannya di bab selanjutnya tentang pencarian teks lengkap dan kueri menggunakan tipe geometri.

Untuk saat ini, mari kita lihat indeks B-Tree mempercepat permintaan pencarian sederhana. Untuk latihan ini, kami akan menggunakan kumpulan data besar yang terdiri dari lebih dari 900.000 alamat jalan Kota New York, yang disusun oleh proyek OpenAddresses di https:// openaddresses.io/. File dengan data, city_of_new_york.csv, tersedia untuk Anda unduh bersama dengan semua sumber daya untuk buku ini dari
https://www.nostarch.com/practicalSQL/

Setelah Anda mengunduh file, gunakan kode ini untuk membuat tabel new_york_addresses dan mengimpor data alamat. Anda sudah ahli dalam hal ini sekarang, meskipun impor akan memakan waktu lebih lama daripada kumpulan data kecil yang telah Anda muat sejauh ini. Tabel terakhir yang dimuat adalah 126MB, dan di salah satu sistem saya, butuh hampir satu menit untuk menyelesaikan perintah COPY.

```
CREATE TABLE new_york_addresses (
            longitude numeric(9,6),
            latitude numeric(9,6),

            street_number varchar(10),
            street varchar(32),
            unit varchar(7),
            postcode varchar(5),
            id_integer CONSTRAINT ney_york_key PRIMARY KEY
);

COPY new_york_address
FROM 'C:\YourDirectory\city_of_newyork.csv
WITH (FORMAT CSV, HEADER);
```

Saat data dimuat, jalankan kueri SELECT cepat untuk memeriksa secara visual bahwa Anda memiliki 940.374 baris dan tujuh kolom. Penggunaan umum untuk data ini mungkin untuk mencari kecocokan di kolom jalan, jadi kami akan menggunakan contoh itu untuk menjelajahi kinerja indeks.

Membandingkan Kinerja Kueri dengan EXPLAIN

Kami akan mengukur seberapa baik indeks dapat meningkatkan kecepatan kueri dengan memeriksa kinerja sebelum dan sesudah menambahkannya. Untuk melakukan ini, kami akan menggunakan perintah EXPLAIN PostgreSQL, yang khusus untuk PostgreSQL dan bukan bagian dari SQL standar. Perintah EXPLAIN menyediakan output yang mencantumkan rencana kueri untuk kueri database tertentu. Ini mungkin termasuk bagaimana database berencana untuk memindai tabel, apakah akan menggunakan indeks atau tidak, dan seterusnya. Jika kita menambahkan kata kunci ANALYZE, EXPLAIN akan menjalankan query dan menunjukkan waktu eksekusi yang sebenarnya, yang kita inginkan untuk latihan saat ini.

Merekam Beberapa Waktu Eksekusi Kontrol

Jalankan masing-masing dari tiga kueri di Daftar 7-12 satu per satu. Kami menggunakan kueri SELECT yang khas dengan klausa WHERE tetapi dengan kata kunci EXPLAIN ANALYZE disertakan di awal. Alih-alih menampilkan hasil kueri, kata kunci ini memberi tahu database untuk mengeksekusi kueri dan menampilkan statistik tentang proses kueri dan berapa lama waktu yang dibutuhkan untuk mengeksekusi.

```
EXPLAIN ANALYZE SELECT * FROM new_york_addresses
WHERE street = 'BROADWAY';

EXPLAIN ANALYZE SELECT * FROM new_york_addresses
WHERE street = '52 STREET';

EXPLAIN ANALYZE SELECT * FROM new_york_addresses
WHERE street = 'ZWICKY AVENUE';
```

Di sistem saya, kueri pertama mengembalikan statistik ini:

```
Seq Scan on new_york_address (Cost=0.00..20730.68 rows=3730 width=46)
(actual time=0.055..289.426 rows=3336 loops=1)
    Filter: ((street)::text = 'BROADWAY'::text)
    Rows Removed by Filters: 937038
Planning time: 0.617 ms
Execution time: 289.838 ms
```

Tidak semua output relevan di sini, jadi saya tidak akan mendekode semuanya, tetapi dua baris relevan. Yang pertama menunjukkan bahwa untuk menemukan baris mana pun di mana jalan = 'BROADWAY', database akan melakukan pemindaian berurutan ❶ dari tabel. Itu sinonim untuk pemindaian tabel penuh: setiap baris akan diperiksa, dan basis data akan menghapus setiap baris yang tidak cocok dengan BROAWAY. Waktu eksekusi (di komputer saya sekitar 290 milidetik) adalah berapa lama waktu yang dibutuhkan. Waktu Anda akan tergantung pada faktor-faktor termasuk perangkat keras komputer Anda.

Menambahkan Indeks

Sekarang, mari kita lihat bagaimana menambahkan indeks mengubah metode pencarian kueri dan seberapa cepat kerjanya. Listing ini menunjukkan pernyataan SQL untuk membuat indeks dengan PostgreSQL:

```
CREATE INDEX street_idx ON new_york_address (street);
```

Perhatikan bahwa ini mirip dengan perintah untuk membuat batasan yang telah kita bahas di bab ini. (Sistem database lain memiliki varian dan opsi sendiri untuk membuat indeks, dan tidak ada standar ANSI.) Kami memberikan kata kunci CREATE INDEX diikuti dengan nama yang kami pilih untuk indeks, dalam hal ini street_idx. Kemudian ON ditambahkan, diikuti oleh tabel dan kolom target.

Jalankan pernyataan CREATE INDEX, dan PostgreSQL akan memindai nilai di kolom jalan dan membuat indeks darinya. Kita hanya perlu membuat indeks sekali. Saat tugas selesai, jalankan kembali masing-masing dari tiga kueri di listing dibawah ini dan catat waktu eksekusi yang dilaporkan oleh EXPLAIN ANALYZE. Sebagai contoh:

```
Bitmap Heap Scan on new_york_addresses (cost=65.80..5962.17 rows=2758 width = 46)
(actual time=1.792..9.816 rows=3336 loops=1)
      Recheck Cond: ((street)::text = 'BROADWAY'::text)
      Heap Blocks: exact=2157
      -> Bitmap Index Scan on street_idx (cost=0.00..65.11 rows=2758 width=0)
      (actual time=1.253..1.253 rows=3336 loops=1)
            Index Cond: ((street)::text = 'BROADWAY'::text)
Planning time: 0.163 ms
Execution tim: 5.887 ms
```

Apakah Anda melihat perubahan? Pertama, alih-alih pemindaian sekuensial, statistik EXPLAIN ANALYZE untuk setiap kueri menunjukkan bahwa basis data sekarang menggunakan pemindaian indeks pada street_idx alih-alih mengunjungi setiap baris. Juga, kecepatan kueri sekarang jauh lebih cepat . Tabel 7.1 menunjukkan waktu eksekusi (dibulatkan) dari komputer saya sebelum dan sesudah menambahkan file index.

Tabel 7.1: Mengukur Kinerja Indeks

Filter Kueri	Sebelum Indeks	Setelah Indeks
WHERE street = 'BROADWAY'	290 ms	6 ms
WHERE street = '52 STREET'	271 ms	6 ms
WHERE street = 'ZWICKY AVENUE'	306 ms	1 ms

Waktu eksekusi jauh, jauh lebih baik, secara efektif seperempat detik lebih cepat atau lebih per kueri. Apakah seperempat detik itu mengesankan? Baik, apakah Anda mencari jawaban dalam data menggunakan kueri berulang atau membuat sistem basis data untuk ribuan pengguna, penghematan waktu bertambah.

Jika Anda perlu menghapus indeks dari tabel, mungkin jika Anda menguji kinerja beberapa jenis indeks, gunakan perintah DROP INDEX diikuti dengan nama indeks yang akan dihapus.

Pertimbangan Saat Menggunakan Indeks

Anda telah melihat bahwa indeks memiliki manfaat kinerja yang signifikan, jadi apakah itu berarti Anda harus menambahkan indeks ke setiap kolom dalam tabel? Tidak begitu cepat! Indeks itu berharga, tetapi tidak selalu dibutuhkan. Selain itu, mereka memperbesar database dan membebankan biaya pemeliharaan pada penulisan data. Berikut adalah beberapa tip untuk menilai kapan harus menggunakan indeks:

- Lihat dokumentasi untuk manajer database yang Anda gunakan untuk mempelajari tentang jenis indeks yang tersedia dan yang akan digunakan pada tipe data tertentu. PostgreSQL, misalnya, memiliki lima jenis indeks selain B-Tree. Satu, yang disebut GiST, sangat cocok untuk tipe data geometri yang akan saya bahas nanti di buku ini. Pencarian teks lengkap, yang akan Anda pelajari di Bab 13, juga mendapat manfaat dari pengindeksan.
- Pertimbangkan untuk menambahkan indeks ke kolom mana pun yang akan Anda gunakan dalam gabungan tabel. Kunci utama diindeks secara default di PostgreSQL, tetapi kolom kunci asing di tabel terkait tidak dan merupakan target yang baik untuk indeks.
- Tambahkan indeks ke kolom yang akan sering berakhir di klausa WHERE query. Seperti yang Anda lihat, kinerja pencarian meningkat secara signifikan melalui indeks.
- Gunakan EXPLAIN ANALYZE untuk menguji kinerja di bawah berbagai konfigurasi jika Anda tidak yakin. Optimalisasi adalah sebuah proses!

Dengan alat yang telah Anda tambahkan ke kotak alat di bab ini, Anda siap untuk memastikan bahwa database yang Anda bangun atau warisi paling cocok untuk pengumpulan dan eksplorasi data Anda. Kueri Anda akan berjalan lebih cepat, Anda dapat mengecualikan nilai yang tidak diinginkan, dan objek database Anda akan memiliki organisasi yang konsisten. Itu adalah keuntungan bagi Anda dan orang lain yang membagikan data Anda.

Bab ini menyimpulkan bagian pertama dari buku ini, yang berfokus pada memberi Anda hal-hal penting untuk menggali database SQL. Saya akan terus membangun fondasi ini saat kami menjelajahi kueri dan strategi yang lebih kompleks untuk analisis data. Di bab berikutnya, kita akan menggunakan fungsi agregat SQL untuk menilai kualitas kumpulan data dan mendapatkan informasi yang dapat digunakan darinya.

Latihan Soal

Apakah Anda siap untuk menguji diri Anda pada konsep-konsep yang dibahas dalam bab ini? Pertimbangkan dua tabel berikut dari database yang Anda buat untuk melacak koleksi vinyl LP Anda. Mulailah dengan meninjau pernyataan CREATE TABLE ini:

```
CREATE TABLE albums (
      album_id bigserial,
      album_catalog_code varchar(100),
      album_title text,
      album_artist text,
      album_release_date date,
      album_genre varchar(40),
      album_description text
);

CREATE TABLE song (
      song_id bigserial,
      song_title text,
      song_artis text,
      album_id bigint
);
```

Tabel album mencakup informasi khusus untuk keseluruhan koleksi lagu pada disk. Tabel lagu mengkatalogkan setiap lagu di album. Setiap lagu memiliki judul dan kolom artisnya sendiri, karena setiap lagu mungkin menampilkan koleksi artisnya sendiri.

Gunakan tabel untuk menjawab pertanyaan berikut:

1. Ubah pernyataan CREATE TABLE ini untuk memasukkan kunci utama dan kunci asing ditambah batasan tambahan pada kedua tabel. Jelaskan mengapa Anda membuat pilihan Anda.
2. Alih-alih menggunakan album_id sebagai kunci pengganti untuk kunci utama Anda, apakah ada kolom dalam album yang dapat berguna sebagai kunci alami? Apa yang harus Anda ketahui untuk memutuskan?
3. Untuk mempercepat kueri, kolom mana yang merupakan kandidat yang baik untuk indeks?

BAB VIII
MENGGALI INFORMASI DENGAN MENGELOMPOKKAN JAWABAN

Setiap kumpulan data menceritakan sebuah cerita, dan tugas analis data adalah mencari tahu apa cerita itu. Di Bab 2, Anda mempelajari tentang mewawancarai data menggunakan pernyataan SELECT, yang mencakup pengurutan kolom, menemukan nilai yang berbeda, dan memfilter hasil. Anda juga telah mempelajari dasar-dasar matematika SQL, tipe data, desain tabel, dan menggabungkan tabel. Dengan semua alat ini di bawah ikat pinggang Anda, Anda siap untuk meringkas data menggunakan pengelompokan dan fungsi SQL.

Meringkas data memungkinkan kami mengidentifikasi informasi berguna yang tidak dapat kami lihat sebaliknya. Dalam bab ini, kami akan menggunakan institusi perpustakaan lokal Anda yang terkenal sebagai contoh kami.

Meskipun ada perubahan dalam cara orang mengkonsumsi informasi, perpustakaan tetap menjadi bagian penting dari komunitas di seluruh dunia. Namun internet dan kemajuan teknologi perpustakaan telah mengubah cara kita menggunakan perpustakaan. Misalnya, ebook dan akses online ke materi digital sekarang memiliki tempat permanen di perpustakaan bersama dengan buku dan majalah.

Di Amerika Serikat, Institute of Museum and Library Services (IMLS) mengukur aktivitas perpustakaan sebagai bagian dari Survei Perpustakaan Umum tahunan. Survei ini mengumpulkan data dari lebih dari 9.000 entitas administrasi perpustakaan, yang didefinisikan oleh survei sebagai lembaga yang menyediakan layanan perpustakaan ke lokasi tertentu. Beberapa lembaga adalah sistem perpustakaan kabupaten, dan yang lainnya adalah bagian dari distrik sekolah. Data masing-masing instansi meliputi jumlah cabang, staf, buku, jam buka per tahun, dan sebagainya. IMLS telah mengumpulkan data setiap tahun sejak 1988 dan mencakup semua lembaga perpustakaan umum di 50 negara bagian ditambah Distrik Columbia dan beberapa wilayah, seperti Samoa Amerika. (Baca lebih lanjut tentang program ini di https://www.imls.gov/research-evaluation/data-collection/public-libraries-survey/.)

Untuk latihan ini, kami akan mengambil peran sebagai analis yang baru saja menerima salinan baru kumpulan data perpustakaan untuk menghasilkan laporan yang menjelaskan tren dari data. Kita perlu membuat dua tabel, satu dengan data dari survei 2014 dan yang kedua dari survei 2009. Kemudian kami akan merangkum data yang lebih menarik di setiap tabel dan menggabungkan tabel untuk melihat tren lima tahun. Selama analisis, Anda akan mempelajari teknik SQL untuk meringkas data menggunakan fungsi dan pengelompokan agregat.

Membuat Tabel Survei Perpustakaan

Mari buat tabel survei perpustakaan 2014 dan 2009 dan impor datanya. Kami akan menggunakan tipe data yang sesuai untuk setiap kolom dan menambahkan batasan dan indeks ke setiap tabel untuk menjaga integritas data dan mempercepat kueri.

Membuat Tabel Data Perpustakaan 2014

Kita akan mulai dengan membuat tabel untuk data perpustakaan 2014. Menggunakan pernyataan CREATE TABLE, sintaks berikut ini untuk membuat pls_fy2014_pupld14a, tabel untuk File Data Perpustakaan Umum tahun fiskal 2014 dari Survei Perpustakaan Umum. File Data Perpustakaan Umum merangkum data di tingkat instansi, menghitung aktivitas di semua outlet instansi, yang meliputi perpustakaan pusat, perpustakaan cabang, dan toko buku. Survei tahunan menghasilkan dua file tambahan yang tidak akan kami gunakan: satu merangkum data di tingkat negara bagian, dan yang lainnya memiliki data di masing-masing outlet. Untuk latihan ini, file-file tersebut berlebihan, tetapi Anda dapat membaca tentang data yang ada di dalam kamus data 2014, tersedia dari IMLS di https://www.imls.gov/sites/default/files/fy2014_pls_data_file_documentation.pdf.

Untuk kenyamanan, saya telah membuat skema penamaan untuk tabel: pls mengacu pada judul survei, fy2014 adalah tahun fiskal yang dicakup data, dan pupld14a adalah nama file tertentu dari survei. Untuk kesederhanaan, saya telah memilih hanya 72 kolom yang lebih relevan dari 159 di file survei asli untuk mengisi tabel pls_fy2014_pupld14a, tidak termasuk data seperti kode yang menjelaskan sumber tanggapan individu. Ketika perpustakaan tidak menyediakan data, agensi memperoleh data menggunakan cara lain, tetapi kami tidak memerlukan informasi itu untuk latihan ini.

Perhatikan bahwa Daftar 8-1 disingkat untuk kemudahan. Kumpulan data lengkap dan kode untuk membuat dan memuat tabel ini tersedia untuk diunduh dengan semua sumber buku di https://www.nostarch.com/practicalSQL/.

```
CREATE TABLE pls_fy2014_pupld14a (
      stabr varchar(2) NOT NULL,
      libid varchar(6) CONSTRAINT fscskey2014_key PRIMARY KEY,
      libname varchar(100) NOT NULL,
      obereg varchar(2) NOT NULL,
      rstatus integer NOT NULL,
      statstru varchar(2) NOT NULL,
      statname varchar(2) NOT NULL,
      stataddr varchar(2) NOT NULL,
      --snip--
      wifisess integer NOT NULL,
      yr_sub integer NOT NULL
);

CREATE INDEX libname2014_idx ON pls_fy2014_pupld14a (libname);
CREATE INDEX statbr2014_idx ON pls_fy2014_pupld14a (stabr);
CREATE INDEX city2014_idx ON pls_fy2014_pupld14a (city);
CREATE INDEX visits2014_idx ON pls_fy2014_pupld14a (visits);

COPY pls_fy2014_pupld14a
FROM 'C:\YourDirectory\pls_fy2014_pupld14a.csv'
WITH (FORMAT CSV, HEADER);
```

Setelah menemukan kode dan file data untuk Daftar 8-1, sambungkan ke database analisis Anda di pgAdmin dan jalankan. Ingatlah untuk mengubah C:\YourDirectory\ ke jalur tempat Anda menyimpan file CSV.

Inilah yang dilakukannya: pertama, kode membuat tabel melalui CREATE TABLE. Kami menetapkan batasan kunci utama ke kolom bernama fscskey , kode unik yang menurut kamus data ditetapkan untuk setiap perpustakaan. Karena unik, hadir di setiap baris, dan tidak mungkin berubah, ia dapat berfungsi sebagai kunci utama alami.

Definisi untuk setiap kolom mencakup tipe data yang sesuai dan batasan NOT NULL di mana kolom tidak memiliki nilai yang hilang. Jika Anda perhatikan baik-baik di kamus data, Anda akan melihat bahwa saya mengubah kolom bernama database di file CSV menjadi database di tabel. Alasannya adalah bahwa database adalah kata kunci yang dicadangkan SQL, dan tidak bijaksana untuk menggunakan kata kunci sebagai pengidentifikasi karena dapat menyebabkan konsekuensi yang tidak diinginkan dalam kueri atau fungsi lainnya.

Kolom startdat dan enddat berisi tanggal, tetapi kami telah menyetel tipe datanya ke varchar(10) dalam kode karena dalam file CSV kolom tersebut menyertakan nilai bukan tanggal, dan impor kami akan gagal jika kami mencoba menggunakan data tanggal Tipe. Di Bab 9, Anda akan belajar cara membersihkan kasus seperti ini. Untuk saat ini, kolom-kolom itu baik-baik saja.

Setelah membuat tabel, kami menambahkan indeks ke kolom yang akan kami gunakan untuk kueri. Ini memberikan hasil yang lebih cepat ketika kami mencari kolom untuk perpustakaan tertentu. Pernyataan COPY mengimpor data dari file CSV bernama pls_fy2014_pupld14a.csv menggunakan jalur file yang Anda berikan.

Membuat Tabel Data Perpustakaan 2009

Membuat tabel untuk data perpustakaan tahun 2009 mengikuti langkah yang sama, seperti yang ditunjukkan pada listing dibawah ini. Sebagian besar survei yang sedang berlangsung akan memiliki beberapa perubahan dari tahun ke tahun karena pembuat survei memikirkan pertanyaan baru atau memodifikasi yang sudah ada, sehingga kolom yang disertakan akan sedikit berbeda dalam tabel ini. Itulah salah satu alasan penyedia data membuat tabel baru alih-alih menambahkan baris ke tabel kumulatif. Misalnya, file 2014 memiliki kolom wifisess, yang mencantumkan jumlah tahunan sesi Wi-Fi yang disediakan perpustakaan, tetapi kolom ini tidak ada dalam data 2009. Kamus data untuk tahun survei ini ada di https://www.imls.gov/sites/default/files/fy2009_pls_data_file_documentation.pdf.

Setelah Anda membuat tabel ini, impor file CSV pls_fy2009_pupld09a. File ini juga tersedia untuk diunduh bersama dengan semua sumber buku di https://www.nostarch.com/practicalSQL/. Ketika Anda telah menyimpan file dan menambahkan jalur file yang benar ke pernyataan COPY, jalankan kode dibawah ini:

```
CREATE TABLE pls_fy2009_pupld09a (
      stbr varchar(2) NOT NULL,
      fscskey varchar(6) CONSTRAINT fxckey2009_key PRIMARY KEY,
      libid varchar(20) NOT NULL,
      address varchar(35) NOT NULL,
      city varchar(20) NOT NULL,
      zip varchar(5) NOT NULL,
      zip4 varchar(4) NOT NULL,
      cnty varchar(20) NOT NULL,
```

```
        --snip--
        fipsst varchar(2) NOT NULL,
        fipsco varchar(3) NOT NULL,
    );

    CREATE INDEX libname2009_idx ON pls_fy2009_pupld09a (libname);
    CREATE INDEX stabr2009_idx ON pls_fy2009_pupld09a (stabr);
    CREATE INDEX city2009_idx ON pls_fy2009_pupld09a (city);
    CREATE INDEX visits2009_idx ON pls_fy2009_pupld09a (visits);

    COPY pls_fy2009_pupld09a
    FROM 'C:\YourDirectory\pls_fy2009_pup109a.csv'
    WITH (FORMAT CSV, HEADER);
```

Kami menggunakan fscskey sebagai kunci utama lagi , dan kami membuat indeks pada libname dan kolom lainnya. Sekarang, mari kita menambang dua tabel data perpustakaan dari 2014 dan 2009 untuk menemukan history mereka.

Menjelajahi Data Perpustakaan Menggunakan Fungsi Agregat

Fungsi agregat menggabungkan nilai dari beberapa baris dan mengembalikan satu hasil berdasarkan operasi pada nilai tersebut. Misalnya, Anda dapat mengembalikan nilai rata-rata dengan fungsi avg(), seperti yang Anda pelajari di Bab 5. Itu hanya salah satu dari banyak fungsi agregat di SQL. Beberapa adalah bagian dari standar SQL, dan lainnya khusus untuk PostgreSQL dan manajer database lainnya. Sebagian besar fungsi agregat yang digunakan dalam bab ini adalah bagian dari SQL standar (daftar lengkap agregat PostgreSQL ada di https://www.postgresql.org/docs/current/static/functions-aggregate.html).

Di bagian ini, kita akan membahas data pustaka menggunakan agregat pada satu dan beberapa kolom, lalu menjelajahi bagaimana Anda dapat memperluas penggunaannya dengan mengelompokkan hasil yang mereka kembalikan dengan nilai dari kolom tambahan.

Menghitung Baris dan Nilai Menggunakan count()

Setelah mengimpor kumpulan data, langkah pertama yang masuk akal adalah memastikan tabel memiliki jumlah baris yang diharapkan. Misalnya, dokumentasi IMLS untuk data 2014 mengatakan file yang kami impor memiliki 9.305 baris, dan file 2009 memiliki 9.299 baris. Saat kita menghitung jumlah baris dalam tabel tersebut, hasilnya harus sesuai dengan jumlah tersebut.

Fungsi agregat count(), yang merupakan bagian dari standar ANSI SQL, memudahkan untuk memeriksa jumlah baris dan melakukan tugas penghitungan lainnya. Jika kita memberikan asterisk sebagai input, seperti count(*), asterisk bertindak sebagai wildcard, sehingga fungsi mengembalikan jumlah baris tabel terlepas dari apakah mereka menyertakan nilai NULL. Kami melakukan ini di kedua pernyataan di kode bawah ini:

```
SELECT count(*)
    FROM pls_fy2014_pupld14a;

    SELECT count(*)
    FROM pls_fy2009_pupld09a;
```

Jalankan koding tersebut untuk melihat jumlah baris tabel. Untuk pls_fy2014_pupld14a, hasilnya harus:

```
count
-----
9305
```

Dan untuk pls_fy2009_pupld09a, hasilnya harus:

```
count
-----
9299
```

Kedua hasil cocok dengan jumlah baris yang kami harapkan.

Catatan: Anda juga dapat memeriksa jumlah baris menggunakan antarmuka pgAdmin, tetapi ini kikuk. Mengklik kanan nama tabel di browser objek pgAdmin dan memilih Lihat/Edit Data Semua Baris menjalankan kueri SQL untuk semua baris. Kemudian, pesan pop-up di panel hasil menunjukkan jumlah baris, tetapi menghilang setelah beberapa detik.

Membandingkan jumlah baris tabel dengan apa yang dikatakan dokumentasi adalah penting karena akan mengingatkan kita akan masalah seperti baris yang hilang atau kasus di mana kita mungkin telah mengimpor file yang salah.

Menghitung Nilai yang Ada dalam Kolom

Untuk mengembalikan jumlah baris dalam kolom tertentu yang berisi nilai, kami menyediakan nama kolom sebagai input ke fungsi count() daripada tanda bintang. Misalnya, jika Anda memindai pernyataan CREATE TABLE untuk kedua tabel perpustakaan dengan cermat, Anda akan melihat bahwa kami menghilangkan batasan NOT NULL untuk kolom gaji ditambah beberapa lainnya. Alasannya adalah tidak semua agen perpustakaan melaporkan gaji, dan beberapa baris memiliki nilai NULL.

Untuk menghitung jumlah baris di kolom gaji dari 2014 yang memiliki nilai, jalankan fungsi count():

```
SELECT count(salaries)
FROM pls_fy2014_pupld014a;
```

Hasilnya menunjukkan 5.983 baris memiliki nilai gaji:

```
count
-----
5983
```

Jumlah ini jauh lebih rendah dari jumlah baris yang ada pada tabel. Dalam data tahun 2014, sedikit kurang dari dua pertiga agen melaporkan gaji, dan Anda ingin mencatat fakta itu ketika melaporkan hasil perhitungan yang dilakukan pada kolom tersebut. Pemeriksaan ini penting karena sejauh mana nilai yang ada dalam kolom dapat memengaruhi keputusan Anda apakah akan melanjutkan analisis sama sekali. Memeriksa dengan ahli tentang topik dan menggali lebih dalam data biasanya merupakan ide yang baik, dan saya merekomendasikan mencari nasihat ahli sebagai bagian dari metodologi analisis yang lebih luas.

Menghitung Nilai Berbeda dalam Kolom

Dalam Bab 2, saya membahas kata kunci DISTINCT, yang merupakan bagian dari standar SQL. Saat ditambahkan setelah SELECT dalam kueri, DISTINCT mengembalikan daftar nilai unik. Kita bisa menggunakannya untuk melihat nilai unik dalam satu kolom, atau kita bisa melihat kombinasi nilai unik dari beberapa kolom. Penggunaan lain dari DISTINCT adalah menambahkannya ke fungsi count() , yang menyebabkan fungsi mengembalikan hitungan nilai yang berbeda dari kolom.

Koding dibawah ini menunjukkan dua kueri. Yang pertama menghitung semua nilai di kolom libname tabel 2014. Yang kedua melakukan hal yang sama tetapi menyertakan DISTINCT di depan nama kolom. Jalankan keduanya, satu per satu.

```
SELECT count(libname)
FROM pls_fy2014_pupld14a;

SELECT count(DISTINCT libname)
FROM pls_fy2014_pupld14a;
```

Kueri pertama mengembalikan jumlah baris yang cocok dengan jumlah baris dalam tabel yang kami temukan menggunakan koding dibawah ini:

```
count
-----
9305
```

Itu bagus. Kami berharap nama lembaga perpustakaan tercantum di setiap baris. Tetapi kueri kedua mengembalikan angka yang lebih kecil:

```
count
-----
8515
```

Menggunakan DISTINCT untuk menghapus duplikat mengurangi jumlah nama perpustakaan menjadi 8.515 yang unik. Pemeriksaan saya lebih dekat terhadap data menunjukkan bahwa 530 lembaga perpustakaan berbagi nama dengan satu atau lebih lembaga lain. Sebagai salah satu contoh, sembilan lembaga perpustakaan bernama OXFORD PUBLIC LIBRARY dalam tabel, masing-masing di kota atau kota kecil bernama Oxford di negara bagian yang berbeda, termasuk Alabama, Connecticut, Kansas, dan Pennsylvania, antara lain. Kami akan menulis kueri untuk melihat kombinasi nilai yang berbeda di “Menggabungkan Data Menggunakan GROUP BY” di halaman 120.

Menemukan Nilai Maksimum dan Minimum Menggunakan max() dan min()

Mengetahui angka terbesar dan terkecil dalam kolom berguna untuk beberapa alasan. Pertama, ini membantu kita memahami cakupan nilai yang dilaporkan untuk variabel tertentu. Kedua, fungsi yang digunakan, max() dan min(), dapat mengungkapkan masalah tak terduga dengan data, seperti yang akan Anda lihat sekarang dengan data perpustakaan.

Baik max() dan min() bekerja dengan cara yang sama: Anda menggunakan pernyataan SELECT yang diikuti oleh fungsi dengan nama kolom yang disediakan. Koding dibawah ini menggunakan max() dan min() pada tabel 2014 dengan kolom kunjungan sebagai input. Kolom

kunjungan mencatat jumlah kunjungan tahunan ke badan perpustakaan dan semua cabangnya. Jalankan kodenya, lalu kami akan meninjau hasilnya.

```
SELECT max(visits), min(visits)
FROM pls_fy2014_pupld14a;
```

Kueri yang dihasilkan adalah sebagai berikut:

```
max         min
--------    ---
17729020    -3
```

Nah, itu menarik. Nilai maksimum lebih dari 17,7 juta masuk akal untuk sistem perpustakaan kota besar, tetapi -3 sebagai minimum? Di permukaan, hasil itu tampak seperti kesalahan, tetapi ternyata pembuat survei perpustakaan menggunakan konvensi yang bermasalah namun umum dalam pengumpulan data: menggunakan angka negatif atau nilai artifisial tinggi sebagai indikator.

Dalam hal ini, pembuat survei menggunakan angka negatif untuk menunjukkan kondisi berikut:

1. Nilai -1 menunjukkan "nonresponse" untuk pertanyaan itu.
2. Nilai -3 menunjukkan “tidak berlaku” dan digunakan pada saat suatu instansi perpustakaan tutup baik untuk sementara maupun permanen.

Kita harus memperhitungkan dan mengecualikan nilai negatif saat menjelajahi data, karena menjumlahkan kolom dan menyertakan nilai negatif akan menghasilkan total yang salah. Kita dapat melakukan ini dengan menggunakan klausa WHERE untuk memfilternya. Ada baiknya kami menemukan masalah ini sekarang daripada nanti setelah menghabiskan banyak waktu untuk analisis yang lebih dalam!

Catatan : Alternatif yang lebih baik untuk skenario nilai negatif ini adalah menggunakan NULL dalam baris di kolom kunjungan yang tidak memiliki data respons, lalu membuat kolom bendera_kunjungan terpisah untuk menyimpan kode yang menjelaskan alasannya. Teknik ini memisahkan nilai angka dari informasi tentangnya.

Menggabungkan Data Menggunakan GROUP BY

Saat Anda menggunakan klausa GROUP BY dengan fungsi agregat, Anda dapat mengelompokkan hasil menurut nilai dalam satu atau beberapa kolom. Ini memungkinkan kita untuk melakukan operasi seperti sum() atau count() untuk setiap status di tabel kita atau untuk setiap jenis agensi perpustakaan.

Mari kita jelajahi bagaimana menggunakan GROUP BY dengan agregat bekerja. Dengan sendirinya, GROUP BY, yang juga merupakan bagian dari standar ANSI SQL, menghilangkan nilai duplikat dari hasil, mirip dengan DISTINCT. Kueri menunjukkan klausa GROUP BY beraksi:

```
SELECT stabr
FROM pls_fy2014_pupld14a
GROUP BY stabr
ORDER BY stabr;
```

Klausa GROUP BY mengikuti klausa FROM dan menyertakan nama kolom ke grup. Dalam hal ini, kami memilih stabr, yang berisi singkatan status, dan mengelompokkan menurut kolom yang sama. Kami kemudian menggunakan ORDER BY stabr juga sehingga hasil yang dikelompokkan dalam urutan abjad. Ini akan menghasilkan hasil dengan singkatan status unik dari tabel 2014. Berikut sebagian hasilnya:

```
stabr
-----
AK
AL
AR
AS
AZ
CA
--snip--
WV
WY
```

Perhatikan bahwa tidak ada duplikat di 56 baris yang dikembalikan. Singkatan pos dua huruf standar ini mencakup 50 negara bagian ditambah Washington, D.C., dan beberapa wilayah AS, seperti Samoa Amerika dan Kepulauan Virgin AS.

Anda tidak dibatasi untuk mengelompokkan satu kolom saja. List program dibawah ini, kami menggunakan klausa GROUP BY pada data 2014 untuk menentukan kota dan kolom stabr untuk pengelompokan:

```
SELECT city, stabr
FROM pls_fy2014_pupld14a
GROUP BY city, stabr
ORDER BY city, stabr;
```

Hasilnya diurutkan berdasarkan kota dan kemudian menurut negara bagian, dan hasilnya menunjukkan kombinasi unik dalam urutan itu:

```
city        stabr
----------  -----
ABBEVILLE   AL
ABBEVILLE   LA
ABBEVILLE   SC
ABBOTSFORD  WI
ABERDEEN    ID
ABERDEEN    SD
ABERNATHY   TX
--snip--
```

Pengelompokan ini mengembalikan 9.088 baris, 217 lebih sedikit dari total baris tabel. Hasilnya menunjukkan ada beberapa kesempatan di mana file menyertakan lebih dari satu agen perpustakaan untuk kombinasi kota dan negara bagian tertentu.

Menggabungkan GROUP BY dengan count()

Jika kita menggabungkan GROUP BY dengan fungsi agregat, seperti count(), kita dapat menarik lebih banyak informasi deskriptif dari data kita. Misalnya, kita tahu 9.305 lembaga perpustakaan ada di tabel 2014. Kita bisa mendapatkan jumlah agensi berdasarkan negara bagian dan mengurutkannya untuk melihat negara bagian mana yang paling banyak.

```
SELECT stabr, count(*)
FROM pls_fy2014_pupld14a
GROUP BY stabr
ORDER BY count(*) DESC;
```

Tidak seperti pada contoh sebelumnya, kami sekarang meminta nilai di kolom stabr dan jumlah nilai tersebut. Dalam daftar kolom untuk kueri , kami menetapkan fungsi stabr dan count() dengan tanda bintang sebagai inputnya. Seperti sebelumnya, tanda bintang menyebabkan count() menyertakan nilai NULL. Juga, ketika kita memilih kolom individu bersama dengan fungsi agregat, kita harus menyertakan kolom dalam klausa GROUP BY. Jika tidak, database akan mengembalikan kesalahan yang menyuruh kami melakukannya. Alasannya adalah Anda tidak dapat mengelompokkan nilai dengan menggabungkan dan memiliki nilai kolom yang tidak dikelompokkan dalam kueri yang sama.

Untuk mengurutkan hasil dan memiliki status dengan jumlah agensi terbesar di atas, kita dapat ORDER BY fungsi count() dalam urutan menurun menggunakan DESC.

Jalankan kode dibawah ini. Hasilnya menunjukkan New York, Illinois, dan Texas sebagai negara bagian dengan jumlah agen perpustakaan terbanyak pada tahun 2014:

```
stabr count
----- -----
NY    756
IL    625
TX    556
IA    543
PA    455
MI    389
WI    381
MA    370
--snip--
```

Ingat bahwa tabel kami mewakili lembaga perpustakaan yang melayani suatu lokalitas. Hanya karena New York, Illinois, dan Texas memiliki jumlah agen perpustakaan terbesar tidak berarti mereka memiliki jumlah gerai terbanyak di mana Anda dapat masuk dan membaca rak dengan teliti. Sebuah lembaga mungkin hanya memiliki satu perpustakaan pusat, atau mungkin tidak memiliki perpustakaan pusat tetapi 23 cabang tersebar di seluruh daerah. Untuk menghitung outlet, setiap baris dalam tabel juga memiliki nilai di kolom centlib dan branlib, yang masing-masing mencatat jumlah perpustakaan pusat dan cabang. Untuk menemukan total, kita akan menggunakan fungsi agregat sum() pada kedua kolom.

Menggunakan GROUP BY pada Beberapa Kolom dengan count()

Kami dapat mengumpulkan lebih banyak informasi dari data kami dengan menggabungkan GROUP BY dengan fungsi count() dan beberapa kolom. Misalnya, kolom stataddr di kedua tabel berisi kode yang menunjukkan apakah alamat agensi berubah dalam setahun terakhir. Nilai dalam stataddr adalah:

00 Tidak ada perubahan dari tahun lalu
07 Pindah ke lokasi baru
15 Perubahan alamat kecil

Daftar 8-10 menunjukkan kode untuk menghitung jumlah agensi di setiap negara bagian yang pindah, mengalami sedikit perubahan alamat, atau tidak ada perubahan menggunakan GROUP BY dengan stabr dan stataddr dan menambahkan count():

```
SELECT stabr, stataddr, count(*)
FROM pls_fy2014_pupld14a
GROUP BY stabr, stataddr
ORDER BY stabr ASC, count(*) DESC;
```

Bagian kunci dari kueri adalah nama kolom dan fungsi count() setelah SELECT , dan memastikan kedua kolom tercermin dalam klausa GROUP BY . Efek pengelompokan berdasarkan dua kolom adalah count() akan menampilkan jumlah kombinasi unik stabr dan stataddr.

Untuk membuat output lebih mudah dibaca, mari kita urutkan dulu kode negara bagian dalam urutan menaik dan kemudian menurut hitungan dalam urutan menurun . Berikut adalah hasilnya:

```
stabr       stataddr    count
-----       --------    -----
AK          00          70
AK          15          10
AK          07          5
AL          00          221
AL          07          3
AR          00          58
AS          00          1
AZ          00          91
--snip--
```

Beberapa baris pertama dari hasil menunjukkan bahwa kode 00 (tidak ada perubahan alamat) adalah nilai yang paling umum untuk setiap negara bagian. Kami mengharapkan itu karena kemungkinan ada lebih banyak agen perpustakaan yang belum mengubah alamat daripada yang telah. Hasilnya membantu meyakinkan kami bahwa kami menganalisis data dengan cara yang baik. Jika kode 07 (dipindahkan ke lokasi baru) adalah yang paling sering di setiap negara bagian, itu akan menimbulkan pertanyaan tentang apakah kami telah menulis kueri dengan benar atau apakah ada masalah dengan data.

Meninjau kembali sum() untuk Memeriksa Kunjungan Perpustakaan

Sejauh ini, kami telah menggabungkan pengelompokan dengan fungsi agregat, seperti count(), pada kolom dalam satu tabel untuk memberikan hasil yang dikelompokkan berdasarkan nilai kolom. Sekarang mari kita perluas teknik untuk menyertakan pengelompokan dan agregasi di seluruh tabel yang digabungkan menggunakan data pustaka 2014 dan 2009. Tujuan kami adalah mengidentifikasi tren kunjungan perpustakaan selama periode lima tahun tersebut. Untuk melakukan ini, kita perlu menghitung total menggunakan fungsi agregat sum().

Sebelum kita menggali kueri ini, mari kita bahas masalah penggunaan nilai -3 dan -1 untuk menunjukkan "tidak berlaku" dan "nonresponse". Untuk mencegah angka negatif tanpa arti sebagai kuantitas memengaruhi analisis, kami akan memfilternya menggunakan klausa WHERE untuk membatasi kueri ke baris dengan nilai kunjungan nol atau lebih besar.

Mari kita mulai dengan menghitung jumlah kunjungan tahunan ke perpustakaan dari tabel individu tahun 2014 dan 2009. Jalankan setiap pernyataan SELECT secara terpisah:

```
SELECT sum(visits) AS visits_2004
FROM pls_fy2014_pupld14a
WHERE visits >= 0;

SELECT sum(visits) AS visits_2009
FROM pls_fy2009_pupld09a
WHERE visits >= 0;
```

Untuk tahun 2014, total kunjungan mencapai sekitar 1,4 miliar.

```
visits_2014
-----------
1425930900
```

Untuk tahun 2009, jumlah kunjungan mencapai sekitar 1,6 miliar. Kami menuju sesuatu di sini, tapi itu mungkin bukan kabar baik. Tren tampaknya mengarah ke bawah dengan kunjungan turun sekitar 10 persen dari 2009 ke 2014.

```
visits_2009
-----------
1591799201
```

Kueri ini menjumlahkan keseluruhan kunjungan. Namun dari jumlah baris yang kita jalankan sebelumnya di bab ini, kita tahu bahwa setiap tabel berisi jumlah lembaga perpustakaan yang berbeda: 9.305 pada tahun 2014 dan 9.299 pada tahun 2009 karena pembukaan, penutupan, atau penggabungan lembaga. Jadi, mari kita tentukan bagaimana jumlah kunjungan akan berbeda jika kita membatasi analisis pada lembaga perpustakaan yang ada di kedua tabel. Kita dapat melakukannya dengan menggabungkan tabel, seperti yang ditunjukkan ini:

```
SELECT sum(pls14.visits) AS visits_2014,
      sum(pls09.visits) AS visits_2009
FROM pls_fy2014_pupld14a pls14 JOIN pls_fy2009_pupld09a pls09

ON pls14.fscskey = pls09.fscskey
WHERE pls14.visits >= AND pls09.visits >= 0;
```

Kueri ini menyatukan beberapa konsep yang telah kita bahas di bab sebelumnya, termasuk gabungan tabel. Di bagian atas, kami menggunakan fungsi agregat sum() untuk menjumlahkan kolom kunjungan dari tabel 2014 dan 2009. Saat kita menggabungkan tabel pada kunci utama tabel, kita mendeklarasikan alias tabel seperti yang kita pelajari di Bab 6. Di sini, kita mendeklarasikan pls14 sebagai alias untuk tabel 2014 dan pls09 sebagai alias untuk tabel 2009 untuk menghindari untuk menulis nama tabel lengkap yang lebih panjang di seluruh kueri.

Perhatikan bahwa kami menggunakan JOIN standar, juga dikenal sebagai INNER JOIN. Itu berarti hasil kueri hanya akan menyertakan baris di mana nilai kunci utama dari kedua tabel (kolom fscskey) cocok.

Menggunakan klausa WHERE , kami mengembalikan baris di mana kedua tabel memiliki nilai nol atau lebih besar di kolom kunjungan. Seperti yang kami lakukan di Daftar 8-11, kami

menetapkan bahwa hasilnya harus menyertakan hanya baris yang kunjungannya lebih besar dari atau sama dengan 0 di kedua tabel. Ini akan mencegah nilai negatif artifisial mempengaruhi jumlah.

Jalankan kueri. Hasilnya akan terlihat seperti ini:

```
visits_2014       visits_2009
-----------       -----------
1417299243        15854550205
```

Hasilnya mirip dengan apa yang kami temukan dengan menanyakan tabel secara terpisah, meskipun totalnya enam hingga delapan juta lebih kecil. Alasannya adalah bahwa kueri hanya merujuk agensi dengan fscskey di kedua tabel. Namun, tren penurunan tetap ada. Kita perlu menggali lebih dalam untuk mendapatkan cerita lengkapnya.

Catatan ; Meskipun kami bergabung dengan tabel di fscskey, sangat mungkin bahwa beberapa agen perpustakaan yang muncul di kedua tabel bergabung atau terpisah antara 2009 dan 2014. Panggilan ke IMLS menanyakan tentang peringatan untuk bekerja dengan data ini adalah ide yang bagus.

Mengelompokkan Jumlah Kunjungan berdasarkan Negara

Sekarang kita tahu bahwa kunjungan perpustakaan menurun di Amerika Serikat secara keseluruhan antara tahun 2009 dan 2014, Anda mungkin bertanya pada diri sendiri, “Apakah setiap bagian negara mengalami penurunan, atau apakah tingkat tren bervariasi menurut wilayah?” Kami dapat menjawab pertanyaan ini dengan memodifikasi kueri sebelumnya untuk dikelompokkan berdasarkan kode negara bagian. Mari kita juga menggunakan perhitungan persen-perubahan untuk membandingkan tren berdasarkan negara bagian. Daftar 8-13 berisi kode lengkap:

```
SELECT pls14.stabr,
      sum(pls14.visits) AS visits_2014,
      sum(pls09.visits) AS visits_2009,
      round( (CAST(sum(pls14.visits) AS decimal(10,1) - sum(pls09.visits)) /
            sum(pls09.visits) * 100, 2) AS pct_change

FROM pls_fy2014_pupld14a pls14 JOIN pls_fy2009_pupld09a pls09
ON pls14.fsckey = pls09.fscskey
WHERE pls14.visits >= 0 AND pls09.visits >=0
GROUP BY pls14.stabr
ORDER BY pct_change DESC;
```

Kami mengikuti kata kunci SELECT dengan kolom stabr dari tabel 2014; kolom yang sama muncul di klausa GROUP BY. Tidak masalah kolom stabr tabel mana yang kami gunakan karena kami hanya menanyakan agensi yang muncul di kedua tabel. Setelah SELECT, kami juga menyertakan penghitungan persentase perubahan yang sekarang sudah Anda pelajari di Bab 5, yang mendapatkan alias pct_change agar mudah dibaca. Kami mengakhiri kueri dengan klausa ORDER BY , menggunakan alias kolom pct_change.

Saat Anda menjalankan kueri, hasil teratas menunjukkan 10 negara bagian atau teritori dengan peningkatan kunjungan dari 2009 hingga 2014. Hasil lainnya menunjukkan penurunan. Oklahoma, di peringkat terbawah, mengalami penurunan 35 persen.

```
stabr       visits_2014       visits_2009       pct_change
-----       -----------       -----------       ----------
GU          103593            60763             70.49
DC          4230790           2944774           43.67
LA          17242110          15591805          10.58
MT          4582604           4386504           4.47
AL          17113602          16933967          1.06
AR          10762521          10660058          0.96
KY          19256394          19113478          0.75
CO          32978245          32782247          0.60
SC          18178677          18105931          0.40
SD          3899554           3890392           0.24
MA          42011647          42237888          -0.54
AK          3486955           3525093           -1.08
ID          8730670           8847034           -1.32
NH          7508751           7675823           -2.18
WY          3666825           3756294           -2.38
--snip--
RI          5259143           6612167           -20.46
NC          33952977          43111094          -21.24
PR          193279            257032            -24.80
GA          28891017          40922598          -29.40
OK          13678542          21171452          -35.39
```

Data yang berguna ini harus mengarahkan analis data untuk menyelidiki apa yang mendorong perubahan, terutama yang terbesar. Analisis data terkadang dapat menimbulkan pertanyaan sebanyak yang dijawab, tetapi itu adalah bagian dari proses. Itu selalu bernilai panggilan telepon ke orang yang memiliki pengetahuan tentang data untuk memberikan konteks untuk hasil. Terkadang, mereka mungkin memiliki penjelasan yang sangat bagus. Di lain waktu, seorang ahli akan berkata, "Kedengarannya tidak benar." Jawaban itu mungkin mengirim Anda kembali ke penjaga data atau dokumentasi untuk mengetahui apakah Anda mengabaikan kode atau nuansa dengan data.

Memfilter Kueri Agregat Menggunakan HAVING

Kami dapat menyempurnakan analisis kami dengan memeriksa subset negara bagian dan teritori yang memiliki karakteristik serupa. Dengan perubahan persentase kunjungan, masuk akal untuk memisahkan negara bagian besar dari negara bagian kecil. Di negara bagian kecil seperti Rhode Island, satu kali penutupan perpustakaan bisa berdampak signifikan. Satu penutupan di California mungkin jarang diperhatikan dalam hitungan di seluruh negara bagian. Untuk melihat status dengan volume kunjungan yang serupa, kami dapat mengurutkan hasil menurut salah satu kolom kunjungan, tetapi akan lebih mudah untuk mendapatkan kumpulan hasil yang lebih kecil dalam kueri kami.

Untuk memfilter hasil fungsi agregat, kita perlu menggunakan klausa HAVING yang merupakan bagian dari ANSI SQL standar. Anda sudah terbiasa menggunakan WHERE untuk memfilter, tetapi fungsi agregat, seperti sum(), tidak dapat digunakan dalam klausa WHERE karena beroperasi pada tingkat baris, dan fungsi agregat bekerja di seluruh baris. Klausa HAVING

menempatkan kondisi pada grup yang dibuat dengan menggabungkan. Kode dibawah ini memodifikasi query di diatas dengan memasukkan klausa HAVING setelah GROUP BY:

```
SELECT pls14.stabr,
      sum(pls14.visits) AS visits_2014,
      sum(pls09.visits) AS visits_2009,
      round( (CAST(sum(pls14.visits) AS decimal(10,1)) - sum(pls09.visits)) /
                  sum(pls09.visits) * 100, 2) AS pct_change
FROM pls_fy2014_pipld14a pls14 JOIN pls_fy2009_pupld09a pls09
ON pls14.fsckey = pls.fscskey
WHERE pls14.visits >= 0 AND pls09.visits >= 0
GROUP BY pls14.stabr
HAVING sum(pls14.visits) > 50000000
ORDER BY pct_change DESC;
```

Dalam hal ini, kami telah menetapkan hasil kueri kami untuk hanya menyertakan baris dengan jumlah kunjungan pada tahun 2014 lebih dari 50 juta. Itu nilai arbitrer yang saya pilih untuk hanya menunjukkan negara bagian yang paling besar. Menambahkan klausa HAVING mengurangi jumlah baris dalam output menjadi hanya enam. Dalam praktiknya, Anda mungkin bereksperimen dengan berbagai nilai. Berikut adalah hasilnya:

```
stabr       visits_2014       visits_2009       pct_change
-----       -----------       -----------       ----------
TX          72876601          78838400          -7.56
CA          162787836         182181408         -10.65
OH          82495138          92402369          -10.72
NY          106453546         119810969         -11.15
IL          72598213          82438755          -11.94
FL          73165352          87730886          -16.60
```

Masing-masing dari enam negara bagian telah mengalami penurunan kunjungan, tetapi perhatikan bahwa variasi perubahan persentase tidak seluas di set lengkap negara bagian dan teritori. Bergantung pada apa yang kita pelajari dari pakar perpustakaan, melihat keadaan dengan aktivitas paling banyak sebagai sebuah kelompok mungkin membantu dalam menggambarkan tren, seperti halnya melihat pengelompokan lainnya. Pikirkan kalimat atau poin-poin Anda mungkin menulis yang akan mengatakan, "Di negara bagian terbesar di negara itu, kunjungan menurun antara 8 persen dan 17 persen antara 2009 dan 2014." Anda dapat menulis kalimat serupa tentang negara bagian berukuran sedang dan negara bagian kecil.

Jika bab ini telah mengilhami Anda untuk mengunjungi perpustakaan setempat dan melihat beberapa buku, tanyakan kepada pustakawan apakah cabang mereka mengalami peningkatan atau penurunan kunjungan selama beberapa tahun terakhir. Kemungkinannya, Anda bisa menebak jawabannya sekarang. Dalam bab ini, Anda mempelajari cara menggunakan teknik SQL standar untuk meringkas data dalam tabel dengan mengelompokkan nilai dan menggunakan beberapa fungsi agregat. Dengan bergabung dengan kumpulan data, Anda dapat mengidentifikasi beberapa tren lima tahun yang menarik.

Anda juga belajar bahwa data tidak selalu dikemas dengan sempurna. Penggunaan nilai negatif dalam kolom sebagai indikator daripada sebagai nilai numerik yang sebenarnya memaksa kami untuk menyaring baris tersebut. Sayangnya, kumpulan data menawarkan tantangan semacam itu lebih sering daripada tidak. Di bab berikutnya, Anda akan mempelajari teknik untuk membersihkan kumpulan data yang memiliki sejumlah masalah. Di bab-bab berikutnya, Anda

juga akan menemukan lebih banyak fungsi agregat untuk membantu Anda menemukan cerita dalam data Anda.

Latihan Soal

Uji keterampilan pengelompokan dan agregasi Anda dengan tantangan berikut:

1. Kami melihat bahwa kunjungan perpustakaan akhir-akhir ini menurun di banyak tempat. Namun bagaimana pola pemanfaatan teknologi di perpustakaan? Baik tabel survei perpustakaan 2014 dan 2009 berisi kolom gpterms (jumlah komputer yang terhubung ke internet yang digunakan oleh publik) dan pitusr (penggunaan komputer internet publik per tahun). Ubah kode program jumlah pengunjung untuk menghitung persentase perubahan jumlah setiap kolom dari waktu ke waktu. Hati-hati dengan nilai negatif!
2. Kedua tabel survei perpustakaan berisi kolom yang disebut obereg, Kode Biro Analisis Ekonomi dua digit yang mengklasifikasikan setiap lembaga perpustakaan menurut wilayah Amerika Serikat, seperti New England, Rocky Mountains, dan sebagainya. Sama seperti kami menghitung persentase perubahan kunjungan yang dikelompokkan menurut negara bagian, lakukan hal yang sama untuk mengelompokkan persentase perubahan kunjungan menurut wilayah AS menggunakan obereg. Lihat dokumentasi survei untuk menemukan arti dari setiap kode wilayah. Untuk tantangan bonus, buat tabel dengan kode obereg sebagai kunci utama dan nama wilayah sebagai teks, dan gabungkan ke kueri ringkasan untuk dikelompokkan menurut nama wilayah, bukan kode.
3. Mengingat kembali jenis gabungan yang Anda pelajari di Bab 6, jenis gabungan mana yang akan menunjukkan kepada Anda semua baris di kedua tabel, termasuk yang tidak cocok? Tulis kueri seperti itu dan tambahkan filter IS NULL dalam klausa WHERE untuk menunjukkan agensi yang tidak termasuk dalam satu atau tabel lainnya.

BAB IX
MEMERIKSA DAN MENGUBAH DATA

Jika Anda meminta saya untuk bersulang untuk kelas analis data yang baru dicetak, saya mungkin akan mengangkat gelas saya dan berkata, "Semoga data Anda selalu bebas dari kesalahan dan semoga selalu tiba dengan terstruktur dengan sempurna!" Hidup akan ideal jika sentimen ini layak. Pada kenyataannya, terkadang Anda akan menerima data dalam keadaan yang sangat menyedihkan sehingga sulit untuk dianalisis tanpa memodifikasinya dengan cara tertentu. Ini disebut data kotor, yang merupakan label umum untuk data dengan kesalahan, nilai yang hilang, atau organisasi yang buruk yang membuat kueri standar tidak efektif. Saat data dikonversi dari satu tipe file ke tipe file lainnya atau saat kolom menerima tipe data yang salah, informasi bisa hilang. Kesalahan ketik dan inkonsistensi ejaan juga dapat mengakibatkan data kotor. Apa pun penyebabnya, data kotor adalah kutukan bagi analis data.

Dalam bab ini, Anda akan menggunakan SQL untuk membersihkan data kotor serta melakukan tugas pemeliharaan berguna lainnya. Anda akan belajar cara memeriksa data untuk menilai kualitasnya dan cara memodifikasi data dan tabel untuk mempermudah analisis. Tetapi teknik yang akan Anda pelajari akan berguna untuk lebih dari sekadar membersihkan data.

Kemampuan untuk membuat perubahan pada data dan tabel memberi Anda opsi untuk memperbarui atau menambahkan informasi baru ke database Anda saat tersedia, meningkatkan database Anda dari koleksi statis menjadi catatan hidup. Mari kita mulai dengan mengimpor data kita.

Impor Data

Untuk contoh ini, kami akan menggunakan direktori U.S. produsen daging, unggas, dan telur. Food Safety and Inspection Service (FSIS), sebuah lembaga di AS Departemen Pertanian, mengkompilasi dan memperbarui database ini setiap bulan. FSIS bertanggung jawab untuk memeriksa hewan dan makanan di lebih dari 6.000 pabrik pengolahan daging, rumah pemotongan hewan, peternakan, dan sejenisnya. Jika inspektur menemukan masalah, seperti kontaminasi bakteri atau makanan yang salah label, badan tersebut dapat mengeluarkan penarikan. Siapa pun yang tertarik dengan bisnis pertanian, rantai pasokan makanan, atau wabah penyakit bawaan makanan akan merasakan manfaat direktori ini. Baca lebih lanjut tentang agensi tersebut di situsnya di https://www.fsis.usda.gov/.

File yang akan kami gunakan berasal dari halaman direktori di https://www.data.gov/, situs web yang dijalankan oleh A.S. pemerintah federal yang membuat katalog ribuan kumpulan data dari berbagai lembaga federal (https://catalog.data.gov/dataset/meat-poultry-and-egg-inspection-directory-by-establishment-name/). Kami akan memeriksa data asli karena tersedia untuk diunduh, dengan pengecualian kolom Kode Pos (saya akan menjelaskan alasannya nanti). Anda akan menemukan data dalam file MPI_Directory_by_Establishment_Name.csv bersama dengan sumber lain untuk buku ini di https://www.nostarch.com/practicalSQL/.

Untuk mengimpor file ke PostgreSQL, gunakan kode di bawah ini untuk membuat tabel bernama meat_poultry_egg_inspect dan gunakan COPY untuk menambahkan file CSV ke tabel. Seperti pada contoh sebelumnya, gunakan pgAdmin untuk menghubungkan ke database analisis Anda, lalu buka Alat Kueri untuk menjalankan kode. Ingatlah untuk mengubah jalur dalam pernyataan COPY untuk mencerminkan lokasi file CSV Anda.

```
CREATE TABLE meat_poultry_egg_inspect (
      est_number varchar(50) COINSTRAINT est_number_key PRIMARY KEY,
      company varchar(100),
      street varchar(100),
      city varchar(30),
      st varchar(2),
      zip varchar(5),
      phone varchar(14),
      grant_date date,
      activities text,
      dbas text,
);

COPY meat_poultry_egg_inspect
FROM 'C:\YourDirectory\MPI_Directory_by_Establishment_Name.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');

CREATE INDEX company_idx ON meat_poultry_egg_inspect (company);
```

Tabel meat_poultry_egg_inspect memiliki 10 kolom. Kami menambahkan batasan kunci primer alami ke kolom est_number , yang berisi nilai unik untuk setiap baris yang mengidentifikasi pembentukan. Sebagian besar kolom yang tersisa berhubungan dengan nama dan lokasi perusahaan. Anda akan menggunakan kolom aktivitas , yang menjelaskan aktivitas di perusahaan, dalam latihan "Coba Sendiri" di akhir bab ini. Kami mengatur kolom aktivitas dan dbas ke teks, tipe data yang di PostgreSQL berikan kepada kami hingga 1GB karakter, karena beberapa string dalam kolom memiliki panjang ribuan karakter. Kami mengimpor file CSV dan kemudian membuat indeks pada kolom perusahaan untuk mempercepat pencarian perusahaan tertentu.

Untuk latihan, mari gunakan fungsi agregat count () yang diperkenalkan di Bab 8 untuk memeriksa berapa banyak baris dalam tabel meat_poultry_egg_inspect:

```
SELECT count(*) FROM meat_poultry_egg_inspect;
```

Hasilnya harus menunjukkan 6.287 baris. Sekarang mari kita cari tahu apa isi data dan tentukan apakah kita dapat memperoleh informasi yang berguna darinya sebagaimana adanya, atau jika kita perlu memodifikasinya dengan cara tertentu.

Meneliti Kumpulan Data

Meneliti data adalah bagian favorit saya dari analisis. Kami meneliti kumpulan data untuk menemukan detailnya: apa yang dipegangnya, pertanyaan apa yang dapat dijawabnya, dan seberapa cocoknya untuk tujuan kami, dengan cara yang sama wawancara kerja mengungkapkan apakah seorang kandidat memiliki keterampilan yang dibutuhkan untuk posisi tersebut.

Kueri agregat yang Anda pelajari di Bab 8 adalah alat wawancara yang berguna karena sering kali mengungkapkan keterbatasan kumpulan data atau mengajukan pertanyaan yang mungkin ingin Anda tanyakan sebelum menarik kesimpulan dalam analisis Anda dan mengasumsikan validitas temuan Anda.

Misalnya, baris tabel meat_poultry_egg_inspect menggambarkan produsen makanan. Sepintas, kita mungkin berasumsi bahwa setiap perusahaan di setiap baris beroperasi di alamat yang berbeda. Tapi tidak pernah aman untuk berasumsi dalam analisis data, jadi mari kita periksa menggunakan kode ini:

```
SELECT company,
      street,
      city,
      st,
      count(*) AS address_count
FROM meat_poultry_egg_inspect
GROUP BY company, street, city, st
HAVING count(8) > 1
ORDER BY company, street, city, st;
```

Di sini, kami mengelompokkan perusahaan berdasarkan kombinasi unik kolom perusahaan, jalan, kota, dan st. Kemudian kami menggunakan count (*), yang mengembalikan jumlah baris untuk setiap kombinasi kolom tersebut dan memberinya alias address_count. Menggunakan klausa HAVING yang diperkenalkan di Bab 8, kami memfilter hasil untuk menunjukkan hanya kasus di mana lebih dari satu baris memiliki kombinasi nilai yang sama. Ini akan mengembalikan semua alamat duplikat untuk sebuah perusahaan.

Kueri mengembalikan 23 baris, yang berarti ada hampir dua lusin kasus di mana perusahaan yang sama terdaftar beberapa kali di alamat yang sama:

```
company                 street                   city        st    address_count
----------------        --------------------     ----------  --    -------------
Acre Station Meat Farm  17076 Hwy 32 N           Pinetown    NC                2
Beltex Corporation      3801 North Grove Street  Fort Worth  TX                2
Cloverleaf Cold Storage 111 Imperial Drive       Sanford     NC                3
```

Ini belum tentu menjadi masalah. Mungkin ada alasan yang sah bagi perusahaan untuk muncul beberapa kali di alamat yang sama. Misalnya, dua jenis pabrik pengolahan bisa ada dengan nama yang sama. Di sisi lain, kami mungkin menemukan kesalahan entri data. Selain itu adalah praktik yang baik untuk menghilangkan kekhawatiran tentang validitas kumpulan data sebelum mengandalkannya, dan hasilnya akan mendorong kami untuk menyelidiki kasus individu sebelum kami menarik kesimpulan. Namun, kumpulan data ini memiliki masalah lain yang perlu kita perhatikan sebelum kita dapat memperoleh informasi yang berarti darinya. Mari kita bekerja melalui beberapa contoh.

Memeriksa Nilai yang Hilang

Mari kita mulai memeriksa nilai-nilai yang hilang dengan mengajukan pertanyaan dasar: berapa banyak perusahaan pengolahan daging, unggas, dan telur di setiap negara bagian? Mencari tahu apakah kami memiliki nilai dari semua status dan apakah ada baris yang kehilangan kode status akan berfungsi sebagai pemeriksaan berguna lainnya pada data. Kami

akan menggunakan jumlah fungsi agregat () bersama dengan GROUP BY untuk menentukan ini, seperti yang ditunjukkan pada kode dibawah ini:

```
SELECT st,
      count(*) AS st_count
FROM meat_poultry_egg_inspect
GROUP BY st
ORGER BY st;
```

Kueri adalah hitungan sederhana yang mirip dengan contoh di Bab 8. Saat Anda menjalankan kueri, kueri menghitung berapa kali setiap kode pos negara bagian (st) muncul di tabel. Hasil Anda harus mencakup 57 baris, dikelompokkan berdasarkan kode pos negara bagian di kolom st. Mengapa lebih dari 50 AS negara bagian? Karena datanya mencakup Puerto Rico dan AS lainnya yang tidak berhubungan. wilayah, seperti Guam dan Samoa Amerika. Alaska (AK) berada di urutan teratas hasil dengan jumlah 17 pendirian:

```
st    st_count
--    --------
AK          17
AL          93
AR          87
AS          1
--snip--
WA          139
WI          184
WV          23
WY          1
            3
```

Namun, baris di bagian bawah daftar memiliki hitungan 3 dan nilai NULL di kolom st_count. Untuk mengetahui apa artinya ini, mari kita kueri baris di mana kolom st memiliki nilai NULL.

Catatan Tergantung pada implementasi database, nilai NULL akan muncul pertama atau terakhir dalam kolom yang diurutkan. Di PostgreSQL, mereka muncul terakhir secara default. Standar ANSI SQL tidak menentukan satu atau yang lain, tetapi memungkinkan Anda menambahkan NULLS FIRST atau NULLS LAST ke klausa ORDER BY untuk menentukan preferensi. Misalnya, untuk membuat nilai NULL muncul terlebih dahulu di kueri sebelumnya, klausa akan membaca ORDER BY st NULLS FIRST.

Kami menggunakan teknik yang tercakup dalam “Menggunakan NULL untuk Menemukan Baris dengan Nilai yang Hilang” di halaman 83, menambahkan klausa WHERE dengan kolom pertama dan kata kunci IS NULL untuk menemukan baris mana yang tidak memiliki kode status:

```
SELECT est_number,
      company,
      city,
      st,
      zip,
      FROM meat_poultry_egg_inspect
      WHERE st IS NULL;
```

Kueri ini mengembalikan tiga baris yang tidak memiliki nilai di kolom pertama:

```
est_number        company                          city        st    zip
----------------- -------------------------------  ------      --    -----
V18677A           Atlas Inspection, Inc.           Blaine            55449
M45319+P45319     Hall-Namie Packing Company, Inc                    36671
M263A+P263A+V263A Jones Dairy Farm                                   53538
```

Jika kita menginginkan penghitungan yang akurat dari pendirian per negara bagian, nilai-nilai yang hilang ini akan menyebabkan hasil yang salah. Untuk menemukan sumber data kotor ini, ada baiknya melakukan pemeriksaan visual cepat dari file asli yang diunduh dari https://www.data.gov/. Kecuali Anda bekerja dengan file dalam kisaran gigabyte, Anda biasanya dapat membuka file CSV di editor teks dan mencari baris. Jika Anda bekerja dengan file yang lebih besar, Anda mungkin dapat memeriksa data sumber menggunakan utilitas seperti grep (di Linux dan macOS) atau findstr (di Windows). Dalam hal ini, pemeriksaan visual mengonfirmasi bahwa, memang, tidak ada status yang tercantum dalam baris tersebut dalam file CSV, sehingga kesalahan bersifat organik pada data, bukan salah satu yang diperkenalkan selama impor.

Dalam wawancara kami tentang data sejauh ini, kami menemukan bahwa kami perlu menambahkan nilai yang hilang ke kolom pertama untuk membersihkan tabel ini. Mari kita lihat masalah lain apa yang ada di kumpulan data kami dan buat daftar tugas pembersihan.

Memeriksa Nilai Data yang Tidak Konsisten

Data yang tidak konsisten adalah faktor lain yang dapat menghambat analisis kami. Kami dapat memeriksa data yang dimasukkan secara tidak konsisten dalam kolom dengan menggunakan GROUP BY dengan count (). Saat Anda memindai nilai yang tidak terduplikasi dalam hasil, Anda mungkin dapat menemukan variasi dalam ejaan nama atau atribut lainnya.

Misalnya, banyak dari 6.200 perusahaan dalam tabel kami adalah beberapa lokasi yang dimiliki oleh beberapa perusahaan makanan multinasional, seperti Cargill atau Tyson Foods. Untuk mengetahui berapa banyak lokasi yang dimiliki setiap perusahaan, kami akan mencoba menghitung nilai di kolom perusahaan. Mari kita lihat apa yang terjadi ketika kita melakukannya, menggunakan kueri:

```
SELECT company,
      count(*) AS company_count
FROM meat_poultry_egg_inspect
GROUP BY company
ORDER BY company ASC;
```

Menggulir hasil mengungkapkan sejumlah kasus di mana nama perusahaan dieja dengan beberapa cara berbeda. Misalnya, perhatikan entri untuk merek Armor-Eckrich:

```
company                      company_count
---------------------------  -------------
--snip--
Armour – Eckrich Meats, LLC              1
Armour-Eckrich Meats LLC                 3
Arnour-Eckrich Meats, Inc.               1
Armour-Eckrich Meats, LLC                2
--snipp--
```

Setidaknya empat ejaan berbeda ditampilkan untuk tujuh perusahaan yang kemungkinan dimiliki oleh perusahaan yang sama. Jika nanti kita melakukan agregasi berdasarkan perusahaan, akan membantu untuk membakukan nama sehingga semua item yang dihitung atau dijumlahkan dikelompokkan dengan benar. Mari tambahkan itu ke daftar item untuk diperbaiki.

Memeriksa Nilai yang Salah Bentuk Menggunakan length()

Sebaiknya periksa nilai tak terduga di kolom yang harus diformat secara konsisten. Misalnya, setiap entri dalam kolom zip di tabel meat_poultry_egg_inspect harus diformat dengan gaya A.S. Kode ZIP dengan lima digit. Namun, bukan itu yang ada di kumpulan data kami. Semata-mata untuk tujuan contoh ini, saya mereplikasi kesalahan yang telah saya lakukan sebelumnya. Ketika saya mengonversi file Excel asli ke file CSV, saya menyimpan Kode ZIP dalam format angka "Umum" di spreadsheet alih-alih sebagai nilai teks. Dengan melakukannya, Kode ZIP apa pun yang dimulai dengan nol, seperti 07502 untuk Paterson, NJ, kehilangan nol di depannya karena bilangan bulat tidak dapat dimulai dengan nol. Akibatnya, 07502 muncul di tabel sebagai 7502. Anda dapat membuat kesalahan ini dalam berbagai cara, termasuk dengan menyalin dan menempelkan data ke kolom Excel yang disetel ke "Umum." Setelah dibakar beberapa kali, saya belajar untuk lebih berhati-hati dengan angka yang harus diformat sebagai teks.

Kesalahan saya yang disengaja muncul ketika kami menjalankan listing dibawah ini. Contoh memperkenalkan length(), fungsi string yang menghitung jumlah karakter dalam string. Kami menggabungkan length() dengan count() dan GROUP BY untuk menentukan berapa banyak baris yang memiliki lima karakter di bidang zip dan berapa banyak yang memiliki nilai selain lima. Untuk memudahkan pemindaian hasil, kami menggunakan panjang () dalam klausa ORDER BY.

```
SELECT length(zip),
      count(*) AS length_count
FROM meat_poultry_egg_inspect
GROUP BY length(zip)
ORDER BY length(zip) ASC;
```

Hasilnya mengkonfirmasi kesalahan pemformatan. Seperti yang Anda lihat, 496 Kode ZIP panjangnya empat karakter, dan 86 panjangnya tiga karakter, yang berarti angka-angka ini awalnya memiliki dua nol di depan yang salah dihilangkan oleh konversi saya:

```
length      length_count
------      ------------
3           86
4           496
5           5705
```

Dengan menggunakan klausa WHERE, kita dapat memeriksa detail hasil untuk melihat status mana yang sesuai dengan Kode Pos yang disingkat ini, seperti yang ditunjukkan List program dibawah ini:

```
SELECT st,
      Count(*) AS st_count
```

```
FROM meat_puoltyr_egg_inspect
WHERE length(zip) < 5

GROUP BY st
ORDER BY st ASC;
```

Fungsi length() di dalam klausa WHERE mengembalikan hitungan baris di mana Kode ZIP kurang dari lima karakter untuk setiap kode negara bagian. Hasilnya adalah apa yang kita harapkan. Negara bagian sebagian besar berada di wilayah Timur Laut Amerika Serikat di mana Kode ZIP sering dimulai dengan nol:

```
st    st_count
--    --------
CT          55
MA         101
ME          24
NH          18
NJ         244
PR          84
RI          27
VI           2
VT          27
```

Jelas, kami tidak ingin kesalahan ini berlanjut, jadi kami akan menambahkannya ke daftar item kami untuk diperbaiki. Sejauh ini, kami perlu memperbaiki masalah berikut dalam kumpulan data kami:

- Nilai yang hilang untuk tiga baris di kolom pertama
- Ejaan yang tidak konsisten untuk setidaknya satu nama perusahaan
- Kode ZIP tidak akurat karena konversi file

Selanjutnya, kita akan melihat cara menggunakan SQL untuk memperbaiki masalah ini dengan memodifikasi data Anda.

Memodifikasi Tabel, Kolom, dan Data

Hampir tidak ada apa pun dalam database, dari tabel hingga kolom dan tipe data serta nilai yang dikandungnya, diatur secara konkret setelah dibuat. Saat kebutuhan Anda berubah, Anda bisa menambahkan kolom ke tabel, mengubah tipe data pada kolom yang ada, dan mengedit nilai. Untungnya, Anda dapat menggunakan SQL untuk memodifikasi, menghapus, atau menambah data dan struktur yang ada. Mengingat masalah yang kami temukan di tabel meat_poultry_egg_inspect, kemampuan untuk memodifikasi database kami akan sangat berguna.

Untuk membuat perubahan pada database kami, kami akan menggunakan dua perintah SQL: perintah pertama, ALTER TABLE, adalah bagian dari standar ANSI SQL dan menyediakan opsi untuk ADD COLUMN, ALTER COLUMN, dan DROP COLUMN, antara lain. Biasanya, PostgreSQL dan database lainnya menyertakan ekstensi khusus implementasi ke ALTER TABLE yang menyediakan serangkaian opsi untuk mengelola objek database (lihat https://www.postgresql.org/docs/current/static/sql-altertable.html) .

Untuk latihan kami, kami akan tetap menggunakan opsi inti. Perintah kedua, UPDATE, juga termasuk dalam standar SQL, memungkinkan Anda untuk mengubah nilai dalam kolom tabel. Anda dapat memberikan kriteria menggunakan klausa WHERE untuk memilih baris mana yang akan diperbarui.

Mari kita jelajahi sintaks dan opsi dasar untuk kedua perintah, lalu gunakan untuk memperbaiki masalah di kumpulan data kita.

Membuang DATA

Jika meneliti tentang data mengungkapkan terlalu banyak nilai yang hilang atau nilai yang bertentangan dengan kenyataan seperti angka yang berkisar dalam miliaran ketika Anda mengharapkan ribuan data, inilah saatnya untuk mengevaluasi kembali penggunaannya. Data mungkin tidak cukup andal untuk dijadikan sebagai dasar analisis Anda.

Jika Anda curiga, langkah pertama adalah meninjau kembali file data asli. Pastikan Anda mengimpornya dengan benar dan nilai di semua kolom sumber terletak di kolom yang sama di tabel. Anda mungkin perlu membuka spreadsheet atau file CSV asli dan melakukan perbandingan visual. Langkah kedua adalah menelepon instansi atau perusahaan yang menghasilkan data untuk mengkonfirmasi apa yang Anda lihat dan mencari penjelasan. Anda juga dapat meminta saran dari orang lain yang telah menggunakan data yang sama.

Lebih dari sekali saya harus membuang kumpulan data setelah menentukan bahwa itu tidak dirakit dengan baik atau tidak lengkap. Terkadang, jumlah pekerjaan yang diperlukan untuk membuat kumpulan data dapat digunakan merusak kegunaannya. Situasi ini mengharuskan Anda untuk membuat keputusan yang sulit. Tetapi lebih baik memulai dari awal atau mencari alternatif daripada menggunakan data buruk yang dapat mengarah pada kesimpulan yang salah.

Memodifikasi Tabel dengan ALTER TABLE

Kita dapat menggunakan pernyataan ALTER TABLE untuk memodifikasi struktur tabel. Contoh berikut menunjukkan sintaks untuk operasi umum yang merupakan bagian dari standar ANSI SQL. Kode untuk menambahkan kolom ke tabel terlihat seperti ini:

```
ALTER TABLE table ADD COLUMN column data_type;
```

Demikian pula, kita dapat menghapus kolom dengan sintaks berikut:

```
ALTER TABLE table DROP COLUMN column;
```

Untuk mengubah tipe data kolom, kami akan menggunakan kode ini:

```
ALTER TABLE table ALTER COLUMN column SET DATA TYPE data_type;
```

Menambahkan batasan NOT NULL ke kolom akan terlihat seperti berikut:

```
ALTER TABLE table ALTER Column column SET NOT NULL;
```

Perhatikan bahwa di PostgreSQL dan beberapa sistem lain, menambahkan batasan ke tabel menyebabkan semua baris diperiksa untuk melihat apakah mereka mematuhi batasan. Jika tabel memiliki jutaan baris, ini bisa memakan waktu cukup lama.

Menghapus batasan NOT NULL terlihat seperti ini:

```
ALTER TABLE table ALTER COLUMN column DROP NOT NULL;
```

Saat Anda menjalankan pernyataan ALTER TABLE dengan placeholder terisi, Anda akan melihat pesan yang berbunyi ALTER TABLE di layar keluaran pgAdmin. Jika operasi melanggar batasan atau jika Anda mencoba mengubah tipe data kolom dan nilai yang ada di kolom tidak sesuai dengan tipe data baru, PostgreSQL mengembalikan kesalahan. Tetapi PostgreSQL tidak akan memberi Anda peringatan apa pun tentang menghapus data saat Anda menjatuhkan kolom, jadi berhati-hatilah sebelum menjatuhkan kolom.

Memodifikasi Nilai dengan UPDATE

Pernyataan UPDATE memodifikasi data dalam kolom di semua baris atau di subset baris yang memenuhi kondisi. Sintaks dasarnya, yang akan memperbarui data di setiap baris dalam kolom, mengikuti formulir ini:

```
UPDATE table
SET column = value;
```

Kami pertama-tama meneruskan UPDATE nama tabel yang akan diperbarui, dan kemudian meneruskan klausa SET ke kolom yang berisi nilai yang akan diubah. Nilai baru untuk ditempatkan di kolom bisa berupa string, angka, nama kolom lain, atau bahkan kueri atau ekspresi yang menghasilkan nilai. Kami dapat memperbarui nilai dalam beberapa kolom sekaligus dengan menambahkan kolom tambahan dan nilai sumber, dan memisahkan setiap kolom dan pernyataan nilai dengan koma:

```
UPDATE table
SET column_a = value,
      Column_b = value;
```

Untuk membatasi pembaruan pada baris tertentu, kami menambahkan klausa WHERE dengan beberapa kriteria yang harus dipenuhi sebelum pembaruan dapat terjadi:

```
UPDATE table
SET column = value
WHERE criteria;
```

Kami juga dapat memperbarui satu tabel dengan nilai dari tabel lain. SQL ANSI standar mengharuskan kami menggunakan subquery, kueri di dalam kueri, untuk menentukan nilai dan baris mana yang akan diperbarui:

```
UPDATE table
SET column = (SELECT column FROM table_b WHERE table.column = table_b.column)
WHERE EXIST (SELECT column FROM table_b WHERE table.column = table_b.column);
```

Bagian nilai dari klausa SET adalah subquery, yang merupakan pernyataan SELECT di dalam tanda kurung yang menghasilkan nilai untuk pembaruan. Demikian pula, klausa WHERE EXISTS menggunakan pernyataan SELECT untuk menghasilkan nilai yang berfungsi sebagai filter untuk pembaruan. Jika kami tidak menggunakan klausa ini, kami mungkin secara tidak sengaja menetapkan beberapa nilai ke NULL tanpa direncanakan. (Jika sintaks ini terlihat agak rumit, tidak apa-apa. Saya akan membahas subquery secara rinci di Bab 12.)

Beberapa manajer database menawarkan sintaks tambahan untuk memperbarui seluruh tabel. PostgreSQL mendukung standar ANSI tetapi juga sintaks yang lebih sederhana menggunakan klausa FROM untuk memperbarui nilai di seluruh tabel:

```
UPDATE table
SET column = table_b.column
FROM table_b
WHERE table.column = table_b.column;
```

Saat Anda menjalankan pernyataan UPDATE, PostgreSQL mengembalikan pesan yang menyatakan UPDATE bersama dengan jumlah baris yang terpengaruh.

Membuat Tabel Cadangan

Sebelum memodifikasi tabel, sebaiknya buat salinan untuk referensi dan cadangan jika Anda tidak sengaja merusak beberapa data. Daftar 9-8 menunjukkan bagaimana menggunakan variasi dari pernyataan CREATE TABLE yang sudah dikenal untuk membuat tabel baru berdasarkan data dan struktur tabel yang ada yang ingin kita duplikat:

```
CREATE TABLE meat_poultry_egg_inspect_backup AS
SELECT * FROM meat_poultry_egg_inspect;
```

Setelah menjalankan pernyataan CREATE TABLE, hasilnya harus berupa salinan murni dari tabel Anda dengan nama baru yang ditentukan. Anda dapat mengonfirmasi ini dengan menghitung jumlah rekaman di kedua tabel dengan satu kueri:

```
(SELECT count(*) FROM meat_poultry_egg_inspect) AS original,
(SELECT count(*) FROM meat_poultry_egg_inspect_backup) AS backup;
```

Hasilnya harus mengembalikan hitungan 6.287 dari kedua tabel, seperti ini:

```
original    backup
--------    ------
6287        6287
```

Jika jumlahnya cocok, Anda dapat yakin bahwa tabel cadangan Anda adalah salinan persis dari struktur dan isi tabel asli. Sebagai ukuran tambahan dan untuk referensi mudah, kami akan menggunakan ALTER TABLE untuk membuat salinan data kolom dalam tabel yang kami perbarui.

Catatan: Indeks tidak disalin saat membuat cadangan tabel menggunakan pernyataan CREATE TABLE. Jika Anda memutuskan untuk menjalankan kueri pada cadangan, pastikan untuk membuat indeks terpisah di tabel itu.

Memulihkan Nilai Kolom yang Hilang

Sebelumnya di bab ini, kueri dibawah ini mengungkapkan bahwa tiga baris dalam tabel meat_poultry_egg_inspect tidak memiliki nilai di kolom pertama:

```
est_number        company                          city        st    zip
----------        -------------------------------  ------      --    ---
V18677a           Atlas, Inspection, Inc.          Blaine            55449
M45319+P45319     Hall-Namie Packing Company, Inc                    36671
M263A+P263A+V263A Jones Dairy Farm                                   53538
```

Untuk mendapatkan jumlah lengkap pendirian di setiap negara bagian, kita perlu mengisi nilai yang hilang tersebut menggunakan pernyataan UPDATE.

Membuat Salinan Kolom

Meskipun kita telah mencadangkan tabel ini, mari kita lebih berhati-hati dan membuat salinan kolom pertama di dalam tabel sehingga kita masih memiliki data asli jika kita membuat kesalahan besar di suatu tempat! Mari buat salinan dan isi dengan nilai kolom st yang ada menggunakan pernyataan SQL ini:

```
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN st_copy varchar(2);

UPDATE meat_polutry_egg_inspect
SET st_copy = st;
```

Pernyataan ALTER TABLE menambahkan kolom bernama st_copy menggunakan tipe data varchar yang sama dengan kolom st asli. Selanjutnya, klausa SET pernyataan UPDATE mengisi kolom st_copy yang baru kita buat dengan nilai di kolom st. Karena kami tidak menentukan kriteria apa pun menggunakan klausa WHERE, nilai di setiap baris diperbarui, dan PostgreSQL mengembalikan pesan UPDATE 6287. Sekali lagi, perlu dicatat bahwa pada tabel yang sangat besar, operasi ini bisa memakan waktu dan juga secara substansial meningkatkan ukuran tabel. Membuat salinan kolom selain cadangan tabel tidak sepenuhnya diperlukan, tetapi jika Anda adalah tipe orang yang sabar dan berhati-hati, itu bisa bermanfaat.

Kami dapat mengonfirmasi bahwa nilai telah disalin dengan benar dengan kueri SELECT sederhana di kedua kolom, seperti pada listing dibawah ini:

```
SELECT st,
      st_copy
FROM meat_poultry_egg_inspect
ORDER BY st;
```

Kueri SELECT mengembalikan 6.287 baris yang menunjukkan kedua kolom yang menyimpan nilai kecuali tiga baris dengan nilai yang hilang:

```
st    st_copy
--    -------
AK    AK
AK    AK
AK    AK
AK    AK
--snip--
```

Sekarang, dengan data asli kami disimpan dengan aman di kolom st_copy, kami dapat memperbarui tiga baris dengan kode status yang hilang. Ini sekarang adalah cadangan dalam tabel kami, jadi jika terjadi kesalahan drastis saat kami memperbarui data yang hilang di kolom asli, kami dapat dengan mudah menyalin kembali data asli. Saya akan menunjukkan caranya setelah kami menerapkan pembaruan pertama.

Memperbarui Baris dimana Nilai Hilang

Untuk memperbarui baris nilai yang hilang, pertama-tama kita temukan nilai yang kita butuhkan dengan pencarian online cepat: Atlas Inspection berlokasi di Minnesota; Hall-Namie Packing berada di Alabama; dan Jones Dairy ada di Wisconsin. Tambahkan status tersebut ke baris yang sesuai menggunakan kode dibawah ini:

```
UPDATE meat_poultry_egg_inspect
SET st = 'MN'
WHERE est_number = 'V18677A':

UPDATE meat_poultry_egg_inspect
SET st = 'AL'
WHERE est_number = 'M45319+p45319';

UPDATE meat_poultry_egg_inspect
SET st = 'AL'
WHERE est_number = 'M263A+P263A+V263A';
```

Karena kami ingin setiap pernyataan UPDATE memengaruhi satu baris, kami menyertakan klausa WHERE untuk setiap pernyataan yang mengidentifikasi est_number unik perusahaan, yang merupakan kunci utama tabel. Saat kami menjalankan setiap kueri, PostgreSQL merespons dengan pesan UPDATE 1, menunjukkan bahwa hanya satu baris yang diperbarui untuk setiap kueri.

Jika kita menjalankan kembali kode untuk menemukan baris di mana st adalah NULL, kueri tidak akan menghasilkan apa-apa. Keberhasilan! Hitungan kami berdasarkan negara bagian sekarang selesai.

Mengembalikan Nilai Asli

Apa yang terjadi jika kami merusak pembaruan dengan memberikan nilai yang salah atau memperbarui baris yang salah? Karena kami telah mencadangkan seluruh tabel dan kolom pertama di dalam tabel, kami dapat dengan mudah menyalin kembali data dari kedua lokasi tersebut.

```
UPDATE meat_poultry_egg_inspect
SET st = st_copy;

UPDATE meat_poultry_egg_inspect original
SET st = backup.st;
FROM meat_poultry_egg_inspect_backup backup
WHERE original.est_number = backup.est_number;
```

Untuk mengembalikan nilai dari kolom cadangan di meat_poultry_egg_inspect yang Anda buat, jalankan kueri UPDATE yang menetapkan st ke nilai di st_copy. Kedua kolom harus kembali memiliki nilai asli yang identik.

Atau, Anda dapat membuat UPDATE yang menetapkan nilai st ke kolom st dari tabel meat_poultry_egg_inspect_backup yang Anda buat.

Memperbarui Nilai untuk Konsistensi

Kami menemukan beberapa kasus di mana satu nama perusahaan dimasukkan secara tidak konsisten. Jika kami ingin menggabungkan data berdasarkan nama perusahaan, inkonsistensi seperti itu akan menghalangi kami untuk melakukannya. Berikut adalah variasi ejaan Armor-Eckrich Meats.

```
--snip–
Armour – Eckrich Meats, LLC
Armour-Eckrich Meats LLC
Armou-Eckrich Meats, Inc.
Armour-Eckrich, LLC
--snip--
```

Kita bisa membakukan ejaan nama perusahaan ini dengan menggunakan pernyataan UPDATE. Untuk melindungi data kami, kami akan membuat kolom baru untuk ejaan standar, menyalin nama di perusahaan ke kolom baru, dan bekerja di kolom baru untuk menghindari gangguan dengan data asli.

```
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN company_standart varchar(100);

UPDATE meat_poultry_egg_inspect
SET company_standart = company;
```

Sekarang, katakanlah kita ingin nama apa pun di perusahaan yang berisi string Armor muncul di company_standard sebagai Armor-Eckrich Meats. (Ini mengasumsikan kami telah memeriksa semua entri yang berisi Armor dan ingin menstandarkannya.) Kami dapat memperbarui semua baris yang cocok dengan string Armor dengan menggunakan klausa WHERE. Jalankan dua pernyataan di bawah ini:

```
UPDATE meat_poultry_egg_inspect
SET company_standart = 'Armour-Eckrich Meats'
WHERE company LIKE 'Armour%';

SELECT company, company_standart
FROM meat_poultry_egg_inspect
WHERE company LIKE 'Armour%';
```

Bagian penting dari kueri ini adalah klausa WHERE yang menggunakan kata kunci LIKE yang diperkenalkan dengan pemfilteran di Bab 2. Menyertakan sintaks wildcard % di akhir string Armor memperbarui semua baris yang dimulai dengan karakter tersebut terlepas dari apa yang muncul setelahnya mereka. Klausa memungkinkan kita menargetkan semua ejaan bervariasi yang digunakan untuk nama perusahaan. Pernyataan SELECT di listing dibawah ini mengembalikan hasil dari kolom company_standard yang diperbarui di sebelah kolom perusahaan asli:

```
company                          company_standart
---------------------------      --------------------
Armour-Ecrich Meats LLC          Armour-Eckrich Meats
Armour – Eckrich Meats, LLC      Armour-Eckrich Meats
```

```
Armour-Eckrich Meats LLC          Armour-Eckrich Meats
Armour-Eckrich Meats LLC          Armour-Eckrich Meats
Armour-Eckrich Meats, INC         Armour-Eckrich Meats
Armour-Eckrich Meats, LLC         Armour-Eckrich Meats
Armour-Eckrich Meats, LLC         Armour-Eckrich Meats
```

Nilai untuk Armor-Eckrich di company_standard sekarang distandarisasi dengan ejaan yang konsisten. Jika kita ingin membakukan nama perusahaan lain dalam tabel, kita akan membuat pernyataan UPDATE untuk setiap kasus. Kami juga akan menyimpan kolom perusahaan asli untuk referensi.

Memperbaiki Kode ZIP Menggunakan Penggabungan

Perbaikan terakhir kami memperbaiki nilai di kolom zip yang kehilangan angka nol di depan sebagai akibat dari kecerobohan data saya yang disengaja. Untuk perusahaan di Puerto Rico dan A.S. Kepulauan Virgin, kami perlu memulihkan dua angka nol di depan ke nilai dalam zip karena (selain dari fasilitas pemrosesan IRS di Holtsville, NY) mereka adalah satu-satunya lokasi di Amerika Serikat di mana Kode Pos dimulai dengan dua angka nol. Kemudian, untuk negara bagian lain, yang sebagian besar berlokasi di New England, kami akan mengembalikan satu nol di depan.

Kami akan menggunakan UPDATE lagi tetapi kali ini bersama dengan operator string pipa ganda (||), yang melakukan penggabungan. Penggabungan menggabungkan dua atau lebih nilai string atau non-string menjadi satu. Misalnya, menyisipkan || antara string abc dan 123 menghasilkan abc123. Operator pipa ganda adalah standar SQL untuk penggabungan yang didukung oleh PostgreSQL. Anda dapat menggunakannya dalam banyak konteks, seperti kueri UPDATE dan SELECT, untuk memberikan keluaran khusus dari data yang sudah ada maupun yang baru.

Pertama, listing berikut ini membuat salinan cadangan kolom zip dengan cara yang sama seperti kita membuat cadangan kolom st sebelumnya:

```
ALTER TABLE meat_poultry_egg_inspect ADD COLUMN zip_copy varchar(5);

UPDATE meat_poultry_egg_inspect
SET zip_copy = zip;
```

Selanjutnya, kami menggunakan kode dibawah ini untuk melakukan pembaruan pertama:

```
UPDATE meat_poultry_egg_inspect
SET zip = '00' || zip
WHERE st IN('PR', 'VI') AND length(zip) = 3;
```

Kami menggunakan SET untuk mengatur kolom zip ke nilai yang merupakan hasil dari rangkaian string 00 dan konten kolom zip yang ada. Kami membatasi UPDATE hanya pada baris di mana kolom pertama memiliki kode status PR dan VI menggunakan operator perbandingan IN dari Bab 2 dan menambahkan tes untuk baris dengan panjang zip 3. Seluruh pernyataan ini kemudian hanya akan memperbarui nilai zip untuk Puerto Rico dan Kepulauan Virgin.

Jalankan kueri; PostgreSQL harus mengembalikan pesan UPDATE 86, yang merupakan jumlah baris yang kita harapkan untuk diubah berdasarkan hitungan kita sebelumnya di listing sebelumnya

Mari kita perbaiki Kode ZIP yang tersisa menggunakan kueri serupa kode dibawah ini:

```
UPDATE meat_poultry_egg_inspect
SET zip = '0' || zip
WHERE st IN('CT','MA','ME','NH','NJ','RI','VT') AND length(zip) = 4;
```

PostgreSQL seharusnya mengembalikan pesan UPDATE 496. Sekarang, mari kita periksa kemajuan kita. Sebelumnya di bab ini, ketika kami menggabungkan baris dalam kolom zip berdasarkan panjangnya, kami menemukan 86 baris dengan tiga karakter dan 496 dengan empat:

```
length      count
------      -----
     3         86
     4        496
     5       5705
```

Menggunakan kueri yang sama untuk mengembalikan hasil yang lebih diinginkan: semua baris memiliki Kode Pos lima digit.

```
length      count
------      -----
     5       6287
```

Dalam contoh ini kami menggunakan penggabungan, tetapi Anda dapat menggunakan fungsi string SQL tambahan untuk memodifikasi data dengan UPDATE dengan mengubah kata dari huruf besar ke huruf kecil, memangkas spasi yang tidak diinginkan, mengganti karakter dalam string, dan banyak lagi. Saya akan membahas fungsi string tambahan di Bab 13 ketika kita mempertimbangkan teknik lanjutan untuk bekerja dengan teks.

Memperbarui Nilai di Seluruh Tabel

Dalam "Memodifikasi Nilai dengan UPDATE", saya menunjukkan standar ANSI SQL dan sintaks khusus PostgreSQL untuk memperbarui nilai dalam satu tabel berdasarkan nilai di tabel lain. Sintaks ini sangat berharga dalam basis data relasional di mana kunci utama dan kunci asing membentuk hubungan tabel. Ini juga berguna ketika data dalam satu tabel mungkin merupakan konteks yang diperlukan untuk memperbarui nilai di tabel lain.

Misalnya, katakanlah kami menetapkan tanggal inspeksi untuk masing-masing perusahaan di tabel kami. Kami ingin melakukan ini oleh A.S. daerah, seperti Timur Laut, Pasifik, dan seterusnya, tetapi sebutan regional tersebut tidak ada di tabel kami. Namun, mereka memang ada dalam kumpulan data yang dapat kita tambahkan ke database kita yang juga berisi kode status st yang cocok. Ini berarti kami dapat menggunakan data lain tersebut sebagai bagian dari pernyataan UPDATE kami untuk memberikan informasi yang diperlukan. Mari kita mulai dengan wilayah New England untuk melihat cara kerjanya.

Masukkan kode dibawah ini, yang berisi pernyataan SQL untuk membuat tabel state_regions dan mengisi tabel dengan data:

```
CREATE TABLE state_regions (
      St varchar(2) CONSTRAINT st_key PRIMARY KEY,
      region varchar(20) NOT NULL );

COPY state_regions
FROM 'C:\YourDirectory\state_regions.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

Kami akan membuat dua kolom dalam tabel state_regions: satu berisi kode negara dua karakter st dan yang lainnya berisi nama wilayah. Kami menetapkan batasan kunci utama ke kolom st, yang menyimpan nilai st_key unik untuk mengidentifikasi setiap status. Dalam data yang Anda impor, setiap negara bagian ada dan ditetapkan ke A.S. Wilayah sensus, dan wilayah di luar Amerika Serikat diberi label sebagai daerah terpencil. Kami akan memperbarui tabel satu wilayah pada satu waktu.

Selanjutnya, mari kembali ke tabel meat_poultry_egg_inspect, tambahkan kolom untuk tanggal pemeriksaan, lalu isi kolom tersebut dengan negara bagian New England. Daftar 9-19 menunjukkan kode:

```
ALTER TABLE meat_poultry_egg_inspect

UPDATE meat_poultry_egg_inspect inspect
SET inspection_date = '2019-12-01'
WHERE EXISTS (SELECT state_regions.region
            FROM state_regions
            WHERE inspect.st = state_regions.st
                  AND state_regions.region = 'New England');
```

Pernyataan ALTER TABLE membuat kolom inspect_date dalam tabel meat_poultry_egg_inspect. Dalam pernyataan UPDATE, kita mulai dengan memberi nama tabel menggunakan alias dari inspect untuk membuat kode lebih mudah dibaca . Selanjutnya, klausa SET menetapkan nilai tanggal 01-12-2019 ke kolom tanggal_pemeriksaan baru . Terakhir, klausa WHERE EXISTS menyertakan subkueri yang menghubungkan tabel meat_poultry_egg_inspect ke tabel state_regions yang kita buat di Daftar 9-18 dan menentukan baris mana yang akan diperbarui . Subquery (dalam tanda kurung, dimulai dengan SELECT) mencari baris dalam tabel state_regions di mana kolom region cocok dengan string New England. Pada saat yang sama, ia bergabung dengan tabel meat_poultry_egg_inspect dengan tabel state_regions menggunakan kolom st dari kedua tabel. Akibatnya, kueri memberi tahu database untuk menemukan semua kode st yang sesuai dengan wilayah New England dan menggunakan kode tersebut untuk memfilter pembaruan.

Saat Anda menjalankan kode, Anda akan menerima pesan UPDATE 252, yang merupakan jumlah perusahaan di New England. Anda dapat menggunakan kode ini untuk melihat efek dari perubahan:

```
SELECT st, inspection_date
FROM meat_poultry_egg_inspect
GROUP BY st, inspection_date
ORDER BY st;
```

Hasilnya harus menunjukkan tanggal inspeksi yang diperbarui untuk semua perusahaan New England. Bagian atas output menunjukkan Connecticut telah menerima tanggal, misalnya, tetapi negara bagian di luar New England tetap NULL karena kami belum memperbaruinya:

```
st    inspection_date
--    ---------------
--snip--
CA
CO
CT    2019-12-01
DC
--ship--
```

Untuk mengisi tanggal untuk wilayah tambahan, ganti wilayah yang berbeda untuk New England di Daftar 9-19 dan jalankan kembali kueri.

Menghapus Data yang Tidak Perlu

Cara yang paling tidak dapat dibatalkan untuk mengubah data adalah dengan menghapusnya sepenuhnya. SQL menyertakan opsi untuk menghapus baris dan kolom dari tabel bersama dengan opsi untuk menghapus seluruh tabel atau database. Kami ingin melakukan operasi ini dengan hati-hati, hanya menghapus data atau tabel yang tidak kami perlukan. Tanpa backup, data akan hilang untuk selamanya.

Catatan: Sangat mudah untuk mengecualikan data yang tidak diinginkan dalam kueri menggunakan klausa WHERE, jadi putuskan apakah Anda benar-benar perlu menghapus data atau hanya memfilternya. Kasus di mana menghapus mungkin merupakan solusi terbaik termasuk data dengan kesalahan atau data yang diimpor secara tidak benar.

Di bagian ini, kita akan menggunakan berbagai pernyataan SQL untuk menghapus data yang tidak perlu. Untuk menghapus baris dari tabel, kita akan menggunakan pernyataan DELETE FROM. Untuk menghapus kolom dari tabel, kita akan menggunakan ALTER TABLE. Dan untuk menghapus seluruh tabel dari database, kita akan menggunakan pernyataan DROP TABLE.

Menulis dan mengeksekusi pernyataan-pernyataan ini cukup sederhana, tetapi melakukannya disertai dengan peringatan. Jika menghapus baris, kolom, atau tabel akan menyebabkan pelanggaran batasan, seperti batasan kunci asing yang dibahas dalam Bab 7, Anda harus menangani batasan itu terlebih dahulu. Itu mungkin melibatkan penghapusan batasan, penghapusan data di tabel lain, atau penghapusan tabel lain. Setiap kasus adalah unik dan akan membutuhkan cara yang berbeda untuk mengatasi kendala tersebut.

Menghapus Baris dari Tabel

Menggunakan pernyataan DELETE FROM, kita dapat menghapus semua baris dari tabel, atau kita dapat menggunakan klausa WHERE untuk menghapus hanya bagian yang cocok dengan ekspresi yang kita berikan. Untuk menghapus semua baris dari tabel, gunakan sintaks berikut:

```
DELETE FROM table_name;
```

Jika tabel Anda memiliki banyak baris, mungkin akan lebih cepat untuk menghapus tabel dan membuat versi baru menggunakan pernyataan CREATE TABLE asli. Untuk menghapus tabel,

Untuk menghapus hanya baris yang dipilih, tambahkan klausa WHERE bersama dengan nilai atau pola yang cocok untuk menentukan mana yang ingin Anda hapus:

```
DELETE FROM table_name WHERE expression;
```

Misalnya, jika kami ingin meja pengolah daging, unggas, dan telur kami hanya menyertakan perusahaan di 50 AS. menyatakan, kita dapat menghapus perusahaan di Puerto Rico dan Kepulauan Virgin dari tabel menggunakan sintaks dibawah ini:

```
DELETE FROM meat_poultry_egg_inspect
WHERE st IN('PR','VI');
```

Jalankan kodenya; PostgreSQL harus mengembalikan pesan DELETE 86. Ini berarti 86 baris di mana kolom pertama berisi PR atau VI telah dihapus dari tabel.

Menghapus Kolom dari Tabel

Saat mengerjakan kolom zip di tabel meat_poultry_egg_inspect di awal bab ini, kami membuat kolom cadangan yang disebut zip_copy. Sekarang setelah kami selesai memperbaiki masalah di zip, kami tidak lagi membutuhkan zip_copy. Kita dapat menghapus kolom cadangan, termasuk semua data di dalam kolom, dari tabel dengan menggunakan kata kunci DROP dalam pernyataan ALTER TABLE.

Sintaks untuk menghapus kolom mirip dengan pernyataan ALTER TABLE lainnya:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Dan untuk menghapus kolom zip_copy:

```
ALTER TABLE meat_poultry_egg_inspect DROP COLUMN zip_copy;
```

PostgreSQL mengembalikan pesan ALTER TABLE, dan kolom zip_copy harus dihapus.

Menghapus Tabel dari Database

Pernyataan DROP TABLE adalah fitur ANSI SQL standar yang menghapus tabel dari database. Pernyataan ini mungkin berguna jika, misalnya, Anda memiliki koleksi cadangan, atau tabel kerja, yang sudah tidak berguna lagi. Ini juga berguna dalam situasi lain, seperti ketika Anda perlu mengubah struktur tabel secara signifikan; dalam hal ini, daripada menggunakan terlalu banyak pernyataan ALTER TABLE, Anda dapat menghapus tabel dan membuat yang lain dengan menjalankan pernyataan CREATE TABLE baru.

Sintaks untuk perintah DROP TABLE sederhana:

```
DROP TABLE table_name;
```

Misalnya, sintaks ini untuk menghapus versi cadangan tabel meat_poultry_egg_inspect:

```
DROP TABLE meat_poultry_egg_inspect_backup;
```

Jalankan kueri; PostgreSQL harus merespons dengan pesan DROP TABLE untuk menunjukkan bahwa tabel telah dihapus.

Menggunakan Blok Transaksi untuk Menyimpan atau Mengembalikan Perubahan

Perubahan yang Anda buat pada data menggunakan teknik dalam bab ini sejauh ini bersifat final. Artinya, setelah Anda menjalankan kueri DELETE atau UPDATE (atau kueri lain apa pun yang mengubah data atau struktur database Anda), satu-satunya cara untuk membatalkan perubahan adalah dengan memulihkan dari cadangan. Namun, Anda dapat memeriksa perubahan Anda sebelum menyelesaikannya dan membatalkan perubahan jika itu bukan yang Anda inginkan. Anda melakukan ini dengan membungkus pernyataan SQL di dalam blok transaksi, yang merupakan sekelompok pernyataan yang Anda definisikan menggunakan kata kunci berikut di awal dan akhir kueri:

MULAI TRANSAKSI menandakan dimulainya blok transaksi. Di PostgreSQL, Anda juga dapat menggunakan kata kunci BEGIN SQL non-ANSI.

COMMIT menandakan akhir blok dan menyimpan semua perubahan.

ROLLBACK menandakan akhir blok dan mengembalikan semua perubahan.

Biasanya, programmer database menggunakan blok transaksi untuk menentukan awal dan akhir dari urutan operasi yang melakukan satu unit kerja dalam database. Contohnya adalah ketika Anda membeli tiket pertunjukan Broadway. Transaksi yang berhasil mungkin melibatkan dua langkah: menagih kartu kredit Anda dan memesan tempat duduk Anda sehingga orang lain tidak dapat membelinya. Pemrogram basis data ingin kedua langkah dalam transaksi terjadi (misalnya, ketika tagihan kartu Anda berhasil) atau keduanya tidak terjadi (jika kartu Anda ditolak atau Anda membatalkannya saat checkout). Mendefinisikan kedua langkah sebagai satu transaksi membuat mereka sebagai satu kesatuan; jika satu langkah gagal, yang lain juga dibatalkan. Anda dapat mempelajari lebih detail tentang transaksi dan PostgreSQL di https://www.postgresql.org/docs/current/static/tutorial-transactions.html.

Kita dapat menerapkan teknik blok transaksi ini untuk meninjau perubahan yang dibuat kueri dan kemudian memutuskan apakah akan menyimpan atau membuangnya. Menggunakan tabel meat_poultry_egg_inspect, misalkan kita sedang membersihkan data kotor yang terkait dengan perusahaan AGRO Merchants Oakland LLC. Tabel memiliki tiga baris yang mencantumkan perusahaan, tetapi satu baris memiliki koma tambahan pada namanya:

```
company
--------------------------
AGRO Merchants Oakland LLC
AGRO Merchants Oakland LLC
AGRO Merchants Oakland, LLC
```

Kami ingin namanya konsisten, jadi kami akan menghapus koma dari baris ketiga menggunakan kueri UPDATE, seperti yang kami lakukan sebelumnya. Tapi kali ini kami akan memeriksa hasil pembaruan kami sebelum kami membuatnya final (dan kami sengaja membuat kesalahan yang ingin kami buang). Sintaks dibawah ini menunjukkan bagaimana melakukan ini menggunakan blok transaksi:

```
START TRANSACTION;
```

```
UPDATE meat_poultry_egg-inspect
SET company = 'AGRO Merchantss Oakland LLC'
WHERE company = 'AGRO Merchants Oakland, LLC';

SELECT company
FROM meat_poultry_egg_inspect
WHERE company LIKE 'AGRO%'
ORDER BY company;

ROLLBACK;
```

Kami akan menjalankan setiap pernyataan secara terpisah, dimulai dengan MULAI TRANSAKSI; Basis data merespons dengan pesan MULAI TRANSAKSI, memberi tahu Anda bahwa perubahan apa pun yang Anda buat pada data tidak akan dibuat permanen kecuali Anda mengeluarkan perintah COMMIT. Selanjutnya, kami menjalankan pernyataan UPDATE, yang mengubah nama perusahaan di baris yang memiliki koma tambahan. Saya sengaja menambahkan 's' tambahan pada nama yang digunakan dalam klausa SET untuk memperkenalkan kesalahan.

Ketika kita melihat nama perusahaan yang dimulai dengan huruf AGRO menggunakan pernyataan SELECT , kita melihat bahwa, oops, satu nama perusahaan salah eja sekarang:

```
company
--------------------------
AGRO Merchants Oakland LLC
AGRO Merchants Oakland LLC
AGRO Merchantss Oakland LLC
```

Alih-alih menjalankan kembali pernyataan UPDATE untuk memperbaiki kesalahan ketik, kita cukup membuang perubahan dengan menjalankan ROLLBACK; Perintah. Saat kami menjalankan kembali pernyataan SELECT untuk melihat nama perusahaan, kami kembali ke tempat kami memulai:

```
company
--------------------------
AGRO Merchants Oakland LLC
AGRO Merchants Oakland LLC
AGRO Merchants Oakland, LLC
```

Dari sini, Anda dapat memperbaiki pernyataan UPDATE Anda dengan menghapus s tambahan dan menjalankannya kembali, dimulai dengan pernyataan MULAI TRANSAKSI lagi. Jika Anda senang dengan perubahannya, jalankan COMMIT; untuk membuat mereka permanen.

Catatan: Saat Anda memulai transaksi, setiap perubahan yang Anda buat pada data tidak akan terlihat oleh pengguna database lain hingga Anda menjalankan COMMIT.

Blok transaksi sering digunakan dalam sistem database yang lebih kompleks. Di sini Anda telah menggunakannya untuk mencoba kueri dan menerima atau menolak perubahan, menghemat waktu dan sakit kepala Anda. Selanjutnya, mari kita lihat cara lain untuk menghemat waktu saat memperbarui banyak data.

Meningkatkan Kinerja Saat Memperbarui Tabel Besar

Karena cara kerja PostgreSQL secara internal, menambahkan kolom ke tabel dan mengisinya dengan nilai dapat dengan cepat memperbesar ukuran tabel. Alasannya adalah database membuat versi baru dari baris yang ada setiap kali nilai diperbarui, tetapi tidak menghapus versi baris lama. (Anda akan mempelajari cara membersihkan baris lama ini ketika saya membahas pemeliharaan basis data di "Memulihkan Ruang yang Tidak Digunakan dengan VAKUM" di halaman 314.) Untuk kumpulan data kecil, peningkatannya dapat diabaikan, tetapi untuk tabel dengan ratusan ribu atau jutaan baris, waktu yang diperlukan untuk memperbarui baris dan penggunaan disk ekstra yang dihasilkan bisa sangat besar.

Alih-alih menambahkan kolom dan mengisinya dengan nilai, kita dapat menghemat ruang disk dengan menyalin seluruh tabel dan menambahkan kolom yang terisi selama operasi. Kemudian, kami mengganti nama tabel sehingga salinan menggantikan yang asli, dan yang asli menjadi cadangan.

Kode dibawah ini menunjukkan cara menyalin meat_poultry_egg_inspect ke dalam tabel baru sambil menambahkan kolom yang terisi. Untuk melakukan ini, pertama jatuhkan tabel meat_poultry_egg_inspect_backup yang kita buat sebelumnya. Kemudian jalankan pernyataan CREATE TABLE.

```
CREATE TABLE meat_poultry_egg_inspect_backup AS
SELECT *,
      '2018-02-07'::date AS reviewed_date
FROM meat_poultry_egg_inspect;
```

Kueri adalah versi skrip cadangan yang dimodifikasi di listing sebelumnya. Di sini, selain memilih semua kolom menggunakan wildcard asterisk, kami juga menambahkan kolom yang disebut review_date dengan memberikan nilai cast sebagai tipe data tanggal dan kata kunci AS. Sintaks itu menambahkan dan mengisi tanggal_review, yang mungkin kita gunakan untuk melacak kapan terakhir kali kita memeriksa status setiap pabrik. Kode dibawah ini untuk mengganti nama tabel menggunakan ALTER TABLE

```
ALTER TABLE meat_polutry_egg_inspect RENAME TO meat_poultry_egg_inspect_temp;
ALTER TABLE meat_poultry_egg_inspect_backup
      RENAME TO meat_poultry_egg_inspect;

ALTER TABLE meat_poultry_egg_inspect_temp
      RENAME TO meat_poultry_egg_inspect_backup;
```

Di sini kita menggunakan ALTER TABLE dengan klausa RENAME TO untuk mengubah nama tabel. Kemudian kami menggunakan pernyataan pertama untuk mengubah nama tabel asli menjadi yang diakhiri dengan _temp . Pernyataan kedua mengganti nama salinan yang kita buat dengan listing sebelumnya menjadi nama asli tabel . Terakhir, kami mengganti nama tabel yang berakhiran _temp menjadi _backup. Tabel aslinya sekarang disebut meat_poultry_egg_inspect_backup, dan salinan dengan kolom tambahan disebut meat_poultry_egg_inspect.

Dengan menggunakan proses ini, kami menghindari memperbarui baris dan database memperbesar ukuran tabel. Ketika kami akhirnya menjatuhkan tabel _backup, tabel data yang tersisa lebih kecil dan tidak memerlukan pembersihan.

Mengumpulkan informasi yang berguna dari data terkadang memerlukan modifikasi data untuk menghilangkan inkonsistensi, memperbaiki kesalahan, dan membuatnya lebih cocok untuk mendukung analisis yang akurat. Dalam bab ini Anda mempelajari beberapa alat yang berguna untuk membantu Anda menilai data kotor dan membersihkannya. Di dunia yang sempurna, semua set data akan tiba dengan semuanya bersih dan lengkap. Tetapi dunia yang sempurna seperti itu tidak ada, jadi kemampuan untuk mengubah, memperbarui, dan menghapus data sangat diperlukan.

Izinkan saya menyatakan kembali tugas penting bekerja dengan aman. Pastikan untuk membuat cadangan tabel Anda sebelum Anda mulai membuat perubahan. Buat salinan kolom Anda juga, untuk tingkat perlindungan ekstra. Saat saya membahas pemeliharaan database untuk PostgreSQL nanti di buku ini, Anda akan mempelajari cara mencadangkan seluruh database. Beberapa langkah pencegahan ini akan menyelamatkan Anda dari dunia yang menyakitkan.

Di bab berikutnya, kita akan kembali ke matematika untuk mengeksplorasi beberapa fungsi statistik canggih dan teknik analisis SQL.

Latihan Soal

Dalam latihan ini, Anda akan mengubah tabel meat_poultry_egg_inspect menjadi informasi yang berguna. Anda perlu menjawab dua pertanyaan: berapa banyak tanaman di meja yang mengolah daging, dan berapa banyak yang mengolah unggas?

Jawaban atas dua pertanyaan ini terletak pada kolom kegiatan. Sayangnya, kolom tersebut berisi bermacam-macam teks dengan input yang tidak konsisten. Berikut adalah contoh jenis teks yang akan Anda temukan di kolom aktivitas:

```
Poultry Processing, Poultry Slaughter
Meat Processing, Poultry Processing
Poultry Processing, Poultry Slaughter
```

Percampuran teks tidak memungkinkan untuk melakukan penghitungan biasa yang memungkinkan Anda mengelompokkan pabrik pemrosesan berdasarkan aktivitas. Namun, Anda dapat membuat beberapa modifikasi untuk memperbaiki data ini. Tugas Anda adalah sebagai berikut:

1. Buat dua kolom baru bernama meat_processing dan birds_processing di tabel Anda. Masing-masing dapat bertipe boolean.
2. Menggunakan UPDATE, atur meat_processing = TRUE pada setiap baris di mana kolom aktivitas berisi teks Meat Processing. Lakukan update yang sama pada kolom unggas_processing, tapi kali ini cari teks Poultry Processing in activities.

Gunakan data dari kolom baru yang diperbarui untuk menghitung berapa banyak tanaman yang melakukan setiap jenis aktivitas. Untuk tantangan bonus, hitung berapa banyak tanaman yang melakukan kedua aktivitas.

BAB X
FUNGSI STATISTIK DALAM SQL

Basis data SQL biasanya bukan alat pertama yang dipilih analis data saat melakukan analisis statistik yang membutuhkan lebih dari sekadar menghitung jumlah dan rata-rata. Biasanya, perangkat lunak pilihan adalah paket statistik berfitur lengkap, seperti SPSS atau SAS, bahasa pemrograman R atau Python, atau bahkan Excel. Namun, ANSI SQL standar, termasuk implementasi PostgreSQL, menawarkan beberapa fungsi statistik yang kuat yang mengungkapkan banyak hal tentang data Anda tanpa harus mengekspor kumpulan data Anda ke program lain.

Dalam bab ini, kita akan menjelajahi fungsi statistik SQL ini bersama dengan panduan tentang kapan menggunakannya. Statistik adalah subjek luas yang layak untuk bukunya sendiri, jadi kami hanya akan membaca sekilas di sini. Namun demikian, Anda akan belajar bagaimana menerapkan konsep statistik tingkat tinggi untuk membantu Anda memperoleh makna dari data Anda menggunakan kumpulan data baru dari Biro Sensus AS. Anda juga akan belajar menggunakan SQL untuk membuat perbandingan menggunakan peringkat dan tarif dengan data kejahatan FBI sebagai subjek kami.

Membuat Tabel Statistik Sensus

Mari kembali ke salah satu sumber data favorit saya, Biro Sensus AS. Di Bab sebelumnya, Anda menggunakan Sensus Tahunan 2010 untuk mengimpor data dan melakukan matematika dasar dan statistik. Kali ini Anda akan menggunakan titik data daerah yang dikumpulkan dari Perkiraan 5 Tahun Survei Komunitas Amerika (ACS) 2011–2015, survei terpisah yang dikelola oleh Biro Sensus.

Gunakan kode di bawah ini untuk membuat tabel acs_2011_2015_stats dan mengimpor file CSV acs_2011_2015_stats.csv. Kode dan data tersedia dengan semua sumber buku di https://www.nostarch.com/practicalSQL/. Ingatlah untuk mengubah C:\YourDirectory\ ke lokasi file CSV.

```
CREATE TABLE acs_2011_2015_stats (
      geoid varchar(14) CONSTRAINT geoid_key PRIMARY KEY,
      county varchar(50) NOT NULL,
      st varchar(20) NOT NULL,
      pct_travel+60_min numeric(5,3) NOT NULL,
      pct_bachelors_higher numeric(5,3) NOT NULL,
      median_hh_income integer,
      CHECK (pct_masters_higher <= pct_bachelors_higher)
);

COPY acs_2011_2015_stats
FROM 'C:\YourDirectory\acs_2011-2015_stats.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');

SELECT * FROM acs_2011_2015_stats;
```

Tabel acs_2011_2015_stats memiliki tujuh kolom. Tiga kolom pertama termasuk geoid unik yang berfungsi sebagai kunci utama, nama county, dan nama negara bagian st. Empat kolom

berikutnya menampilkan tiga persentase berikut I yang diperoleh untuk setiap kabupaten dari data mentah dalam rilis ACS, ditambah satu indikator ekonomi lagi:

- **pct_travel_60_min** Persentase pekerja berusia 16 tahun ke atas yang melakukan perjalanan lebih dari 60 menit ke tempat kerja.
- **pct_bachelors_higher** Persentase orang berusia 25 tahun ke atas yang tingkat pendidikannya sarjana atau lebih tinggi. (Di Amerika Serikat, gelar sarjana biasanya diberikan setelah menyelesaikan pendidikan perguruan tinggi empat tahun.)
- **pct_masters_higher** Persentase orang berusia 25 tahun ke atas yang tingkat pendidikannya adalah gelar master atau lebih tinggi. (Di Amerika Serikat, gelar master adalah gelar lanjutan pertama yang diperoleh setelah menyelesaikan gelar sarjana.)
- **median_hh_income** Pendapatan rumah tangga rata-rata county pada tahun 2015 dolar yang disesuaikan dengan inflasi. Seperti yang Anda pelajari di Bab 5, nilai median adalah titik tengah dalam kumpulan angka yang berurutan, di mana setengah nilainya lebih besar dari titik tengah dan setengahnya lebih kecil. Karena rata-rata dapat dimiringkan oleh beberapa nilai yang sangat besar atau sangat kecil, pelaporan pemerintah tentang data ekonomi, seperti pendapatan, cenderung menggunakan median. Di kolom ini, kami menghilangkan batasan NOT NULL karena satu county tidak memiliki data yang dilaporkan.

Kami menyertakan batasan CHECK yang Anda pelajari di Bab 7 untuk memeriksa bahwa angka untuk gelar sarjana sama dengan atau lebih tinggi daripada angka untuk gelar master, karena di Amerika Serikat, gelar sarjana diperoleh sebelum atau bersamaan dengan gelar master . Daerah yang menunjukkan kebalikannya dapat menunjukkan data yang diimpor salah atau kolom salah diberi label. Data kami keluar: saat diimpor, tidak ada kesalahan yang menunjukkan pelanggaran batasan CHECK.

Kami menggunakan pernyataan SELECT untuk melihat semua 3.142 baris yang diimpor, masing-masing sesuai dengan daerah yang disurvei dalam rilis Sensus ini.

Selanjutnya, kita akan menggunakan fungsi statistik dalam SQL untuk lebih memahami hubungan antara persentase.

Survei Komunitas Amerika, Setiap produk data Sensus A.S. memiliki metodologinya sendiri. Sensus Sepuluh Tahun adalah penghitungan penuh penduduk AS, dilakukan setiap 10 tahun melalui formulir yang dikirimkan ke setiap rumah tangga di negara tersebut. Salah satu tujuan utamanya adalah untuk menentukan jumlah kursi yang dimiliki setiap negara bagian di Dewan Perwakilan Rakyat AS. Sebaliknya, ACS adalah survei tahunan yang sedang berlangsung terhadap sekitar 3,5 juta rumah tangga AS. Ini menanyakan rincian tentang pendapatan, pendidikan, pekerjaan, keturunan, dan perumahan. Organisasi sektor swasta dan sektor publik sama-sama menggunakan data ACS untuk melacak tren dan membuat berbagai keputusan.

Saat ini, Biro Sensus mengemas data ACS menjadi dua rilis: kumpulan data 1 tahun yang memberikan perkiraan untuk geografi dengan populasi 20.000 atau lebih, dan kumpulan data 5 tahun yang mencakup semua geografi. Karena ini adalah survei, hasil ACS adalah perkiraan dan memiliki margin kesalahan, yang telah saya hilangkan untuk singkatnya tetapi yang akan Anda lihat disertakan dalam kumpulan data ACS lengkap.

Mengukur Korelasi dengan corr(Y, X)

Peneliti sering ingin memahami hubungan antar variabel, dan salah satu ukuran hubungan tersebut adalah korelasi. Di bagian ini, kita akan menggunakan fungsi corr(Y, X) untuk mengukur korelasi dan menyelidiki hubungan apa yang ada, jika ada, antara persentase orang di suatu daerah yang telah mencapai gelar sarjana dan pendapatan rumah tangga rata-rata di daerah itu. daerah. Kami juga akan menentukan apakah, menurut data kami, populasi yang berpendidikan lebih baik biasanya setara dengan pendapatan yang lebih tinggi dan seberapa kuat hubungan antara tingkat pendidikan dan pendapatan jika memang demikian.

Pertama, beberapa latar belakang. Koefisien korelasi Pearson (umumnya dilambangkan sebagai r) adalah ukuran untuk mengukur kekuatan hubungan linier antara dua variabel. Ini menunjukkan sejauh mana kenaikan atau penurunan dalam satu variabel berkorelasi dengan perubahan variabel lain. Nilai r berada di antara 1 dan 1. Salah satu ujung rentang menunjukkan korelasi yang sempurna, sedangkan nilai yang mendekati nol menunjukkan distribusi acak tanpa korelasi. Nilai r positif menunjukkan hubungan langsung: ketika satu variabel meningkat, yang lain juga. Ketika digambarkan pada sebar, titik data yang mewakili setiap pasangan nilai dalam hubungan langsung akan miring ke atas dari kiri ke kanan. Nilai r negatif menunjukkan hubungan terbalik: ketika satu variabel meningkat, yang lain menurun. Titik-titik yang mewakili hubungan terbalik akan miring ke bawah dari kiri ke kanan pada sebar.

Tabel 10.1 memberikan pedoman umum untuk menafsirkan nilai r positif dan negatif, meskipun seperti biasa dengan statistik, ahli statistik yang berbeda mungkin menawarkan interpretasi yang berbeda.

Tabel 10.1: Menafsirkan Koefisien Korelasi

Koefisien korelasi (+/−)	**Apa artinya?**
0	Tidak ada hubungan
.01 hingga .29	Hubungan yang lemah
.3 hingga .59	Hubungan sedang
.6 hingga .99	Hubungan yang kuat hingga hampir sempurna
1	Hubungan yang sempurna

Dalam standar ANSI SQL dan PostgreSQL, kami menghitung koefisien korelasi Pearson menggunakan corr(Y, X). Ini adalah salah satu dari beberapa fungsi agregat biner dalam SQL dan dinamakan demikian karena fungsi ini menerima dua input.

Dalam fungsi agregat biner, input Y adalah variabel dependen yang variasinya bergantung pada nilai variabel lain, dan X adalah variabel independen yang nilainya tidak bergantung pada variabel lain.

Catatan : Meskipun SQL menetapkan input Y dan X untuk fungsi corr(), perhitungan korelasi tidak membedakan antara variabel dependen dan independen. Mengganti urutan input di corr() menghasilkan hasil yang sama. Namun, untuk kenyamanan dan keterbacaan, contoh-contoh ini mengurutkan variabel input menurut dependen dan independen.

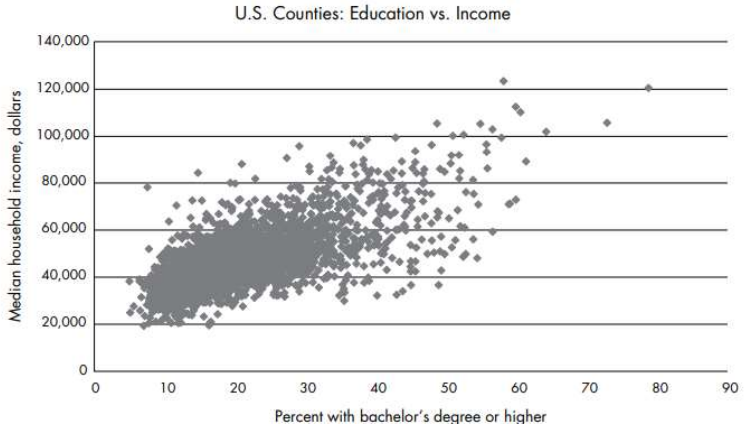
Kami akan menggunakan fungsi corr(Y, X) untuk menemukan hubungan antara tingkat pendidikan dan pendapatan. Masukkan kode di dibawah ini untuk menggunakan corr(Y, X) dengan variabel median_hh_income dan pct_bachelors_higher sebagai input:

```
SELECT corr(median_hh_income, pct_bachelors_higher)
      AS bachelors_income_r
FROM acs_2011_2015_stats;
```

Jalankan kueri; hasil Anda harus berupa nilai r tepat di atas 0,68 yang diberikan sebagai tipe data presisi ganda floating-point:

```
bachelors_income_r
------------------
0.682185675451399
```

Nilai r positif ini menunjukkan bahwa semakin tinggi tingkat pendidikan suatu daerah maka pendapatan rumah tangga cenderung meningkat. Hubungan tersebut tidak sempurna, namun nilai r menunjukkan hubungan yang cukup kuat. Kita dapat memvisualisasikan pola ini dengan memplot variabel pada scatterplot menggunakan Excel, seperti yang ditunjukkan pada Gambar 10-1. Setiap titik data mewakili satu wilayah AS; posisi titik data pada sumbu x menunjukkan persentase penduduk berusia 25 tahun ke atas yang berpendidikan sarjana ke atas. Posisi titik data pada sumbu y mewakili pendapatan rumah tangga rata-rata kabupaten.



Gambar 10.1: Diagram sebar yang menunjukkan hubungan antara pendidikan dan pendapatan

Perhatikan bahwa meskipun sebagian besar titik data dikelompokkan bersama di sudut kiri bawah grafik, mereka umumnya miring ke atas dari kiri ke kanan. Juga, titik-titik menyebar daripada mengikuti garis lurus secara ketat. Jika mereka berada dalam garis lurus miring dari kiri ke kanan, nilai r akan menjadi 1, menunjukkan hubungan linier positif yang sempurna.

Memeriksa Korelasi Tambahan

Sekarang mari kita hitung koefisien korelasi untuk pasangan variabel yang tersisa menggunakan sintaks ini:

```
SELECT
      round(
            corr(median_hh_income, pct_bachelors_higher)::numeric, 2
            ) AS bachelors_income_r,

      round(
            corr(pct_travel_60_min, median_hh_income)::numeric, 2
            ) AS income_travel_r,

      round(
            corr(pct_travel_60_min, pct_bachelors_hgher)::numeric, 2
            ) AS bachelors_travel_r
FROM acs_2011_2015_stats:
```

Kali ini kita akan membuat output lebih mudah dibaca dengan membulatkan nilai desimal. Kita akan melakukan ini dengan membungkus fungsi corr(Y, X) di dalam fungsi round() SQL , yang mengambil dua input: nilai numerik yang akan dibulatkan dan nilai integer yang menunjukkan jumlah tempat desimal untuk membulatkan nilai pertama. Jika parameter kedua dihilangkan, nilainya dibulatkan ke bilangan bulat terdekat. Karena corr(Y, X) mengembalikan nilai floating-point secara default, kami akan mengubahnya ke tipe numerik menggunakan :: notasi yang Anda pelajari di penjelasan sebelumnya. Berikut hasilnya:

```
bachelors_income_r      income_travel_r   bachelor_travel_r
------------------      ---------------   -----------------
0.68                    0.05              -0.14
```

Nilai bujangan_pendapatan_r adalah 0,68, yang sama dengan lari pertama kami tetapi dibulatkan ke dua tempat desimal. Dibandingkan dengan bujangan_pendapatan_r, dua korelasi lainnya lemah. Nilai income_travel_r menunjukkan bahwa korelasi antara pendapatan dan persentase mereka yang bepergian lebih dari satu jam ke tempat kerja praktis adalah nol. Ini menunjukkan bahwa pendapatan rumah tangga rata-rata suatu daerah tidak banyak berhubungan dengan berapa lama waktu yang dibutuhkan orang untuk mulai bekerja.

Nilai bachelors_travel_r menunjukkan bahwa korelasi gelar sarjana dan komuter juga rendah yaitu -0,14. Nilai negatif menunjukkan hubungan terbalik: dengan meningkatnya pendidikan, persentase penduduk yang menempuh perjalanan lebih dari satu jam untuk bekerja menurun. Meskipun ini menarik, koefisien korelasi yang mendekati nol menunjukkan hubungan yang lemah.

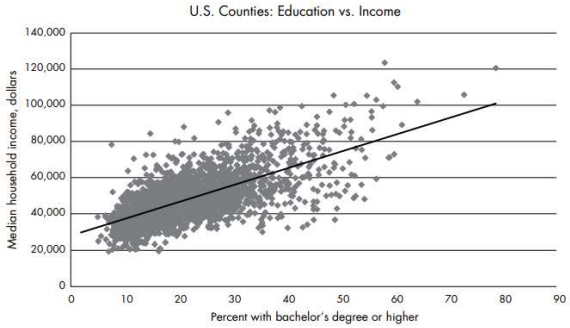
Saat menguji korelasi, kita perlu mencatat beberapa peringatan. Yang pertama adalah bahwa bahkan korelasi yang kuat tidak menyiratkan kausalitas. Kita tidak dapat mengatakan bahwa perubahan pada satu variabel menyebabkan perubahan pada variabel lainnya, hanya saja perubahan tersebut bergerak bersama. Yang kedua adalah bahwa korelasi harus diuji untuk menentukan apakah korelasi tersebut signifikan secara statistik. Tes-tes tersebut berada di luar cakupan buku ini tetapi layak untuk dipelajari sendiri.

Namun demikian, fungsi SQL corr(Y, X) adalah alat yang berguna untuk memeriksa korelasi antar variabel dengan cepat.

Memprediksi Nilai dengan Analisis Regresi

Peneliti tidak hanya ingin memahami hubungan antar variabel; mereka juga ingin memprediksi nilai menggunakan data yang tersedia. Misalnya, katakanlah 30 persen dari populasi kabupaten memiliki gelar sarjana atau lebih tinggi. Mengingat tren dalam data kami, apa yang kami harapkan dari pendapatan rumah tangga rata-rata county itu? Demikian pula, untuk setiap persen peningkatan pendidikan, berapa rata-rata peningkatan pendapatan yang kita harapkan?

Kita dapat menjawab kedua pertanyaan tersebut dengan menggunakan regresi linier. Sederhananya, metode regresi menemukan persamaan linier terbaik, atau garis lurus, yang menggambarkan hubungan antara variabel independen (seperti pendidikan) dan variabel dependen (seperti pendapatan). Standar ANSI SQL dan PostgreSQL menyertakan fungsi yang melakukan regresi linier. Pada gambar 10.2 menunjukkan sebar kami sebelumnya dengan garis regresi ditambahkan.



Gambar 10.2: Scatterplot dengan garis regresi kuadrat terkecil menunjukkan hubungan antara pendidikan dan pendapatan

Garis lurus yang melalui tengah semua titik data disebut garis regresi kuadrat terkecil, yang mendekati "paling cocok" untuk garis lurus yang paling menggambarkan hubungan antara variabel. Persamaan untuk garis regresi seperti rumus perpotongan kemiringan yang mungkin Anda ingat dari matematika sekolah menengah tetapi ditulis dengan menggunakan variabel bernama berbeda: Y = bX + a. Berikut adalah komponen rumus:

Y adalah nilai prediksi, yang juga merupakan nilai pada sumbu y, atau variabel dependen.

b adalah kemiringan garis, yang bisa positif atau negatif. Ini mengukur berapa banyak unit nilai sumbu y akan naik atau turun untuk setiap unit nilai sumbu x.

X mewakili nilai pada sumbu x, atau variabel independen.

a adalah perpotongan y, nilai di mana garis memotong sumbu y ketika nilai X adalah nol.

Mari kita terapkan rumus ini menggunakan SQL. Sebelumnya, kami mempertanyakan berapa pendapatan rumah tangga rata-rata yang diharapkan di suatu daerah jika persentase orang dengan gelar sarjana atau lebih tinggi di daerah itu adalah 30 persen.

Dalam scatterplot kami, persentase dengan gelar sarjana jatuh di sepanjang sumbu x, diwakili oleh X dalam perhitungan. Mari kita masukkan nilai itu ke dalam rumus garis regresi sebagai ganti X:

Y = b(30) + a

Untuk menghitung Y, yang merupakan prediksi pendapatan rumah tangga rata-rata, kita memerlukan kemiringan garis, b, dan perpotongan y, a. Untuk mendapatkan nilai ini, kita akan menggunakan fungsi SQL regr_slope(Y, X) dan regr_intercept(Y, X), seperti yang ditunjukkan pada sintaks dibawah ini:

```
SELECT
      round(
            regr_slope(medium_hh_income, pct_bachelors_higher)::numeric, 2
            ) AS slope,
      round(
            regr_intercept(median_hh_income, pct_bachelors_higher)::numeric, 2
            ) AS y_intercept
From ACS_2011_2015_stats;
```

Menggunakan variabel median_hh_income dan pct_bachelors_higher sebagai input untuk kedua fungsi, kita akan menetapkan nilai yang dihasilkan dari fungsi regr_slope(Y, X) sebagai kemiringan dan output untuk fungsi regr_intercept(Y, X) sebagai y_intercept.

Jalankan kueri; hasilnya harus menunjukkan yang berikut:

```
slope       y_intercept
------      -----------
926.95         27901.15
```

Nilai kemiringan menunjukkan bahwa untuk setiap peningkatan satu unit dalam persentase gelar sarjana, kita dapat mengharapkan pendapatan rumah tangga rata-rata suatu daerah akan meningkat sebesar 926,95. Kemiringan selalu mengacu pada perubahan per satu unit X

Nilai y_intercept menunjukkan bahwa ketika garis regresi memotong sumbu y, di mana persentase gelar sarjana adalah 0, nilai sumbu y adalah 27901,15. Sekarang mari kita masukkan kedua nilai ke dalam persamaan untuk mendapatkan nilai Y:

Y = 926,95(30) + 27901,15
Y = 55709,65

Berdasarkan perhitungan kami, di daerah di mana 30 persen orang berusia 25 dan lebih tua memiliki gelar sarjana atau lebih tinggi, kami dapat mengharapkan pendapatan rumah tangga rata-rata di daerah itu menjadi sekitar $55.710. Tentu saja, data kami mencakup kabupaten yang pendapatan mediannya turun di atas dan di bawah nilai prediksi tersebut, tetapi kami berharap hal ini terjadi karena titik data kami di scatterplot tidak berbaris sempurna di sepanjang garis regresi. Ingatlah bahwa koefisien korelasi yang kami hitung adalah 0,68, menunjukkan hubungan yang kuat tetapi tidak sempurna antara pendidikan dan pendapatan. Faktor-faktor lain mungkin berkontribusi terhadap variasi pendapatan juga.

Mencari Pengaruh Variabel Independen dengan r-kuadrat

Sebelumnya dalam bab ini, kita menghitung koefisien korelasi, r, untuk menentukan arah dan kekuatan hubungan antara dua variabel. Kita juga dapat menghitung sejauh mana variasi variabel x (independen) menjelaskan variasi variabel y (dependen) dengan mengkuadratkan nilai r untuk mencari koefisien determinasi, yang lebih dikenal dengan r-kuadrat. Nilai r-

kuadrat berada di antara nol dan satu dan menunjukkan persentase variasi yang dijelaskan oleh variabel bebas. Misalnya, jika r-kuadrat sama dengan 0,1, kita akan mengatakan bahwa variabel bebas menjelaskan 10 persen variasi dalam variabel terikat, atau tidak banyak sama sekali.

Untuk mencari r-kuadrat, kita menggunakan fungsi regr_r2(Y, X) dalam SQL. Mari kita terapkan pada variabel pendidikan dan pendapatan kita menggunakan sintaks dibawah ini:

```
SELECT round(
      regr_r2(median_hh_income, pct_bachelors_higher)::numeric, 3
      ) AS r_squared
FROM acs_2011_2015_stats;
```

Kali ini kita akan membulatkan output ke tempat seperseribu terdekat dan mengatur hasilnya ke r_squared. Kueri harus mengembalikan hasil berikut:

```
r_squared
---------
    0.465
```

Nilai r-kuadrat sebesar 0,465 menunjukkan bahwa sekitar 47 persen variasi median pendapatan rumah tangga di suatu daerah dapat dijelaskan oleh persentase orang dengan gelar sarjana atau lebih tinggi di daerah tersebut. Apa yang menjelaskan 53 persen lainnya dari variasi pendapatan rumah tangga? Sejumlah faktor dapat menjelaskan variasi lainnya, dan ahli statistik biasanya akan menguji banyak kombinasi variabel untuk menentukan apa itu.

Tetapi sebelum Anda menggunakan angka-angka ini dalam judul atau presentasi, ada baiknya meninjau kembali poin-poin berikut:

1. Korelasi tidak membuktikan kausalitas. Untuk verifikasi, lakukan pencarian Google pada “korelasi dan kausalitas.” Banyak variabel berkorelasi dengan baik tetapi tidak memiliki arti. (Lihat http://www.tylervigen.com/spurious-correlations untuk contoh korelasi yang tidak membuktikan kausalitas, termasuk korelasi antara tingkat perceraian di Maine dan konsumsi margarin.) Ahli statistik biasanya melakukan pengujian signifikan pada hasil untuk pastikan nilai bukan hanya hasil keacakan.
2. Ahli statistik juga menerapkan tes tambahan pada data sebelum menerima hasil analisis regresi, termasuk apakah variabel mengikuti distribusi kurva lonceng standar dan memenuhi kriteria lain untuk hasil yang valid.

Mengingat faktor-faktor ini, fungsi statistik SQL berguna sebagai survei pendahuluan terhadap data Anda sebelum melakukan analisis yang lebih teliti. Jika pekerjaan Anda melibatkan statistik, studi lengkap tentang melakukan regresi bermanfaat.

Membuat Peringkat dengan SQL

Peringkat sering menjadi berita. Anda akan melihatnya digunakan di mana saja mulai dari grafik box office akhir pekan hingga klasemen liga tim olahraga. Anda telah mempelajari cara mengurutkan hasil kueri berdasarkan nilai dalam kolom, tetapi SQL memungkinkan Anda melangkah lebih jauh dan membuat peringkat bernomor. Pemeringkatan berguna untuk analisis data dalam beberapa cara, seperti melacak perubahan dari waktu ke waktu jika Anda

memiliki data selama beberapa tahun. Anda juga dapat menggunakan peringkat sebagai fakta tersendiri dalam laporan. Mari kita jelajahi cara membuat peringkat menggunakan SQL.

Peringkat dengan rank() dan density_rank()

Standar ANSI SQL mencakup beberapa fungsi peringkat, tetapi kami hanya akan fokus pada dua: rank() dan density_rank(). Keduanya adalah fungsi jendela, yang melakukan perhitungan di seluruh rangkaian baris yang kami tentukan menggunakan klausa OVER. Tidak seperti fungsi agregat, yang mengelompokkan baris saat menghitung hasil, fungsi jendela menyajikan hasil untuk setiap baris dalam tabel.

Perbedaan antara rank() dan density_rank() adalah cara mereka menangani nilai peringkat berikutnya setelah seri: rank() menyertakan celah dalam urutan peringkat, tetapi density_rank() tidak. Konsep ini lebih mudah dipahami dalam tindakan, jadi mari kita lihat sebuah contoh. Pertimbangkan seorang analis Wall Street yang meliput pasar manufaktur widget yang sangat kompetitif. Analis ingin membuat peringkat perusahaan berdasarkan output tahunan mereka. Pernyataan SQL di bawah ini untuk membuat dan mengisi tabel dengan data ini dan kemudian memberi peringkat perusahaan berdasarkan output widget:

```
CREATE TABLE widget_companies (
      id bigserial,
      company varchar(30) NOT NULL,
      widget_output integer NOT NULL
);

INSERT INTO widget_companies (company, widget_output)
VALUES
      ('Morse Widgets', 125000),
      ('Springfield Widget Masters', 143000),
      ('Best Widgets', 196000),
      ('Acme Inc.', 133000),
      ('District Widget Inc.', 201000),
      ('Clarke Amalgamated', 620000),
      ('Stavesacre Industries', 244000),
      ('Bowers Widget Emporium', 201000);

SELECT
      company,
      widget_output,
      rank() OVER (ORDER BY widget_output DESC),
      dense_rank() OVER (ORDER BY widget_output DESC)
FROM widget_companies;
```

Perhatikan sintaks dalam pernyataan SELECT yang menyertakan rank() dan density_rank() . Setelah nama fungsi, kita menggunakan klausa OVER dan dalam tanda kurung menempatkan ekspresi yang menentukan "jendela" baris yang harus dioperasikan oleh fungsi. Dalam hal ini, kami ingin kedua fungsi berfungsi di semua baris kolom widget_output, diurutkan dalam urutan menurun. Berikut outputnya:

```
company                    widget_output     rank  dense_rank
-------------------------  -------------     ----  ----------
Clarke Amalgamated           620000      1     1
Stavesacre Industries        244000      2     2
Bowers Widget Emporium       201000      3     3
```

```
District Widget Inc.          201000      3     3
Best Widgets                  196000      5     4
Springfield Widget Masters    143000      6     5
Acme Inc.                     133000      7     6
Morse Widgets                 125000      8     7
```

Kolom yang dihasilkan oleh fungsi rank() dan density_rank() menunjukkan peringkat masing-masing perusahaan berdasarkan nilai widget_output dari tertinggi ke terendah, dengan Clarke Amalgamated di nomor satu. Untuk melihat perbedaan rank() dan density_rank(), periksa daftar baris kelima, Widget Terbaik.

Dengan rank(), Best Widgets adalah perusahaan dengan peringkat tertinggi kelima, menunjukkan ada empat perusahaan dengan output lebih banyak dan tidak ada perusahaan yang menempati peringkat keempat, karena rank() memungkinkan adanya kesenjangan urutan ketika terjadi seri. Sebaliknya, density_rank(), yang tidak memungkinkan kesenjangan dalam urutan peringkat, mencerminkan fakta bahwa Widget Terbaik memiliki jumlah output tertinggi keempat terlepas dari berapa banyak perusahaan yang memproduksi lebih banyak. Oleh karena itu, Widget Terbaik menempati peringkat keempat dengan menggunakan density_rank().

Kedua cara menangani ikatan memiliki kelebihan, tetapi dalam praktiknya rank() paling sering digunakan. Itu juga yang saya sarankan untuk digunakan, karena lebih akurat mencerminkan jumlah total perusahaan yang diperingkat, ditunjukkan oleh fakta bahwa Widget Terbaik memiliki empat perusahaan di depannya dalam total output, bukan tiga.
Mari kita lihat contoh peringkat yang lebih kompleks.

Peringkat Dalam Subgrup dengan PARTITION BY

Pemeringkatan yang baru saja kita lakukan adalah pemeringkatan sederhana secara keseluruhan berdasarkan keluaran widget. Tetapi terkadang Anda ingin menghasilkan peringkat dalam kelompok baris dalam sebuah tabel. Misalnya, Anda mungkin ingin memberi peringkat pegawai pemerintah berdasarkan gaji di setiap departemen atau memberi peringkat film berdasarkan pendapatan box office dalam setiap genre.

Untuk menggunakan fungsi jendela dengan cara ini, kita akan menambahkan PARTITION BY ke klausa OVER. Klausa PARTITION BY membagi baris tabel menurut nilai dalam kolom yang kita tentukan.

Berikut adalah contoh menggunakan data yang dibuat-buat tentang toko kelontong. Masukkan kode pada listing dibawah ini untuk mengisi tabel bernama store_sales:

```
CREATE TABLE store_sales (
      store varchar(30),
      category varchar(30) NOT NULL,
      unit_sales bigint NOT NULL,
      CONSTRAINT store_category_key PRIMARY KEY (store, category);

INSERT INTO store_sales (store, category, unit_sales)
VALUES
      ('Broders', 'Cereal', 1104),
      ('Wallace', 'Ice Cream', 1863),
```

```
      ('Broders', 'Ice Cream', 2517),
      ('Cramers', 'Ice Cream', 2112),
      ('Broders', 'Beer', 641),
      ('Cramers', 'Cereal', 1003),
      ('Cramers', 'Beer', 640),
      ('Wallace', 'Cereal', 980),
      ('Wallace', 'Beer', 988);

SELECT
      category,
      store,
      unit_sales,
      rank() OVER (PARTITION BY category ORDER BY unit_sales DESC)
FROM store_sales;
```

Dalam tabel, setiap baris menyertakan kategori produk toko dan penjualan untuk kategori tersebut. Pernyataan SELECT terakhir membuat kumpulan hasil yang menunjukkan bagaimana peringkat penjualan setiap toko dalam setiap kategori. Elemen baru adalah penambahan PARTITION BY pada klausa OVER. Akibatnya, klausa tersebut memberi tahu program untuk membuat peringkat satu kategori pada satu waktu, menggunakan penjualan unit toko dalam urutan menurun. Berikut outputnya:

```
company     store       unit_sales  rank
-------     -------     ----------  ----
Beer        Wallace     988         1
Beer        Broders     641         2
Beer        Creamers    640         3
Cereal      Broders     1104        1
Cereal      Cramers     1003        2
Cereal      Wallace     980         3
Ice Cream   Broders     2517        1
Ice Cream   Cramers     2112        2
Ice Cream   Wallace     1863        3
```

Perhatikan bahwa nama kategori diurutkan dan dikelompokkan dalam kolom kategori sebagai hasil dari PARTITION BY dalam klausa OVER. Baris untuk setiap kategori diurutkan berdasarkan penjualan unit kategori dengan kolom peringkat menampilkan peringkat.

Dengan menggunakan tabel ini, kita dapat melihat sekilas bagaimana peringkat setiap toko dalam kategori makanan. Misalnya, Broders menduduki puncak penjualan untuk sereal dan es krim, tetapi Wallace menang dalam kategori bir. Anda dapat menerapkan konsep ini ke banyak skenario lain: misalnya, untuk setiap produsen mobil, menemukan kendaraan yang paling banyak dikeluhkan konsumen; mencari tahu bulan mana yang memiliki curah hujan paling banyak dalam 20 tahun terakhir; menemukan tim dengan kemenangan terbanyak melawan pelempar kidal; dan seterusnya.

SQL menawarkan fungsi jendela tambahan. Periksa dokumentasi PostgreSQL resmi di https://www.postgresql.org/docs/current/static/tutorial-window.html untuk ikhtisar fungsi jendela, dan periksa https://www.postgresql.org/docs/current/static/functions-window.html untuk daftar fungsi jendela.

Menghitung Hasil untuk Perbandingan

Meskipun bermanfaat dan menarik, peringkat berdasarkan jumlah mentah tidak selalu berarti; pada kenyataannya, mereka sebenarnya bisa menyesatkan. Pertimbangkan contoh statistik kejahatan ini: menurut Biro Investigasi Federal AS (FBI), pada tahun 2015, Kota New York melaporkan sekitar 130.000 kejahatan properti, yang meliputi perampokan, pencurian, pencurian kendaraan bermotor, dan pembakaran. Sementara itu, Chicago melaporkan sekitar 80.000 kejahatan properti pada tahun yang sama.

Jadi, Anda lebih mungkin menemukan masalah di New York City, bukan? Belum tentu. Pada 2015, New York City memiliki lebih dari 8 juta penduduk, sedangkan Chicago memiliki 2,7 juta. Mengingat konteks itu, hanya membandingkan jumlah total kejahatan properti di kedua kota tidak terlalu berarti.

Cara yang lebih akurat untuk membandingkan angka-angka ini adalah dengan mengubahnya menjadi tarif. Analis sering menghitung tarif per 1.000 orang, atau beberapa kelipatan dari angka itu, untuk perbandingan apel-ke-apel. Untuk kejahatan properti dalam contoh ini, perhitungannya sederhana: bagi jumlah pelanggaran dengan populasi dan kemudian kalikan hasil bagi itu dengan 1.000. Misalnya, jika sebuah kota memiliki 80 pencurian kendaraan dan berpenduduk 15.000, Anda dapat menghitung tingkat pencurian kendaraan per 1.000 orang sebagai berikut:

(80 / 15.000) × 1.000 = 5,3 pencurian kendaraan per seribu penduduk

Ini adalah matematika yang mudah dengan SQL, jadi mari kita coba menggunakan data tingkat kota terpilih I dikompilasi dari laporan Kejahatan 2015 di Amerika Serikat FBI tersedia di https://ucr.fbi.gov/crime-in-the-u.s/2015/crime-in-the-u.s.-2015/home. Daftar 10-8 berisi kode untuk membuat dan mengisi tabel. Ingatlah untuk mengarahkan skrip ke lokasi tempat Anda menyimpan file CSV, yang dapat Anda unduh di https://www.nostarch.com/practicalSQL/.

```
CREATE TABLE fbi_crime_data_2015 (
      st varchar(20),
      city varchar(50),
      population integer,
      violent_crime integer,
      property_crime integer,
      burglary integer,
      larceny_theft integer,
      motor_vehicle_theft integer,
      CONSTRAINT st_city_key PRIMARY KEY (st, city)
);

COPY fbi_crime_data_2015
FROM 'C:\YourDirectory\fbi_crime_data_2015.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');

SELECT * FROM fbi_crime_data_2005
ORDER BY population DESC;
```

Tabel fbi_crime_data_2015 mencakup negara bagian, nama kota, dan populasi untuk kota tersebut. Berikutnya adalah jumlah kejahatan yang dilaporkan oleh polisi dalam kategori, termasuk kejahatan kekerasan, pencurian kendaraan, dan kejahatan properti. Untuk menghitung kejahatan properti per 1.000 orang di kota-kota dengan lebih dari 500.000 orang dan memesannya, kami akan menggunakan kode ini:

```
SELECT
      city,
      st,
      population,
      property_crime,
      round(
            (property_crime::numeric / population) * 1000, 1) AS pc_per_1000
FROM fbi_crime_data_2015
WHERE population >= 500000
ORDER BY (property_crime::numeric / popyation) DESC;
```

Dalam Bab 5, Anda mempelajari bahwa ketika membagi bilangan bulat dengan bilangan bulat, salah satu nilainya harus berupa angka atau desimal agar hasilnya menyertakan tempat desimal. Kami melakukannya dalam perhitungan tarif dengan singkatan titik dua PostgreSQL. Karena kita tidak perlu melihat banyak tempat desimal, kita membungkus pernyataan dalam fungsi round() untuk membulatkan output ke persepuluhan terdekat. Kemudian kami memberikan kolom terhitung alias pc_per_1000 untuk referensi mudah. Berikut sebagian dari kumpulan hasil:

city	**st**	**population**	**property_crime**	**pc_per_1000**
Tucson	Arizona	529675	35185	66.4
San Francisco	California	863782	53019	61.4
Albuquerque	New Mexico	559721	33993	60.7
Memphis	Tennessee	657936	37047	56.3
Seattle	Washington	683700	37754	55.2
--snip--				
El Paso	Texas	686077	13133	19.1
New York	New York	8550861	129860	15.2

Tucson, Arizona, memiliki tingkat kejahatan properti tertinggi, diikuti oleh San Francisco, California. Di bagian bawah adalah Kota New York, dengan tarif seperempat dari Tucson. Jika kami membandingkan kota-kota hanya berdasarkan jumlah kejahatan properti, kami akan mendapatkan hasil yang jauh berbeda dari yang kami peroleh dengan menghitung tarif per seribu.

Saya akan lalai untuk tidak menunjukkan bahwa situs web FBI di https://ucr.fbi.gov/ucr-statistics-their-proper-use/ tidak menganjurkan pembuatan peringkat dari data kejahatannya, yang menyatakan bahwa hal itu menciptakan “persepsi yang menyesatkan yang merugikan entitas geografis dan penduduknya.” Mereka menunjukkan bahwa variasi dalam kejahatan dan tingkat kejahatan di seluruh negeri seringkali disebabkan oleh sejumlah faktor mulai dari kepadatan penduduk hingga kondisi ekonomi dan bahkan iklim. Selain itu, data kejahatan FBI memiliki kekurangan yang terdokumentasi dengan baik, termasuk pelaporan yang tidak lengkap oleh lembaga kepolisian.

Oleh karena itu, menanyakan mengapa suatu daerah memiliki tingkat kejahatan yang lebih tinggi atau lebih rendah daripada yang lain masih layak dikejar, dan tingkat memang memberikan beberapa ukuran perbandingan meskipun ada batasan tertentu.

Itu mengakhiri eksplorasi kami tentang fungsi statistik dalam SQL, peringkat, dan tarif. Sekarang toolkit analisis SQL Anda mencakup cara untuk menemukan hubungan antar variabel menggunakan fungsi statistik, membuat peringkat dari data yang dipesan, dan membandingkan angka mentah dengan benar dengan mengubahnya menjadi tarif. Toolkit itu mulai terlihat mengesankan!

Selanjutnya, kita akan lebih mendalami data tanggal dan waktu, menggunakan fungsi SQL untuk mengekstrak informasi yang kita butuhkan.

Latihan Soal

Uji keterampilan baru Anda dengan pertanyaan-pertanyaan berikut:

1. Koefisien korelasi, atau nilai r, dari variabel pct_sarjana_tinggi dan pendapatan_hh_median adalah sekitar 0,68. Tulis kueri menggunakan kumpulan data yang sama untuk menunjukkan korelasi antara pct_masters_higher dan median_hh_income. Apakah nilai r lebih tinggi atau lebih rendah? Apa yang mungkin menjelaskan perbedaannya?
2. Dalam data kejahatan FBI, kota manakah yang berpenduduk 500.000 atau lebih yang memiliki tingkat pencurian kendaraan bermotor (kolom motor_vehicle_theft) tertinggi? Mana yang memiliki tingkat kejahatan kekerasan tertinggi (kolom kekerasan_kejahatan)?
3. Sebagai tantangan bonus, kunjungi kembali data perpustakaan di tabel pls_fy2014_pupld14a di Bab 8. Beri peringkat lembaga perpustakaan berdasarkan tingkat kunjungan per 1.000 penduduk (kolom popu_lsa), dan batasi kueri ke lembaga yang melayani 250.000 orang atau lebih.

BAB XI
KUERI TANGGAL DAN WAKTU

Kolom yang diisi dengan tanggal dan waktu dapat menunjukkan kapan peristiwa terjadi atau berapa lama, dan itu dapat mengarah pada garis penyelidikan yang menarik. Pola apa yang ada pada momen pada garis waktu? Peristiwa mana yang terpendek atau terpanjang? Hubungan apa yang ada antara par --aktivitas tertentu dan waktu hari atau musim terjadinya?

Dalam bab ini, kita akan menjelajahi jenis pertanyaan ini menggunakan tipe data SQL untuk tanggal dan waktu dan fungsi terkaitnya. Kita akan mulai dengan melihat lebih dekat tipe data dan fungsi yang terkait dengan tanggal dan waktu. Kemudian kita akan menjelajahi Kami juga akan menjelajahi zona waktu menggunakan data Amtrak untuk menghitung durasi perjalanan kereta api di seluruh kumpulan data yang berisi informasi tentang perjalanan taksi Kota New York untuk mencari pola dan mencoba menemukan apa, jika ada, kisah yang diceritakan oleh data tersebut. Serikat.

Tipe dan Fungsi Data untuk Tanggal dan Waktu

Bab 3 mengeksplorasi tipe data SQL utama, tetapi untuk meninjau, berikut adalah empat tipe data yang terkait dengan tanggal dan waktu: tanggal Hanya mencatat tanggal PostgreSQL menerima beberapa format tanggal Misalnya, format yang valid untuk menambahkan hari ke-21 September 2018 adalah 21 September 2018 atau 21/9/2018. Saya sarankan menggunakan YYYY-MM-DD (atau 21-09-2018), yang merupakan format standar internasional ISO 8601 dan juga keluaran tanggal PostgreSQL default. Menggunakan format ISO membantu menghindari kebingungan saat berbagi data secara internasional.

Format ISO 8601 adalah HH: MM: SS, di mana HH mewakili jam, MM menit, dan SS detik. Anda dapat menambahkan penunjuk zona waktu opsional. Misalnya, 14:24 di San Francisco selama waktu standar di musim gugur dan musim dingin adalah 14:24 PST.

Timestamp merekam tanggal dan waktu. Anda dapat menambahkan dengan zona waktu untuk membuat zona waktu kolom sadar. Format timestamp dengan zona waktu adalah bagian dari standar SQL, tetapi dengan PostgreSQL, Anda dapat menggunakan timestamptz singkatan, yang menggabungkan tanggal dan format waktu ditambah penanda zona waktu di akhir: YYYY-MM-DD HH: MM: SS TZ. Anda dapat menentukan zona waktu dalam tiga format berbeda: offset UTC, penanda area / lokasi, atau singkatan standar.

Itu tidak mencatat awal atau akhir suatu periode, hanya durasinya.Contohnya termasuk 12 hari atau 8 jam.

Tiga tipe data pertama, tanggal, waktu, dan cap waktu, dikenal sebagai tipe datetime yang nilainya disebut datetimes. Nilai interval adalah tipe interval yang nilainya adalah interval. Keempat tipe data dapat melacak jam sistem dan nuansa kalender. Misalnya, tanggal dan stempel waktu mengakui bahwa Juni memiliki 30 hari. Oleh karena itu, 31 Juni adalah nilai

tanggal waktu yang tidak valid yang menyebabkan database membuang kesalahan. Demikian pula, tanggal 29 Februari hanya valid di tahun kabisat, seperti 2020 ..

Mengubah Tanggal dan Waktu

Kita dapat menggunakan fungsi SQL untuk melakukan perhitungan pada tanggal dan waktu atau mengekstrak komponen darinya. Misalnya, kita dapat mengambil hari dalam seminggu dari stempel waktu atau mengekstrak hanya bulan dari tanggal. ANSI SQL menguraikan beberapa fungsi yang harus dilakukan ini, tetapi banyak manajer database (termasuk MySQL dan Microsoft SQL Server) menyimpang dari standar untuk menerapkan tipe data tanggal dan waktu, sintaks, dan nama fungsi mereka sendiri.Jika Anda menggunakan database selain PostgreSQL, periksa dokumentasinya.

Mari kita tinjau cara memanipulasi tanggal dan waktu menggunakan fungsi PostgreSQL.

Mengekstrak Komponen dari Nilai stempel waktu

Bukan hal yang aneh jika hanya membutuhkan satu bagian dari nilai tanggal atau waktu untuk analisis, terutama saat Anda menggabungkan hasil berdasarkan bulan, tahun, atau bahkan menit. Kita dapat mengekstrak komponen ini menggunakan fungsi PostgreSQL date_part (). Formatnya terlihat seperti ini:

```
date_part(text, value)
```

Fungsi mengambil dua input. Yang pertama adalah string dalam format teks yang mewakili bagian dari tanggal atau waktu untuk mengekstrak, seperti jam, menit, atau minggu. Yang kedua adalah nilai tanggal, waktu, atau stempel waktu. Untuk melihat date_part () berfungsi, kami akan mengeksekusinya beberapa kali pada nilai yang sama menggunakan sintaks dibawah ini. Dalam sintaks ini, kami memformat string sebagai stempel waktu dengan zona waktu menggunakan stempel waktu steno khusus PostgreSQL. Kami juga menetapkan nama kolom untuk masing-masing dengan AS.

```
SELECT
      date_part('year', '2019-12-01 18:37:12 EST'::timestamptz) AS "year",
      date_part('month', '2019-12-01 18:37:12 EST'::timestamptz) AS "month",
      date_part('day', '2019-12-01 18:37:12 EST'::timestamptz) AS "day",
      date_part('hour', '2019-12-01 18:37:12 EST'::timestamptz) AS "hour",
      date_part('minute', '2019-12-01 18:37:12 EST'::timestamptz) AS "minute",
      date_part('second', '2019-12-01 18:37:12 EST'::timestamptz) AS "second",
      date_part('timezone_hour', '2019-12-01 18:37:12 EST'::timestamptz) AS "tz",
      date_part('week', '2019-12-01 18:37:12 EST'::timestamptz) AS "week",
      date_part('quarter', '2019-12-01 18:37:12 EST'::timestamptz) AS "quarter",
      date_part('epoch', '2019-12-01 18:37:12 EST'::timestamptz) AS "epoch";
```

Setiap pernyataan kolom dalam kueri SELECT ini pertama-tama menggunakan string untuk menamai komponen yang ingin kita ekstrak: tahun, bulan, hari, dan seterusnya.Input kedua menggunakan string 2019-12-01 18:37:12 EST dilemparkan sebagai timestamp dengan zona waktu dengan sintaks titik dua PostgreSQL dan steno timestamptz.

Pada bulan Desember, Amerika Serikat mengamati waktu standar, itulah sebabnya kami dapat menentukan zona waktu Timur menggunakan penunjukan Waktu Standar Timur (EST).

Inilah output seperti yang ditunjukkan di komputer saya, yang terletak di Zona waktu Timur AS (Basis data mengonversi nilai untuk mencerminkan pengaturan zona waktu PostgreSQL Anda, sehingga output Anda mungkin berbeda; misalnya, jika diatur ke zona waktu Pasifik AS, jam akan ditampilkan sebagai):

```
year  month day  hour  minute     seconds     tz    week  quarter     epoch
----  ----- ---  ----  ------     -------     --    ----  -------     ----------
2019  12    1    18    37         12          -5    48    4           1575243432
```

Setiap kolom berisi satu nilai yang mewakili 18:37:12 pada 1 Desember 2019, di zona waktu Timur AS. Meskipun Anda menetapkan zona waktu menggunakan EST dalam string, PostgreSQL melaporkan kembali offset UTC dari itu zona waktu, yaitu jumlah jam plus atau minus dari UTC. UTC mengacu pada Waktu Universal Terkoordinasi, standar waktu dunia, serta nilai UTC +/- 00: 00, zona waktu yang mencakup Amerika Kerajaan dan Afrika Barat Di sini, offset UTC adalah -5 (karena EST lima jam di belakang UTC).

Catatan: Anda dapat memperoleh offset UTC dari zona waktu tetapi tidak sebaliknya. Setiap offset UTC dapat merujuk ke beberapa zona waktu bernama ditambah varian waktu musim panas dan standar.

Tujuh nilai pertama mudah dikenali dari stempel waktu asli, tetapi tiga yang terakhir adalah nilai yang dihitung yang layak mendapat penjelasan.

Kolom minggu menunjukkan tanggal 1 Desember 2019, jatuh pada minggu ke 48 dalam setahun. Angka ini ditentukan oleh standar ISO 8601, yang dimulai setiap minggu pada hari Senin. Artinya, satu minggu di akhir tahun dapat diperpanjang dari Desember memasuki bulan Januari tahun berikutnya.

Kolom epoch menunjukkan pengukuran, yang digunakan dalam sistem komputer dan bahasa pemrograman, yang mewakili jumlah detik yang berlalu sebelum atau setelah pukul 12 pagi, 1 Januari 1970, pada UTC 0.
Nilai positif menunjukkan waktu sejak titik itu; nilai negatif menunjukkan waktu sebelumnya. Dalam contoh ini, 1.575.243.432 detik berlalu antara 1 Januari 1970, dan stempel waktu. Epoch berguna jika Anda perlu membandingkan dua stempel waktu secara matematis pada skala mutlak.
PostgreSQL juga mendukung fungsi SQL-standard extract(), yang memparsing datetimes dengan cara yang sama seperti fungsi date_part(). Saya telah menampilkan date_part() di sini bukan karena dua alasan. Pertama, namanya membantu mengingatkan kita apa fungsinya. Kedua, ekstrak () tidak didukung secara luas oleh manajer basis data. Terutama, itu tidak ada di SQL Server Microsoft. Namun, jika Anda perlu menggunakan ekstrak (), sintaksnya mengambil bentuk ini:

```
extract(text from value)
```

Untuk mereplikasi contoh date_part () pertama di pada sintaks di mana kami menarik tahun dari stempel waktu, kami akan menyiapkan fungsi seperti ini:

```
extract('year' from '2019-12-01 18:37:12 EST'::timestamptz)
```

PostgreSQL menyediakan komponen tambahan yang dapat Anda ekstrak atau hitung dari tanggal dan waktu. Untuk daftar lengkap fungsi, lihat dokumentasi di https://www.postgresql.org/docs/current/static/functions-datetime.html ..

Membuat Nilai Datetime dari Komponen timestamp

Bukan hal yang aneh untuk menemukan kumpulan data di mana tahun, bulan, dan hari ada di kolom terpisah, dan Anda mungkin ingin membuat nilai waktu-tanggal dari komponen ini.Untuk melakukan perhitungan pada tanggal, ada baiknya menggabungkan dan memformatnya. potongan dengan benar menjadi satu kolom.

Anda dapat menggunakan fungsi PostgreSQL berikut untuk membuat objek datetime:
make_date (tahun, bulan, hari) Mengembalikan nilai tipe tanggal
make_time (jam, menit, detik) Mengembalikan nilai tipe waktu tanpa zona waktu
make_timestamptz (tahun, bulan, hari, jam, menit, detik, zona waktu) Mengembalikan stempel waktu dengan zona waktu

Variabel untuk ketiga fungsi ini mengambil tipe integer sebagai input, dengan dua pengecualian: detik adalah tipe presisi ganda karena Anda dapat menyediakan pecahan detik, dan zona waktu harus ditentukan dengan string teks yang memberi nama zona waktu.

Listing ini menunjukkan contoh dari tiga fungsi yang sedang beraksi menggunakan komponen 22 Februari 2018, untuk tanggal, dan 6: 04:30.3 PM di Lisbon, Portugal untuk waktu:

```
SELECT make_date(2018, 2, 22);
SELECT make_time(18, 4, 30.3);
SELECT make_timestamptz(2008, 2, 22, 18, 4, 30.3, 'Europe?Lisbon');
```

Saat saya menjalankan setiap kueri secara berurutan, output di komputer saya di zona waktu Timur AS adalah sebagai berikut. Sekali lagi, Anda mungkin berbeda tergantung pada setelan zona waktu Anda:

```
2018-02-22
18:04:30.3
2018-02-22 13:04:30.3-05
```

Perhatikan bahwa stempel waktu di baris ketiga menunjukkan 13:04:30.3, yang merupakan Waktu Standar Timur dan lima jam di belakang (-05) waktu yang dimasukkan ke fungsi: 18:04:30.3 Dalam diskusi kita tentang zona waktu–diaktifkan kolom "Tanggal dan Waktu", saya mencatat bahwa PostgreSQL menampilkan waktu relatif terhadap zona waktu klien atau zona waktu yang ditetapkan dalam sesi database. Output ini mencerminkan waktu yang tepat karena lokasi saya lima jam di belakang Lisbon. Kami akan mengeksplorasi bekerja dengan zona

waktu secara lebih rinci, dan Anda akan belajar menyesuaikan tampilannya di “Bekerja dengan Zona Waktu”.

Mengambil Tanggal dan Waktu Saat Ini

Jika Anda perlu merekam tanggal atau waktu saat ini sebagai bagian dari kueri—saat memperbarui baris, misalnya—SQL standar juga menyediakan fungsi untuk itu.Fungsi berikut mencatat waktu sejak kueri dimulai:

current_date Mengembalikan tanggal.

current_time Mengembalikan waktu saat ini dengan zona waktu.

current_timestamp Mengembalikan stempel waktu saat ini dengan zona waktu Singkatan versi khusus PostgreSQL sekarang ().

localtime Mengembalikan waktu saat ini tanpa zona waktu.

localtimestamp Mengembalikan stempel waktu saat ini tanpa zona waktu.

Karena fungsi ini mencatat waktu di awal kueri (atau kumpulan kueri yang dikelompokkan dalam transaksi, yang saya bahas di Bab 9), mereka akan menyediakan waktu yang sama selama eksekusi kueri terlepas dari berapa lama kueri Jadi, jika kueri Anda memperbarui 100.000 baris dan membutuhkan waktu 15 detik untuk dijalankan, setiap stempel waktu yang direkam di awal kueri akan diterapkan ke setiap baris, sehingga setiap baris akan menerima stempel waktu yang sama.

Sebaliknya, jika Anda ingin tanggal dan waktu mencerminkan bagaimana jam berubah selama eksekusi kueri, Anda dapat menggunakan fungsi clock_timestamp() khusus PostgreSQL untuk merekam waktu saat ini saat berlalu. memperbarui 100.000 baris dan memasukkan stempel waktu setiap kali, setiap baris mendapatkan waktu pembaruan baris daripada waktu di awal kueri. Perhatikan bahwa clock_timestamp () dapat memperlambat kueri besar dan mungkin tunduk pada batasan sistem.

Sintaks dibawah ini menunjukkan current_timestamp dan clock_timestamp() beraksi saat menyisipkan baris dalam tabel:

```
CREATE TABLE current_time_example (
      time_id bigserial,
      current_timestamp_col timestamp with time zone,
      clock_timestamp_col timestamp with time zone);

  INSERT INTO current_time_example (current_timestamp_col, clock_timestamp_col)
      (SELECT current_timestamp,
            clock_timestamp()
      FROM generate_series(1,1000));

  SELECT * FROM current_time_example;
```

Kode membuat tabel yang menyertakan dua kolom cap waktu dengan zona waktu. Yang pertama menyimpan hasil fungsi current_timestamp , yang mencatat waktu di awal pernyataan INSERT yang menambahkan 1.000 baris ke tabel. Untuk melakukannya, kita gunakan fungsi generate_series (), yang mengembalikan sekumpulan bilangan bulat yang dimulai dengan 1 dan diakhiri dengan 1.000. Kolom kedua berisi hasil dari fungsi clock_timestamp () , yang mencatat waktu penyisipan setiap baris. Anda memanggil kedua fungsi sebagai bagian dari pernyataan INSERT Jalankan kueri, dan hasil dari pernyataan SELECT

akhir akan menunjukkan bahwa waktu di current_timestamp_col adalah sama untuk semua baris, terutama waktu di clock_timestamp_col meningkat dengan setiap baris dimasukkan.

Bekerja dengan Zona Waktu

Data zona waktu memungkinkan tanggal dan waktu di database Anda mencerminkan lokasi di seluruh dunia tempat tanggal dan waktu tersebut berlaku dan offset UTC-nya. Stempel waktu 1 PM hanya berguna, misalnya, jika Anda tahu apakah nilainya mengacu pada waktu lokal di Asia, Eropa Timur, salah satu dari 12 zona waktu Antartika, atau di mana pun di dunia.

Tentu saja, sangat sering Anda akan menerima kumpulan data yang tidak berisi data zona waktu di kolom datetime-nya. Ini tidak selalu menjadi pemecah masalah dalam hal apakah Anda harus terus menggunakan data atau tidak. Jika Anda tahu bahwa setiap peristiwa dalam data yang terjadi di lokasi yang sama, memiliki zona waktu di stempel waktu kurang penting, dan relatif mudah untuk mengubah semua stempel waktu data Anda untuk mencerminkan satu zona waktu itu.
Mari kita lihat beberapa strategi untuk bekerja dengan zona waktu di data Anda.

Menemukan Pengaturan Zona Waktu Anda

Saat bekerja dengan zona waktu di SQL, pertama-tama Anda perlu mengetahui pengaturan zona waktu untuk server database Anda. Jika Anda menginstal PostgreSQL di komputer Anda sendiri, defaultnya adalah zona waktu lokal Anda. Jika Anda terhubung ke database PostgreSQL di tempat lain, Untuk membantu menghindari kebingungan, administrator database sering menyetel zona waktu server bersama ke UTC. Untuk mengetahui zona waktu default server PostgreSQL Anda, gunakan perintah SHOW dengan zona waktu, seperti yang ditunjukkan ini:

```
SHOW timezone;
```

Memasukkan sintaks ke pgAdmin dan menjalankannya di komputer saya mengembalikan US / Eastern, salah satu dari beberapa nama lokasi yang termasuk dalam zona waktu Timur, yaitu Kanada timur dan Amerika Serikat, Karibia, dan sebagian Meksiko.

Catatan: Anda dapat menggunakan SHOW ALL; untuk melihat pengaturan setiap parameter di server PostgreSQL Anda.

Anda juga dapat menggunakan dua listing ini untuk mencantumkan semua nama zona waktu, singkatan, dan offset UTC-nya:

```
SELECT * FROM pg_timezone_abbrevs;
SELECT * FROM pg_timezone_names;
```

Anda dapat dengan mudah memfilter salah satu dari pernyataan SELECT ini dengan klausa WHERE untuk mencari nama lokasi atau zona waktu tertentu:

```
SELECT * FROM pg_timezone_names
WHERE name LIKE ‘Europe%’;
```

Kode ini harus mengembalikan daftar tabel yang menyertakan nama zona waktu, singkatan, offset UTC, dan kolom boolean is_dst yang mencatat apakah zona waktu saat ini mengamati waktu musim panas:

```
 name                 abbrev     utc_offset       is_dst
----------------      ------     ----------       ------
Europe/Amsterdam      CEST       02:00:00         t
Europe/Andorra        CEST       02:00:00         t
Europe/Astrakhan      +04        04:00:00         f
Europe/Athens         EEST       03:00:00         t
Europe/Belfast        BST        01:00:00         t
--snip--
```

Ini adalah cara yang lebih cepat untuk mencari zona waktu daripada menggunakan Wikipedia Sekarang mari kita lihat cara mengatur zona waktu ke nilai tertentu.

Mengatur Zona Waktu

Saat Anda menginstal PostgreSQL, zona waktu default server ditetapkan sebagai parameter di postgresql.conf, file yang berisi lusinan nilai yang dibaca oleh PostgreSQL setiap kali dimulai. Lokasi postgresql.conf di sistem file Anda bervariasi tergantung pada sistem operasi Anda dan terkadang saat Anda menginstal PostgreSQL. Untuk membuat perubahan permanen pada postgresql.conf, Anda perlu mengedit file dan memulai ulang server, yang mungkin tidak mungkin dilakukan jika Anda bukan pemilik mesin. Perubahan pada configurasi mungkin juga memiliki konsekuensi yang tidak diinginkan bagi pengguna atau aplikasi lain.

Untuk mengatur dan mengubah zona waktu klien pgAdmin, kami menggunakan perintah SET zona waktu TO, seperti yang ditunjukkan pada sintkas berikut ini:

```
SET timezone TO ‘US/Pacific’;

CREATE TABLE time_zone_test (
      test_date timestamp with time zone );

INSERT INTO time_zone_test VALUES (‘2020-01-01 4:00’);

SELECT test_date
FROM time_zone_test;

SET timezone TO ‘US/Eastern’;

SELECT test_date AT TIME ZONE ‘Asia/Seoul’
FROM time_zone_test
```

Pertama, kita atur zona waktu menjadi US/Pasifik , yang menunjukkan zona waktu Pasifik yang meliputi Kanada bagian barat dan Amerika Serikat beserta Baja California di Meksiko.Kedua, kita membuat tabel satu kolom dengan tipe data timestamp Dengan zona waktu dan sisipkan satu baris untuk menampilkan hasil tes. Perhatikan bahwa nilai yang dimasukkan, 01-01-2020 4:00, adalah stempel waktu tanpa zona waktu . Anda akan sering menemukan stempel waktu tanpa zona waktu, terutama ketika Anda memperoleh kumpulan data yang terbatas pada lokasi tertentu.

Saat dijalankan, pernyataan SELECT pertama mengembalikan 01-01-2020 4:00 sebagai stempel waktu yang sekarang berisi data zona waktu:

```
test_date
---------------------
2020-01-01 04:00:00-08
```

Ingat dari diskusi kita tentang tipe data di Bab 3 bahwa -08 di akhir stempel waktu ini adalah offset UTC. Dalam hal ini, -08 menunjukkan bahwa zona waktu Pasifik delapan jam di belakang UTC. Karena kita awalnya menyetel pgAdmin zona waktu klien ke AS / Pasifik untuk sesi ini, nilai apa pun yang sekarang kita masukkan ke dalam kolom yang sadar zona waktu akan berada dalam waktu Pasifik dan diberi kode yang sesuai. Namun, perlu dicatat bahwa di server, stempel waktu dengan tipe data zona waktu selalu menyimpan data sebagai UTC secara internal; pengaturan zona waktu mengatur bagaimana data itu ditampilkan.

Sekarang saatnya bersenang-senang. Kami mengubah zona waktu untuk sesi ini ke zona waktu Timur menggunakan perintah SET dan tindakan desain AS / Timur. Kemudian, ketika kami menjalankan pernyataan SELECT lagi, hasilnya akan seperti berikut:

```
test_date
---------------------
2020-01-01 07:00:00-05
```

Dalam contoh ini, dua komponen stempel waktu telah berubah: waktu sekarang 07:00, dan offset UTC adalah -05 karena kita melihat stempel waktu dari perspektif zona waktu Timur: 4 AM Pasifik adalah 7 AM Eastern Nilai waktu Pasifik asli tetap tidak berubah dalam tabel, dan database mengonversinya untuk menunjukkan waktu di zona waktu apa pun yang kami tetapkan.

Yang lebih nyaman lagi adalah kita dapat melihat stempel waktu melalui lensa zona waktu mana pun tanpa mengubah pengaturan sesi. Pernyataan SELECT terakhir menggunakan kata kunci AT TIME ZONE untuk menampilkan stempel waktu di sesi kita sebagai waktu standar Korea (KST) dengan menyebutkan Asia/Seoul:

```
test_date
------------------
2020-01-01 21:00:00
```

Sekarang kita tahu bahwa nilai database jam 4 pagi di AS/Pasifik pada tanggal 1 Januari 2020 sama dengan jam 9 malam di hari yang sama di Asia/Seoul. Sekali lagi, sintaks ini mengubah tipe data keluaran, tetapi data di server tetap Jika nilai asli adalah stempel waktu dengan zona waktu, output menghapus zona waktu. Jika nilai asli tidak memiliki zona waktu, output adalah stempel waktu dengan zona waktu.

Kemampuan database untuk melacak zona waktu sangat penting untuk perhitungan interval yang akurat, seperti yang akan Anda lihat selanjutnya.

Perhitungan dengan Tanggal dan Waktu

Kita dapat melakukan aritmatika sederhana pada tipe datetime dan interval dengan cara yang sama seperti pada bilangan.Penjumlahan, pengurangan, perkalian, dan pembagian semua dimungkinkan di PostgreSQL menggunakan operator matematika +,-, *, dan /. Misalnya, Anda dapat mengurangi Kode berikut mengembalikan bilangan bulat 3: satu tanggal dari tanggal lain untuk mendapatkan bilangan bulat yang mewakili perbedaan hari antara dua tanggal.

```
SELECT '9/30/1929'::date - '9/27/1929'::date;
```

Hasilnya menunjukkan bahwa kedua tanggal ini berjarak tepat tiga hari.
Demikian pula, Anda dapat menggunakan kode berikut untuk menambahkan interval waktu ke tanggal untuk mengembalikan tanggal baru:

```
SELECT '9/30/1929'::date + '5 years'::interval;
```

Kode ini menambahkan lima tahun ke tanggal 30/9/1929 untuk mengembalikan nilai stempel waktu 30/9/1934.

Anda dapat menemukan lebih banyak contoh fungsi matematika yang dapat Anda gunakan dengan tanggal dan waktu dalam dokumentasi PostgreSQL di https://www.postgresql.org/docs / current / static / functions-datetime.html Mari kita jelajahi beberapa contoh praktis lainnya menggunakan aktual data transportasi.

Menemukan Pola di Data Taksi Kota New York

Ketika saya mengunjungi New York City, saya biasanya naik setidaknya satu kali di salah satu dari 13.500 mobil kuning ikonik yang mengangkut ratusan ribu orang melintasi lima wilayah kota setiap hari. The New York City Taxi and Limousine Commission merilis data kuning bulanan perjalanan taksi ditambah kendaraan sewaan lainnya Kami akan menggunakan kumpulan data yang besar dan kaya ini untuk menempatkan fungsi tanggal untuk penggunaan praktis.

File yellow_tripdata_2016_06_01.csv tersedia dari sumber buku
(di https://www.nostarch.com/practicalSQL/) menyimpan catatan perjalanan taksi kuning satu hari mulai 1 Juni 2016. Simpan file ke komputer Anda dan jalankan kode ini untuk membuat tabel nyc_yellow_taxi_trips_2016_06_01 Ingatlah untuk mengubah jalur file dalam perintah COPY ke lokasi tempat Anda menyimpan file dan menyesuaikan format jalur untuk mencerminkan apakah Anda menggunakan Windows, macOS, atau Linux.

```
CREATE TABLE nyc_yellow_taxi_trips_2006_06-01 (
      trip_id bigserial PRIMARY KEY,
      vendor_id varchar(1) NOT NULL,
      tpep_pickup_datetime timestamp with time zone NOT NULL,
      tpep_dropoff_datetime timestamp with time zone NOT NULL,
      passenger_count integer NOT NULL,
      trip_distance numeric(8,2) NOT NULL,
      pickup_longitude numeric(18,15) NOT NULL,
      pickup_latitude numeric(18,15) NOT NULL,
      rate_code_id varchar(2) NOT NULL,
      store_and_fwd_flag varchar(1) NOT NULL,
      dropoff_longitude numeric(18,15) NOT NULL,
      dropoff_latitde numeric(18,15) NOT NULL,
      payment_type varchar(1) NOT NULL,
```

```
      fare_amount numeric(9,2) NOT NULL,
      extra numeric(9,2) NOT NULL,
      mta_tax numeric(5,2) NOT NULL,
      tip_amount numeric(9,2) NOT NULL,
      tolls_amount numeric(9,2) NOT NULL,
      improvement_surcharge numeric(9,2) NOT NULL,
      total_amount numeric(9,2) NOT NULL,
);

COPY nyc_yellow_taxi_trips_2016_06_01 (
      vendor_id,
      tpep_pickup_datetime,
      tpep_dropoff_datetime,
      passenger_count,
      trip_distance,
      pickup_longitude,
      pickup_latitude,
      rate_code_id,
      store_and_fed_flag,
      dropoff_longitude,
      dropoff_latitude,
      payment_type,
      fare_amount,
      extra,
      mta_tax,
      tip_amount,
      tolls_amount,
      improvement_surcharge,
      total_amount
)
FROM 'C:\YourDirectory\yellow_tripdata_2016_06_01.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');

CREATE INDEX tpep_pickup_idx
ON nyc_yellow_taxi_trip-2016_06_01 (tpep_pickup_datetime);
```

Sintaks tersebit diatas untuk membuat tabel , mengimpor baris , dan membuat indeks. Dalam pernyataan COPY, kami memberikan nama kolom karena file CSV input tidak menyertakan kolom trip_id yang ada di tabel target. Kolom itu bertipe bigserial, yang telah Anda pelajari adalah bilangan bulat yang bertambah otomatis dan akan terisi secara otomatis. Setelah impor Anda selesai, Anda harus memiliki 368.774 baris, satu untuk setiap naik taksi kuning pada 1 Juni 2016. Anda dapat memeriksa jumlah baris di tabel Anda dengan hitungan menggunakan kode berikut:

```
SELECT count(*) FROM nyc_yellow_taxi_trips_2016_06_01;
```

Setiap baris memuat data jumlah penumpang, lokasi penjemputan dan pengantaran di lintang dan bujur, serta tarif dan tip dalam dolar AS Kamus data yang menjelaskan semua kolom dan kode tersedia di
http: //www.nyc.gov/html/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf.

Untuk latihan ini, kami paling tertarik pada kolom stempel waktu tpep_pickup_datetime dan tpep_dropoff_datetime, yang mewakili waktu mulai dan akhir perjalanan. (Proyek Peningkatan Penumpang Teknologi [TPEP] adalah program yang di bagian termasuk pengumpulan data otomatis tentang perjalanan taksi.)

Nilai di kedua kolom stempel waktu termasuk zona waktu yang disediakan oleh Komisi Taksi dan Limusin. Di semua baris file CSV, zona waktu yang disertakan dengan stempel waktu ditampilkan sebagai -4, yang merupakan offset UTC musim panas untuk Wilayah Timur zona waktu Jika Anda tidak atau server PostgreSQL Anda tidak berada di waktu Timur, saya sarankan untuk mengatur zona waktu Anda menggunakan kode berikut sehingga hasil Anda akan cocok dengan saya:

```
SET timezone TO 'US/Eastern':
```

Sekarang mari kita jelajahi pola yang dapat kita identifikasi dalam data yang terkait dengan waktu ini.

Waktu tersibuk dalam sehari

Satu pertanyaan yang mungkin Anda tanyakan setelah melihat kumpulan data ini adalah kapan taksi menyediakan paling banyak tumpangan. Apakah jam sibuk pagi atau sore hari, atau ada waktu lain—setidaknya, pada hari ini—saat jumlah kendaraan melonjak? Anda dapat menentukan jawabannya dengan kueri agregasi sederhana yang menggunakan date_part().

Kueri untuk menghitung perjalanan berdasarkan jam menggunakan waktu penjemputan sebagai input:

```
SELECT
      Date_part('hour', tpep_pickup_datetime) AS trip_hour,
      Count(*)
FROM nyc_yeloow_taxi_trps_2016_06_01
GROUP BY trip_hour
ORDER BY trip_hour;
```

Di kolom pertama kueri , date_part() mengekstrak jam dari tpep_pickup_datetime sehingga kami dapat mengelompokkan jumlah perjalanan berdasarkan jam. Kemudian kami menggabungkan jumlah perjalanan di kolom kedua melalui fungsi count(). Sisa kueri mengikuti pola standar untuk mengelompokkan dan mengurutkan hasil, yang seharusnya mengembalikan 24 baris, satu untuk setiap jam dalam sehari:

```
trip_hour   count
---------   -----
0           8182
1           5003
2           3070
3           2275
4           2229
5           3925
6           10825
7           18287
8           21062
9           19875
10          17367
11          17383
12          18031
13          17998
14          19125
15          18053
16          15069
17          18513
```

```
18          22689
19          23190
20          23098
21          24106
22          22554
23          17765
```

Melihat angka-angkanya, terlihat bahwa pada 1 Juni 2016, taksi New York City memiliki penumpang paling banyak antara pukul 18:00 dan 22:00, mungkin mencerminkan perjalanan pulang ditambah banyaknya aktivitas kota pada malam musim panas. Mari kita lakukan ini selanjutnya Pola keseluruhan, yang terbaik adalah memvisualisasikan data.

Mengekspor ke CSV untuk Visualisasi di Excel

Memetakan data dengan alat seperti Microsoft Excel memudahkan untuk memahami pola, jadi saya sering mengekspor hasil kueri ke file CSV dan membuat bagan cepat sintaks dibawah ini menggunakan kueri dari contoh sebelumnya dalam COPY.

```
COPY
      (SELECT
            date_part('hour', tpep-pickup_datetime) AS trip_hour,
            count(*)
      FROM nyc_yellow_taxi_trips_2016_06_01

      GROUP BY trip_hour
      ORDER BY trip_hour
      )
TO 'C:\YourDiretory\hourly_pickups_2016_06_01.csv'
WITH (FORMAT CSV, HEADER, DELIMITER ',');
```

Saat saya memuat data ke Excel dan membuat grafik garis, pola hari itu menjadi lebih jelas dan menggugah pikiran, seperti yang ditunjukkan pada Gambar 11-1.

Gambar 11.1: Penjemputan taksi kuning NYC per jam

Perjalanan mencapai titik terendah pada dini hari sebelum meningkat tajam antara pukul 05.00 dan 08.00. Volume tetap relatif stabil sepanjang hari dan meningkat lagi untuk jam sibuk malam hari setelah pukul 17.00. Namun ada penurunan antara pukul 15.00 dan 16.00 PM—kenapa?

Untuk menjawab pertanyaan itu, kita perlu menggali lebih dalam untuk menganalisis data yang berlangsung beberapa hari atau bahkan beberapa bulan untuk melihat apakah data kita dari 1 Juni 2016 adalah tipikal. Kita bisa menggunakan fungsi date_part() untuk membandingkan volume perjalanan. Agar lebih ambisius, kita dapat memeriksa laporan cuaca dan

membandingkan perjalanan pada hari hujan versus hari cerah Ada banyak cara berbeda untuk mengiris kumpulan data untuk mendapatkan kesimpulan.

Kapan Perjalanan Paling Lama?

Mari kita selidiki pertanyaan menarik lainnya: pada jam berapa perjalanan taksi memakan waktu paling lama? Salah satu cara untuk menemukan jawaban adalah dengan menghitung waktu perjalanan rata-rata untuk setiap jam. Median adalah nilai tengah dalam serangkaian nilai yang diurutkan; seringkali lebih akurat daripada rata-rata untuk membuat perbandingan karena beberapa nilai yang sangat kecil atau sangat besar dalam himpunan tidak akan mengubah hasil seperti yang terjadi pada rata-rata.

Di Bab 5, kita menggunakan fungsi persentil_cont() untuk mencari median.
Kami menggunakannya lagi di listing berikut ini untuk menghitung waktu perjalanan rata-rata:

```
SELECT
      date_part('hour', tpep_pickupdate) AS trip_hour,
      percentile_cont(.5)
            WITHIN GROUP (ORDER BY
                  Tpep_dropoff_datetime – tpep_pickup_datetime) AS median_trip
FROM nyc_yellow_taxi_trip_2016_06_01
GROUP BY trip_hour
ORDER BY trip_hour;
```

Kami menggabungkan data dengan bagian jam dari kolom stempel waktu tpep_pickup_datetime lagi, yang kami ekstrak menggunakan date_part(). Untuk input ke fungsi persentil_cont() , kami mengurangi waktu pengambilan dari waktu drop-off di dalam Klausa GROUP Hasil penelitian menunjukkan bahwa jam 1 PM memiliki median waktu perjalanan tertinggi yaitu 15 menit:

```
date_part   median_trip
---------   -----------
0           00:10:04
1           00:09:27
2           00:08:59
3           00:09:57
4           00:10:06
5           00:07:37
6           00:07:54
7           00:10:23
8           00:12:28
9           00:13:11
10          00:13:46
11          00:14:20
12          00:14:49
13          00:15:00
14          00:14:35
15          00:14:43
16          00:14:42
17          00:14:15
18          00:13:19
19          00:12:25
20          00:11:46
21          00:11:54
22          00:11:37
23          00:11:14
```

Seperti yang kita duga, waktu perjalanan terpendek di pagi hari Hasil ini masuk akal karena lebih sedikit lalu lintas di pagi hari berarti penumpang lebih mungkin untuk mencapai tujuan mereka lebih cepat.

Sekarang kita telah menjelajahi cara untuk mengekstrak bagian dari stempel waktu untuk analisis, mari gali lebih dalam analisis yang melibatkan interval.

Menemukan Pola dalam Data Amtrak

Amtrak, layanan kereta api nasional di Amerika, menawarkan beberapa paket perjalanan di seluruh Amerika Serikat.The All American, misalnya, adalah kereta api yang berangkat dari Chicago dan berhenti di New York, New Orleans, Los Angeles, San Francisco, dan Denver sebelum kembali ke Chicago. Menggunakan data dari situs web Amtrak (http://www.amtrak.com/), kami akan membuat tabel yang menunjukkan informasi untuk setiap segmen perjalanan. Perjalanan mencakup empat zona waktu, jadi kami akan Kemudian kita akan menghitung durasi perjalanan di setiap segmen dan mengetahui panjang seluruh perjalanan.

Menghitung Durasi Perjalanan Kereta

Mari kita buat tabel yang membagi rute kereta The All American menjadi enam segmen. Listing dibawah ini berisi SQL untuk membuat dan mengisi tabel dengan waktu keberangkatan dan kedatangan untuk setiap bagian perjalanan:

```
SET timezone TO 'US/Central';

      CREATE TABLE train_rides (
            trip_id bigserial PRIMARY KEY,
            segment varchar(50) NOT NULL,
            departure timestamp with timezone NOT NULL,
            arrival timestamp with timezone NOT NULL,
      );

      INSERT INTO train rides (segment, departure, arrival)
      VALUES
            ('Chicago to New York', '2017-11-13 21:30 CST', '2017-11-14 18:23 EST'),
            ('New York to New Orleans', '2017-11-15 14:55 EST', 2017-11-16 19:32 CST'),
            ('New Orleans to Los Angeles', 2017-11-17 13:45 CST', '2017-11-18 9:00 PST'),
            ('Los Angelesmto San Francisco', '2017-11-19 10:10 PST', '2017-11-19 21:24
            PST')
            ('San Francisco to Denver', '2017-11-20 9:10 PST', '2017-11-21 18:38 MST'),
            ('Dnvers to Chicago', '2017-11-22 19:10 MST', '2017-11-2 14:50 CST');

      SELECT * FROM train_rides;
```

Pertama, kami menetapkan sesi ke zona waktu Tengah, nilai untuk Chicago, menggunakan penunjuk AS / Pusat . Kami akan menggunakan waktu Tengah sebagai referensi kami saat melihat stempel waktu data yang kami masukkan sehingga terlepas dari Anda dan saya zona waktu default mesin, kami akan berbagi tampilan data yang sama.

Selanjutnya, kami menggunakan pernyataan CREATE TABLE standar. Perhatikan bahwa kolom untuk waktu keberangkatan dan kedatangan diatur ke stempel waktu dengan zona waktu . Terakhir, kami menyisipkan baris yang mewakili enam kaki perjalanan . Setiap masukan stempel waktu mencerminkan zona waktu dari Menentukan zona waktu kota adalah kunci

untuk mendapatkan penghitungan durasi perjalanan yang akurat dan memperhitungkan perubahan zona waktu. Ini juga memperhitungkan perubahan tahunan ke dan dari waktu musim panas jika terjadi selama rentang waktu yang Anda periksa. Pernyataan SELECT terakhir harus mengembalikan isi tabel seperti ini:

```
trip_id     segment                       departure              arrival
-------     ---------------------------- ---------------------- ----------------------
1           Chicago to New York           2017-11-13 21:30:00-06 2017-11-14 17:23:00-06
2           New York to New Orleans       2017-11-15 13:15:00-06 2017-11-16 19:32:00-06
3           New Orleans to Los Angeles    2017-11-17 13:45:00-06 2017-11-18 11:00:00-06
4           Los Angeles to San Francisco  2017-11-19 12:10:00-06 2017-11-19 23:24:00-06
5           San Francisco to Denver       2017-11-20 11:10:00-06 2017-11-21 19:38:00-06
6           Denver to Chicago             2017-11-22 20:10:00-06 2017-11-23 14:50:00-06
```

Semua stempel waktu sekarang harus membawa offset UTC sebesar -06, yang setara dengan zona waktu Tengah di Amerika Serikat selama bulan November, setelah negara tersebut beralih ke waktu standar. Terlepas dari zona waktu yang kami berikan pada sisipan, tampilan data sekarang dalam waktu Tengah, dan waktunya disesuaikan jika berada di zona waktu lain.

Sekarang setelah kami membuat segmen yang sesuai dengan setiap bagian perjalanan, kami akan menggunakan listing ini untuk menghitung durasi setiap segmen:

```
SELECT segment,
      to_char(departure, 'YYYY-MM-DD HH12:MI a.m. TZ') AS departure,
      arrival – departure AS segment_time
FROM train_rides;
```

Kueri ini mencantumkan segmen perjalanan, waktu keberangkatan, dan durasi perjalanan segmen. Sebelum kita melihat perhitungan, perhatikan kode tambahan di sekitar kolom keberangkatan . Ini adalah fungsi pemformatan khusus PostgreSQL yang menentukan cara memformat Dalam hal ini, fungsi to_char () mengubah kolom cap waktu keberangkatan menjadi string karakter yang diformat sebagai YYYY-MM-DD HH12: MI am TZ. Bagian YYYY-MM-DD menentukan format ISO untuk tanggal, dan bagian HH12: MI am menunjukkan waktu dalam jam dan menit. Bagian HH12 menentukan penggunaan jam 12 jam daripada waktu militer 24 jam. Bagian am menentukan bahwa kita ingin menunjukkan waktu pagi atau malam menggunakan huruf kecil karakter yang dipisahkan oleh titik, dan bagian TZ menunjukkan zona waktu.
Untuk daftar lengkap fungsi pemformatan, lihat dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/functions-format.html.

Terakhir, kami mengurangi keberangkatan dari kedatangan untuk menentukan segmen_time . Saat Anda menjalankan kueri, hasilnya akan terlihat seperti ini:

```
segment                       departure                segment_time
----------------------------- ------------------------ --------------
Chicago to New York           2017-11-13 09:30 p.m. CST 19:53:00
New York to New Orleans       2017-11-15 01:15 p.m. CST 1 day 06:17:00
New Orleans to Los Angeles    2017-11-17 01:45 p.m. CST 21:51:00
Los Angeles to San Francisco  2017-11-19 12:10 p.m. CST 11:14:00
San Francisco to Denver       2017-11-20 11:10 a.m. CST 1 day 08:28:00
Denver to Chicago             2017-11-22 08:10 p.m. CST 18:40:00
```

Mengurangi satu stempel waktu dari yang lain menghasilkan tipe data interval, yang diperkenalkan di Bab 3. Selama nilainya kurang dari 24 jam, PostgreSQL menyajikan interval dalam format HH: MM: SS Untuk nilai yang lebih besar dari 24 jam , ini mengembalikan format 1 hari 08:28:00, seperti yang ditunjukkan di segmen San Francisco ke Denver.

Dalam setiap perhitungan, PostgreSQL memperhitungkan perubahan zona waktu sehingga kami tidak secara tidak sengaja menambah atau kehilangan jam saat mengurangi. Jika kami menggunakan stempel waktu tanpa tipe data zona waktu, kami akan berakhir dengan panjang perjalanan yang salah jika segmen membentang beberapa zona waktu.

Menghitung Waktu Perjalanan Kumulatif

Ternyata, San Francisco ke Denver adalah perjalanan terpanjang dari perjalanan kereta api All American. Tapi berapa lama waktu yang dibutuhkan untuk seluruh perjalanan? Untuk menjawab pertanyaan ini, kita akan meninjau kembali fungsi jendela, yang telah Anda pelajari di "Peringkat dengan peringkat () dan density_rank () ”.

Kueri kami sebelumnya menghasilkan interval, yang kami beri label segment_time. Tampaknya langkah alami berikutnya adalah menulis kueri untuk menambahkan nilai tersebut, membuat interval kumulatif setelah setiap segmen. Dan memang, kami dapat menggunakan sum () sebagai fungsi jendela, dikombinasikan dengan klausa OVER yang disebutkan dalam Bab 10, untuk membuat total berjalan. Tetapi ketika kita melakukannya, nilai yang dihasilkan ganjil. Untuk melihat apa yang saya maksud, jalankan kode di bawah ini:

```
SELECT segment,
      arrival – departure AS segment_time,
      sum(arrival – departure) OVER (ORDER BY trip_id) AS cume_time
FROM train_rides;
```

Di kolom ketiga, kami menjumlahkan interval yang dihasilkan saat kami mengurangi keberangkatan dari kedatangan. Total berjalan yang dihasilkan di kolom cume_time akurat tetapi diformat dengan cara yang tidak membantu:

```
segment                      segment_time      cume_time
---------------------------- ------------      --------------
Chicago to New York          19:53:00          19:53:00
New York to New Orleans      1 day 06:17:00    1 day 26:10:00
New Orleans to Los Angeles   21:51:00          1 day 47:25:00
Los Angeles to San Francisco 11:14:00          1 day 58:39:00
San Francisco to Denver      1 day 08:28:00    2 days 67:07:00
Denver to Chicago            18:40:00          2 days 85:47:00
```

PostgreSQL membuat satu jumlah untuk porsi hari dari interval dan satu lagi untuk jam dan menit Jadi, alih-alih waktu kumulatif 5 hari yang lebih mudah dipahami 13:47:00, database melaporkan 2 hari 85:47: 00. Kedua hasil jumlah waktu yang sama, tetapi 2 hari 85:47:00 lebih sulit untuk diuraikan Ini adalah keterbatasan yang tidak menguntungkan dalam menjumlahkan interval basis data menggunakan sintaks ini.

Sebagai solusinya, kami akan menggunakan kode dibawah ini:

```
SELECT segment,
      arrival – departure AS segment_time,
      sum(date_part ('epoch', (arrival – departure)))
            OVER (ORDER BY trip_id) * interval '1 second' AS cume_time
FROM train_rides;
```

Ingat dari awal bab ini bahwa epoch adalah jumlah detik yang telah berlalu sejak tengah malam pada tanggal 1 Januari 1970, yang membuatnya berguna untuk menghitung durasi Pada Listing dibawah ini, kami menggunakan date_part (). dengan pengaturan epoch Kemudian kita kalikan setiap jumlah dengan interval 1 detik untuk mengubah detik tersebut menjadi nilai interval.Hasilnya lebih jelas menggunakan metode ini:

```
segment                     segment_time      cume_time
--------------------------- ------------      --------------
Chicago to New York         19:53:00          19:53:00
New York to New Orleans     1 day 06:17:00    50:10:00
New Orleans to Los Angeles  21:51:00          71:25:00
Los Angeles to San Francisco 11:14:00         82:39:00
San Francisco to Denver     1 day 08:28:00    115:07:00
Denver to Chicago           18:40:00          133:47:00
```

Cume_time terakhir, sekarang dalam format HH: MM: SS, menambahkan semua segmen untuk mengembalikan total panjang perjalanan 133 jam dan 47 menit. Itu waktu yang lama untuk dihabiskan di kereta, tapi saya yakin pemandangannya sangat berharga.

Menangani waktu dan tanggal dalam database SQL menambahkan dimensi yang menarik pada analisis Anda, memungkinkan Anda menjawab pertanyaan tentang kapan suatu peristiwa terjadi bersama dengan masalah temporal lainnya dalam data Anda Dengan pemahaman yang kuat tentang format waktu dan tanggal, zona waktu, dan fungsi untuk dibedah komponen stempel waktu, Anda dapat menganalisis hampir semua kumpulan data yang Anda temukan.

Selanjutnya, kita akan melihat teknik kueri lanjutan yang membantu menjawab pertanyaan yang lebih kompleks.

Latihan Soal

Cobalah latihan ini untuk menguji keterampilan Anda pada tanggal dan waktu.

1. Dengan menggunakan data taksi Kota New York, hitung panjang setiap perjalanan menggunakan stempel waktu penjemputan dan pengantaran. Urutkan hasil kueri dari perjalanan terpanjang hingga terpendek. Apakah Anda melihat sesuatu tentang perjalanan terpanjang atau terpendek yang mungkin Anda lakukan ingin bertanya kepada pejabat kota tentang?
2. Dengan menggunakan kata kunci AT TIME ZONE, tulis kueri yang menampilkan tanggal dan waktu London, Johannesburg, Moskow, dan Melbourne saat 1 Januari 2100 tiba di New York City.
3. Sebagai tantangan bonus, gunakan fungsi statistik di Bab 10 untuk menghitung koefisien korelasi dan nilai r-kuadrat menggunakan waktu perjalanan dan kolom jumlah_total di data taksi Kota New York, yang mewakili jumlah total yang dibebankan penumpang. Lakukan hal yang sama dengan kolom trip_distance dan total_amount. Batasi kueri untuk perjalanan yang berlangsung tiga jam atau kurang.

BAB XII
TEKNIK KUERI TINGKAT LANJUT

Terkadang analisis data memerlukan teknik SQL tingkat lanjut yang melampaui gabungan tabel atau kueri SELECT dasar. Misalnya, untuk menemukan cerita dalam data Anda, Anda mungkin perlu menulis kueri yang menggunakan hasil kueri lain sebagai input. Atau Anda mungkin perlu mengklasifikasi ulang nilai numerik ke dalam kategori sebelum menghitungnya. Seperti bahasa pemrograman lainnya, SQL menyediakan kumpulan fungsi dan pilihan yang penting untuk memecahkan masalah yang lebih kompleks, dan itulah yang akan kita jelajahi dalam bab ini.

Untuk latihan, saya akan memperkenalkan kumpulan data suhu yang direkam di kota-kota tertentu di AS dan kami akan meninjau kembali kumpulan data yang telah Anda buat di bab sebelumnya. Kode untuk latihan tersedia, bersama dengan semua sumber daya buku, di https //www.nostarch.com/practicalSQL/. Anda akan terus menggunakan basis data analisis yang telah Anda buat. Mari kita mulai.

Menggunakan Subquery

Sintaksnya tidak biasa: kami hanya menyertakan sub. Subquery bersarang di dalam kueri lain. Biasanya, ini digunakan untuk penghitungan atau pengujian logika yang memberikan nilai atau kumpulan data untuk diteruskan ke bagian utama kueri. - -query dalam tanda kurung dan menggunakannya jika diperlukan. Misalnya, kita dapat menulis subquery yang mengembalikan beberapa baris dan memperlakukan hasilnya sebagai tabel dalam klausa FROM dari kueri utama. Atau kita dapat membuat subquery skalar yang mengembalikan nilai tunggal Ini adalah penggunaan subkueri yang paling umum dan menggunakannya sebagai bagian dari ekspresi untuk memfilter baris melalui klausa WHERE, IN, dan HAVING.

Anda pertama kali menemukan subquery di Bab 9 dalam sintaks standar ANSI SQL untuk tabel UPDATE, yang ditampilkan lagi di sini Baik data untuk pembaruan dan kondisi yang menentukan baris mana yang akan diperbarui dihasilkan oleh subquery yang terlihat untuk nilai yang cocok dengan kolom di tabel dan table_b:

```
UPDATE table
SET column = (SELECT column FROM table_b WHERE table_b.column)
WHERE EXIST (SELECT column  FROM table_b WHERE table.column = table_b.column);
```

Contoh kueri ini memiliki dua subkueri yang menggunakan sintaks yang sama. Kami menggunakan pernyataan SELECT di dalam tanda kurung sebagai subkueri pertama dalam klausa SET, yang menghasilkan nilai untuk pembaruan. Demikian pula, kami menggunakan subkueri kedua di WHERE klausa EXISTS. , sekali lagi dengan pernyataan SELECT untuk memfilter baris yang ingin kita perbarui. Kedua subkueri adalah subkueri yang berkorelasi dan dinamai demikian karena mereka bergantung pada nilai atau nama tabel dari kueri utama yang mengelilinginya. Dalam hal ini, keduanya subqueries bergantung pada Subquery yang tidak berkorelasi tidak memiliki referensi ke objek dalam kueri utama.

Lebih mudah untuk memahami konsep-konsep ini dengan bekerja dengan data aktual, jadi mari kita lihat beberapa contoh Kita akan meninjau kembali dua kumpulan data dari bab-bab sebelumnya: tabel Sensus Tahunan 2010 us_counties_2010 yang Anda buat di Bab 4 dan tabel meat_poultry_egg_inspect di Bab 9.

Memfilter dengan Subquery dalam Klausa WHERE

Anda tahu bahwa klausa WHERE memungkinkan Anda memfilter hasil kueri berdasarkan kriteria yang Anda berikan, menggunakan ekspresi seperti WHERE kuantitas> 1000. Tetapi ini mengharuskan Anda sudah mengetahui nilai yang akan digunakan untuk perbandingan. Bagaimana jika Anda tidak tahu?

Itulah salah satu cara subquery berguna: ini memungkinkan Anda menulis kueri yang menghasilkan satu atau lebih nilai untuk digunakan sebagai bagian dari ekspresi dalam klausa WHERE.

Menghasilkan Nilai untuk Ekspresi Kueri

Misalnya Anda ingin menulis kueri untuk menunjukkan negara bagian AS mana yang berada pada atau di atas persentil ke-90, atau 10 persen teratas, untuk populasi. Daripada menulis dua kueri terpisah, satu untuk menghitung persentil ke-90 dan yang lainnya untuk memfilter menurut wilayah. Anda dapat melakukan keduanya sekaligus menggunakan subquery dalam klausa WHERE, seperti yang ditunjukkan pada sintaks dibawah ini:

```
SELECT geo_name,
      state_us_abbreviation, p00010001
FROM us_counties_2010 WHERE p0010001 >= ( SELECT percentile_con(.9)
      WITHIN GROUP (ORDER BY p0010001) FROM us_counties_2010 )
ORDER BY p0010001 DESC;
```

Kueri ini standar dalam hal apa yang telah kami lakukan sejauh ini kecuali bahwa klausa WHERE , yang memfilter menurut kolom populasi total p001001, tidak menyertakan nilai seperti biasanya. Sebagai gantinya, setelah > = compari-son operator, kami menyediakan kueri kedua dalam tanda kurung. Kueri kedua ini menggunakan fungsi persentil_cont () di Bab 5 untuk menghasilkan satu nilai: titik batas persentil ke-90 di kolom p0010001, yang kemudian akan digunakan dalam kueri utama.

Catatan: Menggunakan persentil_cont () untuk memfilter dengan subkueri hanya berfungsi jika Anda memasukkan satu input, seperti yang ditunjukkan. Jika Anda meneruskan larik, seperti pada Daftar 5-12 di halaman 68, persentil_cont () mengembalikan larik, dan kueri akan gagal mengevaluasi> = terhadap tipe array.

Jika Anda menjalankan subquery secara terpisah dengan menyorotnya di pgAdmin, Anda akan melihat hasil subquery, nilai 197444.6. Tetapi Anda tidak akan melihat angka itu saat Anda menjalankan seluruh kueri di Daftar 12-1, karena hasil dari subquery itu diteruskan langsung ke klausa WHERE untuk digunakan dalam memfilter hasil.

Seluruh kueri harus mengembalikan 315 baris, atau sekitar 10 persen dari 3.143 baris di us_counties_2010.

```
geo_name              state_us_abbreviation        p0010001
------------------    ---------------------        --------
Los Angeles County                      CA         9818605
Cook County                             IL         5194675
Harris County                           TX         4092459
Maricopa County                         AZ         3817117
San Diego County                        CA         3095313
--snip--
Elkhart County                          IN          197559
Sangamon County                         IL          197465
```

Hasilnya mencakup semua kabupaten dengan populasi lebih besar atau sama dengan 197444,6, nilai subquery yang dihasilkan.

Menggunakan Subquery untuk Mengidentifikasi Baris yang Akan Dihapus

Menambahkan subquery ke klausa WHERE dapat berguna dalam pernyataan kueri selain SELECT. Misalnya, kita dapat menggunakan subquery serupa dalam pernyataan DELETE untuk menentukan apa yang harus dihapus dari tabel. Bayangkan Anda memiliki tabel dengan 100 juta baris itu, karena ukurannya, membutuhkan waktu lama untuk kueri. Jika Anda hanya ingin mengerjakan subset data (seperti status tertentu), Anda dapat membuat salinan tabel dan menghapus apa yang tidak Anda perlukan dari itu.

Listing program dibawah ini menunjukkan contoh pendekatan ini, membuat salinan tabel sensus menggunakan metode yang Anda pelajari di Bab 9 dan kemudian menghapus semuanya dari cadangan itu kecuali 315 kabupaten di 10 persen populasi teratas:

```
CREATE TABLE us_counties_2010_top10 AS
SELECT * FROM us_counties_2010;

DELETE FROM us_counties_2010_top10
WHERE p0010001 < (
      SELECT percentile_cont(.9) WITHIN GROUP (ORDER BY p0010001)
      FROM us_counties_2010_top10
      );
```

Jalankan sintaks diatas tersebut, lalu jalankan SELECT count (*) FROM us_counties_2010_top10; untuk menghitung baris yang tersisa di tabel. Hasilnya harus 315 baris, yang merupakan 3.143 asli dikurangi 2.828 sub-kueri yang dihapus.

Membuat Tabel Turunan dengan Subqueries

Jika subkueri Anda mengembalikan baris dan kolom data, Anda dapat mengonversi data tersebut menjadi tabel dengan menempatkannya dalam klausa FROM, yang hasilnya dikenal sebagai tabel turunan. Tabel turunan berperilaku sama seperti tabel lainnya, sehingga Anda dapat kueri atau gabungkan ke tabel lain, bahkan tabel turunan lainnya. Pendekatan ini berguna ketika satu kueri tidak dapat melakukan semua operasi yang Anda perlukan.

Mari kita lihat contoh sederhana. Di Bab 5, Anda mempelajari perbedaan antara nilai rata-rata dan median. Saya menjelaskan bahwa median seringkali dapat menunjukkan nilai pusat kumpulan data dengan lebih baik karena beberapa nilai yang sangat besar atau kecil (atau outlier) dapat memiringkan rata-rata. Oleh karena itu, saya sering menyarankan untuk membandingkan rata-rata dan median. Jika keduanya dekat, data mungkin jatuh dalam distribusi normal (kurva lonceng yang sudah dikenal), dan rata-rata adalah representasi yang baik dari nilai pusat Jika rata-rata dan median berjauhan, beberapa outlier mungkin berpengaruh atau distribusinya miring, tidak normal.
Menemukan rata-rata dan median populasi negara bagian AS serta perbedaan di antara mereka adalah proses dua langkah. Kita perlu menghitung rata-rata dan median, dan kemudian kita perlu mengurangi keduanya. Kita dapat melakukan kedua operasi di satu gerakan dengan subquery dalam klausa FROM, seperti yang ditunjukkan ini.

```
SELECT round(calcs.average, 0) AS average,
      calcs.median,
      round(calsc.average - calcs.median, 0) AS median_average_diff
FROM (
      SELECT avg(p0010001) AS average,
            percentile_cont(.5)
                  WITHIN GROUP (ORDER BY p0010001)::numeric(10,1) AS median )
AS calcs;
```

Subquery sangat mudah. Kami menggunakan avg () dan persentil_cont() berfungsi untuk mencari rata-rata dan median dari tabel sensus p0010001 jumlah penduduk kolom dan menamai setiap kolom dengan alias, kemudian kita menamai subquery dengan alias dari calcs sehingga kita bisa mereferensikannya sebagai tabel di tabel utama. pertanyaan.

Pengurangan median dari rata-rata, yang keduanya dikembalikan oleh subquery, dilakukan di kueri utama; kemudian kueri utama membulatkan hasilnya dan melabelinya dengan alias median_average_diff. Jalankan kueri, dan hasilnya harus sebagai berikut:

```
average     median           median_average_diff
-------     -------          -------------------
98233       25857.0          72376
```

Perbedaan antara median dan rata-rata, 72.736, hampir tiga kali ukuran median. Itu membantu menunjukkan bahwa jumlah kabupaten berpenduduk tinggi yang relatif kecil mendorong ukuran kabupaten rata-rata lebih dari 98.000, median semua kabupaten jauh kurang dari 25.857.

Bergabung dengan Tabel Turunan

Karena tabel turunan berperilaku seperti tabel biasa, Anda dapat menggabungkannya. Bergabung dengan tabel turunan memungkinkan Anda melakukan beberapa langkah prapemrosesan sebelum sampai pada hasilnya. Misalnya, kita ingin menentukan negara bagian mana yang memiliki pabrik pemrosesan daging, telur, dan unggas paling banyak per juta populasi; sebelum kita dapat menghitung laju itu, kita perlu mengetahui jumlah tanaman di setiap negara bagian dan populasi setiap negara bagian.

Kita mulai dengan menghitung produsen menurut negara bagian menggunakan tabel meat_poultry_egg_inspect di Bab 9. Kemudian kita dapat menggunakan tabel us_counties_2010 untuk menghitung populasi menurut negara bagian dengan menjumlahkan dan mengelompokkan nilai county, sintaks dibawah ini menunjukkan cara menulis subkueri untuk kedua tugas dan menggabungkannya menghitung tarif keseluruhan.

```
SELECT cencus.state_us_abbreviation AS st,
      cencus.st_population,
      plants.plants_count,
      round((plants.plants_count/cencus.st_population::numeric(10,1))*1000000, 1)
            AS plants_per_million

FROM
      (
            SELECT st,
                  count(*) AS plant_count
            FROM meat_poultry_egg_inspect
            GROUP BY st
      )
      AS plants

JOIN
      (
            SELECT state_us_abbreviation,
                  Sum(p0010001) AS st_population
            FROM us_counties_2010
            GROUP BY state_us_abbreviation
      )
      AS census

ON plants.st = cencus.state_us_abbreviation
ORDER BY plants_permillion DESC;
```

Anda telah mempelajari cara menghitung tarif di Bab 10, jadi matematika dan sintaks dalam kueri utama untuk menemukan tanaman_per_juta seharusnya sudah familiar. Kami membagi jumlah tanaman dengan populasi, dan kemudian mengalikan hasil bagi itu dengan 1 juta. Untuk input, kami menggunakan nilai yang dihasilkan dari tabel turunan menggunakan subquery.

Subkueri pertama menemukan jumlah tanaman di setiap keadaan menggunakan fungsi agregat count() dan kemudian mengelompokkannya berdasarkan status. Kami memberi label subkueri ini dengan alias tanaman untuk referensi di bagian utama kueri. Subkueri kedua menemukan jumlah penduduk menurut negara bagian dengan menggunakan sum() pada kolom jumlah penduduk p0010001 dan kemudian mengelompokkannya berdasarkan state_us_abbreviation. Kami alias tabel turunan ini sebagai sensus.

Selanjutnya, kita gabungkan tabel turunan dengan menghubungkan kolom st pada tumbuhan ke kolom state_us_abbreviation dalam sensus. Kemudian kita daftar hasilnya dalam urutan menurun berdasarkan tarif yang dihitung. Berikut adalah contoh keluaran dari 51 baris yang menunjukkan tarif tertinggi dan terendah:

```
st    st_population     plant_count       plants_per_million
--    -------------     -----------       ------------------
NE          1826341             110                     60.2
```

```
IA          3046355            149                 48.9
VT           625741             27                 43.1
HI          1360301             47                 34.6
ND           671591             22                 32.7
--snip--
SC          4625364             55                 11.9
LA          4533372             49                 10.8
AZ          6392017             37                  5.8
DC           601723              2                  3.3
WY           563626              1                  1.8
```

Hasilnya sesuai dengan apa yang kita harapkan. Negara bagian teratas adalah produsen daging terkenal. Misalnya, Nebraska adalah salah satu pengekspor sapi terbesar di negara itu, dan Iowa memimpin Amerika Serikat dalam produksi daging babi.

Washington, D.C., dan Wyoming di bagian bawah daftar adalah di antara negara bagian dengan tanaman per sejuta paling sedikit.

Menghasilkan Kolom dengan Subqueries

Anda juga dapat membuat kolom data baru dengan subkueri dengan menempatkan subkueri dalam daftar kolom setelah SELECT. Biasanya, Anda akan menggunakan nilai tunggal dari agregat. Misalnya, kueri di bawah ini memilih kolom geo_name dan total populasi p0010001 dari us_counties_2010, lalu tambahkan subquery untuk menambahkan median semua kabupaten ke setiap baris di kolom baru us_median:

```
SELECT geo_name,
      state_us_abbreviation AS st,
      p0010001 AS total_pop,
      (SELECT percentile_cont(.5) WITHIN GROUP (ORDER BY p0010001)
      FROM us_counties_2020) AS us_median
FROM us_counties_2010;
```

Baris pertama dari kumpulan hasil akan terlihat seperti ini:

```
geo_name          st          total_pop   us_median
--------------    --          ---------   ---------
Autauga County    AL          54571       25857
Baldwin County    AL          182265      25857
Barbour County    AL          27457       25857
Bibb County       AL          22915       25857
Blount County     AL          57322       25857
--snip--
```

Dengan sendirinya, nilai us_median yang berulang itu tidak terlalu membantu karena nilainya sama setiap saat. Akan lebih menarik dan berguna untuk menghasilkan nilai yang menunjukkan berapa banyak populasi setiap kabupaten menyimpang dari nilai median. Mari kita lihat bagaimana kita dapat menggunakan sintaks berikut ini yang dibangun pada Listing sebelumnya dengan menambahkan ekspresi subquery setelah SELECT yang menghitung perbedaan antara populasi dan median untuk setiap county:

```
SELECT geo_name,
      state_us_abbreviation AS st,
      p0010001 AS total_pop,
      (SELECT percentile.cont(.5) WITHIN GROUP (ORDER BY p0010001)
```

```
      FROM us_counties_2010) AS us_median,
      P0010001 – (SELECT percentile_cont(.5) WITHIN GROUP (ORGER BY p0010001)
            FROM us_counties_2010) AS diff_from_median
FROM us_counties_200
WHERE (p0010001 – (SELECT percentile_cont(.5) WITHIN GROUP (ORDER BY p0010001)
            FROM us_counties_2010))
      BETWEEN -1000 AND 1000;
```

Subquery yang ditambahkan adalah bagian dari definisi kolom yang mengurangi hasil subquery dari p001001, total populasi. Ini menempatkan data baru dalam kolom dengan alias diff_from_median. Untuk membuat kueri ini lebih berguna, kita dapat mempersempit hasilnya lebih lanjut Untuk menampilkan hanya county yang populasinya berada dalam 1.000 dari median. Ini akan membantu kami mengidentifikasi county mana di Amerika yang memiliki populasi mendekati median county. Untuk melakukannya, kami mengulangi ekspresi subquery dalam klausa WHERE dan memfilter hasil menggunakan BETWEEN -1000 DAN 1000 ekspresi.

Hasilnya akan mengungkapkan 71 kabupaten dengan populasi yang relatif dekat dengan median A.S. Berikut adalah lima baris pertama dari hasil:

```
geo_name          st    total_pop   us_median   diff_from_median
---------------   --    ---------   ---------   ----------------
Cherokee County   AL    25989       25857       132
Clarke County     AL    25833       25857       -24
Geneva County     AL    26790       25857       933
Cleburne County   AR    25970       25857       113
Johnson County    AR    25540       25857       -317
--snip--
```

Ingatlah bahwa subquery menambah waktu eksekusi query secara keseluruhan; oleh karena itu, jika kita bekerja dengan jutaan baris, kita dapat menyederhanakan sintaks tersebut diatas dengan menghilangkan subquery yang menampilkan kolom us_median. untuk referensi Anda.

Ekspresi Subquery

Untuk ini, kita dapat menggunakan beberapa ekspresi subquery ANSI SQL standar, yang merupakan kombinasi kata kunci dengan subquery dan umumnya digunakan dalam klausa WHERE untuk memfilter baris berdasarkan keberadaan nilai di tabel lain.

Dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/ static / functions-subquery.html mencantumkan ekspresi subquery yang tersedia, tetapi di sini kita akan memeriksa sintaks hanya untuk dua di antaranya.

Menghasilkan Nilai untuk Operator IN

Ekspresi subquery IN (subquery) seperti operator perbandingan IN di Bab 2 kecuali kami menggunakan subquery untuk menyediakan daftar nilai yang akan diperiksa daripada harus memberikannya secara manual. Dalam contoh berikut, kami menggunakan a subquery untuk menghasilkan nilai id dari tabel pensiunan, dan kemudian menggunakan daftar itu untuk operator IN di klausa WHERE. Ekspresi NOT IN melakukan kebalikannya untuk menemukan karyawan yang nilai idnya tidak muncul di pensiunan.

```
SELECT first_name,last_name
FROM employees
WHERE is IN (
      SELECT id
      FROM retirees);
```

Kita harapkan outputnya menampilkan nama-nama pegawai yang memiliki nilai id yang sesuai dengan yang ada di pensiunan.

Catatan: Kehadiran nilai NULL dalam kumpulan hasil subkueri akan menyebabkan kueri dengan ekspresi NOT IN tidak mengembalikan baris. Jika data Anda berisi nilai NULL, gunakan ekspresi WHERE NOT EXISTS yang dijelaskan di bagian berikutnya.

Memeriksa Apakah Ada Nilai

Ekspresi subquery lain, EXISTS (subquery), adalah tes benar / salah. Ini mengembalikan nilai true jika subquery dalam tanda kurung mengembalikan setidaknya satu baris. Jika tidak mengembalikan baris, EXISTS bernilai false. Dalam contoh berikut, perintah kueri mengembalikan semua nama dari tabel karyawan selama subkueri menemukan setidaknya satu nilai dalam id dalam tabel pensiunan.

```
SELECT first_name,last_name
FROM employees
WHERE EXISTS (
      SELECT id
      FROM retirees);
```

Daripada mengembalikan semua nama dari karyawan, kami malah dapat meniru perilaku IN dan membatasi nama ke tempat subquery setelah EXISTS menemukan setidaknya satu nilai id yang sesuai di pensiunan.Berikut ini adalah subquery yang berkorelasi karena tabel yang dinamai dalam kueri utama adalah dirujuk dalam subquery.

```
SELECT first_name,last_name
FROM employees
WHERE EXISTS (
      SELECT id
      FROM retirees
      WHERE id = employees.id);
```

Pendekatan ini sangat membantu jika Anda perlu bergabung di lebih dari satu kolom, yang tidak dapat Anda lakukan dengan ekspresi IN. Anda juga dapat menggunakan kata kunci NOT dengan EXISTS. Misalnya, untuk menemukan karyawan tanpa catatan yang sesuai di pensiunan, Anda akan menjalankan kueri ini:

```
SELECT first_name,last_name
FROM employees
WHERE NOT EXISTS (
      SELECT id
      FROM retirees
      WHERE id = employees.id);
```

Teknik menggunakan NOT dengan EXISTS sangat membantu untuk menilai apakah kumpulan data sudah lengkap.

Ekspresi Tabel Umum

Sebelumnya di bab ini, Anda telah mempelajari cara membuat tabel turunan dengan menempatkan subkueri dalam klausa FROM. Pendekatan kedua untuk membuat tabel sementara untuk kueri menggunakan Common Table Expression (CTE), tambahan yang relatif baru untuk SQL standar yang secara informal disebut "DENGAN klausa." Menggunakan CTE, Anda dapat mendefinisikan satu atau lebih tabel di depan dengan subkueri. Kemudian Anda dapat mengkueri hasil tabel sesering yang diperlukan dalam kueri utama berikutnya.

Sintaks dibawah ini menunjukkan CTE sederhana yang disebut large_counties berdasarkan data sensus kami, diikuti dengan kueri tabel tersebut. Kode menentukan berapa banyak county di setiap negara bagian yang memiliki 100.000 orang atau lebih. Mari kita menelusuri contoh ini.

```
WITH
      large_counties (geo_name, st, p0010001)
AS
      (
            SELECT geo_name, state_us_abbreviation, p0010001
            FROM us_counties_2010
            WHERE p0010001 <= 100000

SELECT st, count(*)
FROM large_counties
GROUP BY st
ORDER BY count(*) DESC;
```

Blok WITH ... AS mendefinisikan tabel sementara CTE yang besar_counties. Setelah WITH, kita memberi nama tabel dan mencantumkan nama kolomnya dalam tanda kurung. Tidak seperti definisi kolom dalam pernyataan CREATE TABLE, kita tidak perlu menyediakan tipe data, karena tabel sementara mewarisi dari subquery , yang diapit dalam tanda kurung setelah AS. Subquery harus mengembalikan jumlah kolom yang sama seperti yang didefinisikan dalam tabel sementara, tetapi nama kolom tidak harus cocok. Juga, daftar kolom adalah opsional jika Anda tidak mengganti nama kolom, meskipun menyertakan daftar masih merupakan ide bagus untuk kejelasan bahkan jika Anda tidak mengganti nama kolom.

Kueri utama menghitung dan mengelompokkan baris dalam large_counties berdasarkan st, lalu mengurutkan berdasarkan jumlah dalam urutan menurun. Lima baris teratas dari hasil akan terlihat seperti ini:

```
st    count
--    -----
TX    39
CA    35
FL    33
PA    31
OM    28
--snip--
```

Seperti yang Anda lihat, Texas, California, dan Florida termasuk di antara negara bagian dengan jumlah county tertinggi dengan populasi 100.000 atau lebih.

Anda dapat menemukan hasil yang sama menggunakan kueri SELECT alih-alih CTE, seperti yang ditunjukkan di sini:

```
SELECT state_us_abbreviation, count(*)
FROM us_counties_2010
WHERE p0010001 >= 100000
GROUP BY state_us_abbreviation
ORDER BY count(*) DESC;
```

Jadi mengapa menggunakan CTE? Salah satu alasannya adalah dengan menggunakan CTE, Anda dapat melakukan pra-tahapan subset data untuk dimasukkan ke dalam kueri yang lebih besar untuk analisis yang lebih kompleks. Selain itu, Anda juga dapat menggunakan kembali setiap tabel yang ditentukan dalam CTE di beberapa tempat di kueri utama, yang berarti Anda tidak perlu mengulang kueri SELECT setiap kali. Keuntungan lain yang sering dikutip adalah bahwa kode lebih mudah dibaca daripada jika Anda melakukan operasi yang sama dengan subkueri.

Sintaks dibawah ini menggunakan CTE untuk menulis ulang gabungan tabel turunan di kueri dibawah ini (menemukan negara bagian yang memiliki pabrik pengolahan daging, telur, dan unggas paling banyak per satu juta populasi) ke dalam format yang lebih mudah dibaca:

```
WITH
      counties (st, population) AS
      (SELECT state_us_abbreviation, sum(population_count_100_percent)
      FROM us_counties_2010
      GROUP BY state_us_abbreviation),

      plants (st,plants) AS
      (SELECT st, count(*) AS plants
      FROM meat_polutry_egg_inspect
      GROUP BY st)

SELECT counties.st,
      population,
      plants,
      round((plants/population::numeric(10,1)) * 1000000, 1) AS per_million
FROM counties JOIN plants
ON counties.st = plants.st
ORDER BY per_million DESC;
```

Mengikuti kata kunci WITH, kita mendefinisikan dua tabel menggunakan subquery. Subquery pertama, kabupaten , mengembalikan populasi setiap negara bagian. Yang kedua, tanaman , mengembalikan jumlah tanaman per negara bagian. Dengan tabel tersebut didefinisikan, kita menggabungkannya join pada Hasilnya identik dengan tabel turunan yang digabungkan.

Sebagai contoh lain, Anda dapat menggunakan CTE untuk menyederhanakan kueri dengan kode yang berlebihan. Misalnya, di sintaks sebelumnya, kami menggunakan subquery dengan fungsi persentil_cont() di tiga lokasi berbeda untuk menemukan median populasi county. Listprogram dibawah ini, kita dapat menulis subquery itu sekali saja sebagai CTE:

```
WITH us_median AS
      (SELECT percentile_cont(.5)
      WITHIN GROUP (ORDER BY p0010001) AS us_median_pop
      FROM us_counties_2010)

SELECT geo_name,
      state_us_abbreviation AS st,
      p0010001 AS total_pop,
      us_median_pop,
      p0010001 - us_median_pop AS diff_from_median
FROM us_counties_2010 CROSS JOIN us_median
WHERE (p0010001 - us_median_pop)
      BETWEEN -1000 AND 1000;
```

Setelah kata kunci WITH, kita mendefinisikan us_median sebagai hasil dari subquery yang sama dengan yang digunakan dalam sintaks sebelumnya, yang menemukan populasi median menggunakan persentil_cont (). Kemudian kita mereferensikan kolom us_median_pop sendiri its, sebagai bagian dari kolom terhitung , dan dalam klausa WHERE . Untuk membuat nilai tersedia untuk setiap baris dalam tabel us_counties_2010 selama SELECT, kami menggunakan kueri CROSS JOIN yang Anda pelajari di Bab 6.

Kueri ini memberikan hasil yang identik dengan yang ada, tetapi kami hanya perlu menulis subkueri satu kali untuk menemukan median. Hal ini tidak hanya menghemat waktu, tetapi juga memungkinkan Anda merevisi kueri dengan lebih mudah. Misalnya, untuk menemukan kabupaten yang populasinya mendekati persentil ke-90, Anda dapat mengganti .9 dengan .5 sebagai input ke persentil_cont () hanya di satu tempat.

Tabulasi Silang

Tabulasi silang menyediakan cara sederhana untuk meringkas dan membandingkan variabel dengan menampilkannya dalam tata letak tabel, atau matriks. Dalam matriks, baris mewakili satu variabel, kolom mewakili variabel lain, dan setiap sel di mana baris dan kolom berpotongan menyimpan nilai, seperti sebagai hitungan atau persentase.

Anda akan sering melihat tabulasi silang, juga disebut tabel pivot atau tab silang, digunakan untuk melaporkan ringkasan hasil survei atau untuk membandingkan serangkaian variabel. Contoh yang sering terjadi selama setiap pemilihan saat suara kandidat dihitung berdasarkan geografi:

```
candidate   ward 1      ward 2      ward 3
---------   ------      ------      ------
Disk        602         1,799       2,112
Pratt       599         1,398       1,616
Lerxst      911           902       1,114
```

Dalam hal ini, nama kandidat adalah satu variabel, kelurahan (atau distrik kota) adalah variabel lain, dan sel di persimpangan keduanya menyimpan total suara untuk kandidat di distrik itu. melakukan tabulasi silang.

Memasang fungsi tab silang ()

ANSI SQL standar tidak memiliki fungsi tab silang, tetapi PostgreSQL memilikinya sebagai bagian dari modul yang dapat Anda instal dengan mudah. Modul menyertakan ekstra

PostgreSQL yang bukan bagian dari aplikasi inti; mereka menyertakan fungsi yang terkait dengan keamanan, pencarian teks, dan banyak lagi Anda dapat menemukan daftar modul PostgreSQL di
https://www.postgresql.org/docs/current/static/contrib.html.

Fungsi crosstab () PostgreSQL adalah bagian dari modul tablefunc. Untuk menginstal tablefunc di Alat Kueri pgAdmin, jalankan perintah ini:

```
CREATE EXTENSION tablefunc;
```

PostgreSQL akan menampilkan pesan CREATE EXTENSION setelah selesai menginstal. (Jika Anda bekerja dengan sistem manajemen basis data lain, periksa dokumentasi untuk melihat apakah ia menawarkan fungsionalitas serupa. Misalnya, Microsoft SQL Server memiliki perintah PIVOT.)

Selanjutnya, kami akan membuat tab silang dasar sehingga Anda dapat mempelajari sintaksnya, dan kemudian kami akan menangani kasus yang lebih kompleks.

Tabulasi Hasil Survei

Katakanlah perusahaan Anda membutuhkan aktivitas karyawan yang menyenangkan, jadi Anda mengoordinasikan acara sosial es krim di tiga kantor Anda di kota. Masalahnya, orang-orang itu khusus tentang rasa es krim. Untuk memilih rasa yang disukai orang, Anda memutuskan untuk melakukan survey.

File CSV ice_cream_survey.csv berisi 200 tanggapan terhadap survei Anda. Anda dapat mengunduh file ini, bersama dengan semua sumber buku, di https://www.nostarch.com/practicalSQL/.Setiap baris menyertakan response_id, office, dan flavor Anda harus menghitung berapa banyak orang yang memilih setiap rasa di setiap kantor dan mempresentasikan hasilnya dengan cara yang mudah dibaca kepada rekan kerja Anda.

Dalam database analisis Anda, gunakan kode di bawah ini untuk membuat tabel dan memuat data.Pastikan Anda mengubah jalur file ke lokasi di komputer tempat Anda menyimpan file CSV.

```
CREATE TABLE ice_cream_survey (
      response_id integer PRIMARY KEY,
      office varchar(20),
      flavor varchar(20)
);

COPY ice_cream_survey
FROM 'C:\YourDirectory\ice_cream_survey.csv'
WITH (FORMAT CSV, HEADER);
```

Jika Anda ingin memeriksa data, jalankan perintah berikut untuk melihat lima baris pertama:

```
SELECT *
FROM ice_cream_survey
LIMIT 5;
```

Datanya akan terlihat seperti ini:

```
response_id       office            flavor
-----------       --------          ----------
1                 Uptown            Chocolate
2                 Midtown           Chocolate
3                 Downtown          Strawberry
4                 Uptown            Chocolate
5                 Midtown           Chocolate
```

Sepertinya cokelat memimpin! Tapi mari kita konfirmasikan pilihan ini dengan menggunakan kode di sintaks dibawah ini untuk menghasilkan tab silang dari tabel:

```
SELECT *
FROM crosstab('SELECT office,
                      flavor,
                      count(*)
            FROM ice_cream_survey
            GROUP BY office, flavor
            ORDER BY office,

            'SELECT flavor
            FROM ice_cream_survey
            GROUP BY flavor
            ORDER BY flavor')

AS (office varchar(20),
      chocolate bigint,
      strawberry bigint,
      vanilla bigint;
```

Kueri dimulai dengan pernyataan SELECT * yang memilih semuanya dari isi fungsi tab silang (). Kami menempatkan dua subkueri di dalam fungsi tab silang (). Subkueri pertama menghasilkan data untuk tab silang dan memiliki tiga kolom yang diperlukan. Yang pertama kolom, kantor, memasok nama baris untuk tab silang, dan kolom kedua, rasa , memasok kolom kategori. Kolom ketiga memasok nilai untuk setiap sel tempat baris dan kolom berpotongan dalam tabel. Dalam hal ini kasus, kami ingin sel yang berpotongan untuk menunjukkan hitungan () each dari setiap rasa yang dipilih di setiap kantor. Subquery pertama ini sendiri membuat daftar agregat sederhana.

Subquery kedua menghasilkan set nama kategori untuk kolom. Fungsi tab silang () mengharuskan subquery kedua hanya mengembalikan satu kolom, jadi di sini kita menggunakan SELECT untuk mengambil kolom flavor, dan kita menggunakan GROUP BY untuk mengembalikan keunikan kolom itu nilai-nilai.

Kemudian kita tentukan nama dan tipe data kolom keluaran tab silang mengikuti kata kunci AS. Daftar tersebut harus cocok dengan nama baris dan kolom dalam urutan yang dihasilkan oleh subkueri. Misalnya, karena subkueri kedua yang memasok kolom kategori memesan rasa menurut abjad, daftar kolom keluaran juga.

Saat kami menjalankan kode, data kami ditampilkan dalam tab silang yang bersih dan dapat dibaca:

```
office      chocolate   strawberry  vanilla
--------    ---------   ----------  -------
Downtown    23          32          19
Midtown     41                      23
Uptown      22          17          23
```

Sangat mudah untuk melihat sekilas bahwa kantor Midtown menyukai cokelat tetapi tidak tertarik pada stroberi, yang diwakili oleh nilai NULL yang menunjukkan bahwa stroberi tidak mendapat suara. Tetapi stroberi adalah pilihan utama di Pusat Kota, dan kantor Uptown lebih merata. diantara ketiga rasa tersebut.

Tabulasi Pembacaan Suhu Kota

Mari kita buat tab silang lain, tapi kali ini kita akan menggunakan data nyata File temperature_readings.csv, juga tersedia dengan semua sumber buku di https://www.nostar.com/practicalSQL/, berisi pembacaan suhu harian selama satu tahun dari tiga stasiun pengamatan di seluruh Amerika Serikat: Chicago, Seattle, dan Waikiki, sebuah lingkungan di pantai selatan kota Honolulu. Data berasal dari US National Oceanic and Atmospheric Administration (NOAA) di https: // www.ncdc .noaa.gov/cdo-web/datatools/findstation/.

Setiap baris dalam file CSV berisi empat nilai: nama stasiun, tanggal, suhu maksimum hari itu, dan suhu minimum hari itu. Semua suhu dalam Fahrenheit. Untuk setiap bulan di setiap kota, kita ingin menghitung median tinggi suhu sehingga kita dapat membandingkan iklim yang berisi kode untuk membuat tabel temperature_readings dan mengimpor file CSV:

```
CREATE TABLE temperature_readings (
            reading_id bigserial,
            station_name varchar(50),
            observation_date date,
            max_temp integer,
            min_temp integer
      );

      COPY temperature_reading
            (station_name, observation_date, max_temp, min_temp)
      FROM 'C:\YourDirectory\temperature_reading,csv'
      WITH (FORMAT CSV, HEADER);
```

Tabel berisi empat kolom dari file CSV bersama dengan read_id tambahan dari tipe bigserial yang kami gunakan sebagai kunci utama pengganti. Jika Anda melakukan penghitungan cepat pada tabel, Anda seharusnya memiliki 1.077 baris. Sekarang, mari kita lihat tabulasi silang apa data tidak menggunakan:

```
      SELECT *
      FROM crosstab('SELECT
                  station_name,
                  date_part(''month'', observation_date),
                  percentile_cont(.5)
                        WITHIN GROUP (ORDER BY max_temp)
                  FROM temperature_readings
                  GROUP BY station_name,
                        Date_part(''month"", observation_date)
                  ORDER BY station_name',

                  'SELECT month
```

```
            FROM generate_series(1,12) month')

AS (station varchar(50),
      jan numeric(3,0),
      feb numeric(3,0),
      mar numeric(3,0),
      apr numeric(3,0),
      may numeric(3,0),
      jun numeric(3,0),
      jul numeric(3,0),
      aug numeric(3,0),
      sep numeric(3,0),
      oct numeric(3,0),
      nov numeric(3,0),
      dec numeric(3,0)
);
```

Subquery pertama di dalam fungsi tab silang () menghasilkan data untuk tab silang, menghitung suhu maksimum rata-rata untuk setiap bulan. Ini memasok tiga kolom yang diperlukan. Kolom pertama, nama_stasiun , menamai baris. Kolom kedua menggunakan fungsi tanggal_bagian() yang Anda pelajari di Bab 11 untuk mengekstrak bulan dari tanggal_pengamatan, yang menyediakan kolom tab silang. Kemudian kita menggunakan persentil_cont(.5) untuk menemukan persentil ke-50 , atau median, dari max_temp. Kami mengelompokkan berdasarkan nama stasiun dan bulan sehingga kami memiliki median max_temp untuk setiap bulan di setiap stasiun.

Seperti pada listing dibawah ini, subquery kedua menghasilkan kumpulan nama kategori untuk kolom. Saya menggunakan fungsi yang disebut generate_series () dengan cara yang dicatat dalam dokumentasi PostgreSQL resmi untuk membuat daftar angka dari 1 hingga 12 yang cocok dengan nomor bulan date_part () diekstrak dari observasi_date.

Mengikuti AS, kami menyediakan nama dan tipe data untuk kolom keluaran tab silang. Masing-masing adalah tipe numerik, cocok dengan keluaran fungsi persentil. Keluaran berikut praktis berupa puisi:

```
station           jan   feb   mar   apr   may   jun   jul   aug   sep   oct   nov   dec
----------------- ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---
CHICAGO NORTHERLY
ISLAND IL US      34    36    46    50    66    77    81    80    77    65    57    35
SEATTLE BOEING
FIELD WA US       50    54    56    64    66    71    76    77    69    62    55    42
WAIKIKI 717.2
HI US             83    84    84    86    87    87    88    87    87    86    84    82
```

Kami telah mengubah satu set mentah bacaan harian menjadi tabel kompak yang menunjukkan suhu maksimum rata-rata setiap bulan untuk setiap stasiun Anda dapat melihat sekilas bahwa suhu di Waikiki secara konsisten nyaman, di atas titik beku hingga benar-benar menyenangkan. Seattle jatuh di antara keduanya.

Tab silang memang membutuhkan waktu untuk disiapkan, tetapi melihat kumpulan data dalam matriks sering kali membuat perbandingan lebih mudah daripada melihat data yang sama dalam daftar vertikal. Ingatlah bahwa fungsi tab silang () membutuhkan banyak CPU, jadi berhati-hatilah saat mengkueri kumpulan itu memiliki jutaan atau miliaran baris.

Mengklasifikasi Ulang Nilai dengan CASE

Pernyataan CASE SQL Standar ANSI adalah ekspresi bersyarat, artinya memungkinkan Anda menambahkan beberapa logika "jika ini, maka ..." ke kueri. Anda dapat menggunakan CASE dalam berbagai cara, tetapi untuk analisis data, ini berguna untuk mengklasifikasi ulang nilai ke dalam kategori. Anda dapat membuat kategori berdasarkan rentang dalam data Anda dan mengelompokkan nilai menurut kategori tersebut.

Sintaks CASE mengikuti pola ini:

```
CASE WHEN condition THEN result
     WHEN another_condition THEN result
     ELSE result
END
```

Kami memberikan kata kunci CASE , dan kemudian memberikan setidaknya satu klausa WHEN condition THEN result, di mana condition adalah ekspresi apa pun yang dapat dievaluasi oleh database sebagai benar atau salah, seperti county ='Dutchess County' atau date> '1995-08-09 Jika kondisinya benar, pernyataan CASE mengembalikan hasilnya dan berhenti memeriksa kondisi lebih lanjut. Hasilnya bisa berupa tipe data yang valid. Jika kondisinya salah, database bergerak untuk mengevaluasi kondisi berikutnya.

Untuk mengevaluasi kondisi lainnya, kita dapat menambahkan klausa WHEN ... THEN opsional.

Kami juga dapat menyediakan klausa ELSE opsional untuk mengembalikan hasil jika tidak ada kondisi yang bernilai benar. Tanpa klausa ELSE, pernyataan akan mengembalikan NULL ketika tidak ada kondisi yang benar. Pernyataan diakhiri dengan kata kunci END.

Sintaks ini menunjukkan bagaimana menggunakan pernyataan CASE untuk mengklasifikasi ulang data pembacaan suhu ke dalam kelompok deskriptif (dinamai menurut bias saya sendiri terhadap cuaca dingin):

```
SELECT max_temp,
     CASE  WHEN max_temp >= 90 THEN 'Hot'
           WHEN max_temp BETWEEN 70 AND 89 THEN 'Warm'
           WHEN max_temp BETWEEN 50 AND 69 THEN 'Pleasant'
           WHEN max_temp BETWEEN 33 AND 49 THEN 'Cold'
           WHEN max_temp BETWEEN 20 AND 32 THEN 'Freezing'
           ELSE 'Inhumane'
     END AS temperature_group
  FROM temperature_readings;
```

Kami membuat lima rentang untuk kolom max_temp di temperature_readings, yang kami definisikan menggunakan operator perbandingan. Pernyataan CASE mengevaluasi setiap nilai untuk menemukan apakah salah satu dari lima ekspresi itu benar. Jika demikian, pernyataan menghasilkan teks yang sesuai. Perhatikan bahwa akun rentang Jika tidak ada pernyataan yang benar, maka klausa ELSE memberikan nilai ke kategori Inhumane. Cara saya menyusun rentang, ini hanya terjadi ketika max_temp di bawah 20 derajat. Alternatifnya, kita dapat mengganti ELSE dengan klausa WHEN yang mencari suhu kurang dari atau sama dengan 19 derajat dengan menggunakan max_temp <= 19.

Jalankan kode; lima baris output pertama akan terlihat seperti ini:

```
max_temp    temperature_group
--------    -----------------
31          Freezing
34          Cold
32          Freezing
32          Freezing
34          Cold
--snip--
```

Sekarang setelah kita mengelompokkan kumpulan data ke dalam enam kategori, mari gunakan kategori tersebut untuk membandingkan iklim di antara tiga kota dalam tabel.

Menggunakan CASE dalam Ekspresi Tabel Umum

Operasi yang kami lakukan dengan CASE pada data suhu di bagian sebelumnya adalah contoh yang baik dari langkah pra-pemrosesan yang akan Anda gunakan dalam CTE. Sekarang setelah kita mengelompokkan suhu dalam kategori, mari kita hitung kelompok berdasarkan kota dalam sebuah CTE untuk melihat berapa hari dalam setahun yang termasuk dalam setiap kategori suhu.

Listing dibawah ini menunjukkan kode untuk mengklasifikasi ulang suhu maksimum harian yang disusun ulang untuk menghasilkan temps_collapsed CTE dan kemudian menggunakannya untuk analisis:

```
WITH temps_collapsed (station_name, max_temperature_group) AS
      (SELECT station_name,
            CASE  WHEN max_temp >= 90 THEN 'Hot'
                  WHEN max_temp BETWEEN 70 AND 89 THEN 'Warm'
                  WHEN max_temp BETWEEN 50 AND 69 THEN 'Pleasant'
                  WHEN max_temp BETWEEN 33 AND 49 THEN 'Cold'
                  WHEN max_temp BETWEEN 20 AND 32 THEN 'Freezing'
                  ELSE 'Inhumane'
      END
      FROM temperature_readings)

SELECT station_name, max_temperature_group, count(*)
FROM temps_collapsed
GROUP BY station_name, max_temperature_group
ORDER BY station_name, count(*) DESC;
```

Kode ini mengklasifikasi ulang suhu, lalu menghitung dan mengelompokkan menurut nama stasiun untuk menemukan klasifikasi iklim umum setiap kota. Kata kunci WITH mendefinisikan CTE dari temps_collapsed , yang memiliki dua kolom: station_name dan max_temperature_group. Kemudian kita menjalankan kueri SELECT pada CTE, melakukan operasi penghitungan langsung (*) dan GROUP BY pada kedua kolom. Hasilnya akan terlihat seperti ini:

```
station_name                    max_temperature_group   count
-----------------------------   ---------------------   -----
CHICAGO NORTHELY ISLAND IL US   Warm                    133
CHICAGO NORTHELY ISLAND IL US   Cold                    92
CHICAGO NORTHELY ISLAND IL US   Pleasant               91
CHICAGO NORTHELY ISLAND IL US   Freezing                30
CHICAGO NORTHELY ISLAND IL US   Inhuman                 8
CHICAGO NORTHELY ISLAND IL US   Hot                     8
```

```
SEATTLE BOEING FIELD WA US          Pleasant                198
SEATTLE BOEING FIELD WA US          Warm                    98
SEATTLE BOEING FIELD WA US          Cold                    50
SEATTLE BOEING FIELD WA US          Hot                     3
WAIKIKI 717.2 HI US                 Warm                    361
WAIKIKI 717.2 HI US                 Hot                     5
```

Dengan menggunakan skema klasifikasi ini, cuaca Waikiki yang luar biasa konsisten, dengan suhu maksimum Hangat 361 hari dalam setahun, menegaskan daya tariknya sebagai tujuan liburan. Dari sudut pandang suhu, Seattle juga terlihat bagus, dengan hampir 300 hari suhu tinggi dikategorikan sebagai Menyenangkan atau Hangat (walaupun ini memungkiri curah hujan legendaris Seattle) Chicago, dengan 30 hari suhu maksimum Pembekuan dan 8 hari Tidak manusiawi, mungkin bukan untuk saya.

Dalam bab ini, Anda telah belajar membuat kueri bekerja lebih keras untuk Anda. Kini Anda dapat menambahkan subkueri di beberapa lokasi untuk memberikan kontrol yang lebih baik atas pemfilteran atau prapemrosesan data sebelum menganalisisnya dalam kueri utama. Anda juga dapat memvisualisasikan data dalam matriks menggunakan tabulasi silang dan klasifikasi ulang data ke dalam kelompok; kedua teknik memberi Anda lebih banyak cara untuk menemukan dan menceritakan kisah menggunakan data Anda. Kerja bagus!

Sepanjang bab berikutnya, kita akan menyelami teknik SQL yang lebih spesifik untuk PostgreSQL. Kita akan mulai dengan bekerja dengan dan mencari teks dan string.

Latihan Soal

Berikut adalah dua tugas untuk membantu Anda menjadi lebih akrab dengan konsep-konsep yang diperkenalkan dalam bab ini:

1. Untuk menggali lebih dalam nuansa suhu tinggi Waikiki. Batasi tabel temps_collapsed ke pengamatan suhu harian maksimum Waikiki. Kemudian gunakan klausa WHEN dalam pernyataan CASE untuk mengklasifikasi ulang suhu menjadi tujuh grup yang akan menghasilkan output teks berikut:

   ```
   '90 or more'
   '88-89'
   '86-87'
   '84-85'
   '82-83'
   '80-81'
   '79 or less'
   ```

   Di kelompok manakah suhu maksimum harian Waikiki paling sering turun?

2. Perbaiki tab silang survei es krim untuk membalik tabel. Dengan kata lain, buat baris dan atur kolomnya. Elemen kueri mana yang perlu Anda ubah? Apakah hitungannya berbeda?

BAB XIII
PENCARIAN TEKS UNTUK MENEMUKAN DATA LENGKAP

Meskipun mungkin tidak terlihat jelas pada pandangan pertama, Anda dapat mengekstrak data dan bahkan mengukur data dari teks dalam pidato, laporan, siaran pers, dan dokumen lainnya Meskipun sebagian besar teks ada sebagai data tidak terstruktur atau semi terstruktur, yang tidak diatur dalam baris dan kolom, seperti dalam tabel, Anda dapat menggunakan SQL untuk mendapatkan makna darinya.

Salah satu cara untuk melakukannya adalah dengan mengubah teks menjadi data terstruktur. Anda mencari dan mengekstrak elemen seperti tanggal atau kode dari teks, memuatnya ke dalam tabel, dan menganalisisnya. Cara lain untuk menemukan makna dari data tekstual adalah dengan menggunakan fitur analisis teks lanjutan, seperti pencarian teks lengkap PostgreSQL.

Dalam bab ini, Anda akan mempelajari cara menggunakan SQL untuk menganalisis dan mengubah teks. Anda akan mulai dengan pertengkaran teks sederhana menggunakan pemformatan string dan pencocokan pola sebelum beralih ke fungsi analisis yang lebih lanjut. Kita akan menggunakan dua kumpulan data sebagai contoh: kumpulan kecil laporan kejahatan dari departemen sheriff di dekat Washington, DC, dan satu set pidato kenegaraan yang disampaikan oleh mantan presiden AS.

Memformat Teks Menggunakan Fungsi String

Baik Anda sedang mencari data dalam teks atau hanya ingin mengubah tampilannya dalam laporan, pertama-tama Anda harus mengubahnya menjadi format yang dapat Anda gunakan. PostgreSQL memiliki lebih dari 50 fungsi string bawaan yang menangani tugas rutin tetapi perlu Beberapa adalah bagian dari standar ANSI SQL, dan lainnya khusus untuk PostgreSQL. Anda akan menemukan daftar lengkap fungsi string di https://www.postgresql .org / docs / current / static / functions-string.html, tetapi di bagian ini kita akan memeriksa beberapa yang mungkin paling sering Anda gunakan.

Anda dapat menggunakan fungsi-fungsi ini di dalam berbagai kueri. Mari kita coba sekarang menggunakan kueri sederhana yang menempatkan fungsi setelah SELECT dan menjalankannya di Alat Kueri pgAdmin, seperti ini: SELECT upper ('hello') ;. Contoh setiap fungsi kode plus untuk semua daftar dalam bab ini tersedia di https://www.nostarch.com/practicalSQL/.

Pemformatan Kasus

Fungsi kapitalisasi memformat huruf besar/kecil teks. Fungsi atas (string) mengkapitalisasi semua karakter alfabet dari string yang diteruskan ke teks tersebut. Karakter nonalfabet, seperti angka, tetap tidak berubah. Misalnya, atas ('Neal7') mengembalikan NEAL7. Lebih

rendah (string) berfungsi huruf kecil semua karakter alfabet sambil menjaga karakter nonalfabet tidak berubah. Misalnya, lebih rendah ('Randy') mengembalikan randy.
Fungsi initcap (string) menggunakan huruf kapital pada huruf pertama dari setiap kata. Misalnya, initcap ('di penghujung hari') mengembalikan At The End Of The Day. Fungsi ini berguna untuk memformat judul buku atau film, tetapi karena Itu tidak mengenali akronim, itu tidak selalu merupakan solusi yang sempurna Misalnya, initcap ('Practical SQL') akan mengembalikan Sql Praktis, karena tidak mengenali SQL sebagai akronim.
Fungsi atas () dan bawah () adalah perintah standar SQL ANSI, tetapi initcap () khusus untuk PostgreSQL. Ketiga fungsi ini memberi Anda cukup opsi untuk mengerjakan ulang kolom teks menjadi huruf yang Anda inginkan. Perhatikan bahwa huruf besar tidak bekerja dengan semua lokal atau bahasa.

Informasi Karakter

Beberapa fungsi mengembalikan data tentang string daripada mengubahnya. Fungsi-fungsi ini berguna sendiri atau digabungkan dengan fungsi lain. Misalnya, fungsi char_length (string) mengembalikan jumlah karakter dalam string, termasuk spasi. Untuk contoh, fungsi char_length (string) mengembalikan jumlah karakter dalam string, termasuk spasi. contoh, char_length ('Pat') mengembalikan nilai 5, karena tiga huruf di Pat dan spasi di kedua ujungnya total lima karakter. Anda juga dapat menggunakan panjang (string) fungsi SQL non-ANSI untuk menghitung string, yang memiliki varian yang memungkinkan Anda menghitung panjang string biner.

Catatan: Fungsi length () dapat mengembalikan nilai yang berbeda dari char_length () saat digunakan dengan pengkodean multibyte, seperti kumpulan karakter yang mencakup bahasa Cina, Jepang, atau Korea.

Fungsi posisi (substring dalam string) mengembalikan lokasi karakter substring dalam string. Misalnya, posisi (',' dalam' Tan,Bella') mengembalikan 4, karena karakter koma dan spasi (,) yang ditentukan dalam substring yang dilewatkan sebagai parameter pertama dimulai dari posisi indeks keempat dalam string utama Tan, Bella. Baik char_length() dan position() berada dalam standar SQL ANSI.

Menghapus Karakter

Fungsi trim (karakter dari string) menghapus karakter yang tidak diinginkan dari string. Untuk mendeklarasikan satu atau lebih karakter yang akan dihapus, tambahkan ke fungsi diikuti dengan kata kunci dari dan string utama yang ingin Anda ubah. Opsi untuk menghapus karakter utama (di bagian depan string), karakter tambahan (di akhir string), atau keduanya membuat fungsi ini sangat fleksibel.
Misalnya, trim ('s' from'socks') menghapus semua karakter s dan mengembalikan ock. Untuk menghapus hanya s di akhir string, tambahkan kata kunci trailing sebelum karakter ke trim: trim (trailing's' dari 'socks' ') mengembalikan kaus kaki.

Jika Anda tidak menentukan karakter apa pun untuk dihapus, trim () menghapus spasi apa pun dalam string secara default. Misalnya, trim ('Pat') mengembalikan Pat tanpa spasi awal atau

akhir. Untuk mengonfirmasi panjang string yang dipangkas, kita dapat membuat sarang trim () di dalam char_length () seperti ini:

```
SELECT char_length(trim('Pat'));
```

Kueri ini harus mengembalikan 3, jumlah huruf dalam Pat, yang merupakan hasil trim ('Pat'). Fungsi ltrim (string, karakter) dan rtrim (string, karakter) adalah variasi khusus PostgreSQL dari fungsi trim(). Fungsi ini menghilangkan karakter dari ujung kiri atau kanan string. Misalnya, rtrim ('socks' ,'s') mengembalikan kaus kaki dengan hanya menghapus s di ujung kanan string.

Mengekstrak dan Mengganti Karakter

Fungsi kiri (string, angka) dan kanan (string, angka), keduanya standar ANSI SQL, mengekstrak dan mengembalikan karakter yang dipilih dari string. Misalnya, untuk mendapatkan kode area 703 saja dari nomor telepon 703-555-1212, gunakan kiri ('703-555-1212', 3) untuk menentukan bahwa Anda menginginkan tiga karakter pertama dari string dimulai dari kiri. Demikian pula, kanan ('703-555-1212', 8) mengembalikan delapan karakter dari kanan: 555-1212.

Untuk mengganti karakter dalam string, gunakan fungsi replace (string, from, to).Untuk mengubah bat menjadi cat, misalnya, Anda akan menggunakan replace ('bat','b','c') untuk menentukan yang Anda inginkan untuk mengganti b di kelelawar dengan c.

Sekarang setelah Anda mengetahui fungsi dasar untuk memanipulasi string, mari kita lihat cara mencocokkan pola yang lebih kompleks dalam teks dan mengubah pola tersebut menjadi data yang dapat kita analisis.

Mencocokkan Pola Teks dengan Ekspresi Reguler

Ekspresi reguler (atau regex) adalah jenis bahasa notasi yang menjelaskan pola teks. Jika Anda memiliki string dengan pola yang terlihat (misalnya, empat digit diikuti dengan tanda hubung, lalu dua digit lagi), Anda dapat menulis ekspresi reguler yang menjelaskan Anda kemudian dapat menggunakan notasi dalam klausa WHERE untuk memfilter baris menurut pola atau menggunakan fungsi ekspresi reguler untuk mengekstrak dan mengadu teks yang berisi pola yang sama.

Ekspresi reguler bisa tampak sulit dipahami oleh programmer pemula; mereka membutuhkan latihan untuk memahami karena mereka menggunakan simbol karakter tunggal yang tidak intuitif. Mendapatkan ekspresi untuk mencocokkan pola dapat melibatkan coba-coba, dan setiap bahasa pemrograman memiliki perbedaan halus dalam cara Ini menangani ekspresi reguler Namun, mempelajari ekspresi reguler adalah investasi yang baik karena Anda mendapatkan kemampuan seperti kekuatan super untuk mencari teks menggunakan banyak bahasa pemrograman, editor teks, dan aplikasi lainnya.

Di bagian ini, saya akan memberikan dasar-dasar ekspresi reguler yang cukup untuk mengerjakan latihan. Untuk mempelajari lebih lanjut, saya merekomendasikan penguji kode online interaktif, seperti https://regexr.com/ atau http://www.regexpal.com/ , yang memiliki referensi notasi.

Notasi Ekspresi Reguler

Pencocokan huruf dan angka menggunakan notasi ekspresi reguler adalah lurus ke depan karena huruf dan angka (dan simbol tertentu) adalah literal yang menunjukkan karakter yang sama, misalnya, Al cocok dengan dua karakter pertama di Alicia.

Untuk pola yang lebih kompleks, Anda akan menggunakan kombinasi elemen ekspresi reguler di Tabel 13.1.

Tabel 13.1: Dasar-dasar Notasi Ekspresi Reguler

Ekspresi	**Keterangan**
.	Titik adalah karakter pengganti yang menemukan karakter apa pun kecuali baris baru.
[FGz]	Setiap karakter dalam kurung siku Di sini, F, G, atau z.
[a-z]	Rentang karakter. Di sini, huruf kecil a sampai z.
[^ a-z]	Tanda sisipan meniadakan kecocokan. Di sini, bukan huruf kecil a sampai z.
\ w	Karakter kata apa pun atau garis bawah. Sama seperti [A-Za-z0-9_].
\ d	Angka apa saja.
\s	Sebuah spasi.
\t	Karakter tab.
\n	Karakter baris baru.
\r	Karakter carriage return.
^	Cocokkan di awal string.
$	Cocokkan di akhir string.
?	Dapatkan pertandingan sebelumnya nol atau satu kali.
*	Dapatkan pertandingan sebelumnya nol kali atau lebih.
+	Dapatkan pertandingan sebelumnya satu kali atau lebih.
{m}	Dapatkan kecocokan sebelumnya tepat m kali.
{m,n}	Dapatkan kecocokan sebelumnya antara m dan n kali.
a|b	Pipa menunjukkan pergantian. Temukan a atau b.
( )	Buat dan laporkan grup tangkapan atau atur prioritas.
(?:)	Meniadakan pelaporan grup penangkap.

Dengan menggunakan ekspresi reguler dasar ini, Anda dapat mencocokkan berbagai jenis karakter dan juga menunjukkan berapa kali dan di mana harus mencocokkannya. Misalnya, menempatkan karakter di dalam tanda kurung siku ([]) memungkinkan Anda mencocokkan karakter tunggal atau rentang apa pun. Jadi, [FGz] cocok dengan satu F, G, atau z, sedangkan [A-Za-z] akan cocok dengan huruf besar atau huruf kecil apa pun.

Garis miring terbalik (\) mendahului penanda untuk karakter khusus, seperti tab (\t), digit (\d), atau baris baru (\n), yang merupakan karakter akhir baris dalam file teks.

Ada beberapa cara untuk menunjukkan berapa kali untuk mencocokkan karakter. Menempatkan angka di dalam tanda kurung kurawal menunjukkan bahwa Anda ingin

mencocokkannya berkali-kali. Misalnya, \d{4} mencocokkan empat digit berturut-turut, dan \d{1,4} mencocokkan satu digit antara satu dan empat kali.

Karakter ?, *, dan + memberikan notasi singkatan yang berguna untuk jumlah kecocokan. Misalnya, tanda tambah (+) setelah karakter menunjukkan untuk mencocokkannya satu kali atau lebih. Jadi, ekspresi a+ akan menemukan karakter aa dalam string aardvark.

Selain itu, tanda kurung menunjukkan grup tangkapan, yang dapat Anda gunakan untuk menentukan hanya sebagian dari teks yang cocok untuk ditampilkan dalam hasil kueri. Ini berguna untuk melaporkan kembali hanya sebagian dari ekspresi yang cocok. Misalnya, jika Anda mencari format waktu HH:MM:SS dalam teks dan hanya ingin melaporkan jam, Anda dapat menggunakan ekspresi seperti (\d{2}):\d{2}:\d{ 2}. Ini mencari dua digit (\d{2}) dari jam diikuti oleh titik dua, dua digit lainnya untuk menit dan titik dua, dan kemudian dua digit detik. Dengan menempatkan \d{2} pertama di dalam tanda kurung, Anda hanya dapat mengekstrak dua digit tersebut, meskipun seluruh ekspresi cocok dengan waktu penuh.

Tabel 13.2 menunjukkan contoh menggabungkan ekspresi reguler untuk menangkap bagian yang berbeda dari kalimat “Permainan dimulai pukul 7 malam. pada 2 Mei 2019.”

Tabel 13.2: Contoh Pencocokan Ekspresi Reguler

Ekspresi	Apa yang cocok?	Hasil
.+	Karakter apa pun satu kali atau lebih	Permainan dimulai pukul 7 malam. pada 2 Mei 2019.
\d{1,2} (?:a.m.|p.m.)	Satu atau dua digit diikuti dengan spasi dan	jam 7 malam
^\w+	Satu atau lebih karakter kata di awal	NS
\w+.$	Satu atau lebih karakter kata diikuti oleh	2019.
Mei|Juni	Salah satu dari kata Mei atau Juni	Mungkin
\d{4}	Empat digit	2019
Mei \d, \d{4}	Dapat diikuti dengan spasi, angka, koma,	2 Mei 2019

Hasil ini menunjukkan kegunaan ekspresi reguler untuk memilih hanya bagian dari string yang menarik bagi kita. Misalnya, untuk mencari waktu, kita menggunakan ekspresi \d{1,2} (?:am|p.m.) untuk mencari satu atau dua digit karena waktu bisa berupa satu atau dua digit diikuti dengan spasi. Kemudian kita mencari baik a.m. atau p.m.; simbol pipa yang memisahkan istilah menunjukkan kondisi baik-atau, dan menempatkannya dalam tanda kurung memisahkan logika dari ekspresi lainnya. Kami membutuhkan simbol ?: untuk menunjukkan bahwa kami tidak ingin memperlakukan istilah di dalam tanda kurung sebagai grup tangkapan, yang akan melaporkan a.m. atau p.m. hanya. ?: memastikan bahwa kecocokan penuh akan dikembalikan.

Anda dapat menggunakan ekspresi reguler ini di pgAdmin dengan menempatkan teks dan ekspresi reguler di dalam fungsi substring (string dari pola) untuk mengembalikan teks yang cocok. Misalnya, untuk menemukan tahun empat digit, gunakan kueri berikut:

```
SELECT substring('the game stats at 7 p.m. on May 2, 2019.' From '\d{4};
```

Kueri ini harus mengembalikan 2019, karena kami menetapkan bahwa pola harus mencari digit apa pun yang panjangnya empat karakter, dan 2019 adalah satu-satunya digit dalam string ini yang cocok dengan kriteria ini. Anda dapat melihat contoh kueri substring() untuk semua contoh di Tabel 13.2 di sumber kode buku di https://www.nostarch.com/practicalSQL/.

Pelajaran di sini adalah bahwa jika Anda dapat mengidentifikasi pola dalam teks, Anda dapat menggunakan kombinasi simbol ekspresi reguler untuk menemukannya. Teknik ini sangat berguna ketika Anda memiliki pola berulang dalam teks yang ingin Anda ubah menjadi sekumpulan data untuk dianalisis. Mari berlatih bagaimana menggunakan fungsi ekspresi reguler menggunakan contoh dunia nyata.

Mengubah Teks menjadi Data dengan Fungsi Ekspresi Reguler

Departemen sheriff di salah satu pinggiran kota Washington, D.C. menerbitkan laporan harian yang merinci tanggal, waktu, lokasi, dan deskripsi insiden yang diselidiki departemen. Laporan-laporan ini akan sangat bagus untuk dianalisis, kecuali mereka memposting informasi dalam dokumen Microsoft Word yang disimpan sebagai file PDF, yang bukan format yang paling ramah untuk diimpor ke database.

Jika saya menyalin dan menempel insiden dari PDF ke editor teks, hasilnya adalah blok teks yang terlihat dalam sintaks dibawah ini:

```
4/16/17-4/17/17
2100-0900 hrs.
46000 Block Ashmere Sq.
Sterling
Larency: The victim reported that a
bicycle was stolen from their opened
garage door during the overnight hours.
C0170006614

04/10/17
1605 hrs.
21800 block Newlin Mill Rd.
Middleburg
Larceny: A License plate was reported
Stolen from a vehicle.
S0170006250
```

Setiap blok teks menyertakan tanggal , waktu , alamat jalan , kota atau kota kecil , jenis kejahatan, dan deskripsi kejadian . Informasi terakhir adalah kode yang mungkin merupakan ID unik untuk insiden tersebut, meskipun kami harus memeriksa dengan departemen sheriff untuk memastikannya. Ada sedikit inkonsistensi. Misalnya, blok teks pertama memiliki dua tanggal (4/16/17-4/17/17) dan dua kali (2100-0900 jam), yang berarti waktu pasti kejadian tidak diketahui dan kemungkinan terjadi dalam waktu tersebut. menjangkau. Blok kedua memiliki satu tanggal dan waktu.

Jika Anda menyusun laporan-laporan ini secara teratur, Anda dapat mengharapkan untuk menemukan beberapa wawasan bagus yang dapat menjawab pertanyaan-pertanyaan penting: Di mana kejahatan cenderung terjadi? Jenis kejahatan apa yang paling sering terjadi? Apakah mereka lebih sering terjadi pada akhir pekan atau hari kerja?

Sebelum Anda dapat mulai menjawab pertanyaan-pertanyaan ini, Anda harus mengekstrak teks ke dalam kolom tabel menggunakan ekspresi reguler.

Membuat Tabel untuk Laporan Kejahatan

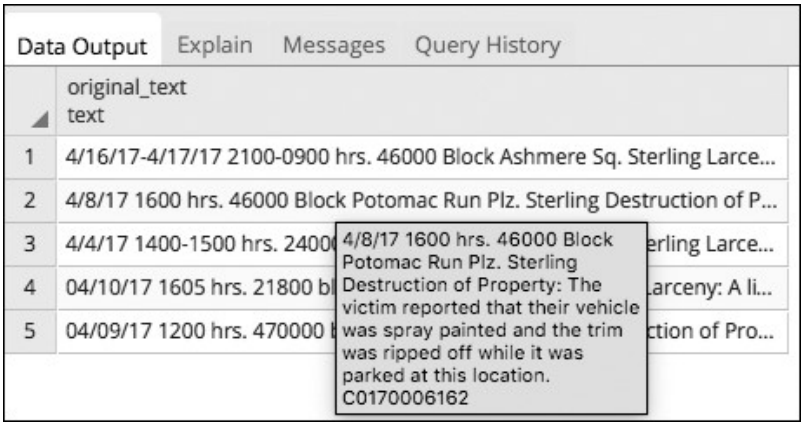
Saya telah mengumpulkan lima insiden kejahatan ke dalam file bernama crime_reports.csv yang dapat Anda unduh di https://www.nostarch.com/practicalSQL/. Unduh file dan simpan di komputer Anda. Kemudian gunakan kode di listing dibawah ini untuk membuat tabel yang memiliki kolom untuk setiap elemen data yang dapat Anda uraikan dari teks menggunakan ekspresi reguler.

```
CREATE TABLE crime_reports (
      crime_id bigserial PRIMARY KEY,
      date_1 timestamp with time zone,
      date_2 timestamp with time zone,
      street varchar(250),
      city varchar(100),
      crime_type varchar(100),
      description text,
      case_number varchar(50),
      original_text text NOT NULL
);

COPY crime_reports (original_text)
FROM 'C\:YourDirectory\crime_reports.csv'
WITH (FORMAT CSV, HEADER OFF, QUOTE '"');
```

Jalankan pernyataan CREATE TABLE di lalu gunakan COPY untuk memuat teks ke dalam kolom original_text. Sisa kolom akan menjadi NULL sampai kita mengisinya.

Saat Anda menjalankan SELECT original_text FROM crime_reports; di pgAdmin, kisi hasil harus menampilkan lima baris dan beberapa kata pertama dari setiap laporan. Saat Anda mengarahkan kursor ke sel mana pun, pgAdmin menampilkan semua teks di baris itu, seperti yang ditunjukkan pada Gambar 13.1.



Gambar 13.1: Menampilkan teks tambahan di kisi hasil pgAdmin

Sekarang setelah Anda memuat teks yang akan Anda parsing, mari jelajahi data ini menggunakan fungsi ekspresi reguler PostgreSQL.

Pencocokan Pola Tanggal Laporan Kejahatan

Bagian pertama dari data yang ingin kami ekstrak dari laporan original_text adalah tanggal atau tanggal kejahatan. Sebagian besar laporan memiliki satu tanggal, meskipun satu memiliki dua. Laporan juga memiliki waktu yang terkait, dan kami akan menggabungkan tanggal dan waktu yang diekstraksi menjadi stempel waktu. Kami akan mengisi date_1 dengan tanggal dan waktu pertama (atau satu-satunya) di setiap laporan. Jika ada tanggal kedua atau kedua kalinya, kami akan membuat stempel waktu dan menambahkannya ke date_2.

Untuk mengekstrak data, kita akan menggunakan fungsi regexp_match(string, pattern), yang mirip dengan substring() dengan beberapa pengecualian. Salah satunya adalah mengembalikan setiap kecocokan sebagai teks dalam array. Juga, jika tidak ada kecocokan, ia mengembalikan NULL. Seperti yang mungkin Anda ingat dari Bab 5, array adalah daftar elemen; dalam satu latihan, Anda menggunakan larik untuk meneruskan daftar nilai ke fungsi persentil_cont() untuk menghitung kuartil. Saya akan menunjukkan kepada Anda cara bekerja dengan hasil yang kembali sebagai larik saat kami mengurai laporan kejahatan.

Catatan : Fungsi regexp_match() diperkenalkan di PostgreSQL 10 dan tidak tersedia di versi sebelumnya.

Untuk memulai, mari gunakan regexp_match() untuk menemukan tanggal di masing-masing dari lima insiden di crime_reports. Pola umum yang cocok adalah MM/DD/YY, meskipun mungkin ada satu atau dua digit untuk bulan dan tanggal. Berikut ekspresi reguler yang cocok dengan polanya:

```
\d{1,2}\/\d{1,2}\/\d{2}
```

Dalam ekspresi ini, \d{1,2} menunjukkan bulan. Angka-angka di dalam kurung kurawal menentukan bahwa Anda menginginkan setidaknya satu digit dan paling banyak dua digit.

Selanjutnya, Anda ingin mencari garis miring (/), tetapi karena garis miring dapat memiliki arti khusus dalam ekspresi reguler, Anda harus menghindari karakter itu dengan menempatkan garis miring terbalik (\) di depannya, seperti ini \/ . Melarikan diri dari karakter dalam konteks ini berarti kita ingin memperlakukannya sebagai literal daripada membiarkannya mengambil makna khusus. Jadi, kombinasi garis miring terbalik dan garis miring (\/) menunjukkan Anda menginginkan garis miring.

\d{1,2} lainnya mengikuti untuk satu atau dua digit hari dalam sebulan. Ekspresi diakhiri dengan garis miring kedua lolos dan \d{2} untuk menunjukkan tahun dua digit. Mari kita meneruskan ekspresi \d{1,2}\/\d{1,2}\/\d{2} ke regexp_match(), seperti yang ditunjukkan pada sintaks dibawah ini:

```
SELECT crime_id
      regexp_match(original_text, '\d{1,2}\/\d{1,2}\/\d{2}')
      FROM crime_report;
```

Jalankan kode itu di pgAdmin, dan hasilnya akan terlihat seperti ini:

```
crime_id    regexp_match
--------    ------------
1           {4/16/17}
2           {4/8/17}
3           {4/4/17}
4           {04/10/17}
5           {04/09/17}
```

Perhatikan bahwa setiap baris menunjukkan tanggal pertama yang terdaftar untuk insiden tersebut, karena regexp_match() mengembalikan kecocokan pertama yang ditemukannya secara default. Perhatikan juga bahwa setiap tanggal diapit tanda kurung kurawal. Itu PostgreSQL yang menunjukkan bahwa regexp_match() mengembalikan setiap hasil dalam array, atau daftar elemen. Saya akan menunjukkan cara mengakses elemen-elemen tersebut dari array. Anda juga dapat membaca lebih lanjut tentang menggunakan array di PostgreSQL di
https://www.postgresql.org/docs/current/ static/arrays.html.

Mencocokkan Kencan Kedua Saat Hadir

Kami telah berhasil mengekstrak tanggal pertama dari setiap laporan. Tapi ingat bahwa salah satu dari lima insiden memiliki kencan kedua. Untuk menemukan dan menampilkan semua tanggal dalam teks, Anda harus menggunakan fungsi regexp_matches() terkait dan memasukkan opsi dalam bentuk flag g, seperti yang ditunjukkan pada kode dibawah ini.

```
SELECT crime_id
      regexp_matches(original_text, '\d{1,2}\/\d{1,2}\/\d{2}', 'g')
FROM crime_report;
```

Fungsi regexp_matches() , ketika diberikan flag g , berbeda dari regexp_match() dengan mengembalikan setiap kecocokan yang ditemukan ekspresi sebagai baris dalam hasil daripada hanya mengembalikan kecocokan pertama saja.

Jalankan kode lagi dengan revisi ini; Anda sekarang akan melihat dua tanggal untuk insiden yang memiliki crime_id 1, seperti ini:

```
crime_id    regexp_matches
--------    --------------
1           {4/16/17}
1           {4/17/17}
2           {4/8/17}
3           {4/4/17}
4           {04/10/17}
5           {04/09/17}
```

Setiap kali laporan kejahatan memiliki tanggal kedua, kami ingin memuatnya dan waktu terkait ke dalam kolom date_2. Meskipun menambahkan flag g menunjukkan kepada kita semua tanggal, untuk mengekstrak hanya tanggal kedua dalam laporan, kita dapat menggunakan pola yang selalu kita lihat ketika ada dua tanggal. Di kode, blok teks pertama menunjukkan dua tanggal yang dipisahkan oleh tanda hubung, seperti ini:

```
4/16/17-4/17/17
```

Ini berarti Anda dapat beralih kembali ke regexp_match() dan menulis ekspresi reguler untuk mencari tanda hubung diikuti dengan tanggal,

```
SELECT crime_id
      regexp_matches(original_text, '\d{1,2}\/\d{1,2}\/\d{2}')
FROM crime_report;
```

Meskipun kueri ini menemukan tanggal kedua di item pertama (dan mengembalikan NULL untuk sisanya), ada konsekuensi yang tidak diinginkan: ini menampilkan tanda hubung bersamanya.

```
crime_id    regexp_matches
--------    --------------
1           {-4/17/17}
2
3
4
5
```

Anda tidak ingin menyertakan tanda hubung, karena ini adalah format yang tidak valid untuk tipe data stempel waktu. Untungnya, Anda dapat menentukan bagian yang tepat dari ekspresi reguler yang ingin Anda kembalikan dengan menempatkan tanda kurung di sekitarnya untuk membuat grup tangkapan, seperti ini:

```
-(\d{1,2}\/\d{1,2}\/\d{1,2})
```

Notasi ini hanya mengembalikan bagian dari ekspresi reguler yang Anda inginkan. Jalankan kueri yang dimodifikasi untuk hanya melaporkan data dalam tanda kurung.

```
SELECT crime_id
      regexp_match(original_text, '-(\d{1,2}\/\d{1,2}\/\d{1,2})')
FROM crime_reports;
```

Kueri di diatas harus mengembalikan hanya tanggal kedua tanpa tanda hubung di depan, seperti yang ditunjukkan di sini:

```
crime_id    regexp_matches
--------    --------------
1           {4/17/17}
2
3
4
5
```

Proses yang baru saja Anda selesaikan adalah tipikal. Anda mulai dengan teks untuk dianalisis, lalu menulis dan menyaring ekspresi reguler hingga menemukan data yang Anda inginkan. Sejauh ini, kami telah membuat ekspresi reguler untuk mencocokkan kencan pertama dan kencan kedua, jika ada. Sekarang, mari gunakan ekspresi reguler untuk mengekstrak elemen data tambahan.

Mencocokkan Elemen Laporan Kejahatan Tambahan

Di bagian ini, kami akan menangkap waktu, alamat, jenis kejahatan, deskripsi, dan nomor kasus dari laporan kejahatan. Berikut adalah ekspresi untuk menangkap informasi ini:

Jam pertama \/\d{2}\n(\d{4})

Jam pertama, yaitu jam terjadinya kejahatan atau awal rentang waktu, selalu mengikuti tanggal dalam setiap laporan kejahatan, seperti ini:

```
4/17/17-4/17/17
2100-0900 hrs.
```

Untuk menemukan jam pertama, kita mulai dengan tebasan maju yang lolos dan \d{2}, yang mewakili tahun dua digit sebelum tanggal pertama (17). Karakter \n menunjukkan baris baru karena jam selalu dimulai pada baris baru, dan \d{4} mewakili empat digit jam (2100). Karena kami hanya ingin mengembalikan empat digit, kami menempatkan \d{4} di dalam tanda kurung sebagai grup tangkap.

Jam kedua \/\d{2}\n\d{4}-(\d{4})

Jika jam kedua ada, itu akan mengikuti tanda hubung, jadi kami menambahkan tanda hubung dan \d{4} lain ke ekspresi yang baru saja kami buat untuk jam pertama. Sekali lagi, \d{4} kedua masuk ke dalam grup tangkapan, karena 0900 adalah satu-satunya jam yang ingin kami kembalikan.

Jam jalan.\n(\d+ .+(?:Sq.|Plz.|Dr.|Ter.|Rd.))

Dalam data ini, jalan selalu mengikuti waktu jam. penunjukan dan baris baru (\n), seperti ini:

```
4/10/17
2100-0900 hrs.
1605 hrs.
21800 block Newlin Mill Rd.
```

Alamat jalan selalu dimulai dengan beberapa nomor yang panjangnya bervariasi dan diakhiri dengan beberapa jenis akhiran yang disingkat. Untuk menggambarkan pola ini, kami menggunakan \d+ untuk mencocokkan digit apa pun yang muncul satu kali atau lebih.

Kemudian kami menentukan spasi dan mencari karakter apa pun satu kali atau lebih menggunakan titik wildcard dan notasi tanda plus (.+). Ekspresi diakhiri dengan serangkaian istilah yang dipisahkan oleh simbol pipa bergantian yang terlihat seperti ini: (?:Sq.|Plz.|Dr.|Ter.|Rd.).

Istilahnya berada di dalam tanda kurung, jadi ekspresinya akan cocok dengan salah satu atau beberapa istilah tersebut. Saat kita mengelompokkan istilah seperti ini, jika kita tidak ingin tanda kurung berfungsi sebagai grup tangkap, kita perlu menambahkan ?: untuk meniadakan efek tersebut.

Catatan: kumpulan data yang besar, kemungkinan nama jalan raya akan diakhiri dengan sufiks di luar lima dalam ekspresi reguler kami. Setelah membuat pass awal saat mengekstrak jalan, Anda dapat menjalankan kueri untuk memeriksa baris yang tidak cocok untuk menemukan sufiks tambahan yang cocok.

Kota (?:Sq.|Plz.|Dr.|Ter.|Rd.)\n(\w+ \w+|\w+)\n
Karena kota selalu mengikuti akhiran jalan, kami menggunakan kembali istilah yang dipisahkan oleh simbol pergantian yang baru saja kami buat untuk jalan. Kami mengikutinya dengan baris baru (\n) dan kemudian menggunakan grup tangkap untuk mencari dua kata atau satu kata (\w+ \w+|\w+) sebelum baris baru terakhir, karena nama kota atau kota bisa lebih dari satu kata.

Jenis kejahatan \n(?:\w+ \w+|\w+)\n(.*):
Jenis kejahatan selalu mendahului titik dua (satu-satunya saat titik dua digunakan dalam setiap laporan) dan mungkin terdiri dari satu atau lebih kata, seperti ini:

```
--snip--
Middleburg
Larceny: A License plate was reported
stolen from a vehicle.
S0170006250
--snip--
```

Untuk membuat ekspresi yang cocok dengan pola ini, kami mengikuti baris baru dengan grup tangkapan nonreporting yang mencari kota satu atau dua kata. Kemudian kami menambahkan baris baru lain dan mencocokkan karakter apa pun yang muncul nol kali atau lebih sebelum titik dua menggunakan (.*):.

Keterangan :\s(.+)(?:C0|SO)
Deskripsi kejahatan selalu berada di antara titik dua setelah jenis kejahatan dan nomor kasus. Ekspresi dimulai dengan titik dua, karakter spasi (\s), lalu grup tangkap untuk menemukan karakter apa pun yang muncul satu kali atau lebih menggunakan notasi .+. Grup penangkap nonreporting (?:C0|SO) memberi tahu program untuk berhenti mencari ketika menemukan C0 atau SO, dua pasangan karakter yang memulai setiap nomor kasus (a C diikuti oleh nol, dan S diikuti oleh a modal O). Kita harus melakukan ini karena deskripsi mungkin memiliki satu atau lebih jeda baris.

Nomor kasus (?:C0|SO)[0-9]+

Nomor kasus dimulai dengan C0 atau SO, diikuti dengan serangkaian digit. Untuk mencocokkan pola ini, ekspresi mencari C0 atau SO dalam grup tangkapan non-pelaporan diikuti oleh digit apa pun dari 0 hingga 9 yang muncul satu kali atau lebih menggunakan notasi rentang [0-9].

Sekarang mari kita berikan ekspresi reguler ini ke regexp_match() untuk melihatnya beraksi. Kueri dibawah ini menunjukkan contoh kueri regexp_match() yang mengambil nomor kasus, kencan pertama, jenis kejahatan, dan kota:

```
SELECT
      regexp_match(original_text, '(?:CO|SO)[0-9]+') AS case_number,
      regexp_match(original_text, '\d{1,2}\/\d{1,2}/\/d{2}') AS date_1,
      regexp_match(original_text, '\n(?:\w+ \w+|\w+)\n(.*):') AS crime_type,
      regexp_match(original_text,  '(?:Sq.|Plz.|Dr.|Ter.|Rd.)\n(\w+  \w+|\w+)\n')
            AS city
FROM crime_reports;
.
```

Jalankan kodenya, dan hasilnya akan terlihat seperti ini:

```
case_number       date_1           crime_type                  city
-------------     ----------       -------------------------   ------------
{C0170006614}     {4/16/17}        {Larceny}                   {Sterling}
{C0170006162}     {4/8/17}         {Destruction of Property}   {Sterling}
{C0170006079}     {4/4/17}         {Larceny}                   {Sterling}
{SO170006250}     {04/10/17}       {Larceny}                   {Middleburg}
{SO170006211}     {04/09/17}       {Destruction of Property}   {Sterling}
```

Setelah semua pertengkaran itu, kami telah mengubah teks menjadi struktur yang lebih cocok untuk analisis. Tentu saja, Anda harus memasukkan lebih banyak insiden untuk menghitung frekuensi jenis kejahatan menurut kota atau jumlah kejahatan per bulan untuk mengidentifikasi tren apa pun.

Untuk memuat setiap elemen yang diurai ke dalam kolom tabel, kami akan membuat kueri UPDATE. Tetapi sebelum Anda dapat menyisipkan teks ke dalam kolom, Anda harus mempelajari cara mengekstrak teks dari larik yang dikembalikan oleh regexp_match().

Mengekstrak Teks dari regexp_match() Hasil

Dalam "Mencocokkan Pola Tanggal Laporan Kejahatan" pada halaman 218, saya menyebutkan bahwa regexp_match() mengembalikan array yang berisi nilai teks. Dua petunjuk mengungkapkan bahwa ini adalah nilai teks. Yang pertama adalah bahwa penunjukan tipe data di header kolom menunjukkan teks[] alih-alih teks. Yang kedua adalah bahwa setiap hasil dikelilingi oleh kurung kurawal. Gambar 13.2 menunjukkan bagaimana pgAdmin menampilkan hasil kuer.

Data Output | Explain | Messages | Query History

	case_number text[]	date_1 text[]	crime_type text[]	city text[]
1	{C0170006614}	{4/16/17}	{Larceny}	{Sterling}
2	{C0170006162}	{4/8/17}	{Destruction of Property}	{Sterling}
3	{C0170006079}	{4/4/17}	{Larceny}	{Sterling}
4	{SO170006250}	{04/10/17}	{Larceny}	{Middleburg}
5	{SO170006211}	{04/09/17}	{Destruction of Property}	{Sterling}

Gambar 13.2: Nilai larik di kisi hasil pgAdmin

Kolom crime_reports yang ingin kita perbarui bukanlah tipe array, jadi daripada meneruskan nilai array yang dikembalikan oleh regexp_match(), kita perlu mengekstrak nilai dari array terlebih dahulu. Kami melakukan ini dengan menggunakan notasi array, seperti yang ditunjukkan:

```
SELECT
      crime_id
      regexp_match(original_text, '(?:C0|SO)[0-9]+'))[1] AS case_number
FROM crime_reports;
```

Koding ditersebut menambahkan Array. Pertama, kita membungkus fungsi regexp_match() dalam tanda kurung. Kemudian, pada akhirnya, kami memberikan nilai 1, yang mewakili elemen pertama dalam array, diapit dalam tanda kurung siku . Kueri harus menghasilkan hasil berikut:

```
crime_id          case_number
--------          -----------
      1           C0170006614
      2           C0170006162
      3           C0170006079
      4           SO170006250
      5           SO170006211
```

Sekarang penunjukan tipe data di tajuk kolom pgAdmin harus menampilkan teks alih-alih teks[], dan nilainya tidak lagi diapit tanda kurung kurawal. Kami sekarang dapat memasukkan nilai-nilai ini ke dalam crime_reports menggunakan kueri UPDATE.

Memperbarui Tabel crime_reports dengan Data yang Diekstraksi

Dengan setiap elemen yang saat ini tersedia sebagai teks, kami dapat memperbarui kolom di tabel crime_reports dengan data yang sesuai dari laporan kejahatan asli. Untuk memulai, Sintaks dibawah ini menggabungkan tanggal dan waktu pertama yang diekstrak menjadi nilai stempel waktu tunggal untuk kolom date_1.

```
UPDATE crime_reports
SET date_1 = (
      (regexp_match(original_text, '\d{1,2}\/\d{1,2}\/\d{2}'))[1]||' ' ||
      (regexp_match(original_text, '\/\d{2}\n(\d{4}'))[1]||' US/Eastern'
)::timestamptz;
SELECT crime_id
      date_1
      original_text
FROM crime_reports;
```

Karena kolom date_1 bertipe timestamp, kita harus memberikan input dalam tipe data tersebut. Untuk melakukannya, kami akan menggunakan operator penggabungan pipa ganda (||) PostgreSQL untuk menggabungkan tanggal dan waktu yang diekstraksi dalam format yang dapat diterima untuk stempel waktu dengan input zona waktu. Dalam klausa SET , kita mulai dengan pola regex yang cocok dengan tanggal pertama . Selanjutnya, kami menggabungkan tanggal dengan spasi menggunakan dua tanda kutip tunggal dan ulangi operator penggabungan. Langkah ini menggabungkan tanggal dengan spasi sebelum menghubungkannya ke pola regex yang cocok dengan waktu . Kemudian kami memasukkan zona waktu untuk area Washington, D.C., dengan menggabungkannya di akhir string menggunakan penunjukan AS/Timur.

Menggabungkan elemen-elemen ini membuat string dalam pola MM/DD/YY HHMM TIMEZONE, yang dapat diterima sebagai input stempel waktu. Kami melemparkan string ke stempel waktu dengan tipe data zona waktu menggunakan steno titik dua PostgreSQL dan singkatan stempel waktu.

Saat Anda menjalankan bagian UPDATE dari kode, PostgreSQL akan mengembalikan pesan UPDATE 5. Menjalankan pernyataan SELECT di pgAdmin akan menampilkan kolom date_1 yang sekarang terisi di samping bagian dari kolom original_text, seperti ini:

```
crime_id    date_1                      original_text
--------    ----------------------      ----------------------------------
1           2017-04-16 21:00:00-04      4/16/17-4/17/17 2100-0900 hrs. 460
2           2017-04-08 16:00:00-04      4/8/17 1600 hrs. 46000 Block Potom
3           2017-04-04 14:00:00-04      4/4/17 1400-1500 hrs. 24000 Block
4           2017-04-10 16:05:00-04      04/10/17 1605 hrs. 21800 block New
5           2017-04-09 12:00:00-04      04/09/17 1200 hrs. 470000 block Fa
```

Sekilas, Anda dapat melihat bahwa date_1 secara akurat menangkap tanggal dan waktu pertama yang muncul dalam teks asli dan memasukkannya ke dalam format yang dapat digunakan yang dapat kami analisis. Perhatikan bahwa jika Anda tidak berada di zona waktu Timur, stempel waktu akan mencerminkan zona waktu klien pgAdmin Anda. Seperti yang Anda pelajari di “Mengatur Zona Waktu” di halaman 178, Anda dapat menggunakan perintah SET zona waktu KE 'AS/Timur'; untuk mengubah klien untuk mencerminkan waktu Timur.

Menggunakan CASE untuk Menangani Instance Khusus

Anda dapat menulis pernyataan UPDATE untuk setiap elemen data yang tersisa, tetapi menggabungkan pernyataan tersebut menjadi satu akan lebih efisien. Kueri dibawah ini untuk memperbarui semua kolom crime_reports menggunakan satu pernyataan sambil menangani nilai yang tidak konsisten dalam data.

```
UPDATE crime_reports
SET date_1,
      (
            (regexp_match(original_text, '\d{1,2}\/\d{1,2}\/\d{2}'))[1]
                  ||' ||
            (regexp_match(original_text, '\/\d{2}\n(\d{4})'))[1]
                  ||'US/Eastern'
      )::timestampz,
```

```
date_2 *
CASE
WHEN (SELECT regexp_match(original_text, '-(\d{1,2}\/\d{1,2}\/\d{1,2})') IS NULL)
AND (SELECT regexp_match(original_text, '/\/d{2}\n\d{4}-(\d{4})') IS NOT NULL)
THEN
      ((regexp_match(original_text, '\d{1,2}\/\d{1,2}\/\d{2}'))[1] ||' '||
      (regexp_match(original_text, '/\/d{2}\n\d{4}-(\d{4})'))[1] ||'US/Eastern'
      )::timestampz

      WHEN (SELECT regexp_match(original_text, '-(\d{1,2}\/\d{1,2}\/\d{1,2})') IS
            NOT NULL
            AND (SELECT regexp_match(original_text, '\/\d{2}\n\d{4}-(\d{4})') IS
      NOT NULL

      THEN
            ((regexp_match(original_text, '-(\d{1,2}\/\d{1,2}\/\d{1,2})'))[1]
            || ' ' ||
            (regexp_match(original_text, '\/\d{2}\n\d{4}-(\d{4})'))[1]
            ||'US/Eastern'
      ))::timestampz

      ELSE NULL
      END,
      street = (regexp_match(original_text,
            'hrs.\n(\d+ .+(?:Sq.|Plz.\Dr.|Ter.|Rd.))'))[1],
      city = (regexp_match(original_text,
            '(?:Sq.|Plz.|Dr.|Ter.|Rd.)\n(\w+ \w+ |\w+)\n))[1],
      crime_type = (regexp_match(original_text, '\n(?:\w+ \w+|\w+)\n(.*):'))[1],
      description = (regexp_match(original_text, ':\s(.+)(?:CO|SO)'))[1],
      case_number = (regexp_match(original_text, '(?:CO|SO)[0-9]+'))[1];
```

Pernyataan UPDATE ini mungkin terlihat menakutkan, tetapi tidak jika kita memecahnya berdasarkan kolom. Pertama, kami menggunakan kode yang sama untuk memperbarui kolom date_1 . Tetapi untuk memperbarui date_2 , kita perlu memperhitungkan kehadiran tanggal dan waktu kedua yang tidak konsisten. Dalam kumpulan data kami yang terbatas, ada tiga kemungkinan:

1. Jam kedua ada tapi bukan kencan kedua. Ini terjadi ketika laporan mencakup rentang jam pada satu tanggal.
2. Kencan kedua dan jam kedua ada. Ini terjadi ketika laporan mencakup lebih dari satu tanggal.
3. Baik kencan kedua maupun jam kedua tidak ada.

Kami menggunakan serangkaian pernyataan WHEN ... THEN untuk memeriksa dua kondisi pertama dan memberikan nilai untuk disisipkan; jika kondisi tidak ada, kami menggunakan kata kunci ELSE untuk memberikan NULL.

Pernyataan WHEN pertama memeriksa apakah regexp_match() mengembalikan NULL untuk tanggal kedua dan nilai untuk jam kedua (menggunakan IS NOT NULL). Jika kondisi tersebut bernilai benar, pernyataan THEN menggabungkan tanggal pertama dengan jam kedua untuk membuat stempel waktu pembaruan.

Pernyataan WHEN kedua memeriksa bahwa regexp_match() mengembalikan nilai untuk jam kedua dan tanggal kedua. Jika benar, pernyataan THEN menggabungkan tanggal kedua dengan jam kedua untuk membuat stempel waktu.

Jika tidak satu pun dari dua pernyataan WHEN kembali benar, pernyataan ELSE memberikan NULL untuk pembaruan karena hanya ada kencan pertama dan pertama kali.

Catatan : Pernyataan WHEN menangani kemungkinan yang ada dalam kumpulan data sampel kecil kami. Jika Anda bekerja dengan lebih banyak data, Anda mungkin perlu menangani variasi tambahan, seperti kencan kedua tetapi tidak untuk kedua kalinya.

Saat kita menjalankan kueri lengkap didiatas tadi, PostgreSQL akan melaporkan UPDATE 5. Sukses! Sekarang kita telah memperbarui semua kolom dengan data yang sesuai sambil menghitung elemen yang memiliki data tambahan, kita dapat memeriksa semua kolom tabel dan menemukan elemen yang diurai dari original_text. 11 menanyakan empat kolom:

```
SELECT date_1
      street,
      city,
      crime_type
FROM crime_reports;
```

Hasil kueri akan menampilkan kumpulan data yang terorganisir dengan baik yang terlihat seperti ini:

```
date_1                 street                            city        crime_type
---------------------  --------------------------------  --------    ----------------
2017-04-16 21:00:00-04  46000 Block Ashmere Sq.           Sterling    Larceny
2017-04-08 16:00:00-04  46000 Block Potomac Run Plz.      Sterling    Destruction of..
2017-04-04 14:00:00-04  24000 Block Hawthorn Thicket Ter.  Sterling    Larceny
2017-04-10 16:05:00-04  21800 block Newlin Mill Rd.       Middleburg  Larceny
2017-04-09 12:00:00-04  470000 block Fairway Dr.          Sterling    Destruction of..
```

Anda telah berhasil mengubah teks mentah menjadi tabel yang dapat menjawab pertanyaan dan mengungkapkan alur cerita tentang kejahatan di area ini.

Nilai Proses

Menulis ekspresi reguler dan mengkode kueri untuk memperbarui tabel bisa memakan waktu, tetapi ada nilai untuk mengidentifikasi dan mengumpulkan data dengan cara ini. Faktanya, beberapa kumpulan data terbaik yang akan Anda temui adalah yang Anda buat sendiri. Semua orang dapat mengunduh kumpulan data yang sama, tetapi kumpulan data yang Anda buat adalah milik Anda sendiri. Anda bisa menjadi orang pertama yang menemukan dan menceritakan kisah di balik data.

Selain itu, setelah Anda menyiapkan database dan kueri, Anda dapat menggunakannya lagi dan lagi. Dalam contoh ini, Anda dapat mengumpulkan laporan kejahatan setiap hari (baik dengan tangan atau dengan mengotomatiskan unduhan menggunakan bahasa pemrograman seperti Python) untuk kumpulan data berkelanjutan yang dapat Anda gali terus-menerus untuk tren.

Di bagian berikutnya, kita akan menyelesaikan eksplorasi ekspresi reguler kita menggunakan fungsi PostgreSQL tambahan.

Menggunakan Ekspresi Reguler dengan WHERE

Anda telah memfilter kueri menggunakan LIKE dan ILIKE di klausa WHERE. Di bagian ini, Anda akan belajar menggunakan ekspresi reguler dalam klausa WHERE sehingga Anda dapat melakukan pencocokan yang lebih kompleks.

Kami menggunakan tilde (~) untuk membuat kecocokan case-sensitive pada ekspresi reguler dan tilde-asterisk (~*) untuk melakukan kecocokan case-insensitive. Anda dapat meniadakan salah satu ekspresi dengan menambahkan tanda seru di depan.

Misalnya, !~* menunjukkan tidak cocok dengan ekspresi reguler yang tidak peka huruf besar-kecil. Listing dibawah ini menunjukkan cara kerjanya menggunakan tabel Sensus 2010 us_counties_2010 dari latihan sebelumnya:

```
SELECT geo_name
FROM us_counties_2010
WHERE geo_name ~* '(.+lade.+|.+lare.+)'
ORDER BY geo_name;

SELECT geo_name
FROM us_counties_2010
WHERE geo_name ~*' .+ash.+' AND geo_name !~ 'Wash.+'
ORDER BY geo_name;
```

Klausa WHERE pertama menggunakan tilde-asterisk (~*) untuk melakukan pencocokan case-insensitive pada ekspresi reguler (.+lade.+|.+lare.+) untuk menemukan nama daerah yang mengandung huruf lade atau mencolok di antara karakter lainnya. Hasilnya harus menunjukkan delapan baris:

```
geo_name
------------------
Bladen County
Clare County
Clarendon County
Glades County
Langlade County
Philadelphia County
Talladega County
Tulare County
```

Seperti yang Anda lihat, nama daerah menyertakan huruf lade atau lare di antara karakter lain. Klausa WHERE kedua menggunakan tilde-asterisk (~*) serta tilde yang dinegasikan (!~) untuk menemukan nama daerah yang mengandung huruf ash tetapi tidak termasuk yang dimulai dengan Wash. Kueri ini harus mengembalikan yang berikut:

```
geo_name
--------------
Nash County
Wabash County
Wabash County
Wabasha County
```

Keempat kabupaten dalam keluaran ini memiliki nama yang mengandung huruf ash tetapi tidak dimulai dengan Wash.

Ini adalah contoh yang cukup sederhana, tetapi Anda dapat melakukan pencocokan yang lebih kompleks menggunakan ekspresi reguler yang tidak dapat Anda lakukan dengan wildcard yang tersedia hanya dengan LIKE dan ILIKE.

Fungsi Ekspresi Reguler Tambahan

Mari kita lihat tiga fungsi ekspresi reguler lainnya yang mungkin berguna bagi Anda saat bekerja dengan teks. Sintaks dibawah ini menunjukkan beberapa fungsi ekspresi reguler yang menggantikan dan memisahkan teks:

```
SELECT regexp_replace('05/12/2008', '\d{4}', '2007');

SELECT regexp_split_to_table('Four,score,and,seven,years,ago', ',');

SELECT regexp_split_to_array('Phil Mike Tony Steve', ',');
```

Fungsi regexp_replace(string, pattern, replacement text) memungkinkan Anda mengganti pola yang cocok dengan teks pengganti. Dalam contoh kami mencari string tanggal 12/05/2018 untuk setiap rangkaian empat digit berturut-turut menggunakan \d{4}. Saat ditemukan, kami menggantinya dengan teks pengganti 2017. Hasil kueri tersebut adalah 05/12/2017 dikembalikan sebagai teks.

Fungsi regexp_split_to_table(string, pattern) membagi teks yang dibatasi menjadi beberapa baris. Listing dibawah ini menggunakan fungsi ini untuk memisahkan string 'Empat,skor,dan,tujuh,tahun,lalu' pada koma , menghasilkan satu set baris yang memiliki satu kata di setiap baris:

```
regexp_split_to_table
---------------------
Four
Score
And
Seven
Years
Ago
```

Ingatlah fungsi ini saat Anda mengerjakan latihan "Coba Sendiri" di akhir bab ini. Fungsi regexp_split_to_array(string, pattern) membagi teks yang dibatasi menjadi sebuah array. Contoh membagi string Phil Mike Tony Steve pada spasi, mengembalikan array teks yang akan terlihat seperti ini di pgAdmin:

```
regexp_split_to_array
----------------------
{Phil,Mike,Tony,Steve}
```

Notasi text[] di header kolom pgAdmin bersama dengan tanda kurung kurawal di sekitar hasil menegaskan bahwa ini memang tipe array, yang menyediakan cara analisis lain. Misalnya, Anda kemudian dapat menggunakan fungsi seperti array_length() untuk menghitung jumlah kata.

```
SELECT array_length(regexp_split_to_array('Phil Mike Tony Steve', ' '),1);
```

Kueri harus mengembalikan karena empat elemen ada dalam larik ini. Anda dapat membaca lebih lanjut tentang array_length() dan fungsi array lainnya di https://www.postgresql.org/docs/current/static/functions-array.html.

Pencarian Teks Lengkap di PostgreSQL

PostgreSQL hadir dengan mesin pencari teks lengkap yang kuat yang memberi Anda lebih banyak opsi saat mencari informasi dalam jumlah besar teks. Anda sudah familiar dengan Google atau mesin pencari web lainnya dan teknologi serupa yang mendukung pencarian di situs web berita atau database penelitian, seperti LexisNexis. Meskipun implementasi dan kemampuan pencarian teks lengkap memerlukan beberapa bab, di sini saya akan memandu Anda melalui contoh sederhana menyiapkan tabel untuk pencarian teks dan fungsi untuk pencarian menggunakan PostgreSQL.

Untuk contoh ini, saya mengumpulkan 35 pidato mantan presiden AS yang menjabat setelah Perang Dunia II melalui pemerintahan Gerald R. Ford. Sebagian besar terdiri dari alamat State of the Union, teks publik ini tersedia melalui Internet Archive di https://archive.org/ dan Proyek Kepresidenan Amerika Universitas California di http://www.presidency.ucsb.edu/ ws/index.php/. Anda dapat menemukan data dalam file sotu-1946-1977.csv bersama dengan sumber buku di https://www.nostarch.com/practicalSQL/.

Mari kita mulai dengan tipe data unik untuk pencarian teks lengkap.

Jenis Data Pencarian Teks

Implementasi pencarian teks PostgreSQL mencakup dua tipe data. Tipe data tsvector mewakili teks yang akan dicari dan disimpan dalam bentuk yang dioptimalkan. Tipe data tsquery mewakili istilah dan operator kueri penelusuran. Mari kita lihat detail keduanya.

Menyimpan Teks sebagai Lexem dengan tsvector

Tipe data tsvector mereduksi teks menjadi daftar leksem yang diurutkan, yang merupakan unit makna dalam bahasa. Pikirkan leksem sebagai kata-kata tanpa variasi yang dibuat oleh sufiks. Misalnya, format tsvector akan menyimpan kata wash, wash, dan wash sebagai leksem wash sambil mencatat posisi setiap kata dalam teks aslinya. Mengonversi teks ke tsvector juga menghapus stopword kecil yang biasanya tidak berperan dalam pencarian, seperti atau itu.

Untuk melihat cara kerja tipe data ini, mari kita konversi string ke format tsvector. Sintaks ini menggunakan fungsi pencarian PostgreSQL to_tsvector(), yang menormalkan teks "Saya berjalan melintasi ruang duduk untuk duduk dengan Anda" menjadi leksem:

```
SELECT totsvector('I am walking across the sitting room to sit with you.');
```

Jalankan kode, dan itu akan mengembalikan output berikut di tsvector format:

```
'across':4 'room':7 'sit':6,9 ''walk':3
```

Fungsi to_tsvector() mengurangi jumlah kata dari sebelas menjadi empat, menghilangkan kata-kata seperti I, am, dan the, yang bukan merupakan istilah pencarian yang berguna. Fungsinya menghilangkan sufiks, mengubah berjalan menjadi berjalan dan duduk menjadi

duduk. Itu juga mengurutkan kata-kata menurut abjad, dan angka yang mengikuti setiap titik dua menunjukkan posisinya dalam string asli, dengan mempertimbangkan kata-kata berhenti. Perhatikan bahwa duduk dikenali dalam dua posisi, satu untuk duduk dan satu untuk duduk.

Membuat Istilah Pencarian dengan tsquery

Tipe data tsquery mewakili kueri pencarian teks lengkap, sekali lagi dioptimalkan sebagai leksem. Ini juga menyediakan operator untuk mengontrol pencarian. Contoh operator termasuk ampersand (&) untuk AND, simbol pipa (|) untuk OR, dan tanda seru (!) untuk NOT. Operator <-> khusus memungkinkan Anda mencari kata-kata yang berdekatan atau kata-kata dengan jarak tertentu.

Sintaks menunjukkan bagaimana fungsi to_tsquery() mengonversi istilah pencarian ke tipe data tsquery.

```
SELECT to_tsquery('Walking & Sitting');
```

Setelah menjalankan kode, Anda akan melihat bahwa tipe data tsquery yang dihasilkan telah menormalkan istilah menjadi leksem, yang cocok dengan format data yang akan dicari:

```
'Walk' & 'sit'
```

Sekarang Anda dapat menggunakan istilah yang disimpan sebagai tsquery untuk mencari teks yang dioptimalkan sebagai
tsvektor.

Menggunakan Operator Pencocokan @@ untuk Pencarian

Dengan teks dan istilah pencarian yang dikonversi ke tipe data pencarian teks lengkap, Anda dapat menggunakan operator pencocokan tanda (@@) ganda untuk memeriksa apakah kueri cocok dengan teks. Query pertama menggunakan to_tsquery() untuk mencari kata berjalan dan duduk, yang kita gabungkan dengan operator &. Ini mengembalikan nilai Boolean true karena berjalan dan duduk ada dalam teks yang dikonversi oleh to_tsvector().

```
SELECT to_tsvector('I am walking across the sitting room') @@ to_tsquery('Walking
& sitting');
SELECT to_tsvector('I am walking across the sitting room') @@ to_tsquery('Walking
& running');
```

Namun, kueri kedua mengembalikan false karena berjalan dan berlari tidak ada dalam teks. Sekarang mari kita buat tabel untuk mencari pidato.

Membuat Tabel untuk Pencarian Teks Lengkap

Mari kita mulai dengan membuat tabel untuk menampung teks pidato. Kode membuat dan mengisi president_speeches sehingga berisi kolom untuk teks pidato asli serta kolom tipe tsvector. Alasannya adalah kita perlu mengonversi teks ucapan asli menjadi kolom tsvector untuk mengoptimalkan pencarian. Kami tidak dapat dengan mudah melakukan konversi itu selama impor, jadi mari kita tangani itu sebagai langkah terpisah. Pastikan untuk mengubah jalur file agar sesuai dengan lokasi file CSV yang Anda simpan:

```
CREATE TABLE president_speeches (
      sotu_id serial PRIMARY KEY,
      president varchar(100) NOT NULL,
      title varchar(250) NOT NULL,
      speech_date date NOT NULL,
      speech_text text NOT NULL,
      search_speech_text tsvector
);

COPY president_speeches (president, title, speech_date, speech_text)
FROM 'C:\YourDirectory\sotu-1946-1977.csv'
WITH (FORMAT CSV, DELIMITER '|', HEADER OFF, QUOTE '@');
```

Setelah menjalankan kueri, jalankan SELECT * FROM president_speeches; untuk melihat datanya. Di pgAdmin, arahkan mouse Anda ke sel mana pun untuk melihat kata tambahan yang tidak terlihat di kisi hasil. Anda akan melihat jumlah teks yang cukup besar di setiap baris kolom speech_text.

Selanjutnya, kita menyalin isi speech_text ke kolom tsvector search_speech_text dan mengubahnya ke tipe data tersebut pada saat yang bersamaan. Kueri UPDATE di menangani tugas:

```
UPDATE president_speeches
SET search_speech_text = totsvector('english', speech_text);
```

Klausa SET mengisi search_speech_text dengan output dari to_tsvector(). Argumen pertama dalam fungsi menentukan bahasa untuk mengurai leksem. Kami menggunakan bahasa Inggris default di sini, tetapi Anda dapat mengganti bahasa Spanyol, Jerman, Prancis, atau bahasa apa pun yang ingin Anda gunakan (beberapa bahasa mungkin mengharuskan Anda mencari dan menginstal kamus tambahan). Argumen kedua adalah nama kolom input. Jalankan kode untuk mengisi kolom.
Terakhir, kami ingin mengindeks kolom search_speech_text untuk mempercepat pencarian. Anda belajar tentang pengindeksan di Bab 7, yang berfokus pada tipe indeks default PostgreSQL, B-Tree. Untuk pencarian teks lengkap, dokumentasi PostgreSQL merekomendasikan penggunaan Generalized Inverted Index (GIN; lihat https://www.postgresql.org/docs/current/static/textsearch-indexes.html). Anda dapat menambahkan indeks GIN menggunakan CREATE INDEX.

```
CREATE INDEX search_idx ON president_speeches USING gin(search_speech_text);
```

Indeks GIN berisi entri untuk setiap leksem dan lokasinya, memungkinkan basis data untuk menemukan kecocokan lebih cepat.

Catatan : Cara lain untuk menyiapkan kolom pencarian adalah dengan membuat indeks pada kolom teks menggunakan fungsi to_tsvector(). Lihat
https://www.postgresql.org/docs/current/static/textsearch-tables.html untuk detailnya. Sekarang Anda siap menggunakan fungsi pencarian.

Mencari Tanggal Teks

Pidato presiden selama tiga puluh dua tahun adalah lahan subur untuk menggali sejarah. Misalnya, kueri dibawah ini mencantumkan pidato-pidato di mana presiden menyebutkan Vietnam:

```
SELECT president, speech_date
FROM president_speches
WHERE search_speech_text @@ to_tsquery('Vietnam')
ORDER BY speech_date;
```

Dalam klausa WHERE, kueri menggunakan operator pencocokan tanda (@@) ganda antara kolom search_speech_text (dari tipe data tsvector) dan istilah kueri Vietnam, yang to_tsquery() ubah menjadi data tsquery. Hasilnya harus mencantumkan 10 pidato, menunjukkan bahwa penyebutan pertama Vietnam muncul dalam pesan khusus tahun 1961 kepada Kongres oleh John F. Kennedy dan menjadi topik yang berulang mulai tahun 1966 ketika keterlibatan Amerika dalam Perang Vietnam meningkat.

```
president              speech_date
-----------------      -----------
John F. Kennedy        1961-05-25
Lyndon B. Johnson      1966-01-12
Lyndon B. Johnson      1967-01-10
Lyndon B. Johnson      1968-01-17
Lyndon B. Johnson      1969-01-14
Richard M. Nixon       1970-01-22
Richard M. Nixon       1972-01-20
Richard M. Nixon       1973-02-02
Gerald R. Ford         1975-01-15
Gerald R. Ford         1977-01-12
```

Sebelum kita mencoba lebih banyak pencarian, mari tambahkan metode untuk menunjukkan lokasi istilah pencarian kita dalam teks.

Menampilkan Hasil Pencarian Lokasi

Untuk melihat di mana istilah pencarian kita muncul dalam teks, kita dapat menggunakan fungsi ts_headline(). Ini menampilkan satu atau lebih istilah pencarian yang disorot dikelilingi oleh kata-kata yang berdekatan. Opsi untuk fungsi ini memberi Anda fleksibilitas dalam cara memformat tampilan. Listing dibawah ini menyoroti cara menampilkan pencarian untuk contoh spesifik Vietnam menggunakan ts_headline():

```
SELECT president,
      speech_date,
      ts_headline(speech_text, to_tsquery('Vietnam),
            'StartSel = <,
            StopSel = >,
            MinWords=5,
            MaxWords=7,
            MaxFragments=1')
FROM president_speeches
WHERE search_speech_text @@ to_tsquery('Vietnam");
```

Untuk mendeklarasikan ts_headline() , kita melewati kolom speech_text asli daripada kolom tsvector yang kita gunakan dalam fungsi pencarian dan relevansi sebagai argumen pertama. Kemudian, sebagai argumen kedua, kami meneruskan fungsi to_tsquery() yang menentukan

kata yang akan disorot. Kami mengikuti ini dengan argumen ketiga yang mencantumkan parameter pemformatan opsional dipisahkan dengan koma. Di sini, kami menentukan karakter untuk mengidentifikasi awal dan akhir kata yang disorot (StartSel dan StopSel). Kami juga mengatur jumlah minimum dan maksimum kata untuk ditampilkan (MinWords dan MaxWords), ditambah jumlah maksimum fragmen untuk ditampilkan menggunakan MaxFragments. Pengaturan ini opsional, dan Anda dapat menyesuaikannya sesuai dengan kebutuhan Anda.

Hasil kueri ini harus menampilkan paling banyak tujuh kata per ucapan, dengan menonjolkan kata Vietnam:

```
president              speech_date ts_headline
-----------------      ----------- ----------------------------------------------
John F. Kennedy        1961-05-25  twleve month in <Vietnam> alone—by subversives
Lyndon B. Johnson      1966-01-12  bitter conflict in <Vietnam>. Later
Lyndon B. Johnson      1967-01-10  <Vietnam>--is not a simple one. There
Lyndon B. Johnson      1968-01-17  been held in <Vietnam>--in the midst
Lyndon B. Johnson      1969-01-14  conflict in <Vietnam>, the danger of Nuclear
Richard M. Nixon       1970-01-22  <Vietnam> in a way that our generation
--snip--
```

Dengan menggunakan teknik ini, kita dapat dengan cepat melihat konteks dari istilah yang kita cari. Anda juga dapat menggunakan fungsi ini untuk menyediakan opsi tampilan fleksibel untuk fitur pencarian pada aplikasi web. Mari kita terus mencoba bentuk pencarian.

Menggunakan Beberapa Istilah Pencarian

Sebagai contoh lain, kita bisa mencari pidato di mana seorang presiden menyebutkan kata transportasi tetapi tidak membahas jalan. Kami mungkin ingin melakukan ini untuk menemukan pidato yang berfokus pada kebijakan yang lebih luas daripada program jalan tertentu.

```
SELECT president,
      speech_date,
      ts_headline(speech_text, to_tsquesry('transportation & !roads'),
            'StartSel = <,
            StopSel = >,
            MinWords=5,
            MaxWords=7,
            MaxFragments=1')
FROM president_speeches
WHERE search_speech_text @@ to_tsquery('transportation & !roads');
```

Sekali lagi, kami menggunakan ts_headline() untuk menyorot istilah yang ditemukan dalam pencarian kami.

Dalam fungsi to_tsquery() dalam klausa WHERE , kita melewati transportasi dan jalan, menggabungkannya dengan operator ampersand (&). Kita menggunakan tanda seru (!) di depan jalan untuk menunjukkan bahwa kita menginginkan pidato yang tidak mengandung kata ini. Permintaan ini harus menemukan delapan pidato yang sesuai dengan kriteria. Berikut adalah empat baris pertama:

```
president         speech_date ts_headline
---------------   ----------- ------------------------------------------------------------
Harry S. Truman   1947-01-06  such industries as <transportation>, coal, oil, steel
Harry S. Truman   1949-01-05  field of <transportation>.
John F. Kennedy   1961-01-30  Obtaining   additional   air   <transport>   mobility--and
obtaining
Lyndon B. Johnson 1964-01-08  reformed our tangled <transportation> and transit policies
--snip--
```

Perhatikan bahwa kata-kata yang disorot di kolom ts_headline termasuk transportasi dan transportasi. Alasannya adalah bahwa fungsi to_tsquery() mengonversi transport ke transport leksem untuk istilah pencarian. Perilaku basis data ini sangat berguna dalam membantu menemukan kata-kata terkait yang relevan.

Mencari Kata Berdekatan

Terakhir, kita akan menggunakan operator jarak (<->), yang terdiri dari tanda hubung antara tanda kurang dari dan lebih besar dari, untuk menemukan kata yang berdekatan. Atau, Anda dapat menempatkan angka di antara tanda-tanda untuk menemukan istilah yang terpisah dari banyak kata.

```
SELECT president,
      speech_date,

      ts_headline(speech_text, to_tsquery('military <-> defense'),
                  'StartSel = <,
                  StopSel = >,
                  MinWords=5,
                  MaxWords=7,
                  MaxFragments=1')
      FROM president_speeches
      WHERE search_speech_text @@ to_tsquery('military <-> defense');
```

Kueri ini harus menemukan empat pidato, dan karena to_tsquery() mengubah istilah pencarian menjadi leksem, kata-kata yang diidentifikasi dalam pidato harus menyertakan bentuk jamak, seperti pertahanan militer. Berikut ini menunjukkan empat pidato yang memiliki istilah yang berdekatan:

```
president              speech_date ts_headline
--------------------   ----------- --------------------------------------------------
Dwight D. Eisenhower   1956-01-05  system our <military> <defenses> are designed
Dwight D. Eisenhower   1958-01-09  direct <military> <defense> efforts, but likewise
Dwight D. Eisenhower   1959-01-09  survival--the <military> <defense> of national life
Richard M. Nixon       1972-01-20  spending. Strong <military> <defenses>
```

Jika Anda mengubah istilah kueri menjadi pertahanan <2> militer, database akan menampilkan kecocokan di mana istilah tersebut hanya terpisah dua kata, seperti dalam frasa “komitmen militer dan pertahanan kami.”

Peringkat Kecocokan Kueri berdasarkan Relevansi

Anda juga dapat memberi peringkat hasil pencarian berdasarkan relevansi menggunakan dua fungsi pencarian teks lengkap PostgreSQL. Fungsi-fungsi ini berguna saat Anda mencoba

memahami bagian teks, atau ucapan mana dalam kasus ini, yang paling relevan dengan istilah penelusuran khusus Anda.

Satu fungsi, ts_rank(), menghasilkan nilai peringkat (dikembalikan sebagai tipe data real presisi variabel) berdasarkan seberapa sering leksem yang Anda cari muncul dalam teks. Fungsi lainnya, ts_rank_cd(), mempertimbangkan seberapa dekat leksem yang dicari satu sama lain. Kedua fungsi dapat mengambil argumen opsional untuk memperhitungkan panjang dokumen akun dan faktor lainnya. Nilai peringkat yang mereka hasilkan adalah desimal arbitrer yang berguna untuk menyortir tetapi tidak memiliki arti yang melekat. Misalnya, nilai 0,375 yang dihasilkan selama satu kueri tidak dapat dibandingkan secara langsung dengan nilai yang sama yang dihasilkan selama kueri yang berbeda.

Sebagai contoh, sintaks dibawah ini menggunakan ts_rank() untuk mengurutkan pidato yang berisi semua kata perang, keamanan, ancaman, dan musuh:

```
SELECT president,
      speech_date,
      ts_rank(search_speech_text,
            to_tsquery('war & security & threat & enemy')) AS score
FROM president_speeches
WHERE search_speech_text @@ to_tsquery('war & security & threat & enemy')
ORDER BY score DESC
LIMIT 5
```

Dalam kueri ini, fungsi ts_rank() mengambil dua argumen: kolom search_speech_text dan output dari fungsi to_tsquery() yang berisi istilah pencarian. Output dari fungsi menerima skor alias. Dalam klausa WHERE kami memfilter hasil ke hanya pidato yang berisi istilah pencarian yang ditentukan. Kemudian kami mengurutkan hasil dalam skor dalam urutan menurun dan mengembalikan hanya lima pidato dengan peringkat tertinggi. Hasilnya harus sebagai berikut:

```
president              speech_date score
-----------------      ----------- --------
Harry S. Truman        1946-01-21  0.257522
Lyndon B. Johnson      1966-01-17  0.186296
Dwight D. Eisenhwer    1967-01-10  0.140851
Harry S. Truman        1968-01-09  0.0982469
Richard M. Nixon       1970-01-20  0.0973585
```

Pesan State of the Union karya Harry S. Truman tahun 1946, hanya empat bulan setelah berakhirnya Perang Dunia II, lebih sering berisi kata-kata perang, keamanan, ancaman, dan musuh daripada pidato-pidato lainnya. Namun, itu juga merupakan pidato terpanjang dalam tabel (yang dapat Anda tentukan dengan menggunakan char_length(), seperti yang Anda pelajari sebelumnya di bab ini). Panjang pidato mempengaruhi peringkat ini karena faktor ts_rank() dalam jumlah istilah yang cocok dalam teks tertentu. Pidato Kenegaraan Lyndon B. Johnson tahun 1968, yang disampaikan pada puncak Perang Vietnam, berada di urutan kedua. Akan sangat ideal untuk membandingkan frekuensi antara pidato dengan panjang yang sama untuk mendapatkan peringkat yang lebih akurat, tetapi ini tidak selalu memungkinkan. Namun, kita dapat memperhitungkan panjang setiap ucapan dengan menambahkan kode normalisasi sebagai parameter ketiga dari fungsi ts_rank(), seperti dibawah ini:

```
SELECT president,
      speech_date,
      ts_rank(search_speech_text,
            to_tsquery('war & security & threat & enemy'),2@)::numeric AS score
FROM president_speeches
WHERE search_speech_text @@ to_tsquery('war & security & threat & enemy')
ORDER BY score DESC
LIMIT 5
```

Menambahkan kode opsional 2 menginstruksikan fungsi untuk membagi skor dengan panjang data di kolom search_speech_text. Hasil bagi ini kemudian mewakili skor yang dinormalisasi dengan panjang dokumen, memberikan perbandingan antara pidato. Dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/textsearch-controls.html mencantumkan semua opsi yang tersedia untuk pencarian teks, termasuk menggunakan panjang dokumen dan membaginya dengan jumlah kata unik.

```
president              speech_date       score
-------------------    ----------------  ------------
Lyndon B. Johnson      1968-01-17        0.0000728288
Dwight D. Eisenhower   1957-01-10        0.0000633609
Richard M. Nixon       1972-01-20        0.0000497998
Harry S. Truman        1952-01-09        0.0000365366
Dwight D. Eisenhower   1958-01-09        0.0000355315
```

Berbeda dengan hasil peringkat, pidato Johnson tahun 1968 sekarang menduduki peringkat teratas, dan pesan tahun 1946 Truman jatuh dari lima besar. Ini mungkin peringkat yang lebih berarti daripada keluaran sampel pertama, karena kami menormalkannya menurut panjangnya. Tetapi empat dari lima pidato peringkat teratas adalah sama di antara kedua set, dan Anda dapat cukup yakin bahwa masing-masing dari keempat pidato ini layak untuk diteliti lebih dekat untuk memahami lebih banyak tentang pidato presiden masa perang.

Jauh dari membosankan, teks menawarkan banyak peluang untuk analisis data. Dalam bab ini, Anda telah mempelajari teknik berharga untuk mengubah teks biasa menjadi data yang dapat Anda ekstrak, kuantifikasi, cari, dan rangking. Dalam pekerjaan atau studi Anda, perhatikan laporan rutin yang memiliki fakta terkubur di dalam potongan teks. Anda dapat menggunakan ekspresi reguler untuk menggalinya, mengubahnya menjadi data terstruktur, dan menganalisisnya untuk menemukan tren. Anda juga dapat menggunakan fungsi pencarian untuk menganalisis teks.

Di bab berikutnya, Anda akan mempelajari bagaimana PostgreSQL dapat membantu Anda menganalisis informasi geografis.

Latihan Soal

Gunakan keterampilan pertengkaran teks baru Anda untuk menangani tugas-tugas ini:

1. Panduan gaya perusahaan penerbitan yang Anda tulis ingin Anda menghindari koma sebelum sufiks dalam nama. Tapi ada beberapa nama seperti Alvarez, Jr. dan Williams, Sr. di database Anda. Fungsi apa yang dapat Anda gunakan untuk menghapus koma? Akankah fungsi ekspresi reguler membantu? Bagaimana Anda hanya menangkap sufiks untuk menempatkannya ke dalam kolom terpisah?

2. Dengan menggunakan salah satu alamat State of the Union, hitung jumlah kata unik yang terdiri dari lima karakter atau lebih. (Petunjuk: Anda dapat menggunakan regexp_split_to_table() dalam subquery untuk membuat tabel kata untuk dihitung.) Bonus: Hapus koma dan titik di akhir setiap kata.

Tulis ulang kueri dimenggunakan fungsi ts_rank_cd() alih-alih ts_rank(). Menurut dokumentasi PostgreSQL, ts_rank_cd() menghitung kerapatan penutup, yang memperhitungkan seberapa dekat istilah pencarian leksem satu sama lain. Apakah menggunakan fungsi ts_rank_cd() secara signifikan mengubah hasil?

BAB IVX
ANALISIS DATA PATIAL DENGAN POSTGIS

Saat ini, aplikasi seluler dapat memberikan daftar kedai kopi di dekat Anda dalam hitungan detik. Mereka dapat melakukannya karena mereka didukung oleh sistem informasi geografis (GIS), yang merupakan sistem apa pun yang memungkinkan untuk menyimpan, mengedit, menganalisis, dan menampilkan data spasial. Seperti yang dapat Anda bayangkan, GIS memiliki banyak aplikasi praktis saat ini, mulai dari membantu perencana kota memutuskan di mana akan membangun sekolah berdasarkan pola populasi hingga menemukan jalan memutar terbaik di sekitar kemacetan lalu lintas.

Data spasial mengacu pada informasi tentang lokasi dan bentuk objek, yang dapat berupa dua dan tiga dimensi. Misalnya, data spasial yang akan kita gunakan dalam bab ini berisi koordinat yang menggambarkan bentuk geometris, seperti titik, garis, dan poligon. Bentuk-bentuk ini pada gilirannya mewakili fitur yang akan Anda temukan di peta, seperti jalan, danau, atau negara.

Dengan mudah, Anda dapat menggunakan PostgreSQL untuk menyimpan dan menganalisis data spasial, yang memungkinkan Anda menghitung jarak antar titik, menghitung ukuran area, dan mengidentifikasi apakah dua objek berpotongan. Namun, untuk mengaktifkan analisis spasial dan menyimpan tipe data spasial di PostgreSQL, Anda perlu menginstal ekstensi open source yang disebut PostGIS. Ekstensi PostGIS juga menyediakan fungsi dan operator tambahan yang bekerja secara khusus dengan data spasial.

Dalam bab ini, Anda akan belajar menggunakan PostGIS untuk menganalisis jalan raya di Santa Fe, New Mexico serta lokasi pasar petani di seluruh Amerika Serikat. Anda akan mempelajari cara menyusun dan membuat kueri tipe data spasial dan cara bekerja dengan format data geografis berbeda yang mungkin Anda temui saat memperoleh data dari sumber data publik dan pribadi.

Anda juga akan belajar tentang proyeksi peta dan sistem grid. Tujuannya adalah memberi Anda alat untuk mengumpulkan informasi dari data spasial, mirip dengan cara Anda menganalisis angka dan teks.

Kita akan mulai dengan menyiapkan PostGIS sehingga kita dapat menjelajahi berbagai jenis data spasial. Semua kode dan data untuk latihan tersedia dengan sumber buku di https://www.nostarch.com/practicalSQL/.

Menginstal PostGIS dan Membuat Basis Data Spasial

PostGIS adalah proyek open source gratis yang dibuat oleh perusahaan geospasial Kanada Refractions Research dan dikelola oleh tim pengembang internasional di bawah Open Source Geospatial Foundation. Anda akan menemukan dokumentasi dan pembaruan di http://postgis.net/. Jika Anda menggunakan Windows atau macOS dan telah menginstal

PostgreSQL dengan mengikuti langkah-langkah di Pendahuluan buku, PostGIS harus ada di komputer Anda. Itu juga sering diinstal pada PostgreSQL di penyedia cloud, seperti Amazon Web Services. Tetapi jika Anda menggunakan Linux atau jika Anda menginstal PostgreSQL dengan cara lain di Windows atau macOS, ikuti petunjuk penginstalan di http://postgis.net/install/.

Mari buat database dan aktifkan PostGIS. Prosesnya mirip dengan yang Anda gunakan untuk membuat database pertama Anda di Bab 1 tetapi dengan beberapa langkah tambahan. Ikuti langkah-langkah ini di pgAdmin untuk membuat database bernama gis_analysis:

1. Di browser objek pgAdmin (panel kiri), sambungkan ke server Anda dan perluas node Databases dengan mengklik tanda plus.
2. Klik sekali pada database analisis yang telah Anda gunakan untuk latihan sebelumnya.
3. Pilih Alat, Alat Kueri.
4. Di Alat Kueri.

```
CREATE DATABASE gis_analysis:
```

PostgreSQL akan membuat database gis_analysis, yang tidak berbeda dengan database lain yang Anda buat. Untuk mengaktifkan ekstensi PostGIS di dalamnya, ikuti langkah-langkah berikut:

1. Tutup tab Alat Kueri.
2. Di browser objek, klik kanan Databases dan pilih Refresh.
3. Klik database gis_analysis baru dalam daftar untuk menyorotnya.
4. Buka tab Alat Kueri baru dengan memilih Alat Alat Kueri. Basis data gis_analysis harus terdaftar di bagian atas panel pengeditan.
5. Di Alat Kueri, jalankan kode dibawah ini:

```
CREATE EXTENSION postgis;
```

Anda akan melihat pesan CREATE EXTENSION. Basis data Anda kini telah diperbarui untuk menyertakan tipe data spasial dan puluhan fungsi analisis spasial. Jalankan SELECT postgis_full_version(); untuk menampilkan nomor versi PostGIS beserta komponen yang terpasang. Versi tidak akan cocok dengan versi PostgreSQL yang diinstal, tetapi tidak apa-apa.

Blok Bangunan Data Spasial

Sebelum Anda belajar membuat kueri data spasial, mari kita lihat bagaimana data tersebut dijelaskan dalam GIS dan format data terkait (walaupun jika Anda ingin langsung mendalami kueri, Anda dapat melompat ke “Menganalisis Data Pasar Petani” di halaman 250 dan kembali ke sini nanti) .

Sebuah titik pada grid adalah blok bangunan terkecil dari data spasial. Kisi mungkin ditandai dengan sumbu x dan y, atau garis bujur dan garis lintang jika kita menggunakan peta. Sebuah grid bisa datar, dengan dua dimensi, atau bisa menggambarkan ruang tiga dimensi seperti kubus. Dalam beberapa format data, seperti GeoJSON berbasis JavaScript, suatu titik mungkin memiliki lokasi di kisi serta atribut yang memberikan informasi tambahan. Misalnya, toko

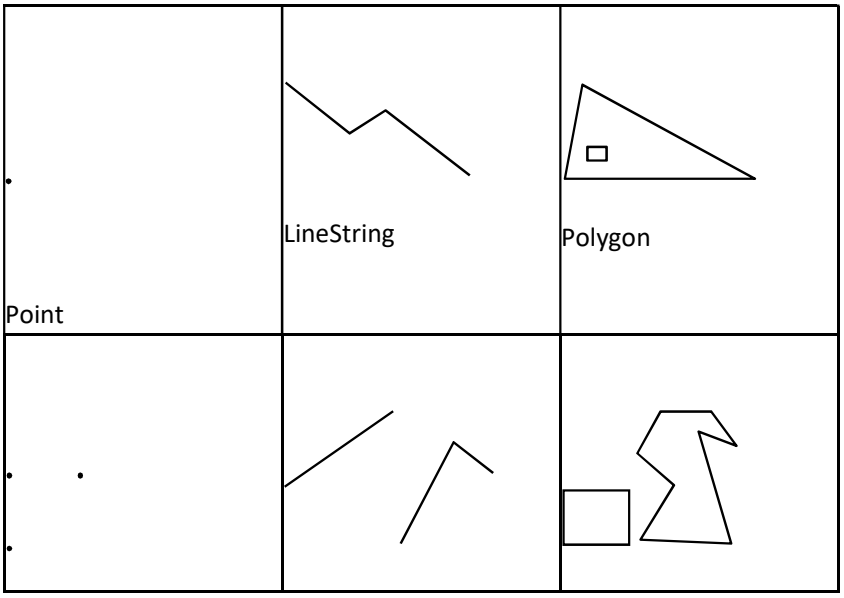
kelontong dapat dideskripsikan dengan titik yang berisi garis bujur dan garis lintangnya serta atribut yang menunjukkan nama toko dan jam operasionalnya.

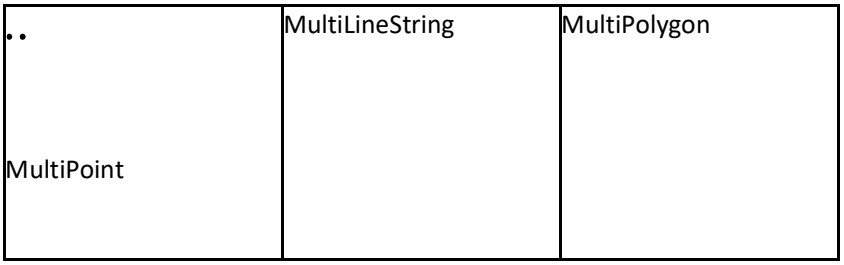
Geometri Dua Dimensi

Untuk membuat data spasial yang lebih kompleks, Anda menghubungkan beberapa titik menggunakan garis. Organisasi Internasional untuk Standardisasi (ISO) dan Open Geospatial Consortium (OGC) telah menciptakan standar fitur sederhana untuk membangun dan mengakses bentuk dua dan tiga dimensi, kadang-kadang disebut sebagai geometri. PostGIS mendukung standar.

Fitur sederhana yang paling umum digunakan yang akan Anda temui saat menanyakan atau membuat data spasial dengan PostGIS adalah sebagai berikut:

- **Titik** Sebuah lokasi tunggal dalam bidang dua atau tiga dimensi. Pada peta, Titik biasanya diwakili oleh titik yang menandai garis bujur dan garis lintang.
- **LineString** Dua atau lebih titik dihubungkan oleh sebuah garis lurus. Dengan LineStrings, Anda dapat mewakili fitur seperti jalan, jalur pendakian, atau aliran sungai.
- **Poligon** Sebuah bentuk dua dimensi, seperti segitiga atau persegi, yang memiliki tiga atau lebih sisi lurus, masing-masing dibangun dari LineString. Dianalisis geografis, Poligon mewakili objek seperti negara, negara bagian, bangunan, dan badan air. Sebuah Poligon juga dapat memiliki satu atau lebih Poligon interior yang bertindak sebagai lubang di dalam Poligon yang lebih besar.
- **MultiPoint** Satu set Po**in**. Misalnya, Anda dapat mewakili beberapa lokasi pengecer dengan satu objek MultiPoint yang berisi garis lintang dan garis bujur setiap toko.
- **MultiLineString** Satu set LineStrings. Anda dapat merepresentasikan, misalnya, sebuah objek seperti jalan dengan beberapa segmen yang tidak kontinu.
- M**ultiPolygon** Satu set Poligon. Misalnya, Anda dapat mewakili sebidang tanah yang dibagi menjadi dua bagian oleh jalan: Anda dapat mengelompokkannya dalam satu objek MultiPolygon daripada menggunakan poligon terpisah.





Gambar 14.1: Contoh visual geometri

Menggunakan fungsi PostGIS, Anda dapat membuat data spasial Anda sendiri dengan membangun objek-objek ini menggunakan titik atau geometri lainnya. Atau, Anda dapat menggunakan fungsi PostGIS untuk melakukan perhitungan pada data spasial yang ada. Umumnya, untuk membuat objek spasial, fungsi memerlukan input string teks terkenal (WKT), yaitu teks yang mewakili geometri, ditambah Pengidentifikasi Sistem Referensi Spasial (SRID) opsional yang menentukan kisi tempat menempatkan objek spasial. objek. Saya akan menjelaskan SRID secara singkat, tetapi pertama-tama, mari kita lihat contoh string WKT dan kemudian bangun beberapa geometri dengan menggunakannya.

Format Teks Terkenal

Format WKT standar OGC mencakup tipe geometri dan koordinatnya di dalam satu atau lebih set tanda kurung. Jumlah koordinat dan tanda kurung bervariasi tergantung pada geometri yang ingin Anda buat. Tabel 14-1 menunjukkan contoh jenis geometri yang lebih sering digunakan dan format WKT-nya. Di sini, saya menunjukkan pasangan garis bujur/lintang untuk koordinat, tetapi Anda mungkin menemukan sistem kisi yang menggunakan ukuran lain.

Catatan : WKT menerima koordinat dalam urutan bujur, lintang, yang mundur dari Google Maps dan beberapa perangkat lunak lainnya. Tom MacWright, sebelumnya dari perusahaan perangkat lunak Mapbox, mencatat di https://macwright.org/lonlat/ bahwa tidak ada urutan yang "benar" dan membuat katalog "inkonsistensi yang membuat frustrasi" di mana kode terkait pemetaan menangani urutan koordinat.

Tabel 14-1: Format Teks Terkenal untuk Geometri

Geometri	Format	Catatan
Point	POINT (-74.9 42.7)	Pasangan koordinat yang menandai sebuah titik di −74.9 bujur dan 42.7 lintang.
LineString	LINESTRING (-74.9 42.7, -75.1 42.7)	Garis lurus dengan titik akhir yang ditandai oleh dua pasangan koordinat.
Polygon	POLYGON ((-74.9 42.7, -75.1 42.7, -75.1 42.6, -74.9 42.7))	Segitiga yang digariskan oleh tiga pasang koordinat yang berbeda. Meskipun terdaftar dua kali, pasangan pertama dan terakhir adalah koordinat yang sama, menutup bentuknya.
MultiPoint	MULTIPOINT (-74.9 42.7, -75.1 42.7)	Dua Titik, satu untuk setiap pasangan koordinat.

MultiLineString	MULTILINESTRING((76.27 43.1, -76.06 43.08),(-76.2 43.3, -76.2 43.4, -76.4 43.1))	Dua LineString. Yang pertama memiliki dua poin; yang kedua memiliki tiga.
MultiPolygon	MULTIPOLYGON(((-74.92 42.7, -75.06 42.71, -75.07 42.64, -74.92 42.7), (-75.0 42.66, -75.0 42.64, -74.98 42.64, -74.98 42.66, -75.0 42.66)))	Dua Poligon. Yang pertama adalah segitiga, dan yang kedua adalah persegi panjang.

Meskipun contoh-contoh ini membuat bentuk sederhana, dalam praktiknya, geometri kompleks dapat terdiri dari ribuan koordinat.

Catatan tentang Sistem Koordinat

Mewakili permukaan bumi yang bulat pada peta dua dimensi tidaklah mudah. Bayangkan mengupas lapisan luar Bumi dari bola dunia dan mencoba menyebarkannya di atas meja sambil menjaga semua bagian benua dan lautan tetap terhubung. Tak pelak, beberapa area peta akan membentang. Inilah yang terjadi ketika kartografer membuat proyeksi peta dengan sistem koordinat proyeksinya sendiri yang meratakan permukaan bumi yang bulat menjadi bidang dua dimensi.

Beberapa proyeksi mewakili seluruh dunia; lainnya khusus untuk wilayah atau tujuan. Misalnya, proyeksi Mercator biasanya digunakan untuk navigasi di aplikasi, seperti Google Maps. Matematika di balik transformasinya mendistorsi wilayah daratan yang dekat dengan Kutub Utara dan Selatan, membuatnya tampak jauh lebih besar daripada kenyataan. Proyeksi Albers adalah yang kemungkinan besar akan Anda lihat ditampilkan di layar TV di Amerika Serikat saat suara dihitung pada malam pemilihan. Ini juga digunakan oleh Biro Sensus AS.

Proyeksi berasal dari sistem koordinat geografis, yang menentukan garis lintang, bujur, dan ketinggian dari setiap titik di dunia bersama dengan faktor-faktor termasuk bentuk bumi. Setiap kali Anda mendapatkan data geografis, penting untuk mengetahui sistem koordinat yang dirujuk untuk memeriksa apakah perhitungan Anda akurat. Seringkali, sistem koordinat atau proyeksi diberi nama dalam dokumentasi pengguna.

Pengidentifikasi Sistem Referensi Spasial

Saat menggunakan PostGIS (dan banyak aplikasi GIS), Anda perlu menentukan sistem koordinat yang Anda gunakan melalui SRID-nya. Saat Anda mengaktifkan ekstensi PostGIS di awal bab ini, proses membuat tabel spatial_ref_sys, yang berisi SRID sebagai kunci utamanya. Tabel juga berisi srtext kolom, yang mencakup representasi WKT dari sistem referensi spasial serta metadata lainnya.

Dalam bab ini, kita akan sering menggunakan SRID 4326, ID untuk sistem koordinat geografis WGS 84. Ini adalah standar Sistem Geodesi Dunia (WGS) terbaru yang digunakan oleh GPS, dan Anda akan sering menemukannya jika memperoleh data spasial. data. Anda dapat melihat

representasi WKT untuk WGS 84 dengan menjalankan kode dibawah ini yang mencari SRID-nya, 4326:

```
SELECT srtext
FROM spatial_ref_sys
WHERE srid = 4326;
```

Jalankan kueri dan Anda akan mendapatkan hasil berikut, yang telah saya indentasi agar mudah dibaca:

```
GEOGCS[“WGS 84”,
      DATUM[“WGS_1984”,
            SPHEROID[“WGS 84”,6378137,298.257223563,
                  AUTHORITY[“ESPG”,”7030”]],
            AUTHORITY[“ESPG”,”6362”]],
      PRIMEN[“Greemwich”,0,
            AUTHORITY[“ESPG”,”8901”]],
      UNIT[“degree”,0.010174532925199433,
            AUTHORITY[“ESPG”,”9122”]],
      AUTHORITY[“ESPG”,”4326”]]
```

Anda tidak perlu menggunakan informasi ini untuk latihan bab ini, tetapi mengetahui beberapa variabel dan bagaimana variabel tersebut mendefinisikan proyeksi akan sangat membantu. Kata kunci GEOGCS menyediakan sistem koordinat geografis yang digunakan. Kata kunci PRIMEM menentukan lokasi Meridian Utama, atau bujur 0. Untuk melihat definisi semua variabel, periksa referensi di
http://docs.geotools.org/stable/javadocs/org/opengis/referencing/doc- file/WKT.html.

Sebaliknya, jika Anda perlu menemukan SRID yang terkait dengan sistem koordinat, Anda dapat menanyakan kolom srtext di spasial_ref_sys untuk menemukannya.

Tipe Data PostGIS

Menginstal PostGIS menambahkan lima tipe data ke database Anda. Dua tipe data yang akan kita gunakan dalam latihan adalah geografi dan geometri. Kedua tipe tersebut dapat menyimpan data spasial, seperti titik, garis, poligon, SRID, dan sebagainya yang baru saja Anda pelajari, tetapi keduanya memiliki perbedaan penting:

geografi Sebuah tipe data berdasarkan bola, menggunakan sistem koordinat bumi bulat (bujur dan lintang). Semua perhitungan terjadi di dunia, dengan mempertimbangkan kelengkungannya. Itu membuat matematika menjadi rumit dan membatasi jumlah fungsi yang tersedia untuk bekerja dengan tipe geografi. Tetapi karena kelengkungan bumi diperhitungkan, perhitungan jarak menjadi lebih tepat; Anda harus menggunakan tipe data geografi saat menangani data yang mencakup area yang luas. Juga, hasil dari perhitungan pada tipe geografi akan dinyatakan dalam meter.

geometri Sebuah tipe data berdasarkan pesawat, menggunakan sistem koordinat Euclidean. Perhitungan terjadi pada garis lurus yang berlawanan dengan kelengkungan bola, membuat perhitungan untuk jarak geografis kurang tepat dibandingkan dengan tipe data geografi; hasil perhitungan dinyatakan dalam satuan sistem koordinat mana pun yang Anda tentukan.

Dokumentasi PostGIS di https://postgis.net/docs/using_postgis_dbmanagement.html menawarkan panduan tentang kapan harus menggunakan satu atau jenis lainnya. Singkatnya,

jika Anda bekerja secara ketat dengan data bujur/lintang atau jika data Anda mencakup area yang luas, seperti benua atau globe, gunakan tipe geografi, meskipun membatasi fungsi yang dapat Anda gunakan. Jika data Anda mencakup area yang lebih kecil, tipe geometri menyediakan lebih banyak fungsi dan kinerja yang lebih baik. Anda juga dapat mengubah satu jenis ke jenis lainnya menggunakan CAST.

Dengan latar belakang yang Anda miliki sekarang, kita dapat mulai bekerja dengan objek spasial.

Membuat Objek Spasial dengan Fungsi PostGIS

PostGIS memiliki lebih dari tiga lusin fungsi konstruktor yang membangun objek spasial menggunakan WKT atau koordinat. Anda dapat menemukan daftarnya di https://postgis.net/docs/reference.html#Geometry_Constructors, tetapi bagian berikut menjelaskan beberapa yang akan Anda gunakan dalam latihan. Sebagian besar fungsi PostGIS dimulai dengan huruf ST, yang merupakan standar penamaan ISO yang berarti tipe spasial.

Membuat Jenis Geometri dari Teks

Fungsi ST_GeomFromText(WKT, SRID) membuat tipe data geometri dari input string WKT dan SRID opsional. Daftar 14-4 menunjukkan pernyataan SELECT sederhana yang menghasilkan tipe data geometri untuk setiap fitur sederhana yang dijelaskan pada Tabel 14-1. Menjalankan pernyataan SELECT ini opsional, tetapi penting untuk mengetahui cara membuat setiap fitur sederhana.

```
SELECT ST_GeoFromText('Point(-74.9233606 42.699992)' 4326);

SELECT ST_GeoFromText('LINESTRING(-74.9 42.7, -75.1 42.7)', 4326);

SELECT ST_GeoFromText('POLYGON((-74.9 42.7, -75.1 42.7,
                               -75.1 42.6, -74.9 42.7))', 4326);

SELECT ST_GeoFromText('MULTIPOINT(-74.9 42.7, -75.1 42.7)', 4326);

SELECT ST_GeoFromText('MULTILINESTRING((-76.27 43.1, -76.06 43.08),
                                       (-76.2 43.3, -76.2 43.4,
                                         -76.4 43.1))', 4326);

SELECT ST_GeoFromText('MULTIPOLYGON((-74.92 42.7, 75.06 42.71,
                                    -75.07 42.64, -74.92 42.7),
                                    (-75.0 42.66, -75.0 42.62,
                                      -74.98 42.64, -74.98 42.66,
                                    -75.0 42.66)))', 4326);
```

Untuk setiap contoh, kami memberikan koordinat sebagai input pertama dan SRID 4326 sebagai input kedua. Pada contoh pertama, kita membuat titik dengan memasukkan string WKT POINT sebagai argumen pertama ke ST_GeomFromText() dengan SRID sebagai argumen opsional kedua. Kami menggunakan format yang sama di sisa contoh. Perhatikan bahwa kita tidak perlu membuat indentasi koordinat. Saya hanya melakukannya di sini untuk membuat pasangan koordinat lebih mudah dibaca.

Pastikan untuk melacak jumlah tanda kurung yang memisahkan objek, terutama dalam struktur kompleks, seperti MultiPolygon. Misalnya, kita perlu menggunakan dua tanda kurung buka dan menyertakan koordinat setiap poligon dalam kumpulan tanda kurung lainnya .

Mengeksekusi setiap pernyataan harus mengembalikan tipe data geometri yang dikodekan dalam string karakter yang terlihat seperti contoh terpotong ini:

```
01010000020E61000008ED0E5AH713N0072KX886612EDA987400. . .
```

Hasil ini menunjukkan bagaimana data disimpan dalam sebuah tabel. Biasanya, Anda tidak akan membaca rangkaian kode itu. Sebagai gantinya, Anda akan menggunakan kolom geometri (atau geografi) sebagai input ke fungsi.

Membuat Jenis Geografi dari Teks Terkenal

Untuk membuat tipe data geografi, Anda dapat menggunakan ST_GeogFromText(WKT) untuk mengonversi WKT atau ST_GeogFromText(EWKT) untuk mengonversi variasi khusus PostGIS yang disebut WKT diperpanjang yang menyertakan SRID. Listing berikut ini menunjukkan cara meneruskan SRID sebagai bagian dari string WKT yang diperluas untuk membuat objek geografi MultiPoint dengan tiga titik:

```
SELECT
ST_GeoFromText('SRID=4326;MULTIPOINT(-74.9 42.7, -75.1 42.7, -74.924 42.6)')
```

Bersama dengan fungsi ST_GeomFromText() dan ST_GeogFromText() serba guna, PostGIS menyertakan beberapa yang khusus untuk membuat objek spasial tertentu. Saya akan membahasnya secara singkat berikutnya.

Fungsi Titik

Fungsi ST_PointFromText() dan ST_MakePoint() akan mengubah WKT POINT menjadi tipe data geometri. Titik menandai koordinat, seperti bujur dan lintang, yang akan Anda gunakan untuk mengidentifikasi lokasi atau digunakan sebagai blok penyusun objek lain, seperti LineStrings. Kode dibawah ini menunjukkan bagaimana fungsi-fungsi ini bekerja:

```
SELECT ST_PointFromText('POINT(-74.9233606 42.6999992)', 4326);

SELECT ST_MakePoint(-74.9233606 42.6999992);

SELECT ST_SetSRID(ST_MakePoint(-74.9233606 42.6999992), 4326);
```

Fungsi ST_PointFromText(WKT, SRID) membuat tipe geometri titik dari WKT POINT dan SRID opsional sebagai input kedua. Dokumen PostGIS mencatat bahwa fungsi tersebut menyertakan validasi koordinat yang membuatnya lebih lambat daripada fungsi ST_GeomFromText().

Fungsi ST_MakePoint(x, y, z, m) membuat tipe geometri titik pada kisi dua, tiga, dan empat dimensi. Dua parameter pertama, x dan y dalam contoh, mewakili koordinat bujur dan lintang. Anda dapat menggunakan z opsional untuk mewakili ketinggian dan m untuk mewakili ukuran dimensi keempat, seperti waktu. Itu akan memungkinkan Anda untuk menandai lokasi pada

waktu tertentu, misalnya. Fungsi ST_MakePoint() lebih cepat daripada ST_GeomFromText() dan ST_PointFromText(), tetapi jika Anda ingin menentukan SRID, Anda harus menetapkannya dengan membungkusnya di dalam fungsi ST_SetSRID() .

Fungsi LineString

Sekarang mari kita periksa beberapa fungsi yang kita gunakan secara khusus untuk membuat tipe data geometri LineString. Listing dibawah ini menunjukkan cara kerjanya:

```
SELECT ST_LineFromText('LINESTRING(-105.90 35.67, -105.91 35.67)', 4326);
SELECT ST_MakeLine(ST_MakePoint(-74.9 42.6), ST_MakePoint(-74.1 42.));
```

Fungsi ST_LineFromText(WKT, SRID) membuat LineString dari LINESTRING WKT dan SRID opsional sebagai input kedua. Seperti ST_PointFromText() sebelumnya, fungsi ini menyertakan validasi koordinat yang membuatnya lebih lambat dari ST_GeomFromText().

Fungsi ST_MakeLine(geom, geom) membuat LineString dari input yang harus bertipe data geometri. Dalam sintaks diatas, contoh menggunakan dua fungsi ST_MakePoint() sebagai input untuk membuat titik awal dan titik akhir baris. Anda juga dapat melewatkan objek ARRAY dengan banyak titik, mungkin dihasilkan oleh subquery, untuk menghasilkan garis yang lebih kompleks.

Fungsi Poligon

Mari kita lihat tiga fungsi Polygon: ST_PolygonFromText(), ST_MakePolygon(), dan ST_MPolyFromText(). Semua membuat tipe data geometri. Sintaks berikut menunjukkan bagaimana Anda dapat membuat Poligon dengan masing-masing:

```
SELECT ST_PolygonFromText('POLYGON((-74.9 42.7, -75.1 42.7,
                                    -75.1 42.6, -74.9 42.7))', 4326);

SELECT ST_MakePolygon(ST_GeoFromText('LINESTRING(-74.92 42.7, -75.06 42.71,
                                    -75.07 42.64, -74.92 42.7))', 4326));

SELECT STMPolyFromText('MULTIPOLYGON((
                                    (-74.92 42.7, -75.06 42.71,
                                    -75.07 42.64, -74.92 42.7),
                                    (-75.0 42.66, -75.0 42.64,
                                    -74.98 42.64, -74.98 42.66,
                                    -75.0 42.66)
                                    ))', 4326);
```

Fungsi ST_PolygonFromText(WKT, SRID) membuat Poligon dari POLYGON WKT dan SRID opsional. Seperti halnya fungsi bernama serupa untuk membuat titik dan garis, ini menyertakan langkah validasi yang membuatnya lebih lambat daripada ST_GeomFromText().

Fungsi ST_MakePolygon(linestring) membuat Poligon dari LineString yang harus dibuka dan ditutup dengan koordinat yang sama, memastikan objek tertutup. Contoh ini menggunakan ST_GeomFromText() untuk membuat geometri LineString menggunakan WKT LINESTRING.
Fungsi ST_MPolyFromText(WKT, SRID) membuat MultiPolygon dari WKT dan SRID opsional.

Sekarang Anda memiliki blok bangunan untuk menganalisis data spasial. Selanjutnya, kami akan menggunakannya untuk menjelajahi sekumpulan data.

Menganalisis Data Pasar Petani

Direktori Pasar Petani Nasional dari Departemen Pertanian AS membuat katalog lokasi dan penawaran lebih dari 8.600 "pasar yang menampilkan dua atau lebih vendor pertanian yang menjual produk pertanian langsung ke pelanggan di lokasi fisik umum yang berulang," menurut https:/ /www.ams.usda.gov/local-food-directories/farmersmarkets/. Menghadiri pasar-pasar ini membuat aktivitas akhir pekan menjadi menyenangkan, jadi akan membantu untuk menemukan pasar-pasar yang berada dalam jarak perjalanan yang masuk akal. Kita dapat menggunakan kueri spasial SQL untuk menemukan pasar terdekat.

File Farmers_markets.csv berisi sebagian data USDA di setiap pasar, dan tersedia bersama dengan sumber daya buku di https://www.nostarch.com/practicalSQL/. Simpan file ke komputer Anda dan jalankan kode di berikut ini untuk membuat dan memuat tabel Farmers_markets. Pastikan Anda terhubung ke database gis_analysis yang Anda buat sebelumnya di bab ini, dan ubah jalur file pernyataan COPY agar sesuai dengan lokasi file Anda.

```
CREATE TABLE farmers_markets (
      fmid bigint PRIMARY KEY,
      market_name varchar(100) NOT NULL,
      street varchar(180),
      city varchar(60),
      county varchar(25),
      st varchar(20) NOT NULL,
      zip varchar(10),
      longitude numeric(10,7),
      latitude numeric(10,7),
      organic varchar(1) NOT NULL
);

COPY farmers_markets
FROM 'C:\YourDirectory\farmers_markets.csv'
WITH (FORMAT CSV, HEADER);
```

Tabel berisi data alamat rutin ditambah garis bujur dan garis lintang untuk sebagian besar pasar. Dua puluh sembilan pasar kehilangan nilai-nilai itu ketika saya mengunduh file dari USDA. Kolom organik menunjukkan apakah pasar menawarkan produk organik; tanda hubung (-) di kolom itu menunjukkan nilai yang tidak diketahui. Setelah Anda mengimpor data, hitung baris menggunakan SELECT count(*) FROM Farmers_markets;. Jika semuanya diimpor dengan benar, Anda harus memiliki 8.681 baris.

Membuat dan Mengisi Kolom Geografi

Untuk melakukan kueri spasial pada garis bujur dan garis lintang pasar, kita perlu mengubah koordinat tersebut menjadi satu kolom dari tipe data spasial. Karena kami bekerja dengan lokasi yang mencakup seluruh Amerika Serikat dan pengukuran yang akurat dari jarak bola yang besar adalah penting, kami akan menggunakan jenis geografi. Setelah membuat kolom, kita dapat memperbaruinya menggunakan Poin yang berasal dari koordinat, dan kemudian menerapkan indeks untuk mempercepat kueri. Sintaks berikut berisi pernyataan untuk melakukan tugas-tugas ini:

```
ALTER TABLE farmers_markets ADD COLUMN geog_point geography(POINT,4326)
```

```
UPDATE farmers_markets
SET geog_point =
      ST_SetSRID(
                  ST_MakePoint(longitude,latitude),4326:::geography;

CREATE INDEX market_pts_idx ON farmer_markets USING GIST (geog_point);

SELECT longitude,
       latitude,
       geog_point,
       ST_AsText(geog_point)
FROM farmers_markets
WHERE longitude IS NOT NULL
LIMIT 5;
```

Pernyataan ALTER TABLE yang Anda pelajari di Bab 9 dengan opsi ADD COLUMN membuat kolom tipe geografi yang disebut geog_point yang akan menampung titik dan merujuk sistem koordinat WSG 84, yang kami tunjukkan menggunakan SRID 4326.

Selanjutnya, kita menjalankan pernyataan UPDATE standar untuk mengisi kolom geog_point. Bersarang di dalam fungsi ST_SetSRID() , fungsi ST_MakePoint() mengambil kolom bujur dan lintang dari tabel sebagai input. Outputnya, yang merupakan tipe geometri secara default, harus ditransmisikan ke geografi agar sesuai dengan tipe kolom geog_point. Untuk melakukan ini, kami menggunakan sintaks titik dua (::) khusus PostgreSQL untuk mentransmisikan tipe data.

Menambahkan Indeks GiST

Sebelum Anda memulai analisis, sebaiknya tambahkan indeks ke kolom baru untuk mempercepat penghitungan. Di Bab 7, Anda mempelajari tentang indeks default PostgreSQL, B-Tree. Indeks B-Tree berguna untuk data yang dapat Anda pesan dan cari menggunakan operator persamaan dan rentang, tetapi kurang berguna untuk objek spasial. Alasannya adalah Anda tidak dapat dengan mudah mengurutkan data GIS sepanjang satu sumbu. Misalnya, aplikasi tidak memiliki cara untuk menentukan pasangan koordinat mana yang terbesar: (0,0), (0,1), atau (1,0).

Sebagai gantinya, untuk data spasial, pembuat PostGIS merekomendasikan untuk menggunakan indeks Generalized Search Tree (GiST). Anggota tim inti PostgreSQL Bruce Momjian menggambarkan GiST sebagai "kerangka kerja pengindeksan umum yang dirancang untuk memungkinkan pengindeksan tipe data yang kompleks," termasuk geometri.
Pernyataan CREATE INDEX di Sintaks sebelum nya menambahkan indeks GiST ke geog_point. Kami kemudian dapat menggunakan pernyataan SELECT untuk melihat data geografi untuk menampilkan kolom geog_points yang baru dikodekan. Untuk melihat geog_point versi WKT, kami membungkusnya dalam fungsi ST_AsText(). Hasilnya akan terlihat mirip dengan ini, dengan geog_point terpotong untuk singkatnya:

longitude	latitude	geog_point	st_astext
-121.9982460	37.5253970	010100002 ...	POINT(-121.998246 37.525397)
-100.5288290	39.8204690	010100002 ...	POINT(-100.528829 39.820469)

-92.6256000	44.8560000	010100002 ...	POINT(-92.6256 44.856)
-104.8997430	39.7580430	010100002 ...	POINT(-104.899743 39.758043)
-101.9175330	33.5480160	010100002 ...	POINT(-101.917533 33.548016)

Menemukan Geografi Dalam Jarak Tertentu

Saat berada di Iowa pada tahun 2014 untuk melaporkan sebuah cerita tentang pertanian, saya mengunjungi Pasar Petani Pusat Kota yang besar di Des Moines. Dengan ratusan vendor, pasar ini membentang di beberapa blok kota di ibu kota Iowa. Pertanian adalah bisnis besar di Iowa, dan meskipun pasar di pusat kota sangat besar, itu bukan satu-satunya di daerah tersebut. Mari gunakan PostGIS untuk menemukan lebih banyak pasar petani dalam jarak dekat dari pasar pusat kota Des Moines.

Fungsi PostGIS ST_DWithin() mengembalikan nilai Boolean true jika satu objek spasial berada dalam jarak tertentu dari objek lain. Jika Anda bekerja dengan tipe data geografi, seperti kami di sini, Anda perlu menggunakan meter sebagai satuan jarak. Jika Anda menggunakan tipe geometri, gunakan satuan jarak yang ditentukan oleh SRID.

Catatan : Pengukuran jarak PostGIS berada pada garis lurus untuk data geometri, sedangkan untuk data geografi berada pada bidang bola. Berhati-hatilah agar tidak bingung dengan jarak mengemudi di sepanjang jalan raya, yang biasanya lebih jauh dari satu titik ke titik lainnya. Untuk melakukan perhitungan yang berkaitan dengan jarak mengemudi, lihat ekstensi pgRouting di http://pgrouting.org/.

Listing dibawah ini menggunakan fungsi ST_DWithin() untuk menyaring petani_pasar untuk menunjukkan pasar dalam jarak 10 kilometer dari Pasar Petani Pusat Kota di Des Moines:

```
SELECT market_name,
      city,
      st
FROM farmers_markets
WHERE  ST_DWithin(geog_point,  ST_GeogFromText('POINT(-93.6204386  41.5853202)'),
      100000)
ORDER BY market_name;
```

Input pertama untuk ST_DWithin() adalah geog_point , yang menyimpan lokasi pasar setiap baris dalam tipe data geografi. Input kedua adalah fungsi ST_GeogFromText() yang mengembalikan geografi titik dari WKT. Koordinat -93.6204386 dan 41.5853202 mewakili garis bujur dan garis lintang Pasar Petani Pusat Kota di Des Moines. Input terakhir adalah 10.000 , yang merupakan jumlah meter dalam 10 kilometer. Basis data menghitung jarak antara setiap pasar di tabel dan pasar di pusat kota. Jika pasar berada dalam jarak 10 kilometer, itu termasuk dalam hasil.

Kami menggunakan poin di sini, tetapi fungsi ini berfungsi dengan jenis geografi atau geometri apa pun. Jika Anda bekerja dengan objek seperti poligon, Anda dapat menggunakan fungsi ST_DFullyWithin() terkait untuk menemukan objek yang sepenuhnya berada dalam jarak tertentu.

Jalankan kueri; itu harus mengembalikan sembilan baris:

market_name	city	st
Beaverdale Farmers Market	Des Moines	Iowa
Capitol Hill Farmers Market	Des Moines	Iowa
Downtown Farmers' Market - Des Moines	Des Moines	Iowa
Drake Neighborhood Farmers Market	Des Moines	Iowa
Eastside Farmers Market	Des Moines	Iowa
Highland Park Farmers Market	Des Moines	Iowa
Historic Valley Junction Farmers Market	West Des Moines	Iowa
LSI Global Greens Farmers' Market	Des Moines	Iowa
Valley Junction Farmers Market	West Des Moines	Iowa

Salah satu dari sembilan pasar ini adalah Pasar Petani Pusat Kota di Des Moines, yang masuk akal karena lokasinya berada pada titik yang digunakan untuk perbandingan. Sisanya adalah pasar lain di Des Moines atau di dekat West Des Moines. Operasi ini seharusnya sudah tidak asing lagi karena ini adalah fitur standar di banyak peta online dan aplikasi produk yang memungkinkan Anda menemukan toko atau tempat menarik di dekat Anda.

Meskipun daftar pasar terdekat ini sangat membantu, akan lebih membantu lagi jika mengetahui jarak pasti pasar dari pusat kota. Kami akan menggunakan fungsi lain untuk melaporkannya.

Mencari Jarak Antara Geografi

Fungsi ST_Distance() mengembalikan jarak minimum antara dua objek spasial. Ini juga mengembalikan meter untuk geografi dan unit SRID untuk geometri. Misalnya, untuk menghitung jarak dalam mil dari Yankee Stadium di Bronx borough New York City ke Citi Field di Queens, markas New York Mets:

```
SELECT ST_Distance(
                ST_GeogFromText('POINT(-73.9283685 40.8296466)'),
                ST_GeogFromText('POINT(-73.8480153 40.7570917)'),
                ) / 1609.344 AS mets_to_yanks;
```

Dalam contoh ini, untuk melihat hasil dalam mil, kami membagi hasil fungsi ST_Distance() dengan 1609,344 (jumlah meter dalam satu mil) untuk mengonversi satuan jarak dari meter ke mil. Hasilnya sekitar 6,5 mil:

```
mets_to_yanks
---------------
6.5438182787521
```

Mari kita terapkan teknik ini untuk mencari jarak antar titik ke data pasar petani menggunakan kode pada Daftar 14-13. Kami akan menampilkan semua pasar petani dalam jarak 10 kilometer dari Pasar Petani Pusat Kota di Des Moines dan menunjukkan jarak dalam mil:

```
SELECT market_name,
      city,
      round(
            (ST_Distance(geog_point,
```

```
                ST_GeogFromText('POINT(-93.62024386 415853202)')
                ) / 1609.344)::numeric(8,5),2
            ) AS miles_from_dt
FROM farmers_markets
WHERE ST_DWithin(geog_point,
                ST_GeogFromText('POINT(-93.62024386 415853202)'),
                10000)
ORDER BY miles_from_dt ASC;
```

Query tersebut mirip dengan menggunakan ST_DWithin() untuk menemukan pasar 10 kilometer atau lebih dekat ke pusat kota, tetapi menambahkan fungsi ST_Distance() sebagai kolom untuk menghitung dan menampilkan jarak dari pusat kota. Saya telah membungkus fungsi di dalam round() untuk memangkas output.

Kami menyediakan ST_Distance() dengan dua input yang sama seperti yang kami berikan ST_DWithin() dalam sintaks siatas tersebut: geog_point dan fungsi ST_GeogFromText(). Fungsi ST_Distance() kemudian menghitung jarak antara titik yang ditentukan oleh kedua input, mengembalikan hasilnya dalam meter. Untuk mengonversi ke mil, kita bagi dengan 1609,344 , yang merupakan perkiraan jumlah meter dalam satu mil. Kemudian, untuk menyediakan fungsi round() dengan tipe data input yang benar, kami memasukkan hasil kolom ke tipe numerik.

Klausa WHERE menggunakan fungsi dan input ST_DWithin() yang sama. Anda akan melihat hasil berikut, diurutkan berdasarkan jarak dalam urutan menaik:

```
market_name                              city              miles_from_dt
---------------------------------------  ---------------   -------------
Downtown Farmers' Market - Des Moines    Des Moines        0.00
Capitol Hill Farmers Market              Des Moines        1.15
Drake Neighborhood Farmers' Market       Des Moines        1.70
LSI Global Greens Farmers Market         Des Moines        2.30
Highland Park Farmers Market             Des Moines        2.93
Eastside Farmers Market                  Des Moines        3.40
Beaverdale Farmers Market                Des Moines        3.74
Historic Valley Junction Farmers Market  West Des Moines   4.68
Valey Junction Farmers Market            West Des Moines   4.70
```

Sekali lagi, ini adalah jenis daftar yang Anda lihat setiap hari di ponsel atau komputer Anda saat Anda mencari toko atau alamat terdekat secara online. Anda mungkin juga merasa terbantu untuk banyak skenario analisis lainnya, seperti menemukan semua sekolah dalam jarak tertentu dari sumber polusi yang diketahui atau semua rumah dalam jarak lima mil dari bandara.

Catatan : Jenis pengukuran jarak lain yang didukung oleh PostGIS, K-Nearest Neighbor, memberikan kemampuan untuk menemukan titik atau bentuk terdekat dengan cepat ke titik atau bentuk yang Anda tentukan. Untuk gambaran panjang tentang cara kerjanya, lihat http://workshops.boundlessgeo.com/postgis-intro/knn.html.

Sejauh ini, Anda telah belajar bagaimana membangun objek spasial dari WKT. Selanjutnya, saya akan menunjukkan format data umum yang digunakan dalam GIS yang disebut shapefile dan bagaimana membawanya ke PostGIS untuk dianalisis.

Bekerja dengan Sensus Shapefiles

Shapefile adalah format data GIS yang dikembangkan oleh Esri, sebuah perusahaan AS yang dikenal dengan platform visualisasi dan analisis pemetaan ArcGIS. Selain berfungsi sebagai format file standar untuk platform GIS—seperti ArcGIS dan QGIS open source—pemerintah, perusahaan, nirlaba, dan organisasi teknis menggunakan shapefile untuk menampilkan, menganalisis, dan mendistribusikan data yang mencakup berbagai fitur geografis. seperti gedung, jalan, dan batas wilayah.

Shapefile berisi informasi yang menjelaskan bentuk fitur (seperti county, jalan, atau danau) serta database yang berisi atribut tentang mereka. Atribut tersebut mungkin termasuk nama mereka dan deskriptor lainnya. Shapefile tunggal hanya dapat berisi satu jenis bentuk, seperti poligon atau titik, dan saat Anda memuat shapefile ke dalam platform GIS yang mendukung visualisasi, Anda dapat melihat bentuk dan menanyakan atributnya. PostgreSQL, dengan ekstensi PostGIS, tidak memvisualisasikan data shapefile, tetapi memungkinkan Anda untuk menjalankan kueri kompleks pada data spasial di shapefile, yang akan kita lakukan di “Menjelajahi Shapefile Kabupaten Sensus 2010” di halaman 259 dan “Melakukan Gabungan Spasial” pada halaman 262.

Pertama, mari kita periksa struktur dan isi dari shapefile.

Isi dari Shapefile

Shapefile mengacu pada kumpulan file dengan ekstensi yang berbeda, dan masing-masing memiliki tujuan yang berbeda. Biasanya, saat Anda mengunduh shapefile dari sumber, file tersebut akan datang dalam arsip terkompresi, seperti .zip. Anda harus membuka ritsletingnya untuk mengakses file individual.

Per dokumentasi ArcGIS, ini adalah ekstensi paling umum yang akan Anda temui:
.shp File utama yang menyimpan geometri fitur.
.shx File indeks yang menyimpan indeks geometri fitur.
.dbf Tabel database (dalam format dBASE) yang menyimpan informasi atribut fitur.
.xml File format XML yang menyimpan metadata tentang shapefile.
.prj File proyeksi yang menyimpan informasi sistem koordinat. Anda dapat membuka file ini dengan editor teks untuk melihat sistem koordinat geografis dan proyeksi. File dengan tiga ekstensi pertama menyertakan data yang diperlukan untuk bekerja dengan shapefile. Jenis file lainnya adalah opsional. Anda dapat memuat shapefile ke dalam PostGIS untuk mengakses objek spasial dan atribut untuk masing-masing objek. Mari lakukan selanjutnya dan jelajahi beberapa fungsi analisis tambahan.

Memuat Shapefile melalui Alat GUI

Ada dua cara untuk memuat shapefile ke dalam database Anda. Paket PostGIS menyertakan Shapefile Import/Export Manager dengan antarmuka pengguna grafis (GUI) sederhana, yang mungkin disukai pengguna.

Importir/Eksportir Windows Shapefile

Di Windows, jika Anda mengikuti langkah-langkah penginstalan di Pendahuluan buku, Anda akan menemukan Shapefile Import/Export Manager dengan memilih Start PostGIS Bundle x.y untuk PostgreSQL x64 x.y PostGIS 2.0 Shapefile dan DBF Loader Exporter.

Apa pun yang Anda lihat di tempat x.y harus cocok dengan versi perangkat lunak yang Anda unduh. Anda dapat langsung melanjutkan ke "Menghubungkan ke Database dan Memuat Shapefile" di halaman 258.

MacOS dan Linux Shapefile Importir/Eksportir

Di macOS, penginstalan postgres.app yang diuraikan dalam Pendahuluan buku tidak menyertakan alat GUI, dan pada saat penulisan ini, satu-satunya versi macOS dari alat yang tersedia (dari perusahaan geospasial Tanpa Batas) tidak berfungsi dengan macOS High Sierra. Saya akan memperbarui status di sumber buku di https: // www.nostarch.com/practicalSQL/ jika itu berubah. Sementara itu, ikuti instruksi yang ada di "Memuat Shapefile dengan shp2pgsql" di halaman 311.

Kemudian lanjutkan ke "Exploring the Census 2010 Counties Shapefile" pada halaman 259.

Untuk pengguna Linux, pgShapeLoader tersedia sebagai aplikasi shp2p- gsql-gui. Kunjungi http://postgis.net/install/ dan ikuti instruksi untuk distribusi Linux Anda. Sekarang, Anda dapat terhubung ke database dan memuat shapefile.

Menghubungkan ke Database dan Memuat Shapefile

Mari hubungkan Shapefile Import/Export Manager ke database Anda dan kemudian muat sebuah shapefile. Saya telah menyertakan beberapa shapefile dengan sumber daya untuk bab ini di https://www.nostarch.com/practicalSQL/. Kita akan mulai dengan TIGER/Line Shapefiles dari A.S. Sensus yang berisi batas-batas untuk setiap county atau county yang setara, seperti paroki atau borough, pada Sensus Desennial 2010. Anda dapat mempelajari lebih lanjut tentang rangkaian shapefile ini di https://www.census.gov/geo/maps-data/data/tiger-line.html.

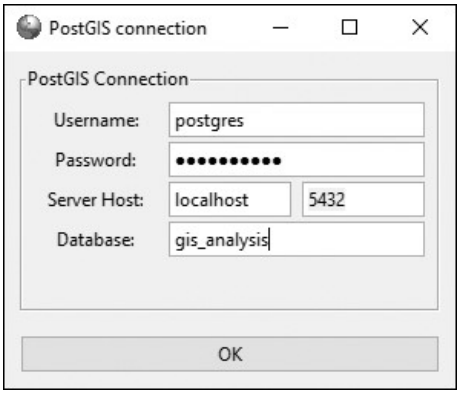
Catatan : Banyak organisasi menyediakan data dalam format shapefile. Mulailah dengan lembaga pemerintah nasional atau lokal Anda atau periksa entri Wikipedia "Daftar sumber data GIS."

Simpan tl_2010_us_county10.zip ke komputer Anda dan unzip; arsip harus berisi lima file dengan ekstensi yang saya cantumkan pada pembahasan sebelumnya. Kemudian buka aplikasi Shapefile dan DBF Loader Exporter.

Pertama, Anda perlu membuat koneksi antara aplikasi dan gis_analisis basis data. Untuk melakukannya, ikuti langkah-langkah berikut:

1. Klik Lihat detail koneksi.
2. Pada dialog yang terbuka, masukkan postgres untuk Nama Pengguna, dan masukkan kata sandi jika Anda menambahkannya untuk server selama pengaturan awal.
3. Pastikan Server Host memiliki localhost dan 5432 secara default. Biarkan itu apa adanya kecuali Anda berada di server atau port yang berbeda.

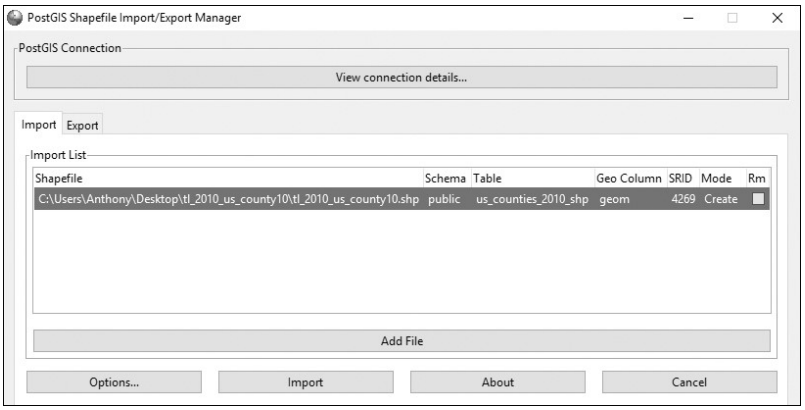
4. Masukkan gis_analysis untuk nama Database. Gambar 14-2 menunjukkan tangkapan layar seperti apa koneksi itu seharusnya.
5. Klik Oke. Anda akan melihat pesan Connection Succeeded di jendela log.



Gambar 14.2: Membuat koneksi PostGIS di pemuat shapefile

Sekarang setelah Anda berhasil membuat koneksi PostGIS, Anda dapat memuat shapefile Anda:

1. Di bawah Opsi, ubah pengkodean karakter file DBF ke Latin1 — kami melakukan ini karena atribut shapefile menyertakan nama daerah dengan karakter yang memerlukan pengkodean ini. Pertahankan kotak centang default, termasuk kotak untuk membuat indeks pada kolom spasial. Klik Oke.
2. Klik Add File dan pilih tl_2010_us_county10.shp dari lokasi Anda menyimpannya. Klik Buka. File akan muncul dalam daftar Shapefile di loader, seperti yang ditunjukkan pada Gambar 14-3.



Gambar 14.3: Menentukan detail unggahan di pemuat shapefile

3. Pada kolom Tabel, klik dua kali untuk memilih nama tabel. Ganti dengan us_counties_2010_shp.
4. Pada kolom SRID, klik dua kali dan masukkan 4269. Itulah ID untuk sistem koordinat North American Datum 1983 yang sering digunakan oleh A.S. lembaga federal termasuk Biro Sensus.
5. Klik Impor.

Di jendela log, Anda akan melihat pesan yang diakhiri dengan pesan berikut:

```
Shapefile type: Polygon
PostGIS type: MULTIPOLYGON[2]
Shapefile import completed.
```

Beralih ke pgAdmin, dan di browser objek, perluas node gis_analysis dan lanjutkan ekspansi dengan memilih Schemas public Tables. Refresh tabel Anda dengan mengklik kanan Tabel dan memilih Refresh dari menu pop-up. Anda akan melihat us_counties_2010_shp terdaftar. Selamat! Anda telah memuat shapefile Anda ke dalam sebuah tabel. Sebagai bagian dari impor, pemuat shapefile juga mengindeks kolom geom.

Menjelajahi Shapefile Kabupaten Sensus 2010

Tabel us_counties_2010_shp berisi kolom-kolom yang mencakup nama setiap county serta kode Federal Information Processing Standards (FIPS) yang secara unik ditetapkan untuk setiap negara bagian dan county. Kolom geom berisi data spasial batas wilayah masing-masing kabupaten. Untuk memulai, mari kita periksa jenis objek spasial yang berisi geom menggunakan fungsi ST_AsText(). Gunakan kode ini untuk menunjukkan representasi WKT dari nilai geom pertama dalam tabel.

```
SELECT ST_AsText(geom)
FROM us_counties_2010_shp
LIMIT 1;
```

Hasilnya adalah MultiPolygon dengan ratusan pasangan koordinat yang menguraikan batas county. Berikut sebagian dari outputnya:

```
MULTIPOLYGON(((-162.636253 54,801121, -162.641178 54.795317, -162.644046
54.789099, -162.653751 54.780339, -162.666629 54.770215, -162677799 54.562712,-
162.965212 54.632833, -162.7062413 54.764281, -162.722841 54.753155, --snip--)))
```

Setiap pasangan koordinat menandai sebuah titik pada batas county. Sekarang, Anda siap untuk menganalisis data.

Menemukan Kabupaten Terbesar di Mil Persegi

Data sensus membawa kita ke pertanyaan alami: kabupaten mana yang memiliki wilayah terluas? Untuk menghitung luas county, Sintaks berikut ini menggunakan fungsi ST_Area(), yang mengembalikan luas objek Polygon atau MultiPolygon. Jika Anda bekerja dengan tipe data geografi, ST_Area () mengembalikan hasilnya dalam meter persegi. Dengan tipe data geometri, fungsi mengembalikan area dalam satuan SRID. Biasanya, unit tidak berguna untuk analisis praktis, tetapi Anda dapat memasukkan data geometri ke geografi untuk mendapatkan meter persegi. Itulah yang akan kami lakukan di sini. Ini adalah perhitungan yang lebih intensif daripada yang lain yang telah kami lakukan sejauh ini, jadi jika Anda menggunakan komputer lama, harapkan waktu ekstra untuk menyelesaikan kueri.

```
SELECT name10,
      statefp10 AS st
      round(
            ( ST_Area(geom::geography) / 2589988.110336 )::numeric, 2
            ) AS square_miles
FROM us_counties_2010_shp
```

```
ORDER BY square_miles DESC
LIMIT 5;
```

Kolom geom adalah tipe data geometri, jadi untuk mencari luas dalam meter persegi, kita menggunakan kolom geom sebagai tipe data geografi menggunakan sintaks titik dua . Kemudian, untuk mendapatkan mil persegi, kita membagi luasnya dengan 2589988.110336, yang merupakan jumlah meter persegi dalam satu mil persegi. Untuk membuat hasilnya lebih mudah dibaca, saya telah membungkusnya dalam fungsi bulat () dan menamai kolom yang dihasilkan square_miles . Terakhir, kami membuat daftar hasil dalam urutan menurun dari area terbesar ke terkecil dan menggunakan LIMIT 5 untuk hanya menampilkan lima hasil pertama, yang akan terlihat seperti ini:

```
name10           st    square_miles
-------------    --    ------------
Yukon-Koyukuk    02    147805.08
North Slope      02    94796.21
Bethel           02    45504.36
Northwest Arthic 02    40748.95
Valdez-Cordova   02    40340.08
```

Lima kabupaten dengan wilayah terluas semuanya berada di Alaska, dilambangkan dengan kode FIPS negara bagian 02. Yukon-Koyukuk, terletak di jantung Alaska, luasnya lebih dari 147.800 mil persegi. (Ingatlah informasi itu untuk latihan “Cobalah Sendiri” di akhir bab ini.)

Menemukan Kabupaten berdasarkan Bujur dan Lintang

Jika Anda pernah bertanya-tanya bagaimana iklan situs web tampaknya mengetahui tempat tinggal Anda (“Anda tidak akan percaya apa yang dilakukan pria Boston ini dengan sepatu lamanya!”), Ini berkat layanan geolokasi yang menggunakan berbagai cara, seperti GPS ponsel Anda , untuk menemukan garis bujur dan garis lintang. Setelah koordinat Anda diketahui, koordinat tersebut dapat digunakan dalam kueri spasial untuk menemukan geografi mana yang berisi titik tersebut.

Anda dapat melakukan hal yang sama menggunakan shapefile sensus dan fungsi ST_Within(), yang mengembalikan nilai true jika satu geometri berada di dalam geometri lainnya. Listing dibawah ini menunjukkan contoh menggunakan garis bujur dan garis lintang pusat kota Hollywood:

```
SELECT name10
      statefp10
FROM us_counties_2010_shp
WHERE ST_Within('SRID=4269;POINT(-118.3419063 34.0977076)'::geometry, geom);
```

Fungsi ST_Within() di dalam klausa WHERE membutuhkan dua input geometri dan memeriksa apakah yang pertama ada di dalam yang kedua. Agar fungsi berfungsi dengan baik, kedua input geometri harus memiliki SRID yang sama. Dalam contoh ini, input pertama adalah representasi WKT yang diperluas dari suatu Titik yang menyertakan SRID 4269 (sama dengan data sensus), yang kemudian ditampilkan sebagai tipe geometri. Fungsi ST_Within () tidak menerima input SRID terpisah, jadi untuk menyetelnya untuk WKT yang disediakan, Anda harus mengawalinya

ke string seperti ini: 'SRID = 4269; POINT (-118.3419063 34.0977076)'. Input kedua adalah kolom geom dari tabel. Jalankan kueri; Anda akan melihat hasil berikut:

```
name10           statefp10
-----------      ---------
Los Angeles      06
```

Kueri menunjukkan bahwa Poin yang Anda berikan berada di wilayah Los Angeles di California (negara bagian FIPS 06). Informasi ini sangat berguna, karena dengan menggabungkan data tambahan ke tabel ini, Anda dapat memberi tahu seseorang tentang demografi atau tempat menarik di dekat mereka. Coba berikan pasangan garis bujur dan garis lintang lainnya untuk melihat A.S. kabupaten tempat mereka jatuh. Jika Anda memberikan koordinat di luar Amerika Serikat, kueri tidak akan memberikan hasil karena shapefile hanya berisi A.S. daerah.

Melakukan Gabungan Spasial

Di Bab 6, Anda mempelajari tentang gabungan SQL, yang melibatkan penautan tabel terkait melalui kolom di mana nilai cocok atau di mana ekspresi benar. Anda juga dapat melakukan penggabungan menggunakan kolom data spasial, yang membuka peluang menarik untuk analisis. Misalnya, Anda dapat menggabungkan tabel kedai kopi (yang mencakup garis bujur dan garis lintangnya) ke tabel kabupaten untuk mengetahui berapa banyak toko yang ada di setiap kabupaten berdasarkan lokasinya.

Atau, Anda dapat menggunakan gabungan spasial untuk menambahkan data dari satu tabel ke tabel lainnya untuk analisis, sekali lagi berdasarkan lokasi. Di bagian ini, kita akan mengeksplorasi gabungan spasial dengan tampilan rinci jalan dan saluran air menggunakan data sensus.

Menjelajahi Data Jalan dan Perairan

Hampir sepanjang tahun, Sungai Santa Fe, yang melintasi ibu kota negara bagian New Mexico, adalah dasar sungai kering yang lebih baik digambarkan sebagai aliran yang terputus-putus.

Menurut situs web kota Santa Fe, sungai ini rentan terhadap banjir bandang dan dinobatkan sebagai sungai paling terancam punah di negara ini pada tahun 2007. Jika Anda seorang perencana kota, akan membantu untuk mengetahui di mana sungai itu melintasi jalan raya sehingga Anda dapat merencanakan tanggap darurat. saat banjir.

Anda dapat menentukan lokasi ini menggunakan kumpulan A.S. Sensus TIGER/Line shapefiles, yang memiliki detail tentang jalan dan saluran air di Santa Fe County. Shapefile ini juga disertakan dengan sumber daya buku. Unduh dan unzip tl_2016_35049_linearwater.zip dan tl_2016_35049_roads.zip, lalu luncurkan Shapefile dan DBF Loader Exporter. Mengikuti langkah yang sama di "Memuat Shapefiles melalui GUI" Alat ”pada halaman 257, impor kedua shapefile ke gis_analysis. Beri nama tabel air santafe_linearwater_2016 dan tabel jalan santafe_roads_2016.

Selanjutnya, segarkan database Anda dan jalankan kueri SELECT * FROM cepat di kedua tabel untuk melihat data. Anda harus memiliki 12.926 baris di tabel jalan dan 1.198 di tabel air linier.

Seperti halnya shapefile county yang Anda impor melalui GUI loader, kedua tabel memiliki kolom geom terindeks dari tipe geometri. Sangat membantu untuk memeriksa jenis objek spasial di kolom sehingga Anda mengetahui jenis fitur spasial yang Anda kueri. Anda dapat melakukannya menggunakan fungsi ST_AsText () yang Anda pelajari di pembahasan sebelumnya atau menggunakan ST_GeometryType (), seperti yang ditunjukkan pada Daftar SIntaks dibawah ini:

```
SELECT ST_GeometryType(geom)
FROM santafe_linearwater_2016
LIMIT 1;

SELECT ST_GeometryType(geom)
FROM santafe_roads_2016
LIMIT 1;
```

Kedua kueri harus mengembalikan satu baris dengan nilai yang sama: ST_MultiLineString. Nilai tersebut menunjukkan bahwa saluran air dan jalan disimpan sebagai objek MultiLineString, yang merupakan rangkaian titik yang dihubungkan oleh garis lurus.

Menggabungkan Jalan Sensus dan Tabel Air

Untuk menemukan semua jalan di Santa Fe yang melintasi Sungai Santa Fe, kita akan menggabungkan tabel menggunakan sintaks JOIN ... ON yang Anda pelajari di Bab 6. Daripada mencari nilai yang cocok di kolom di kedua tabel sebagai biasa, kami akan menulis kueri yang memberi tahu kami di mana objek tumpang tindih. Kami akan melakukan ini menggunakan fungsi ST_Intersects (), yang mengembalikan nilai Boolean true jika dua objek spasial saling kontak. Input dapat berupa jenis geometri atau geografi.

```
SELECT water.fullname AS waterway,
      roads.rttyp,
      roads.fullname AS road
FROM satafe_linearwater_2016 water JOIN santafe_roads_2016 roads
      ON ST_Intersects(water.geom, roads.geom)
WHERE water.fullname = 'Santa Fe Riv'
ORDER BY roads.fullname;
```

Daftar kolom SELECT menyertakan kolom nama lengkap dari tabel santafe_linearwater_2016, yang mendapatkan air sebagai aliasnya dalam klausa FROM . Daftar kolom menyertakan kode rttyp, yang mewakili jenis rute, dan kolom nama lengkap dari tabel santafe_roads_2016, alias sebagai jalan.

Pada bagian ON dari klausa JOIN, kita menggunakan fungsi ST_Intersects() dengan kolom geom dari kedua tabel sebagai input. Ini adalah contoh penggunaan klausa ON dengan ekspresi yang mengevaluasi hasil Boolean, seperti yang disebutkan dalam "Menghubungkan Tabel Menggunakan JOIN". Kemudian kami menggunakan nama lengkap untuk menyaring hasil agar hanya menampilkan yang memiliki string lengkap ' Santa Fe Riv ', begitulah cara Sungai Santa Fe dicantumkan dalam tabel air. Kueri harus mengembalikan 54 baris; inilah lima yang pertama:

```
Waterway          rttyp       road
------------      -----       ---------
Santa Fe Riv      M           Baca Ranch Ln
```

```
Santa Fe Riv      M          Cam Alire
Santa Fe Riv      M          Cam Carlos Rael
Santa Fe Riv      M          Cam Dos Antonios
Santa Fe Riv      M          Cerro Gord Rd
--snip--
```

Setiap jalan di hasil berpotongan dengan sebagian dari Sungai Santa Fe. Kode jenis rute untuk masing-masing hasil pertama adalah M, yang menunjukkan bahwa nama jalan yang ditampilkan adalah nama umum yang berlawanan dengan nama kabupaten atau negara bagian, misalnya. Nama jalan lain dalam hasil lengkap membawa jenis rute C, S, atau U (untuk tidak diketahui). Daftar kode jenis rute lengkap tersedia di https://www.census.gov/geo/reference/rttyp.html.

Menemukan Lokasi Dimana Obyek Berpotongan

Kami berhasil mengidentifikasi semua jalan yang memotong Sungai Santa Fe. Ini adalah awal yang baik, tetapi ini akan membantu survei kami tentang daerah-daerah rawan banjir untuk mengetahui dengan tepat di mana setiap persimpangan terjadi. Kita dapat memodifikasi kueri untuk menyertakan fungsi ST_Intersection(), yang mengembalikan lokasi tempat objek bersilangan. Saya telah menambahkannya sebagai kolom dibawah ini:

```
SELECT water.fullname AS waterway,
      roads.rttyp,
      roads.fullname AS road,
      ST_AsText(ST_Intersection(water.geom, roads.geom))
FROM santafe_linearwater_2016 water JOIN santafe_roads_2016 roads
      ON ST_Intersects(water.geom, roads.geom)
WHERE water.fullname = 'Santa fe Riv'
ORDER BY roads.fullname;
```

Fungsi mengembalikan objek geometri, jadi untuk mendapatkan representasi WKT-nya, kita harus membungkusnya dalam ST_AsText() . Fungsi ST_Intersection () mengambil dua input: kolom geom dari tabel air dan jalan. Jalankan kueri, dan hasilnya sekarang harus menyertakan lokasi koordinat yang tepat, atau lokasi, di mana sungai melintasi jalan:

```
Waterway          rttyp       road              st_astext
------------      -----       ---------------   ------------------
Santa Fe Riv      M           Baca Ranch Ln     POINT(-106.049782 35.642805)
Santa Fe Riv      M           Cam Alire         POINT(-105.967111 35.68479)
Santa Fe Riv      M           Cam Carlos Rael   POINT(-105.986712 35.672483)
Santa Fe Riv      M           Cam Dos Antonios  POINT(-106.007913 35.661576)
Santa Fe Riv      M           Cerro Gord Rd     POINT(-105.895799 35.686198)
--snip--
```

Anda mungkin dapat memikirkan lebih banyak ide untuk menganalisis data spasial. Misalnya, jika Anda memperoleh shapefile yang menunjukkan bangunan, Anda dapat menemukan bangunan yang dekat dengan sungai dan dalam bahaya banjir saat hujan lebat.

Pemerintah dan organisasi swasta secara teratur menggunakan teknik ini sebagai bagian dari proses perencanaan mereka.

Fitur pemetaan adalah alat analisis yang hebat, dan teknik yang Anda pelajari dalam bab ini memberi Anda awal yang kuat untuk menjelajahi lebih banyak dengan PostGIS. Anda mungkin

juga ingin melihat aplikasi pemetaan sumber terbuka QGIS (http://www.qgis.org/), yang menyediakan alat untuk memvisualisasikan data geografis dan bekerja secara mendalam dengan shapefile. QGIS juga bekerja cukup baik dengan PostGIS, memungkinkan Anda menambahkan data dari tabel Anda langsung ke peta.

Anda sekarang telah menambahkan bekerja dengan data geografis ke keterampilan analisis Anda. Di bab-bab selanjutnya, saya akan memberi Anda alat dan kiat tambahan untuk bekerja dengan PostgreSQL dan alat terkait untuk terus meningkatkan keterampilan Anda.

Latihan Soal

Gunakan data spasial yang telah Anda impor dalam bab ini untuk mencoba analisis tambahan:

1. Sebelumnya, Anda menemukan A.S. kabupaten memiliki wilayah terluas. Sekarang, gabungkan data county untuk menemukan luas setiap negara bagian dalam mil persegi. (Gunakan kolom statefp10 di tabel us_counties_2010_shp.) Berapa banyak negara bagian yang lebih besar dari wilayah Yukon-Koyukuk?
2. Dengan menggunakan ST_Distance (), tentukan berapa mil yang memisahkan kedua pasar petani ini: Oakleaf Greenmarket (9700 Argyle Forest Blvd, Jacksonville, Florida) dan Columbia Farmers Market (1701 West Ash Street, Columbia, Missouri). Anda harus terlebih dahulu menemukan koordinat keduanya di tabel Farmers_markets. (Petunjuk: Anda juga dapat menulis kueri ini menggunakan sintaks Ekspresi Tabel Umum yang Anda pelajari di Bab 12.)
3. Lebih dari 500 baris di tabel Farmers_markets tidak memiliki nilai di kolom county, yang merupakan contoh data kotor pemerintah. Menggunakan tabel us_counties_2010_shp dan fungsi ST_Intersects (), lakukan penggabungan spasial untuk menemukan nama daerah yang hilang berdasarkan garis bujur dan garis lintang setiap pasar. Karena geog_point di Farmers_markets bertipe geografi dan SRID-nya 4326, Anda harus memasukkan geom di tabel sensus ke tipe geografi dan mengubah SRID-nya menggunakan ST_SetSRID ().

BAB XV
TAMPILAN WAKTU DAN MENYEDERHANAKAN FUNGSI

Salah satu keuntungan menggunakan bahasa pemrograman adalah memungkinkan kita untuk mengotomatiskan tugas-tugas yang berulang dan membosankan. Jika Anda harus menjalankan kueri yang sama setiap bulan untuk memperbarui tabel yang sama, cepat atau lambat Anda akan mencari jalan pintas untuk menyelesaikan tugas tersebut.

Apakah pintasan itu ada! Dalam bab ini, Anda akan mempelajari teknik untuk merangkum kueri dan logika ke dalam objek database PostgreSQL yang dapat digunakan kembali yang akan mempercepat alur kerja Anda. Saat Anda membaca bab ini, ingatlah prinsip pemrograman KERING: Jangan Ulangi Sendiri Menghindari pengulangan akan menghemat waktu dan mencegah kesalahan yang tidak perlu.

Anda akan mulai dengan belajar menyimpan kueri sebagai tampilan database yang dapat digunakan kembali. Selanjutnya, Anda akan mempelajari cara membuat fungsi Anda sendiri untuk menjalankan opsi pada data Anda. Anda telah menggunakan fungsi, seperti bulat () dan atas ( ), untuk mengubah data; sekarang, Anda akan membuat fungsi untuk melakukan operasi yang Anda tentukan. Kemudian Anda akan menyiapkan pemicu untuk menjalankan fungsi secara otomatis ketika peristiwa tertentu terjadi pada tabel. Dengan menggunakan teknik ini, Anda dapat mengurangi pekerjaan berulang dan membantu mempertahankan integritas data Anda.

Kami akan menggunakan tabel yang dibuat dari contoh di bab sebelumnya untuk mempraktikkan teknik ini. Jika Anda terhubung ke database gis_analysis di pgAdmin saat mengerjakan Bab 14, ikuti instruksi di bab itu untuk kembali ke database analisis. Semua kode untuk bab ini tersedia untuk diunduh bersama dengan sumber buku di https://www.nostarch.com/practicalSQL /. Mari kita mulai.

Menggunakan Tampilan untuk Menyederhanakan Query

Tampilan adalah tabel virtual yang dapat Anda buat secara dinamis menggunakan kueri tersimpan. Misalnya, setiap kali Anda mengakses tampilan, kueri yang disimpan berjalan secara otomatis dan menampilkan hasilnya. Mirip dengan tabel biasa, Anda dapat mengkueri tampilan, bergabung dengan tampilan ke tabel biasa (atau tampilan lain), dan gunakan tampilan untuk memperbarui atau menyisipkan data ke dalam tabel yang menjadi dasarnya, meskipun dengan beberapa peringatan.

Di bagian ini, kita akan melihat tampilan reguler dengan sintaks PostgreSQL yang sebagian besar sejalan dengan standar ANSI SQL. Tampilan ini mengeksekusi kueri dasarnya setiap kali Anda mengakses tampilan, tetapi tidak menyimpan data seperti tabel Kami tidak akan mengeksplorasi tampilan material di sini, tetapi Anda melakukannya. Tampilan material, yang khusus untuk PostgreSQL, Oracle, dan sejumlah sistem database lainnya, menyimpan data yang dibuat oleh tampilan tersebut, dan nanti Anda dapat memperbarui data yang di-cache tersebut. dapat menelusuri

https://www.postgresql.org/docs/current/static/sql-creatematerializedview.html untuk mempelajari lebih lanjut.

Tampilan sangat berguna karena memungkinkan Anda untuk:

- Hindari upaya duplikat dengan membiarkan Anda menulis kueri sekali dan mengakses hasilnya saat diperlukan
- Kurangi kerumitan untuk Anda sendiri atau pengguna database lain dengan hanya menampilkan kolom yang relevan dengan kebutuhan Anda
- Memberikan keamanan dengan membatasi akses hanya ke kolom tertentu dalam tabel

Catatan : Untuk memastikan keamanan data dan sepenuhnya mencegah pengguna melihat informasi sensitif, seperti data gaji pokok di tabel karyawan, Anda harus membatasi akses dengan menyetel izin akun di PostgreSQL Biasanya, administrator database menangani fungsi ini untuk organisasi , tetapi jika Anda ingin menjelajahi masalah ini lebih lanjut, baca dokumentasi PostgreSQL tentang peran pengguna di https://www.postgresql.org/ docs / current / static / sql-createrole.html dan perintah GRANT di https://www.postgresql.org/docs/current/static/sql-grant.html.

Tampilan mudah dibuat dan dipelihara. Mari bekerja melalui beberapa contoh untuk melihat cara kerjanya.

Membuat dan Menanyakan Tampilan

Di bagian ini, kami akan menggunakan data dalam tabel Sensus AS Desenial us_counties_2010 yang Anda impor di Bab 4. Sintaks dibawah ini menggunakan data ini untuk membuat tampilan yang disebut nevada_counties_pop_2010 yang hanya menampilkan empat dari 16 kolom asli, menampilkan data hanya Kabupaten Nevada:

```
CREATE OR REPLACE VIEW Nevada_counties_pop_2010 AS
      SELECT geo_name,
             state_fips,
             county_fips,
             p0010001 AS pop_2010
      FROM us_counties_2010
      WHERE state_us_abbreviation = 'NV'
      ORDER BY county_fips;
```

Di sini, kami mendefinisikan tampilan menggunakan kata kunci CREATE OR REPLACE VIEW , diikuti dengan nama tampilan dan AS. Berikutnya adalah kueri SQL standar SELECT yang mengambil total populasi (kolom p001001) untuk setiap county Nevada dari us_counties_2010 Kemudian kami mengurutkan data dengan kode FIPS (Standar Pemrosesan Informasi Federal) kabupaten , yang merupakan penanda standar yang digunakan Biro Sensus dan intervensi federal lainnya untuk menentukan setiap kabupaten dan negara bagian.

Perhatikan kata kunci OR REPLACE setelah CREATE, yang memberi tahu database bahwa jika tampilan dengan nama ini sudah ada, ganti dengan definisi di sini.

Namun ada peringatan menurut dokumentasi PostgreSQL: kueri yang menghasilkan tampilan harus memiliki kolom dengan nama dan tipe data yang sama dalam urutan yang sama dengan tampilan yang diganti. Namun, Anda dapat menambahkan kolom di akhir daftar kolom.

Jalankan kode tersebut diatas menggunakan pgAdmin. Database akan merespons dengan pesan CREATE VIEW. Untuk menemukan tampilan yang Anda buat, di browser objek pgAdmin, klik kanan database analisis dan pilih Refresh. Pilih Skema publik Tampilan ke lihat tampilan baru Saat Anda mengklik kanan tampilan dan memilih Properti, Anda akan melihat kueri di bawah tab Definisi di dialog yang terbuka.

Catatan: Seperti objek database lainnya, Anda dapat menghapus tampilan menggunakan perintah DROP. Dalam contoh ini, sintaksnya adalah DROP VIEW nevada_counties_pop_2010;.

Setelah membuat tampilan, Anda dapat menggunakan tampilan dalam klausa FROM dari kueri SELECT dengan cara yang sama seperti Anda menggunakan tabel biasa. Masukkan kode ini untuk mengambil lima baris pertama dari tampilan:

```
SELECT *
FROM Nevada_counties_pop_2010
LIMIT 5;
```

Selain batas lima baris, hasilnya harus sama seperti jika Anda menjalankan kueri SELECT yang digunakan untuk membuat tampilan ini:

```
geo_name               state_fips  county_fips       pop_2010
----------------       ----------  -----------       --------
Churchill County       32          001               24877
Clark County           32          003               1951269
Douglas County         32          005               46997
Elko County            32          007               48818
Esmeralda County       32          009               783
```

Contoh sederhana ini tidak terlalu berguna kecuali dengan cepat membuat daftar populasi county Nevada adalah tugas yang akan sering Anda lakukan.Jadi, mari kita bayangkan pertanyaan yang mungkin sering ditanyakan oleh analis yang berpikiran data di organisasi riset politik: berapa persen perubahan populasi untuk setiap county di Nevada (atau negara bagian lainnya) dari tahun 2000 hingga 2010?

Sintaks dibawah ini untuk memodifikasi listing dari Bab 6 menggunakan persenan rumus.

```
CREATE OR REPLACE VIEW county_pop_change_2010_2000 AS
      SELECT c2010.geo_name,
             c2010.state_us_abbreviation AS st,
             c2010.state_fips,
             c2010.county_fips,
             c2010.p0010001 AS pop_2010,
             c2000.p0010001 AS pop_2000,
             round( (CAST(c2010.p0010001 AS numeric(8,1) - c2000.p0010001)
                  / c2000.p0010001 * 100, 1) AS pct_change_2010_2000
FROM us_counties_2010 c2010 INNER JOIN un_counties_2000 c2000
ON c2010.state.fips = c2000.county_fips
      AND c2010.county_fips = c2000.county_fips
ORDER BY c2010.state_fips, c2010.county_fips;
```

Kami memulai definisi tampilan dengan CREATE OR REPLACE VIEW , diikuti dengan nama tampilan dan AS. Query SELECT menamai kolom dari tabel sensus dan menyertakan definisi kolom dengan perhitungan persentase perubahan yang Anda pelajari di Bab 5 Kemudian kita bergabung dengan tabel Sensus 2010 dan 2000 menggunakan kode FIPS negara bagian dan kabupaten. Jalankan kode tersebut, dan database akan kembali merespons dengan CREATE VIEW.

Sekarang setelah kita membuat tampilan, kita dapat menggunakan kode dibawah ini untuk menjalankan kueri sederhana terhadap tampilan baru yang mengambil data untuk county Nevada:

```
SELECT geo_name,
       st,

       pop_2010,
       pct_change_2010_2000
FROM county_pop_change_2010_200
WHERE st = 'NV'
LIMIT 5;
```

Dalam kueri sebelumnya, tampilan pertama yang kita buat, kita mengambil setiap kolom dalam tampilan dengan menggunakan wildcard asterisk setelah kata kunci SELECT. Sintaks tersebut diatas menunjukkan bahwa, seperti kueri pada tabel, Kita dapat menamai kolom tertentu saat mengkueri tampilan. Di sini, kita menetapkan empat dari tujuh kolom tampilan county_pop_change_2010_2000. Salah satunya adalah pct_change_2010_2000 which, yang mengembalikan hasil penghitungan persentase perubahan yang kita cari. Seperti yang Anda lihat, ini adalah Kami juga memfilter hasil menggunakan klausa WHERE , mirip dengan cara kami memfilter kueri apa pun alih-alih mengembalikan semua baris. Setelah menanyakan empat kolom dari tampilan, hasilnya akan terlihat seperti ini:

```
geo_name          st    pop_2010    pct_change_2010_2000
----------------  --    --------    --------------------
Churchill County  NV    24877       3.7
Clark County      NV    1951269     41.8
Douglas County    NV    46997       13.9
Elko County       NV    48818       7.8
Esmeralda County  NV    783         -19.4
```

Sekarang kita dapat meninjau kembali pandangan ini sesering yang kita suka untuk menarik data untuk presentasi atau untuk menjawab pertanyaan tentang persentase perubahan populasi untuk setiap county di Nevada (atau negara bagian lainnya) dari tahun 2000 hingga 2010.

Melihat hanya lima baris ini, Anda dapat melihat bahwa beberapa cerita menarik muncul: efek ledakan perumahan tahun 2000-an di Clark County, yang mencakup kota Las Vegas, serta penurunan tajam populasi di Esmeralda County, yang memiliki salah satu kepadatan penduduk terendah di Amerika Serikat.

Memasukkan, Memperbarui, dan Menghapus Data Menggunakan Tampilan

Anda dapat memperbarui atau menyisipkan data dalam tabel yang mendasari kueri tampilan selama tampilan memenuhi kondisi tertentu. Salah satu persyaratannya adalah tampilan harus mereferensikan satu tabel. Jika kueri tampilan bergabung dengan tabel, seperti tampilan perubahan populasi, kita hanya Selain itu, kueri tampilan tidak boleh berisi DISTINCT, GROUP BY, atau klausa lainnya (Lihat daftar lengkap pembatasan di) https://www.postgresql.org/docs/current/static/sql-createview.html.)

Anda sudah tahu cara menyisipkan dan memperbarui data secara langsung pada tabel, jadi mengapa melakukannya melalui tampilan? Salah satu alasannya adalah dengan tampilan, Anda dapat lebih mengontrol data mana yang dapat diperbarui pengguna. Mari bekerja melalui contoh untuk melihat caranya ini bekerja.

Menciptakan Pandangan Karyawan

Dalam pelajaran Bab 6 tentang bergabung, kami membuat dan mengisi tabel departemen dan karyawan dengan empat baris tentang orang dan tempat mereka bekerja atau karyawan; kueri menunjukkan isi tabel, seperti yang Anda lihat di sini:

```
emp_id      first_name  last_name   salary      dept_id
------      ----------  ---------   ------      -------
1           Nancy       Jones       62500       1
2           Lee         Smith       59300       1
3           Soo         Nguyen      83000       2
4           Janet       King        95000       2
```

Katakanlah kita ingin memberi pengguna di Departemen Pajak (dept_id-nya adalah 1) kemampuan untuk menambah, menghapus, atau memperbarui nama karyawan mereka tanpa membiarkan mereka mengubah informasi gaji atau data karyawan di departemen lain. kita dapat mengatur tampilan menggunakan listing dibawah ini:

```
CREATE OR REPLACE VIEW employees_tax_dept AS
      SELECT emp_id,
             first_name,
             last_name,
             dept_id
      FROM employees
      WHERE dept_id = 1
      ORDER BY emp_id
      WITH LOCAL CHECK OPTION;
```

Mirip dengan tampilan yang telah kami buat sejauh ini, kami hanya memilih kolom yang ingin kami tampilkan dari tabel karyawan dan menggunakan WHERE untuk memfilter hasil pada dept_id = 1 untuk mencantumkan hanya staf Departemen Pajak. Untuk membatasi sisipan atau pembaruan Hanya untuk karyawan Departemen Pajak, kami menambahkan OPSI WITH LOCAL CHECK , yang menolak penyisipan atau pembaruan apa pun yang tidak memenuhi kriteria klausa WHERE. Misalnya, opsi tidak mengizinkan siapa pun untuk menyisipkan atau memperbarui baris di tabel yang mendasari di mana dept_id karyawan adalah 3.

Buat tampilan employee_tax_dept dengan menjalankan sintaks tesebut diatas. Kemudian jalankan SELECT * FROM employee_tax_dept; yang seharusnya menyediakan dua baris ini:

```
emp_id      first_name  last_name   dept_id
------      ----------  ---------   -------
1           Nancy       Jones       1
2           Lee         Smith       1
```

Hasilnya menunjukkan karyawan yang bekerja di Departemen Pajak adalah dua dari empat baris di seluruh tabel karyawan. Sekarang, mari kita lihat bagaimana sisipan dan pembaruan bekerja melalui tampilan ini.

Memasukkan Baris Menggunakan Tampilan employee_tax_dept

Kita juga dapat menggunakan tampilan untuk menyisipkan atau memperbarui data, tetapi alih-alih menggunakan nama tabel dalam pernyataan INSERT atau UPDATE, kita mengganti nama tampilan. Setelah kita menambahkan atau mengubah data menggunakan tampilan, perubahan diterapkan ke tabel yang mendasarinya. , yang dalam hal ini adalah karyawan. Tampilan kemudian mencerminkan perubahan melalui kueri yang dijalankannya.

Kode dibawah ini menunjukkan dua contoh yang mencoba menambahkan catatan karyawan baru melalui tampilan employee_tax_dept. Yang pertama berhasil, tetapi yang kedua gagal.

```
INSERT INTO emploees_tax_dept (first_name, last_name, dept_id)
VALUES ('Suzanne', 'Legere', 1);

INSERT INTO employees_tax_dept (first_name, last_name, dept_id)
VALUES ('Jamil', 'White', 2);

SELECT * FROM employees_tax_dept;

SELECT * FROM employees;
```

Dalam INSERT pertama, yang mengikuti format sisipan yang Anda pelajari di Bab 1, kami menyediakan nama depan dan belakang Suzanne Legere ditambah dept_id-nya. Karena dept_id adalah 1, nilainya memenuhi LOCAL CHECK dalam tampilan, dan sisipan berhasil ketika dijalankan.

Tetapi ketika kami menjalankan INSERT kedua untuk menambahkan karyawan bernama Jamil White menggunakan dept_id 2, operasi gagal dengan pesan kesalahan baris baru melanggar opsi centang untuk tampilan "karyawan_tax_dept". Alasannya adalah ketika kami membuat tampilan dari sintaks diatas, kami menggunakan klausa WHERE untuk menampilkan hanya baris dengan dept_id = 1. Dept_id dari 2 tidak lulus LOCAL CHECK dalam tampilan, dan dicegah untuk disisipkan.

Jalankan pernyataan SELECT pada tampilan untuk memeriksa apakah Suzanne Legere berhasil ditambahkan:

```
emp_id      first_name  last_name   dept_id
------      ----------  ---------   -------
1           Nancy       Jones       1
2           Lee         Smith       1
5           Suzanne     Legere      1
```

Kita juga dapat mengkueri tabel karyawan untuk melihat bahwa, pada kenyataannya, Suzanne Legere telah ditambahkan ke tabel lengkap. Tampilan mengkueri tabel karyawan setiap kali kita mengaksesnya.

```
emp_id      first_name  last_name   salary      dept_id
------      ----------  ---------   ------      -------
1           Nancy       Jones       62500             1
2           Lee         Smith       59300             1
3           Soo         Nguyen      83000             2
4           Janet       King        95000             2
5           Suzanne     Legere                        1
```

Seperti yang Anda lihat dari penambahan “Suzanne Legere”, data yang kami tambahkan menggunakan tampilan juga ditambahkan ke tabel yang mendasarinya. Namun, karena tampilan tidak menyertakan kolom gaji, nilainya di barisnya adalah NULL. Jika Anda mencoba memasukkan nilai gaji menggunakan tampilan ini, Anda akan menerima pesan kesalahan kolom "gaji" dari relasi "employees_tax_dept" tidak ada Alasannya adalah bahwa meskipun kolom gaji ada di tabel karyawan yang mendasarinya, untuk mempelajari lebih lanjut tentang memberikan izin kepada pengguna jika Anda berencana untuk mengambil tanggung jawab administrator basis data.

Memperbarui Baris Menggunakan Tampilan employee_tax_dept

Pembatasan yang sama dalam mengakses data dalam tabel yang mendasari berlaku ketika kami membuat pembaruan pada data dalam tampilan employee_tax_dept. Kode dibawah ini menunjukkan kueri standar untuk memperbarui ejaan nama belakang Suzanne menggunakan UPDATE (sebagai orang dengan lebih dari satu huruf besar) dalam nama belakangnya, saya dapat mengonfirmasi bahwa nama yang salah mengeja bukanlah hal yang aneh).

```
UPDATE employees_tax_dept
SET last_name = ‘Le Gere’
WHERE emp_id = 5;

SELECT * FROM employees_tax_dept;
```

Jalankan kode, dan hasil dari kueri SELECT akan menampilkan nama belakang yang diperbarui, yang muncul di tabel karyawan yang mendasarinya:

```
emp_id      first_name  last_name   dept_id
------      ----------  ---------   -------
1           Nancy       Jones       1
2           Lee         Smith       1
5           Suzanne     Le Gere     1
```

Nama belakang Suzanne sekarang dieja dengan benar sebagai "Le Gere," bukan "Legere." Namun, jika kami mencoba memperbarui nama karyawan yang tidak berada di Departemen Pajak, kueri gagal seperti saat kami mencoba memasukkan Jamil White di Listing tersebut datas. Selain itu, mencoba menggunakan tampilan ini untuk memperbarui gaji seorang karyawan — bahkan satu di Departemen Pajak — akan gagal dengan kesalahan yang sama seperti yang saya sebutkan di bagian sebelumnya. Jika tampilan tidak Jika Anda tidak mereferensikan kolom di tabel yang mendasarinya, Anda tidak dapat mengakses kolom itu

melalui tampilan. Sekali lagi, fakta bahwa pembaruan pada tampilan dibatasi dengan cara ini menawarkan cara untuk memastikan privasi dan keamanan untuk bagian data tertentu.

Menghapus Baris Menggunakan Tampilan employee_tax_dept

Sekarang, mari kita telusuri cara menghapus baris menggunakan tampilan. Pembatasan data mana yang dapat Anda pengaruhi juga berlaku di sini. Misalnya, jika Suzanne Le Gere di Departemen Pajak mendapat tawaran yang lebih baik dari perusahaan lain dan memutuskan untuk bergabung dengan perusahaan lain Daftar 15-8 menunjukkan kueri dalam sintaks DELETE standar:, Anda dapat menghapusnya dari tabel karyawan melalui tampilan employee_tax_dept.

```
DELETE FROM employees_tax_dept
WHERE emp_id = 5
```

Jalankan kueri, dan PostgreSQL akan merespons dengan DELETE 1. Namun, saat Anda mencoba menghapus baris untuk karyawan di departemen selain Departemen Pajak, PostgreSQL tidak akan mengizinkannya dan akan melaporkan DELETE 0.

Singkatnya, tampilan tidak hanya memberi Anda kontrol atas akses ke data, tetapi juga pintasan untuk bekerja dengan data. Selanjutnya, mari kita jelajahi cara menggunakan fungsi untuk menghemat lebih banyak waktu.

Memprogram Fungsi

Anda telah menggunakan banyak fungsi di seluruh buku ini, baik untuk menggunakan huruf besar dengan () atau menambahkan angka dengan jumlah (). Di balik fungsi-fungsi ini terdapat sejumlah besar (terkadang kompleks) pemrograman yang mengambil input, mengubahnya, atau memulai tindakan, dan mengembalikan respons. Anda melihat sejauh mana kode dalam bab 4 saat Anda membuat fungsi median (), yang menggunakan 30 baris kode untuk menemukan nilai tengah dalam sekelompok angka. dalam fungsi dan fungsi lain yang dikembangkan oleh programmer basis data untuk mengotomatisasi proses dapat menggunakan lebih banyak baris kode, termasuk tautan ke kode eksternal yang ditulis dalam bahasa lain, seperti C.

Kami tidak akan menulis kode yang rumit di sini, tetapi kami akan membahas beberapa contoh fungsi bangunan yang dapat Anda gunakan sebagai landasan untuk ide Anda sendiri. Bahkan fungsi sederhana yang dibuat pengguna dapat membantu Anda menghindari pengulangan kode saat Anda sedang menganalisis data.

Kode di bagian ini khusus untuk PostgreSQL dan bukan bagian dari standar ANSI SQL. Di beberapa database, terutama Microsoft SQL Server dan MySQL, penerapan kode yang dapat digunakan kembali terjadi dalam prosedur tersimpan. Jika Anda menggunakan sistem manajemen database lain, periksa dokumentasinya untuk spesifik.

Membuat Fungsi persen_perubahan ()

Untuk mempelajari sintaks untuk membuat fungsi, mari kita tulis fungsi untuk menyederhanakan penghitungan persentase perubahan dari dua nilai, yang merupakan pokok analisis data. Dalam Bab 5, Anda telah mempelajari bahwa rumus persentase perubahan dapat dinyatakan sebagai berikut:

```
percent change = (New Number – Old Number) / Old Number
```

Daripada menulis rumus itu setiap kali kita membutuhkannya, kita dapat membuat fungsi yang disebut persen_perubahan () yang mengambil angka baru dan lama sebagai input dan mengembalikan hasilnya yang dibulatkan ke sejumlah tempat desimal yang ditentukan pengguna. Kode di bawah ini untuk melihat cara mendeklarasikan fungsi SQL sederhana:

```
CREATE OR REPLACE FUNCTION
percent_change(new_value numeric,
               old_value numeric,
               decimal_places integer DEFAULT 1)
RETURNS numeric AS
'SELECT round(
        ((new_value – old_value) / old_value) * 100, decimal_places
);
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
```

Banyak yang terjadi dalam kode ini, tetapi tidak serumit kelihatannya. Kita mulai dengan perintah CREATE OR REPLACE FUNCTION , diikuti dengan nama fungsi dan, dalam tanda kurung, daftar argumen yang merupakan input fungsi Setiap argumen memiliki nama dan tipe data. Sebagai contoh, kami menetapkan bahwa nilai_baru dan nilai_lama adalah numerik, oleh karena itu tempat_desimal (yang menentukan jumlah tempat untuk membulatkan hasil) adalah bilangan bulat. Untuk tempat_desimal, kami menetapkan 1 sebagai DEFAULT Karena kita menetapkan nilai default, argumen akan menjadi opsional saat kita memanggil fungsi nanti.

Kami kemudian menggunakan kata kunci RETURNS numerik AS untuk memberi tahu fungsi untuk mengembalikan perhitungannya sebagai tipe numerik.Jika ini adalah fungsi untuk menggabungkan string, kami mungkin mengembalikan teks.

Selanjutnya, kita menulis daging dari fungsi yang melakukan penghitungan. Di dalam tanda kutip tunggal, kita menempatkan kueri SELECT yang menyertakan penghitungan persentase perubahan yang bersarang di dalam fungsi bulat (). Dalam rumus, kita menggunakan nama argumen fungsi alih-alih angka.

Kami kemudian menyediakan serangkaian kata kunci yang mendefinisikan atribut dan perilaku fungsi. Kata kunci BAHASA menentukan bahwa kami telah menulis fungsi ini menggunakan SQL biasa, yang merupakan salah satu dari beberapa bahasa yang didukung PostgreSQL dalam fungsi. Opsi umum lainnya adalah bahasa prosedural khusus PostgreSQL yang disebut PL / pgSQL yang, selain menyediakan sarana untuk membuat fungsi, menambahkan fitur yang tidak ditemukan dalam SQL standar, seperti struktur kontrol logis (IF ... THEN ... ELSE).PL / pgSQL

adalah bahasa prosedural default yang diinstal dengan PostgreSQL, tetapi Anda dapat menginstal yang lain, seperti PL / Perl dan PL / Python, untuk menggunakan bahasa pemrograman Perl dan Python di database Anda. Nanti di bab ini, saya akan menunjukkan contoh PL / pgSQL dan Python.

Selanjutnya, kata kunci IMMUTABLE menunjukkan bahwa fungsi tidak akan membuat perubahan apa pun pada database, yang dapat meningkatkan kinerja Baris RETURNS NULL ON NULL INPUT menjamin bahwa fungsi akan memberikan respons NULL jika ada input yang tidak disediakan secara default adalah NULL.

Jalankan kode menggunakan pgAdmin untuk membuat fungsi persen_perubahan(). Server harus merespons dengan pesan CREATE FUNCTION.

Menggunakan Fungsi persen_change()

Untuk menguji fungsi persen_change() baru, jalankan dengan sendirinya menggunakan SELECT, seperti yang ditunjukkan pada listing dibawah ini:

```
SELECT percent_change(110, 108, 2);
```

Contoh ini menggunakan nilai 110 untuk angka baru, 108 untuk angka lama, dan 2 sebagai angka desimal yang diinginkan untuk membulatkan hasilnya. Jalankan kodenya; hasilnya akan terlihat seperti ini:

```
percent_change
--------------
          1.85
```

Hasilnya menunjukkan bahwa ada peningkatan 1,85 persen antara 108 dan 110. Anda dapat bereksperimen dengan angka lain untuk melihat bagaimana hasilnya berubah. Juga, coba ubah argumen desimal_places ke nilai termasuk 0, atau hilangkan, untuk melihat bagaimana itu mempengaruhi output. Anda akan melihat hasil yang memiliki lebih banyak atau lebih sedikit angka setelah titik desimal, berdasarkan input Anda.

Tentu saja, kami membuat fungsi ini untuk menghindari keharusan menulis rumus perubahan persentase penuh dalam kueri. Sekarang mari kita gunakan untuk menghitung persentase perubahan menggunakan versi kueri perubahan populasi Sensus Tahunan yang kami tulis di Bab 6, seperti yang ditunjukkan pada Sintaks dibawah ini:

```
SELECT c2010.geo.name,
       c2010.state_us_abbreviation AS st,
       c2010.p0010001 AS pop_2010,
       percent_change(c2010.p0010001, c2000.p0010001) AS pct_chg_func,
       round( (CAST(c2010.p0010001 * 100, 1 ) AS pct_chg_formula
FROM us_counties_2010 c2010 INNER JOIN un_counties_2000 c2000
ON c2010.state_fips = c2000.state_fips
      AND c2010.county_fips = c2000.county_fips
ORDER BY pct_chg_func DESC
LIMIT 5;
```

Sintaks diatas menggunakan kueri asli pada bab 6 dan menambahkan fungsi persen_perubahan () sebagai kolom sebelum rumus sehingga kita dapat membandingkan hasil. Sebagai input, kita menggunakan kolom total populasi 2010 (c2010.p0010001) sebagai jumlah baru dan 2000 total penduduk sebagai yang lama (c2000.p001001).

Saat Anda menjalankan kueri, hasilnya akan menampilkan lima wilayah dengan persentase perubahan populasi terbesar, dan hasil dari fungsi tersebut harus cocok dengan hasil dari rumus yang dimasukkan langsung ke kueri.

```
geo_name          st    pop_2010    pct_chg_func      pct_chg_formula
--------------    --    --------    ------------      ---------------
Kendall County    IL    114736      110.4             110.4
Pinal County      AZ    375770      109.1             109.1
Flagler County    FL    95696       92.0              92.0
Lincoln County    SD    44828       85.8              85.8
Loudoun County    VA    312311      84.1              84.1
```

Setiap hasil menampilkan satu tempat desimal, nilai default fungsi, karena kami tidak memberikan argumen ketiga opsional saat kami memanggil fungsi. Sekarang kami tahu fungsi berfungsi sebagaimana dimaksud, kami dapat menggunakan percent_change() kapan pun kami perlu menyelesaikannya Perhitungan itu Menggunakan fungsi jauh lebih cepat daripada harus menulis rumus setiap kali kita perlu menggunakannya.

Memperbarui Data dengan Fungsi

Kami juga dapat menggunakan fungsi untuk menyederhanakan pembaruan rutin ke data. Di bagian ini, kami akan menulis fungsi yang menetapkan jumlah hari pribadi yang benar yang tersedia untuk seorang guru (selain liburan) berdasarkan tanggal perekrutan mereka. Tabel guru dari pelajaran pertama di Bab 1, “Membuat Tabel” Jika Anda melewatkan bagian itu, Anda dapat kembali ke sana untuk membuat tabel dan menyisipkan data menggunakan kode contoh di awal pembahasan Bab 1. Mari kita mulai dengan menambahkan kolom ke guru untuk mengadakan hari-hari pribadi menggunakan kode di dibawah ini:

```
ALTER TABLE teachers ADD COLUMN personal_days integer;

SELECT first_name,
       last_name,
       hire_date,
       personal_days
FROM teachers;
```

Sintaks diatas tersebut memperbarui tabel pengajar menggunakan ALTER dan menambahkan kolom personal_days menggunakan kata kunci ADD COLUMN. Jalankan pernyataan SELECT untuk melihat data. Ketika kedua kueri selesai, Anda akan melihat enam baris berikut:

```
first_name  last_name   hire_date   personal_days
----------  ---------   ---------   -------------
Janet       Smith       2011-10-30
Lee         Reynolds    1993-05-22
Samuel      Cole        2005-08-01
Samantha    Bush        2011-10-30
Betty       Diaz        2005-08-30
Kathleen    Roush       2010-10-22
```

Kolom personal_days menyimpan nilai NULL karena kami belum memberikan nilai apa pun. Sekarang, mari kita buat sebuah fungsi bernama update_personal_days() yang memperbarui kolom personal_days dengan hari pribadi yang benar berdasarkan tanggal pengangkatan guru. Kami akan menggunakan aturan berikut untuk memperbarui data di kolom personal_days:

- Kurang dari lima tahun sejak disewa: 3 hari pribadi
- Antara lima dan 10 tahun sejak disewa: 4 hari pribadi
- Lebih dari 10 tahun sejak disewa: 5 hari pribadi

Kode di bawah ini mirip dengan kode yang kita gunakan untuk membuat fungsi persen_perubahan (), tapi kali ini kita akan menggunakan bahasa PL / pgSQL sebagai ganti SQL biasa.

```
CREATE OR REPLACE FUNCTION update_personal_days()
RETURN void AS $$
BEGIN
      UPDATE teachers
      SET personal_days =
            CASE WHEN (now() – hire_date) BETWEEN ‘5 years’::interval
                                          AND ’10 years’::interval THEN 4
                 WHEN (now() – hire_date) > ’10 years’::interval THEN 5
                  ELSE 3
            END;
RAISE NOTICE ‘personal_days update!’;
END;
$$ LANGUAGE plpgsql;
```

Kami mulai dengan CREATE OR REPLACE FUNCTION, diikuti dengan nama fungsi. Kali ini, kami tidak memberikan argumen karena tidak ada input pengguna yang diperlukan. Fungsi beroperasi pada kolom yang telah ditentukan dengan aturan yang ditetapkan untuk menghitung interval. Juga, kami menggunakan RETURNS void untuk dicatat bahwa fungsi tidak mengembalikan data; itu hanya memperbarui kolom personal_days.

Seringkali, saat menulis fungsi berbasis PL / pgSQL, konvensi PostgreSQL adalah menggunakan tanda kutip dolar standar non-ANSI SQL ($$) untuk menandai awal dan akhir string yang berisi semua perintah fungsi. dengan fungsi percent_change() sebelumnya, Anda bisa menggunakan tanda kutip tunggal untuk mengapit string, tetapi kemudian tanda kutip tunggal dalam string perlu digandakan, dan itu terlihat berantakan.) Jadi, semua yang ada di antara pasangan $$ adalah The kode yang berfungsi. Anda juga dapat menambahkan beberapa teks di antara tanda dolar, seperti $ namestring $, untuk membuat pasangan unik tanda kutip awal dan akhir. Ini berguna, misalnya, jika Anda perlu mengutip kueri di dalam fungsi.

Tepat setelah $$ pertama kita memulai BEGIN ... END; blok ke fungsi payudara; di dalamnya kita menempatkan pernyataan UPDATE yang menggunakan pernyataan KASUS untuk menentukan jumlah hari yang didapat setiap guru. Kita kurangi tanggal_pekerjaan dari tergantung pada rentang mana now() --hire_date masuk, pernyataan CASE mengembalikan jumlah hari libur yang benar sesuai dengan rentang.

gunakan RAISE NOTICE untuk menampilkan pesan di pgAdmin bahwa fungsi tersebut telah selesai. Pada akhirnya, kami menggunakan kata kunci LANGUAGE untuk menentukan bahwa kami telah menulis fungsi ini menggunakan PL / pgSQL.

Jalankan kode dibawah ini untuk membuat update_personal_days()function. Kemudian gunakan baris berikut untuk menjalankannya di pgAdmin:

```
SELECT update_personal_days();
```

Sekarang ketika Anda menjalankan kembali pernyataan SELECT di atas tersebut. Anda akan melihat bahwa setiap baris kolom personal_days diisi dengan nilai yang sesuai. Perhatikan bahwa hasil Anda dapat bervariasi tergantung pada saat Anda menjalankan fungsi ini, karena hasil dari now ( ) terus diperbarui dengan berlalunya waktu.

```
first_name  last_name   hire_date   personal_days
----------  ---------   ----------  -------------
Janet       Smith       2011-10-30              4
Lee         Reynolds    1993-05-22              5
Samuel      Cole        2005-08-01              5
Samantha    Bush        2011-10-30              4
Betty       Diaz        2005-08-30              5
Kathleen    Roush       2010-10-22              4
```

Anda dapat menggunakan fungsi update_personal_days() untuk memperbarui data secara manual secara berkala setelah melakukan tugas tertentu, atau Anda dapat menggunakan penjadwal tugas seperti pgAgent (alat open source terpisah) untuk menjalankannya secara otomatis. Anda dapat mempelajari tentang pgAgent dan alat lainnya di “Utilitas, Alat, dan Ekstensi PostgreSQL”.

Menggunakan Bahasa Python dalam Fungsi

Sebelumnya saya telah menyebutkan bahwa PL / pgSQL adalah bahasa prosedural default dalam PostgreSQL, tetapi database juga mendukung pembuatan fungsi menggunakan bahasa sumber terbuka, seperti Perl dan Python. Dukungan ini memungkinkan Anda untuk memanfaatkan fitur bahasa tersebut serta yang terkait modul dalam fungsi yang Anda buat. Misalnya, dengan Python, Anda dapat menggunakan perpustakaan pandas untuk analisis data. Dokumentasi di https://www.postgresql.org/docs/current/ static / server-programming.html memberikan tinjauan komprehensif bahasa yang tersedia, tetapi di sini saya akan menunjukkan kepada Anda fungsi yang sangat sederhana menggunakan Python.

Untuk mengaktifkan PL / Python, Anda harus menambahkan ekstensi menggunakan kode di bawah ini. Jika Anda mendapatkan kesalahan, seperti tidak dapat mengakses file "$ libdir / plpython2", itu berarti PL / Python tidak disertakan saat Anda menginstal PostgreSQL Lihat kembali tautan pemecahan masalah untuk setiap sistem operasi di “Menginstal PostgreSQL”

```
CREATE EXTENSION plpythonu;
```

Catatan: Ekstensi plpythonu saat ini menginstal Python versi 2.x. Jika Anda ingin menggunakan Python 3.x, instal ekstensi plpython3u sebagai gantinya. Namun, versi yang tersedia mungkin berbeda berdasarkan distribusi PostgreSQL.

Setelah mengaktifkan ekstensi, buat fungsi mengikuti sintaks yang sama yang baru saja Anda pelajari di Listing program sebelumnya, tetapi gunakan Python untuk badan fungsi sintaks dibawah ini untuk menunjukkan cara menggunakan PL / Python untuk membuat fungsi yang disebut trim_county() yang menghilangkan kata “County” dari akhir sebuah string, kita akan menggunakan fungsi ini untuk membersihkan nama-nama county dalam data sensus.

```
CREATE OR REPLACE FUNCTION trim_county(input_string text)
RETURN text AS $$
      import re
      cleaned = re.sub(r' County', '', input_string)
      return cleaned
$$ LANGUAGE plpythonu;
```

Catatan: Ekstensi plpythonu saat ini menginstal Python versi 2.x. Jika Anda ingin menggunakan Python 3.x, instal ekstensi plpython3u sebagai gantinya. Namun, versi yang tersedia mungkin berbeda berdasarkan distribusi PostgreSQL.

Setelah mengaktifkan ekstensi, buat fungsi mengikuti sintaks yang sama yang baru saja Anda pelajari dipembahasan sebelumnya, tetapi gunakan Python untuk badan fungsi dibawah ini untuk menunjukkan cara menggunakan PL / Python untuk membuat fungsi yang disebut trim_county() yang menghilangkan kata “County” dari akhir sebuah string, kita akan menggunakan fungsi ini untuk membersihkan nama-nama county dalam data sensus.

```
SELECT geo_name,
       trim_county(geo_name)
FROM us_counties_2010
ORDER BY state_fips, county_fips
LIMIT 5;
```

Kami menggunakan kolom geo_name di tabel us_counties_2010 sebagai input ke trim_county(). Itu akan mengembalikan hasil ini:

```
geo_name          trim_county
--------------    -----------
Autauga County    Autauga
Baldwin County    Baldwin
Barbour County    Barbour
Bibb County       Bibb
Blount County     Blount
```

Seperti yang Anda lihat, fungsi trim_county () mengevaluasi setiap nilai di kolom geo_name dan menghapus spasi dan kata County saat ada. Meskipun ini adalah contoh sepele, ini menunjukkan betapa mudahnya menggunakan Python—atau salah satu dari yang lain bahasa prosedural yang didukung—di dalam suatu fungsi.

Selanjutnya, Anda akan belajar cara menggunakan pemicu untuk mengotomatisasi database Anda.

Mengotomatiskan Tindakan Basis Data dengan Pemicu

Pemicu database menjalankan fungsi setiap kali peristiwa tertentu, seperti INSERT, UPDATE, atau DELETE, terjadi pada tabel atau tampilan. Anda dapat menyetel pemicu untuk diaktifkan

sebelum, sesudah, atau sebagai ganti peristiwa, dan Anda juga dapat Misalnya, Anda menghapus 20 baris dari sebuah tabel. Anda dapat menyetel pemicu untuk diaktifkan sekali untuk masing-masing dari 20 baris yang dihapus atau hanya satu kali. ...

Kami akan mengerjakan dua contoh. Contoh pertama menyimpan log perubahan yang dibuat pada nilai di sekolah. Contoh kedua secara otomatis mengklasifikasikan suhu setiap kali kami mengumpulkan bacaan.

Mencatat Pembaruan Nilai ke Tabel

Katakanlah kita ingin secara otomatis melacak perubahan yang dibuat pada tabel nilai siswa di database sekolah kita. Setiap kali baris diperbarui, kita ingin mencatat nilai lama dan baru ditambah waktu perubahan terjadi (cari "David Lightman dan nilai" dan Anda akan melihat mengapa hal ini layak untuk dilacak). Untuk menangani tugas ini secara otomatis, kita memerlukan tiga item:

- Tabel grades_history untuk mencatat perubahan nilai dalam tabel grades
- Pemicu untuk menjalankan fungsi setiap kali terjadi perubahan pada tabel nilai, yang akan kita beri nama grades_update
- Fungsi yang akan dijalankan oleh pemicu; kami akan memanggil fungsi ini
- record_if_grade_changed()

Membuat Tabel untuk Melacak Nilai dan Pembaruan

Mari kita mulai dengan membuat tabel yang kita butuhkan dengan menyertakan kode untuk membuat dan mengisi nilai terlebih dahulu kemudian membuat grades_history:

```
CREATE TABLE grades (
      student_id bigint,
      course_id bigint,
      course varchar(30) NOT NULL,
      grade varchar(5) NOT NULL,
PRIMARY KEY (student_id, course_id)
);

INSERT INTO grades
VALUES
      (1, 1, 'Biology 2', 'F'),
      (1, 2, 'English 11B', 'D'),
      (1, 3, 'World History 11B', 'C'),
      (1, 4, 'Trig 2', 'B');

CREATE TABLE grades_history (
      student_id bigint NOT NULL,
      course_id bigint NOT NULL,
      change_time timestamp with time zone NOT NULL,
      course varchar(30) NOT NULL,
      old_grade varchar(5) NOT NULL,
      new_grade varchar(5) NOT NULL,
PRIMARY KEY (student_id, course_id, change_time)
);
```

Perintah ini sangat mudah. Kami menggunakan CREATE untuk membuat tabel nilai dan menambahkan empat baris menggunakan INSERT , di mana setiap baris mewakili nilai siswa di kelas. Kemudian kami menggunakan CREATE TABLE untuk membuat tabel grades_history untuk menyimpan data Kami mencatat setiap kali nilai yang ada diubah. Tabel grades_history

memiliki kolom untuk nilai baru, nilai lama, dan waktu perubahan. Jalankan kode untuk membuat tabel dan mengisi tabel nilai. Kami tidak memasukkan data ke dalam grades_history di sini karena proses pemicu akan menangani tugas itu.

Membuat Fungsi dan Pemicu

Selanjutnya, mari kita tulis fungsi record_if_grade_changed () yang akan dieksekusi oleh trigger. Kita harus menulis fungsi tersebut sebelum menamakannya di trigger. Mari kita lihat kode dibawah ini:

```
CREATE OR REPLACE FUNCTION record_if_grade_changed()
      RETURNS trigger AS
$$
BEGIN
      IF NEW.grade <> OLD.grade THEN
      INSERT INTO grades_history (
            student_id,
            course_id,
            change_time,
            old_grade,
            new_grade)
      VALUES
            (OLD.student_id,
            OLD.course_id,
            now(),
            OLD.course,
            OLD.grade,
            NEW.grade);
      END IF;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Fungsi record_if_grade_changed () mengikuti pola contoh sebelumnya di bab ini tetapi dengan pengecualian khusus untuk bekerja dengan pemicu. Pertama, kami menetapkan pemicu RETURNS alih-alih tipe data atau batal. Karena record_if_grade_changed () adalah fungsi PL / pgSQL , kami menempatkan prosedur di dalam blok BEGIN ... END; Kami memulai prosedur menggunakan pernyataan IF ... THEN , yang merupakan salah satu struktur kontrol yang disediakan PL / pgSQL. Kami menggunakannya di sini untuk menjalankan pernyataan INSERT hanya jika nilai yang diperbarui berbeda dari nilai yang lama, yang kami periksa menggunakan operator <>.

Saat terjadi perubahan pada tabel nilai, pemicu (yang akan kita buat selanjutnya) akan dieksekusi. Untuk setiap baris yang diubah, pemicu akan meneruskan dua kumpulan data ke record_if_grade_changed(). Yang pertama adalah nilai baris sebelum mereka Fungsi dapat mengakses nilai baris asli dan nilai baris yang diperbarui, yang akan digunakan untuk perbandingan Jika pernyataan IF ... THEN bernilai benar, yang berarti nilai nilai lama dan baru adalah berbeda, kami menggunakan INSERT untuk menambahkan baris ke grades_history yang berisi OLD.grade dan NEW.grade .

Pemicu harus memiliki pernyataan RETURN , meskipun dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/plpgsql-trigger.html merinci skenario di mana nilai pengembalian pemicu benar-benar penting (terkadang diabaikan) Dokumentasi

juga menjelaskan bahwa Anda dapat menggunakan pernyataan untuk mengembalikan NULL atau memunculkan pengecualian jika terjadi kesalahan.

Jalankan kode diatas tadi untuk membuat fungsi Selanjutnya, tambahkan pemicu grades_update ke tabel nilai menggunakan listing dibawah ini:

```
CREATE TRIGGER grades_update
      AFTER UPDATE
            ON grades
FOR EACH Row
EXECUTE PROCEDURE record_if_grade_changed();
```

Di PostgreSQL, sintaks untuk membuat pemicu mengikuti standar ANSI SQL (walaupun konten fungsi pemicu tidak).Kode dimulai dengan pernyataan CREATE TRIGGER , diikuti oleh klausa yang mengontrol kapan pemicu berjalan dan bagaimana perilakunya. Kami menggunakan AFTER UPDATE untuk menentukan bahwa kami ingin pemicu diaktifkan setelah pembaruan terjadi pada baris nilai. Kami juga dapat menggunakan kata kunci BEFORE atau INSTEAD OF tergantung pada situasinya.

Kami menulis UNTUK SETIAP ROW untuk memberi tahu pemicu untuk menjalankan prosedur satu kali untuk setiap baris yang diperbarui dalam tabel. Misalnya, jika seseorang menjalankan pembaruan yang memengaruhi tiga baris, prosedur akan berjalan tiga kali. Alternatif (dan default) adalah UNTUK SETIAP PERNYATAAN, yang menjalankan prosedur satu kali. Jika kita tidak peduli untuk menangkap perubahan pada setiap baris dan hanya ingin mencatat bahwa nilai berubah pada waktu tertentu, kita bisa menggunakan opsi itu.Akhirnya, kita menggunakan EXECUTE PROCEDURE untuk memberi nama record_if_grade_changed () sebagai fungsi yang harus dijalankan oleh pemicu.

Buat pemicu dengan menjalankan kode diatas tersebut di pgAdmin. Basis data harus merespons dengan pesan CREATE TRIGGER.

Menguji Pemicu

Sekarang kita telah membuat pemicu dan fungsi yang harus dijalankan, mari kita pastikan pemicunya berfungsi. Pertama, ketika Anda menjalankan SELECT * FROM grades_history ;, Anda akan melihat bahwa tabel kosong karena kita belum membuat perubahan apa pun pada tabel nilai dan tidak ada yang dilacak. Selanjutnya, ketika Anda menjalankan SELECT*FROM nilai; Anda akan melihat data nilai, seperti yang ditunjukkan di sini:

```
student_id  course_id   course                grade
----------  ---------   -------------         -----
1           1           Biology 2             F
1           2           English 11B           D
1           3           World History 11B     C
1           4           Trig 2                B
```

Kelas Biologi 2 itu tidak terlihat bagus, mari kita perbarui menggunakan kode di bawah ini:

```
UPDATE grades
SET grade = 'C'
WHERE student_id  = 1 AND course_id = 1;
```

Saat Anda menjalankan UPDATE, pgAdmin tidak menampilkan apa pun untuk memberi tahu Anda bahwa pemicu dijalankan di latar belakang. Ini hanya melaporkan UPDATE 1, artinya baris dengan nilai F telah diperbarui. Namun pemicu kami memang berjalan, yang dapat kami konfirmasikan dengan memeriksa kolom di grades_history menggunakan kueri SELECT ini:

```
SELECT student_id,
       change_time,
       course,
       old_grade,
       new_grade
FROM grades_history;
```

Saat Anda menjalankan kueri ini, Anda akan melihat bahwa tabel grades_history, yang berisi semua perubahan pada nilai, sekarang memiliki satu baris:

```
student_id        change_time                   course      old_grade   new_grade
----------        --------------------------    ---------   ---------   ---------
         1        2018-07-09 13:51:45.937-04    Biology 2           F           C
```

Baris ini menampilkan nilai F Biologi 2 lama, nilai baru C, dan change_time, menunjukkan waktu perubahan yang dibuat (hasil Anda harus mencerminkan tanggal dan waktu Anda) Perhatikan bahwa penambahan baris ini ke grades_history terjadi di Tetapi acara UPDATE di atas meja menyebabkan pemicu menyala, yang mengeksekusi fungsi record_if_grade_changed ().

Jika Anda telah menggunakan sistem manajemen konten, seperti WordPress atau Drupal, pelacakan revisi semacam ini mungkin sudah tidak asing lagi. Ini menyediakan catatan perubahan yang berguna untuk konten untuk tujuan referensi dan audit, dan, sayangnya, dapat menyebabkan Tunjuk jari sesekali Terlepas dari itu, kemampuan untuk memicu tindakan pada database secara otomatis memberi Anda lebih banyak kontrol atas data Anda.

Mengklasifikasikan Suhu Secara Otomatis

Di Bab 12, kami menggunakan pernyataan SQL CASE untuk mengklasifikasi ulang pembacaan suhu ke dalam kategori deskriptif. Pernyataan CASE (dengan sintaks yang sedikit berbeda) juga merupakan bagian dari bahasa prosedural PL / pgSQL, dan kami dapat menggunakan kemampuannya untuk menetapkan nilai tovariabel.untuk secara otomatis menyimpan nama kategori tersebut dalam tabel setiap kali kita menambahkan pembacaan suhu.

Jika kami secara rutin mengumpulkan pembacaan suhu, menggunakan teknik ini untuk mengotomatisasi klasifikasi membuat kami tidak perlu menangani tugas secara manual.

Kami akan mengikuti langkah yang sama seperti yang kami gunakan untuk mencatat perubahan nilai: pertama-tama kami membuat fungsi untuk mengklasifikasikan suhu, dan kemudian membuat pemicu untuk menjalankan fungsi setiap kali tabel diperbarui. Gunakan kueri dibawah ini untuk membuat temperature_test meja untuk latihan:

```
CREATE TABLE temperature_test (
      station_name varchar(50),
      observation_date date,
```

```
      max_temp integer,
      min_temp integer,
      max_temp_group varchar(40),
PRIMARY KRY (station_name, observation_date)
);
```

Pada kuari diatas tersebut, tabel temperature_test berisi kolom untuk menyimpan nama stasiun dan tanggal pengamatan suhu Mari kita bayangkan bahwa kita memiliki beberapa proses untuk menyisipkan baris sekali sehari yang memberikan suhu maksimum dan minimum untuk itu lokasi, dan kita perlu mengisi kolom max_temp_group dengan klasifikasi deskriptif pembacaan tinggi hari itu untuk memberikan teks prakiraan cuaca yang kita distribusikan.

Untuk melakukan ini, pertama-tama kita membuat fungsi yang disebut class_max_temp (), seperti yang ditunjukkan pada kueri dibawah ini:

```
CREATE OR REPLACE FUNCTION classify_max_temp()
      RETURN trigger AS
$$
BEGIN
      CASE
            WHEN NEW.max_temp >= 90 THEN
                  NEW.max_temp_group := 'Hot';
            WHEN NEW.max_temp BETWEEN 70 AND 89 THEN
                  NEW.max_temp_group := 'Warm';
            WHEN NEW.max_temp BETWEEN 50 AND 69 THEN
                  NEW.max_temp_group := 'Pleasant';
            WHEN NEW.max_temp BETWEEN 33 AND 49 THEN
                  NEW.max_temp_group := 'Cold';
            WHEN NEW.max_temp BETWEEN 20 AND 32 THEN
                  NEW.max_temp_group := 'Freezing';
            ELSE NEW.max_temp_group := 'Inhuman';
      END CASE;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Sekarang, fungsi-fungsi ini akan terlihat familier. Apa yang baru di sini adalah versi PL / pgSQL dari sintaks CASE , yang sedikit berbeda dari sintaks SQL karena sintaks PL / pgSQL menyertakan titik koma setelah setiap klausa WHEN ... THEN . Juga baru adalah operator penugasan (: =), yang kami gunakan untuk menetapkan nama deskriptif ke kolom NEW.max_temp_group berdasarkan hasil fungsi CASE. Misalnya, pernyataan NEW.max_temp_group : ='Cold' menetapkan string'Cold' ke NEW.max_temp_group ketika nilai suhu antara 33 dan 49 derajat Fahrenheit, dan ketika fungsi mengembalikan baris BARU untuk dimasukkan ke dalam tabel, itu akan menyertakan nilai string Dingin Jalankan kode untuk membuat fungsi.

Selanjutnya, dengan menggunakan sintaks ini untuk pemicu untuk menjalankan fungsi setiap kali baris ditambahkan ke temperature_test:

```
CREATE TRIGGER temperature_insert
      BEFORE INSERT
      ON temperature_test
      FOR EACH ROW
      EXECUTE PROCEDURE classify_max_temp();
```

Dalam contoh ini, kami mengklasifikasikan max_temp dan membuat nilai untuk max_temp_group sebelum memasukkan baris ke dalam tabel. Melakukannya lebih efisien daripada melakukan pembaruan terpisah setelah baris dimasukkan. Untuk menentukan perilaku itu, kami menyetel pemicu temperature_insert untuk menembak SEBELUM INSERT.

Kami juga ingin pemicu diaktifkan UNTUK SETIAP ROW disisipkan karena kami ingin setiap max_temp dicatat dalam tabel untuk mendapatkan klasifikasi deskriptif. Pernyataan EXECUTE PROCEDURE terakhir menamai fungsi class_max_temp () yang baru saja kita buat. Jalankan pernyataan CREATE TRIGGER di pgAdmin , lalu uji penyiapan menggunakan Listing dibawah ini:

```
INSERT INTO temperature_test (station_name, observation_date, max_temp, min_temp)
VALUES
      ('North Station', '1/19/2019', 10, -3),
      ('North Station', '3/20/2019', 28, 19),
      ('North Station', '5/2/2019', 65, 42),
      ('North Station', '8/9/2019', 93,74);

SELECT * FROM temperature_test;
```

Di sini kita memasukkan empat baris ke dalam temperature_test, dan kita mengharapkan pemicu temperature_insert diaktifkan untuk setiap baris dan memang demikian! Pernyataan SELECT dalam daftar akan menampilkan hasil ini:

```
station_name      observation_date  max_temp    min_temp    max_temp_group
-------------     ----------------  --------    --------    --------------
North Station           2019-01-19        10          -3    Inhumane
North Station           2019-03-20        28          19    Freezing
North Station           2019-05-02        65          42    Pleasant
North Station           2019-08-09        93          74    Hot
```

Karena pemicu dan fungsi yang kami buat, setiap max_temp yang dimasukkan secara otomatis menerima klasifikasi yang sesuai di kolom max_temp_group.

Contoh suhu ini dan contoh audit perubahan nilai sebelumnya belumlah sempurna, tetapi mereka memberi Anda gambaran sekilas tentang betapa bergunanya pemicu dan fungsi dalam menyederhanakan pemeliharaan data.

Meskipun teknik yang Anda pelajari dalam bab ini mulai menyatu dengan teknik administrator basis data, Anda dapat menerapkan konsep tersebut untuk mengurangi jumlah waktu yang Anda habiskan untuk mengulangi tugas tertentu. Saya harap pendekatan ini akan membantu Anda meluangkan lebih banyak waktu untuk menemukan cerita yang menarik. dalam data Anda.

Bab ini menyimpulkan diskusi kita tentang teknik analisis dan bahasa SQL. Dua bab berikutnya menawarkan tip alur kerja untuk membantu Anda, mereka mencakup cara menyambungkan ke database dan menjalankan kueri dari baris perintah komputer Anda, dan cara memelihara database Anda.

Latihan Soal

Tinjau kembali konsep-konsep dalam bab dengan latihan-latihan ini:

1. Buat tampilan yang menampilkan jumlah perjalanan taksi Kota New York per jam Gunakan data taksi di Bab 11.
2. Di Bab 10, Anda telah mempelajari cara menghitung tarif per seribu Ubah rumus tersebut menjadi fungsi rate_per_thousand() yang membutuhkan tiga argumen untuk menghitung hasilnya: angka_observasi, angka_dasar, dan tempat_desimal.
3. Pada Bab 9, Anda bekerja dengan tabel meat_poultry_egg_inspect yang mencantumkan fasilitas pemrosesan makanan. Tulis pemicu yang secara otomatis menambahkan tanggal inspeksi setiap kali Anda memasukkan fasilitas baru ke dalam tabel. Gunakan kolom Inspect_date yang ditambahkan dan atur tanggal menjadi enam bulan dari tanggal saat ini. Anda harus dapat menjelaskan langkah-langkah yang diperlukan untuk menerapkan pemicu dan bagaimana langkah-langkah tersebut saling berhubungan.

BAB XVI
MENGGUNAKAN POSTGRESQL DARI COMMANDLINE

Sebelum komputer menampilkan antarmuka pengguna grafis (GUI), yang memungkinkan Anda menggunakan menu, ikon, dan tombol untuk menavigasi aplikasi, cara utama untuk memberikan instruksi kepada mereka adalah dengan memasukkan perintah pada baris perintah. Baris perintah juga disebut antarmuka baris perintah, konsol, shell, atau terminal adalah antarmuka berbasis teks tempat Anda memasukkan nama program atau perintah lain untuk melakukan tugas, seperti mengedit file atau membuat daftar isi file. direktori.

Ketika saya masih kuliah, untuk mengedit file, saya harus memasukkan perintah ke terminal yang terhubung ke komputer mainframe IBM. Rim-rim teks yang kemudian bergulir di layar mengingatkan pada karakter hijau yang mendefinisikan dunia virtual yang digambarkan dalam The Matrix. Rasanya misterius dan seolah-olah saya telah mendapatkan kekuatan baru. Bahkan hari ini, film menggambarkan peretas fiktif dengan menunjukkan mereka memasukkan perintah samar, hanya teks di komputer.

Dalam bab ini, saya akan menunjukkan kepada Anda cara mengakses dunia yang hanya berisi teks ini. Berikut adalah beberapa keuntungan bekerja dari baris perintah daripada GUI, seperti pgAdmin:

- Anda sering dapat bekerja lebih cepat dengan memasukkan perintah singkat daripada mengklik melalui lapisan item menu.
- Anda mendapatkan akses ke beberapa fungsi yang hanya disediakan oleh baris perintah.
- Jika hanya akses baris perintah yang harus Anda kerjakan (misalnya, saat Anda tersambung ke komputer jarak jauh), Anda masih bisa menyelesaikan pekerjaan.

Kami akan menggunakan psql, alat baris perintah di PostgreSQL yang memungkinkan Anda menjalankan kueri, mengelola objek database, dan berinteraksi dengan sistem operasi komputer melalui perintah teks. Pertama-tama Anda akan mempelajari cara mengatur dan mengakses baris perintah komputer Anda, lalu meluncurkan psql.

Dibutuhkan waktu untuk mempelajari cara menggunakan baris perintah, dan bahkan para ahli yang berpengalaman sering kali menggunakan dokumentasi untuk mengingat opsi baris perintah yang tersedia. Tetapi belajar menggunakan baris perintah sangat meningkatkan efisiensi kerja Anda.

Menyiapkan Baris Perintah untuk psql

Untuk memulai, kami akan mengakses baris perintah pada sistem operasi Anda dan mengatur variabel lingkungan yang disebut PATH yang memberi tahu sistem Anda di mana menemukan psql.

Variabel lingkungan menyimpan parameter yang menentukan konfigurasi sistem atau aplikasi, seperti tempat menyimpan file sementara, atau memungkinkan Anda mengaktifkan atau

menonaktifkan opsi. Setting PATH, yang menyimpan nama dari satu atau lebih direktori yang berisi program yang dapat dieksekusi, memberi tahu antarmuka baris perintah lokasi psql, menghindari kerumitan karena harus memasukkan jalur direktori lengkapnya setiap kali Anda meluncurkannya.

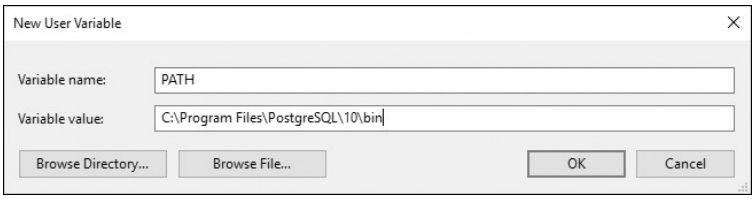
Pengaturan psql Windows

Di Windows, Anda akan menjalankan psql di dalam Command Prompt, aplikasi yang menyediakan antarmuka baris perintah sistem itu. Mari kita mulai dengan menggunakan PATH untuk memberi tahu Command Prompt di mana menemukan psql.exe, yang merupakan nama lengkap aplikasi psql di Windows, serta baris perintah PostgreSQL lainnya
keperluan.

Menambahkan psql dan Utilitas ke Windows PATH

Langkah-langkah berikut mengasumsikan bahwa Anda menginstal PostgreSQL sesuai dengan instruksi yang dijelaskan di "Penginstalan Windows" di halaman xxix. (Jika Anda menginstal PostgreSQL dengan cara lain, gunakan Windows File Explorer untuk mencari drive C: Anda untuk menemukan direktori yang menyimpan psql.exe, dan kemudian ganti C: \ Program Files \ PostgreSQL \ xy \ bin pada langkah 5 dan 6 dengan Anda jalan sendiri.)

1. Buka Panel Kontrol Windows. Masuk ke Control Panel di kotak pencarian di taskbar Windows, lalu klik ikon Control Panel.
2. Di dalam aplikasi Panel Kontrol, masukkan Lingkungan di kotak pencarian di kanan atas. Dalam daftar hasil pencarian yang ditampilkan, klik Edit Variabel Lingkungan Sistem. Dialog System Properties akan muncul.
3. Dalam dialog System Properties, pada tab Advanced, klik Environment Variables. Dialog yang terbuka harus memiliki dua bagian: Variabel pengguna dan Variabel sistem. Di bagian Variabel pengguna, jika Anda tidak melihat variabel PATH, lanjutkan ke langkah a untuk membuat yang baru. Jika Anda melihat variabel PATH yang ada, lanjutkan ke langkah b untuk memodifikasinya.
   a. Jika Anda tidak melihat PATH di bagian Variabel pengguna, klik Baru untuk membuka dialog Variabel Pengguna Baru, yang ditunjukkan pada Gambar 16.1.

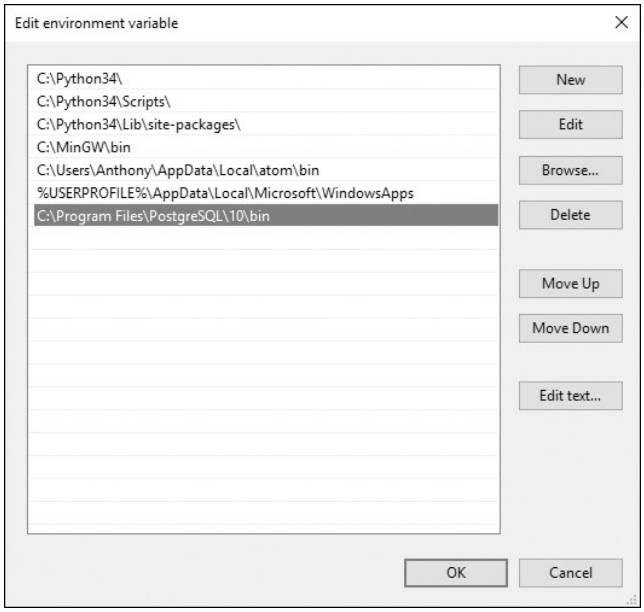


Gambar 16.1: Membuat variabel lingkungan PATH baru di Windows 10

Di kotak Nama variabel, masukkan PATH. Pada kotak Variable value, masukkan C:\Program Files\PostgreSQL\x.y\bin, di mana x.y adalah versi PostgreSQL yang Anda gunakan. Klik OK untuk menutup semua dialog.

   b. Jika Anda melihat variabel PATH yang ada di bagian Variabel pengguna, sorot dan klik Edit. Dalam daftar variabel yang ditampilkan, klik New dan masukkan C: \ Program Files \ PostgreSQL \ x.y \ bin, di mana x.y adalah versi PostgreSQL yang Anda gunakan. Ini

akan terlihat seperti garis yang disorot pada Gambar 16.2. Setelah selesai, klik OK untuk menutup semua dialog.



Gambar 16.2: Mengedit variabel lingkungan PATH yang ada di Windows 10

Sekarang ketika Anda meluncurkan Command Prompt, PATH harus menyertakan direktori. Perhatikan bahwa setiap kali Anda membuat perubahan pada PATH, Anda harus menutup dan membuka kembali Command Prompt agar perubahan diterapkan. Selanjutnya, mari kita atur Command Prompt.

Meluncurkan dan Mengonfigurasi Prompt Perintah Windows

Command Prompt adalah file yang dapat dieksekusi bernama cmd.exe. Untuk meluncurkannya, pilih Mulai Sistem Windows Prompt Perintah. Saat aplikasi terbuka, Anda akan melihat jendela dengan latar belakang hitam yang menampilkan informasi versi dan hak cipta bersama dengan prompt yang menunjukkan direktori Anda saat ini. Pada sistem Windows 10 saya, Command Prompt terbuka ke direktori pengguna default saya dan menampilkan C: \ Users \ Anthony>, seperti yang ditunjukkan pada Gambar 16.3.

Gambar 16.3: Prompt Perintah Saya di Windows 10

Catatan: Untuk akses cepat ke Command Prompt, Anda dapat menambahkannya ke taskbar Windows Anda. Saat Command Prompt sedang berjalan, klik kanan ikonnya di bilah tugas, lalu pilih Sematkan ke bilah tugas.

Baris C:\Users\Anthony > menunjukkan bahwa direktori kerja Command Prompt saat ini adalah drive C: saya, yang biasanya merupakan hard drive utama pada sistem Windows, dan direktori\Users\Anthony pada drive itu. Panah kanan (>) menunjukkan area tempat Anda mengetik perintah.

Anda dapat menyesuaikan font dan warna serta mengakses pengaturan lain dengan mengklik ikon Command Prompt di sebelah kiri bilah jendela dan memilih Properties dari menu. Untuk membuat Command Prompt lebih cocok untuk output kueri, saya sarankan untuk mengatur ukuran jendela (pada tab Layout) ke lebar 80 dan tinggi 25. Font pilihan saya adalah Lucida Console 14, tetapi bereksperimenlah untuk menemukan yang Anda suka.

Memasukkan Instruksi di Command Prompt Windows

Sekarang Anda siap untuk memasukkan instruksi di Command Prompt. Masukkan bantuan saat diminta, dan tekan ENTER pada keyboard Anda untuk melihat daftar perintah yang tersedia. Anda dapat melihat informasi tentang perintah tertentu dengan mengetikkan namanya setelah bantuan. Misalnya, masukkan waktu bantuan untuk menampilkan informasi tentang penggunaan perintah waktu untuk mengatur atau melihat waktu sistem.

Menjelajahi cara kerja Command Prompt sepenuhnya berada di luar cakupan buku ini; namun, Anda harus mencoba beberapa perintah di Tabel 16.1, yang berisi perintah yang sering digunakan yang akan segera berguna tetapi tidak diperlukan untuk latihan di bab ini. Juga, lihat lembar contekan Command Prompt online untuk informasi lebih lanjut.

Tabel 16.1: Perintah Windows yang Berguna

Perintah	Fungsi	Contoh	Tindakan
cd	Ubah direktori	cd C:\my-stuff	Ubah ke direktori my-stuff di drive C:
copy	Salin file	copy C:\my-stuff\ song.mp3 C:\Music\ song_favorite.mp3	Salin file song.mp3 dari my-stuff ke file baru bernama song_favorite.mp3 di direktori Music
del	Menghapus	del *.jpg	Hapus semua file dengan ekstensi .jpg di direktori saat ini (tanda bintang wildcard)
dir	Daftar isi direktori	dir /p	Tampilkan konten direktori satu layar pada satu waktu (menggunakan opsi /p)
findstr	Temukan string dalam file teks yang cocok dengan	findstr "peach" *.txt	Cari teks "peach" di semua file .txt di direktori saat ini
mkdir	Buat direktori baru	makedir C:\my-stuff\ Salad	Buat direktori Salad di dalam direktori my-stuff
move	Pindahkan file	move C:\my-stuff\song .mp3 C:\Music\	Pindahkan file song.mp3 ke direktori C:\Music

Dengan Command Prompt Anda terbuka dan terkonfigurasi, Anda siap untuk melakukan roll. Lewati ke “Bekerja dengan psql”.

Pengaturan macOS psql

Di macOS, Anda akan menjalankan psql di dalam Terminal, aplikasi yang menyediakan akses ke baris perintah sistem itu melalui program shell yang disebut bash. Program Shell pada sistem berbasis Unix atau Linux, termasuk macOS, tidak hanya menyediakan prompt perintah tempat pengguna memasukkan instruksi, tetapi juga bahasa pemrograman mereka sendiri untuk mengotomatisasi tugas. Misalnya, Anda dapat menggunakan perintah bash untuk menulis program untuk masuk ke komputer jarak jauh, mentransfer file, dan keluar. Mari kita mulai dengan memberi tahu bash di mana menemukan psql dan utilitas baris perintah PostgreSQL lainnya dengan mengatur variabel lingkungan PATH. Kemudian kami akan meluncurkan Terminal.

Menambahkan psql dan Utilitas ke macOS PATH

Sebelum Terminal memuat bash shell, ia memeriksa keberadaan beberapa file teks opsional yang dapat menyediakan informasi konfigurasi. Kami akan menempatkan informasi PATH kami di dalam .bash_profile, yang merupakan salah satu file teks opsional ini. Kemudian, setiap kali kita membuka Terminal, proses startup harus membaca bash_profile dan mendapatkan nilai PATH.

Catatan: Anda juga dapat menggunakan .bash_profile untuk mengatur warna baris perintah, menjalankan program secara otomatis, dan membuat pintasan, di antara tugas-tugas lainnya. Lihat https://natelandau.com/ my-mac-osx-bash_profile/ untuk contoh yang bagus dalam menyesuaikan file.

Pada sistem berbasis Unix atau Linux, file yang dimulai dengan titik disebut file titik dan disembunyikan secara default. Kita perlu mengedit .bash_profile untuk menambahkan PATH. Dengan menggunakan langkah-langkah berikut, sembunyikan .bash_profile sehingga muncul di MacOS Finder:

1. Luncurkan Terminal dengan menavigasi ke Applications ➔Utilities ➔Terminal.
2. Pada prompt perintah, yang menampilkan nama pengguna dan nama komputer Anda diikuti dengan tanda dolar ($), masukkan teks berikut dan kemudian tekan RETURN:

```
defaults write com.apple.finder AppleShowAllFiles YES
```

3. Keluar dari Terminal (⌘-Q). Kemudian, sambil menahan tombol OPTION, klik kanan ikon Finder di dok Mac Anda, dan pilih Luncurkan kembali.

Ikuti langkah-langkah ini untuk mengedit atau membuat .bash_profile:

1. Menggunakan MacOS Finder, navigasikan ke direktori pengguna Anda dengan membuka Finder dan klik Macintosh HD lalu Users.
2. Buka direktori pengguna Anda (harus memiliki ikon rumah). Karena Anda mengubah pengaturan untuk menampilkan file tersembunyi, Anda sekarang akan melihat file dan direktori berwarna abu-abu, yang biasanya tersembunyi, bersama dengan file dan direktori biasa.
3. Periksa file .bash_profile yang ada. Jika ada, klik kanan dan buka dengan editor teks pilihan Anda atau gunakan app MacOS TextEdit. Jika .bash_profile tidak ada, buka TextEdit untuk membuat dan menyimpan file dengan nama tersebut ke direktori pengguna Anda.

Selanjutnya, kami akan menambahkan pernyataan PATH ke .bash_profile. Petunjuk ini mengasumsikan Anda menginstal PostgreSQL menggunakan Postgres.app, seperti yang dijelaskan dalam “Instalasi macOS”. Untuk menambahkan ke jalur, tempatkan baris berikut di .bash_profile:

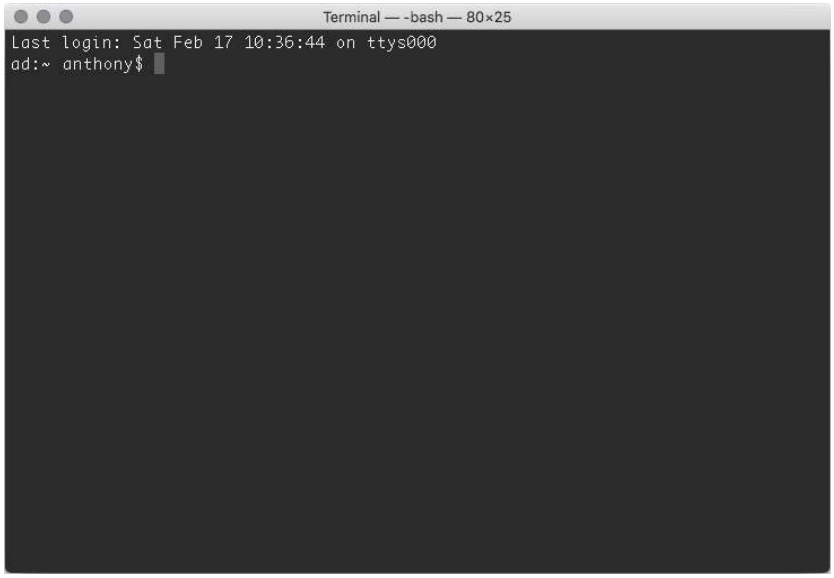
```
Export PATH=”/Application/Postgres.app/Contents/version/lastest/bin:$PATH”
```

Simpan dan tutup file. Jika Terminal terbuka, tutup dan luncurkan kembali sebelum melanjutkan ke bagian berikutnya.

Meluncurkan dan Mengonfigurasi Terminal macOS

Luncurkan Terminal dengan menavigasi ke Aplikasi Utilitas Terminal. Ketika terbuka, Anda akan melihat jendela yang menampilkan tanggal dan waktu login terakhir Anda diikuti dengan prompt yang menyertakan nama komputer Anda, direktori kerja saat ini, dan nama pengguna, diakhiri dengan tanda dolar ($).

Di Mac saya, prompt menampilkan iklan: ~ anthony $, seperti yang ditunjukkan pada Gambar 16.4.



Gambar 16.4: Baris perintah terminal di macOS

Tilde (~) menunjukkan bahwa Terminal sedang bekerja di direktori home saya, yaitu /Users /anthony. Terminal tidak menampilkan jalur direktori lengkap, tetapi Anda dapat melihat informasi itu kapan saja dengan memasukkan perintah pwd (kependekan dari “print working directory”) dan menekan RETURN pada keyboard Anda. Area setelah tanda dolar adalah tempat Anda mengetik perintah.

Catatan; Untuk akses cepat ke Terminal, tambahkan ke Dock macOS Anda. Saat Terminal berjalan, klik kanan ikonnya dan pilih Opsi Simpan di Dock.

Jika Anda belum pernah menggunakan Terminal, skema warna hitam putih defaultnya mungkin tampak membosankan. Anda dapat mengubah font, warna, dan pengaturan lainnya dengan memilih Terminal Preferences. Untuk membuat Terminal lebih besar agar lebih sesuai dengan tampilan keluaran kueri, saya sarankan untuk mengatur ukuran jendela (pada tab Jendela) dengan lebar 80 kolom dan tinggi 25 baris. Font pilihan saya (pada tab Teks) adalah Monaco 14, tetapi bereksperimenlah untuk menemukan yang Anda sukai.

Menjelajahi cara kerja penuh Terminal dan perintah terkait berada di luar cakupan buku ini, tetapi luangkan waktu untuk mencoba beberapa perintah. Tabel 16.2 mencantumkan perintah-perintah yang biasa digunakan yang akan segera Anda temukan berguna tetapi tidak diperlukan untuk latihan dalam bab ini. Masukkan man (kependekan dari "manual") diikuti dengan nama perintah untuk mendapatkan bantuan pada perintah apa pun. Misalnya, Anda dapat menggunakan man ls untuk mengetahui cara menggunakan perintah ls untuk membuat daftar isi direktori.

Tabel 16.2: Perintah Terminal yang Berguna

Perintah	Fungsi	Contoh	Tindakan
cd	Ubah direktori	cd /Users/pparker/ my-stuff/	Ubah ke direktori barang saya
cp	Salin file	cp song.mp3 song_backup.mp3	Salin file song.mp3 ke song_backup.mp3 di direktori saat ini
grep	Temukan string dalam file teks yang cocok dengan yang biasa	grep 'us_counties _2010' *.sql	Temukan semua baris dalam file dengan ekstensi .sql yang memiliki teks “us_counties_2010”
ls	Daftar isi direktori	ls -al	Daftar semua file dan direktori (termasuk yang tersembunyi) dalam format "panjang"
mkdir	Buat direktori baru	mkdir resumes	Buat direktori bernama resume di bawah direktori kerja saat ini
mv	Pindahkan file	mv song.mp3 /Users/ pparker/songs	Pindahkan file song.mp3 dari direktori saat ini ke direktori /songs di bawah direktori pengguna
rm	Hapus file	rm *.jpg	Hapus semua file dengan ekstensi .jpg di direktori saat ini (tanda bintang wildcard)

Pengaturan psql Linux

Ingat dari “Instalasi Linux” pada halaman xxxi bahwa metode untuk menginstal PostgreSQL berbeda-beda menurut distribusi Linux Anda. Namun demikian, psql adalah bagian dari instalasi PostgreSQL standar, dan Anda mungkin sudah menjalankan perintah psql sebagai bagian dari proses instalasi melalui aplikasi terminal baris perintah distribusi Anda. Bahkan jika Anda tidak melakukannya, instalasi Linux standar PostgreSQL akan secara otomatis menambahkan psql ke PATH Anda, jadi Anda seharusnya dapat mengaksesnya.

Luncurkan aplikasi terminal. Pada beberapa distribusi, seperti Ubuntu, Anda dapat membuka terminal dengan menekan CTRL-ALT-T. Perhatikan juga bahwa perintah Terminal macOS pada Tabel 16.2 juga berlaku untuk Linux dan mungkin berguna bagi Anda.

Dengan terminal Anda terbuka, Anda siap untuk melakukan roll. Lanjutkan ke bagian berikutnya, "Bekerja dengan psql."

Bekerja dengan psql

Sekarang setelah Anda mengidentifikasi antarmuka baris perintah Anda dan mengaturnya untuk mengenali lokasi psql, mari luncurkan psql dan sambungkan ke database pada instalasi lokal PostgreSQL Anda. Kemudian kita akan mengeksplorasi mengeksekusi query dan perintah khusus untuk mengambil informasi database.

Meluncurkan psql dan Menghubungkan ke Database

Terlepas dari sistem operasi yang Anda gunakan, Anda memulai psql dengan cara yang sama. Buka antarmuka baris perintah Anda (Command Prompt di Windows, Terminal di macOS atau Linux). Untuk meluncurkan psql, kami menggunakan pola berikut pada command prompt:

```
psql –d database_name –U user_name
```

Mengikuti nama aplikasi psql, kami memberikan nama database setelah argumen -d dan nama pengguna setelah -U.

Untuk nama database, kami akan menggunakan analisis, di mana kami membuat sebagian besar tabel kami untuk latihan buku. Untuk nama pengguna, kami akan menggunakan postgres, yang merupakan pengguna default yang dibuat saat instalasi. Misalnya, untuk menghubungkan mesin lokal Anda ke database analisis, Anda akan memasukkan ini:

```
psql –d analysis –U postgres
```

Anda dapat terhubung ke database di server jauh dengan menentukan argumen -h diikuti dengan nama host. Misalnya, Anda akan menggunakan baris berikut jika Anda terhubung ke komputer di server bernama example.com:

```
psql –d analysis –U postgres –h example.com
```

Jika Anda menetapkan kata sandi selama instalasi, Anda akan menerima prompt kata sandi saat psql diluncurkan. Jika sudah, masukkan kata sandi Anda dan tekan ENTER. Anda kemudian akan melihat prompt yang terlihat seperti ini:

```
psql (10.1
Type "Help" for help.
Analysis=#
```

Di sini, baris pertama mencantumkan nomor versi psql dan server yang terhubung dengan Anda. Versi Anda akan bervariasi tergantung pada saat Anda menginstal PostgreSQL. Prompt di mana Anda akan memasukkan perintah adalah analysis =#, yang mengacu pada nama database, diikuti dengan tanda sama dengan (=) dan tanda hash (#). Tanda hash menunjukkan

bahwa Anda masuk dengan hak pengguna super, yang memberi Anda kemampuan tak terbatas untuk mengakses dan membuat objek serta mengatur akun dan keamanan. Jika Anda masuk sebagai pengguna tanpa hak pengguna super, karakter terakhir dari prompt akan menjadi tanda lebih besar dari (>).

Seperti yang Anda lihat, akun pengguna yang Anda gunakan untuk masuk di sini (postgres) adalah pengguna super.

Catatan: Instalasi PostgreSQL membuat akun pengguna super default bernama postgres. Jika Anda menjalankan postgres.app di macOS, penginstalan itu membuat akun pengguna super tambahan yang memiliki nama pengguna sistem Anda dan tanpa kata sandi.

Mendapatkan bantuan

Pada prompt psql, Anda dapat dengan mudah mendapatkan bantuan dengan perintah psql dan perintah SQL. Tabel 16.3 mencantumkan perintah yang dapat Anda ketikkan pada prompt psql dan menunjukkan informasi yang akan ditampilkan.

Tabel 16.3: Perintah Bantuan Dalam psql

Perintah	Menampilkan
\?	Perintah yang tersedia dalam psql, seperti \ dt untuk membuat daftar tabel.
\? Options	Opsi untuk digunakan dengan perintah psql, seperti -U untuk menentukan nama pengguna.
\? Variables	Variabel untuk digunakan dengan psql, seperti VERSION untuk versi psql saat ini.
\h	Daftar perintah SQL. Tambahkan nama perintah untuk melihat bantuan terperinci untuknya (misalnya, \ h INSERT).

Bahkan pengguna berpengalaman sering membutuhkan penyegaran pada perintah dan opsi, dan memiliki detail dalam aplikasi psql sangat berguna. Mari kita lanjutkan dan jelajahi beberapa perintah.

Mengubah Koneksi Pengguna dan Basis Data

Anda dapat menggunakan serangkaian perintah meta, yang didahului dengan garis miring terbalik, untuk mengeluarkan instruksi ke psql daripada ke database. Misalnya, untuk menyambungkan ke database lain atau mengalihkan akun pengguna yang tersambung, Anda dapat menggunakan perintah meta \ c. Untuk beralih ke database gis_analysis yang kita buat di Bab 14, masukkan \ c diikuti dengan nama database pada prompt psql:

```
analysis=# \c gis_analysis
```

Aplikasi harus merespons dengan pesan berikut:

```
You are now connected to database “gis_analysis” as user “postgres”.
gis_analysis=#
```

Untuk masuk sebagai pengguna yang berbeda, misalnya, menggunakan nama pengguna yang dibuat oleh instalasi macOS untuk saya, saya dapat menambahkan nama pengguna itu setelah nama basis data. Di Mac saya, sintaksnya terlihat seperti ini:

```
analysis=# \c gis_analysis anthony
```

Responsnya harus sebagai berikut:

```
You are now connected to database "gis_analysis" as "anthony".
gis_analysis=#
```

Anda mungkin memiliki berbagai alasan untuk menggunakan beberapa akun pengguna seperti ini. Misalnya, Anda mungkin ingin membuat akun pengguna dengan izin terbatas untuk kolega atau untuk aplikasi database. Anda dapat mempelajari lebih lanjut tentang membuat dan mengelola peran pengguna dengan membaca dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/sql-createrole.html.

Mari beralih kembali ke database analisis menggunakan perintah \ c. Selanjutnya, kita akan memasukkan perintah SQL pada prompt psql.

Menjalankan Kueri SQL di psql

Kami telah mengonfigurasi psql dan terhubung ke database, jadi sekarang mari kita jalankan beberapa kueri SQL, dimulai dengan kueri satu baris dan kemudian kueri multibaris.

Untuk memasukkan SQL ke psql, Anda dapat mengetikkannya langsung pada prompt. Misalnya, untuk melihat beberapa baris dari tabel Sensus 2010 yang telah kami gunakan di seluruh buku, masukkan kueri saat diminta, seperti yang ditunjukkan pada sintaks ini:

```
analysis=# SELECT geo_name FROM us_counties_2010 LIMIT 3;
```

Tekan ENTER untuk menjalankan kueri, dan psql akan menampilkan hasil berikut dalam teks termasuk jumlah baris yang dikembalikan:

```
geo_name
--------------
Autauga County
Baldwin County
Barbour County
(3 rows)

analysis=#
```

Di bawah hasilnya, Anda dapat melihat kembali prompt analysis =#, siap untuk input lebih lanjut dari pengguna. Tekan panah atas dan bawah pada keyboard untuk menggulir kueri terbaru agar tidak perlu mengetik ulang. Atau Anda cukup memasukkan kueri baru.

Memasuki Query Multiline Multi

Anda tidak terbatas pada kueri satu baris. Misalnya, Anda dapat menekan ENTER setiap kali Anda ingin memasukkan baris baru. Perhatikan bahwa psql tidak akan menjalankan kueri sampai Anda memberikan baris yang diakhiri dengan titik koma. Untuk melihat contoh,

masukkan kembali kueri pada kueri diatas menggunakan format yang ditunjukkan pada kueri dibawah ini:

```
analysis=# SELECT geo_name
analysis=# FROM us_counties_2010
analysis=# LIMIT 3;
```

Perhatikan bahwa ketika kueri Anda melampaui satu baris, simbol antara nama database dan tanda pagar berubah dari tanda sama dengan (=) menjadi tanda hubung (-). Kueri multibaris ini hanya dijalankan saat Anda menekan ENTER setelah baris terakhir, yang diakhiri dengan titik koma.

Memeriksa Tanda kurung Terbuka di Prompt psql

Fitur lain yang bermanfaat dari psql adalah ia muncul saat Anda belum menutup sepasang tanda kurung. Kueri dibawah ini menunjukkan ini dalam tindakan:

```
analysis=# CREATE TABLE wineries (
analysis(# id bigint,
analysis(# winery_name varchar(100)
analysis(# );
CREATE TABLE
```

Di sini, Anda membuat tabel sederhana yang disebut kilang anggur yang memiliki dua kolom. Setelah memasukkan baris pertama dari pernyataan CREATE TABLE dan kurung buka, prompt kemudian berubah dari analysis =# menjadi analysis (# untuk menyertakan kurung terbuka yang mengingatkan Anda bahwa kurung terbuka perlu ditutup. Prompt mempertahankan konfigurasi itu sampai Anda tambahkan kurung tutup.

Catatan: Jika Anda memiliki kueri panjang yang disimpan dalam file teks, seperti salah satu dari sumber buku ini, Anda dapat menyalinnya ke papan klip komputer Anda dan menempelkannya ke psql (ctrL-V di Windows, -V di macOS, dan Shift -ctrL-V di Linux). Itu menyelamatkan Anda dari mengetik seluruh kueri. Setelah Anda menempelkan teks kueri ke psql, tekan eNter untuk menjalankannya.

Mengedit Kueri

Jika Anda bekerja dengan kueri di psql dan ingin memodifikasinya, Anda dapat mengeditnya menggunakan \ e atau \ edit meta-command. Masukkan \ e untuk membuka kueri yang terakhir dieksekusi dalam editor teks. Editor mana yang digunakan psql secara default tergantung pada sistem operasi Anda.

Pada Windows, psql default ke Notepad, editor teks GUI sederhana. Di macOS dan Linux, psql menggunakan aplikasi baris perintah yang disebut vim, yang merupakan favorit di antara programmer tetapi tampaknya sulit dipahami oleh pemula. Lihat lembar contekan vim yang bermanfaat di https://vim.rtorr.com/. Untuk saat ini, Anda dapat menggunakan langkah-langkah berikut untuk melakukan pengeditan sederhana:

- Saat vim membuka kueri di jendela pengeditan, tekan I untuk mengaktifkan mode penyisipan.
- Lakukan pengeditan pada kueri.

- Tekan esc lalu SHIFT+: untuk menampilkan prompt perintah titik dua di kiri bawah layar vim, di mana Anda memasukkan perintah untuk mengontrol vim.
- Masukkan wq (untuk "menulis, keluar") dan tekan ENTER untuk menyimpan perubahan Anda.

Sekarang ketika Anda keluar ke prompt psql, itu harus menjalankan kueri Anda yang direvisi. Tekan tombol panah atas untuk melihat teks yang direvisi.

Hasil Navigasi dan Format

Kueri yang Anda jalankan di kueri tersebut diatas hanya mengembalikan satu kolom dan beberapa baris, sehingga outputnya dimuat dengan baik di antarmuka baris perintah Anda. Namun untuk kueri dengan lebih banyak kolom atau baris, output dapat mengambil lebih dari satu layar, sehingga sulit untuk dinavigasi. Untungnya, Anda dapat menggunakan opsi pemformatan menggunakan perintah meta \pset untuk menyesuaikan output ke dalam format yang Anda inginkan.

Mengatur Paging Hasil

Anda dapat menyesuaikan format output dengan menentukan bagaimana psql menampilkan hasil kueri yang panjang. Misalnya, List dibawah ini menunjukkan perubahan format output saat kita menghapus klausa LIMIT dari kueri pada pembahasan sebelumnya dan menjalankannya pada prompt psql:

```
geo_name
-----------------------------------------------
Autauga County
Baldwin County
Barbour County
Bibb County
Blount County
Bullock County
Butler County
Calhoun County
Chambers County
Cherokee County
Chilton County
Choctaw County
Clarke County
Clay County
Cleburne County
Coffee County
Colbert County
:
```

Ingatlah bahwa tabel ini memiliki 3.143 baris. Kueri diatas hanya menampilkan 17 pertama di layar dengan titik dua di bagian bawah (jumlah baris yang terlihat tergantung pada konfigurasi terminal Anda). Titik dua menunjukkan bahwa ada lebih banyak hasil daripada yang ditampilkan; tekan tombol panah bawah untuk menggulirnya. Menggulir melalui banyak baris ini bisa memakan waktu cukup lama. Tekan Q kapan saja untuk keluar dari hasil pengguliran dan kembali ke prompt psql.

Anda dapat meminta hasil Anda segera menggulir sampai akhir dengan mengubah pengaturan pager menggunakan \ pset pager meta-command. Jalankan perintah itu di prompt psql Anda, dan itu akan mengembalikan pesan penggunaan Pager tidak aktif. Sekarang ketika Anda menjalankan kembali kueri pada pembahasan diatas dengan pengaturan pager dimatikan, Anda akan melihat sesuatu seperti ini:

```
--snip--
Niobrara County
Park County
Platte County
Sheridan County
Sublette County
Sweette County
Teton County
Uinta County
Washakie County
Weston County
(3143 rows)

analysis=#
```

Anda langsung dibawa ke akhir hasil tanpa harus menggulir. Untuk mengaktifkan kembali paging, jalankan \ pset pager lagi.

Memformat Hasil Grid

Anda juga dapat menggunakan perintah meta \ pset dengan opsi berikut untuk memformat tampilan hasil:

border int Gunakan opsi ini untuk menentukan apakah kisi hasil tidak memiliki batas (0), garis internal yang membagi kolom (1), atau garis di sekitar semua sel (2). Misalnya, \ pset border 2 menyetel garis di sekitar semua sel.

format unaligned Gunakan opsi \ pset format unaligned untuk menampilkan hasil dalam baris yang dipisahkan oleh pembatas daripada di kolom, mirip dengan apa yang akan Anda lihat di file CSV. Pemisah default ke simbol pipa (|). Anda dapat mengatur pemisah yang berbeda menggunakan perintah fieldsep. Misalnya, untuk menetapkan koma sebagai pemisah, jalankan \ pset fieldsep ','. Untuk kembali ke tampilan kolom, jalankan \ format pset selaras. Anda dapat menggunakan perintah meta psql \ a untuk beralih antara tampilan selaras dan tidak selaras.

footer Gunakan opsi ini untuk mengaktifkan atau menonaktifkan footer hasil, yang menampilkan jumlah baris hasil.

null Gunakan opsi ini untuk mengatur bagaimana nilai null ditampilkan. Secara default, mereka ditampilkan sebagai kosong. Anda dapat menjalankan \ pset null 'NULL' untuk mengganti yang kosong dengan huruf besar semua NULL ketika nilai kolom adalah NULL.

Anda dapat menjelajahi opsi tambahan dalam dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/app-psql.html. Selain itu, dimungkinkan untuk mengatur file .psqlrc di macOS atau Linux atau file psqlrc.conf di Windows untuk menyimpan preferensi konfigurasi Anda dan memuatnya setiap kali psql dimulai. Contoh yang baik disediakan di https://www.citusdata.com/blog/2017/07/16/ customizing-my-postgres-shell-using-psqlrc/.

Melihat Hasil yang Diperluas

Terkadang, sangat membantu untuk melihat hasil sebagai daftar blok vertikal daripada dalam baris dan kolom, terutama ketika data terlalu besar untuk muat di layar dalam kisi hasil horizontal normal. Juga, saya sering menggunakan format ini ketika saya menginginkan cara yang mudah untuk memindai untuk meninjau nilai dalam kolom berdasarkan baris demi baris. Di psql, Anda dapat beralih ke tampilan ini menggunakan perintah meta \ x (untuk diperluas). Cara terbaik untuk memahami perbedaan antara tampilan normal dan tampilan diperluas adalah dengan melihat sebuah contoh. Kueri dibawah ini menunjukkan tampilan normal yang Anda lihat saat menanyakan tabel nilai di Bab 15 menggunakan psql:

```
analysis=# SELECT * FROM grades;
student_id | course_id |    course          | grade
-----------------------------------------------
         1 |         2 | English 11B        | D
         1 |         3 | World History 11B  | C
         1 |         4 | Trig 2             | B
         1 |         1 | Biology 2          | C
(4 rows)
```

Untuk mengubah ke tampilan yang diperluas, masukkan \ x pada prompt psql, yang akan menampilkan pesan Expanded display is on. Kemudian, saat Anda menjalankan kueri yang sama lagi, Anda akan melihat hasil yang diperluas, seperti yang diperlihatkan dalam sintaks dibawah ini:

```
analysis=# SELECT * FROM grades;
-[ RECORD 1 ]------------
student_iq  | 1
course_id   | 2
course      | English 11B
grade       | D

-[ RECORD 2 ]------------
student_iq  | 1
course_id   | 3
course      | World History 11B
grade       | C

-[ RECORD 3 ]------------
student_iq  | 1
course_id   | 4
course      | Trig 2
grade       | B

-[ RECORD 4 ]------------
student_iq  | 1
course_id   | 1
course      | Biology 2
grade       | C
```

Hasilnya muncul dalam blok vertikal yang dipisahkan oleh nomor catatan. Bergantung pada kebutuhan Anda dan tipe data yang sedang Anda kerjakan, format ini mungkin lebih mudah dibaca. Anda dapat kembali ke tampilan kolom dengan memasukkan \ x lagi pada prompt psql. Selain itu, pengaturan \ x auto akan membuat PostgreSQL secara otomatis menampilkan hasil dalam tabel atau tampilan yang diperluas berdasarkan ukuran output.

Selanjutnya, mari kita jelajahi cara menggunakan psql untuk menggali informasi database.

Meta-Commands untuk Informasi Basis Data

Selain menulis kueri dari baris perintah, Anda juga dapat menggunakan psql untuk menampilkan detail tentang tabel dan objek serta fungsi lain di database Anda. Untuk melakukan ini, Anda menggunakan serangkaian perintah meta yang dimulai dengan \ d dan menambahkan tanda plus (+) untuk memperluas output. Anda juga dapat menyediakan pola opsional untuk memfilter output.

Misalnya, Anda dapat memasukkan \ dt+ untuk membuat daftar semua tabel dalam database dan ukurannya. Berikut cuplikan output di sistem saya:

```
                        List of relations
Schema |        Name         | Type  |   Owner  |    Size    | Descripion
-------------------------------------------------------------------------
public | acs_2011_2015_stats | table | postgres | 320 kB     |
public | crime_reports       | table | postgres | 16 kB      |
public | date_time_types     | table | postgres | 8192 bytes |
public | departments         | table | postgres | 8192 bytes |
public | employees           | table | postgres | 8192 bytes |
--snip--
```

Hasil ini mencantumkan semua tabel dalam database saat ini menurut abjad.

Anda dapat memfilter output dengan menambahkan pola untuk dicocokkan menggunakan ekspresi reguler. Misalnya, gunakan \ dt+ us* untuk hanya menampilkan tabel yang namanya dimulai dengan kami (tanda bintang berfungsi sebagai wildcard). Hasilnya akan terlihat seperti ini:

```
                        List of relations
Schema |        Name         | Type  |   Owner  |    Size    | Descripion
-------------------------------------------------------------------------
public | us_counties_2000    | table | postgres | 336 kB     |
public | us_counties_2000    | table | postgres | 1352 kB    |
```

Tabel 16.4 menunjukkan beberapa perintah \ d tambahan yang mungkin berguna bagi Anda.

Tabel 16.4: Contoh Perintah psql \ d

Perintah	**Menampilkan**
\d [pattern]	Kolom, tipe data, ditambah informasi lain tentang objek
\di [pattern]	Indeks dan tabel terkaitnya
\dt [pattern]	Tabel dan akun yang memilikinya
\du [pattern]	Akun pengguna dan atributnya
\dv [pattern]	Tampilan dan akun yang memilikinya
\dx [pattern]	Ekstensi terpasang

Seluruh daftar perintah \ d tersedia dalam dokumentasi PostgreSQL di https://www.postgresql.org/docs/current/static/app-psql.html , atau Anda dapat melihat detailnya dengan menggunakan \? perintah yang disebutkan sebelumnya.

Mengimpor, Mengekspor, dan Menggunakan File

Sekarang mari kita jelajahi cara memasukkan dan mengeluarkan data dari tabel atau menyimpan informasi saat Anda bekerja di server jauh. Alat baris perintah psql menawarkan satu perintah meta untuk mengimpor dan mengekspor data (\ copy) dan satu lagi untuk menyalin output kueri ke file (\ o). Kita akan mulai dengan perintah \ copy.

Menggunakan \copy untuk Impor dan Ekspor

Di Bab 4, Anda telah mempelajari cara menggunakan perintah SQL COPY untuk mengimpor dan mengekspor data. Ini adalah proses yang mudah, tetapi ada satu batasan signifikan: file yang Anda impor atau ekspor harus berada di mesin yang sama dengan server PostgreSQL. Tidak apa-apa jika Anda sedang mengerjakan mesin lokal Anda, seperti yang telah Anda lakukan dengan latihan ini. Tetapi jika Anda terhubung ke database di komputer jarak jauh, Anda mungkin tidak memiliki akses ke sistem file untuk menyediakan file yang akan diimpor atau untuk mengambil file yang telah Anda ekspor. Anda dapat mengatasi batasan ini dengan menggunakan \ copy meta-command di psql. \ copy meta-command bekerja seperti perintah SQL COPY kecuali ketika Anda menjalankannya pada prompt psql, ia dapat merutekan data dari mesin lokal Anda ke server jauh jika itu yang Anda sambungkan. Kami tidak akan benar-benar terhubung ke server jauh untuk mencoba ini, tetapi Anda masih dapat mempelajari sintaksnya.

Dalam kueri dibawah ini, kami menggunakan psql untuk MENGHAPUS tabel state_regions kecil yang Anda buat di Bab 9, lalu membuat ulang tabel dan mengimpor data menggunakan \copy. Anda harus mengubah jalur file agar sesuai dengan lokasi file di komputer Anda.

```
analysis=# DROP TABLE state_regions:
DROP TABLE

analysis=# CREATE TABLE state_regions (
analysis=(# st varchar(2) CONSTRAINT st_key PRIMARY KEY,
analysis=(# region varchar(20) NOT NULL
analysis=(# );

CREATE TABLE

analysis=# \copy state_regions FROM 'C:\YourDirectory\state_regions.csv'
WITH (FORMAT CSV, HEADER);
COPY 56
```

Pernyataan DROP TABLE dan CREATE TABLE pada sintaks diatas tersebut adalah lurus ke depan. Kami pertama-tama menghapus tabel state_regions jika ada, lalu membuatnya kembali. Kemudian, untuk memuat tabel, kami menggunakan \ copy dengan sintaks yang sama yang digunakan dengan SQL COPY, penamaan klausa FROM yang menyertakan jalur file pada mesin Anda, dan klausa WITH yang menentukan file tersebut adalah CSV dan memiliki baris header. Saat Anda menjalankan pernyataan, server akan merespons dengan COPY 56, memberi tahu Anda bahwa baris telah berhasil diimpor.

Jika Anda terhubung melalui psql ke server jauh, Anda akan menggunakan sintaks \ copy yang sama, dan perintah hanya akan merutekan file lokal Anda ke server jauh untuk diimpor. Dalam contoh ini, kami menggunakan \ copy FROM untuk mengimpor file. Kita juga bisa menggunakan \ copy TO untuk mengekspor. Mari kita lihat cara lain untuk mengekspor output ke file.

Menyimpan Output Kueri ke File

Terkadang berguna untuk menyimpan hasil kueri dan pesan yang dihasilkan selama sesi psql ke file, baik untuk menyimpan riwayat pekerjaan Anda atau untuk menggunakan output dalam

spreadsheet atau aplikasi lain. Untuk mengirim output kueri ke file, Anda dapat menggunakan \o perintah meta bersama dengan path lengkap dan nama file output.

Catatan: Di Windows, jalur file untuk perintah \ o harus menggunakan garis miring gaya Linux, seperti C: /my-stuff/my-file.txt, atau garis miring terbalik ganda, seperti C:\\my-stuff\\file-saya.txt.

Misalnya, salah satu trik favorit saya adalah mengatur format output menjadi tidak selaras dengan koma sebagai pemisah bidang dan tidak ada jumlah baris di footer, serupa tetapi tidak identik dengan output CSV. (Ini tidak identik karena file CSV yang sebenarnya, seperti yang Anda pelajari di Bab 4, dapat menyertakan karakter untuk mengutip nilai yang berisi pembatas. Namun, trik ini berfungsi untuk keluaran seperti CSV sederhana.) Kueri dibawah ini menunjukkan urutan perintah pada prompt psql:

```
analysis=# \a \f , \pset footer
Output format is unaligned.
Field separator is “,”.
Default footer is off.

analysis=# SELECT * FROM grades;
student_id, course_id, course, grade
1,2,English 11B,D
1,3,World History 11B,C
1,4,Trig 2,B
1,1,Biology 2,C

analysis=# \o ‘C:/YourDirectory/query_output.csv’

analysis=# SELECT * FROM grades;
analysis=#
```

Pertama, atur format output menggunakan meta-commands \a, \f, dan \pset footer untuk data yang tidak selaras, dipisahkan koma tanpa footer. Saat Anda menjalankan kueri SELECT sederhana pada tabel nilai, keluaran akan kembali sebagai nilai yang dipisahkan dengan koma. Selanjutnya, untuk mengirim data tersebut ke file saat Anda menjalankan kueri berikutnya, gunakan \o meta-command dan kemudian berikan jalur lengkap ke file bernama query_output.csv. Saat Anda menjalankan kueri SELECT lagi, seharusnya tidak ada output ke layar. Sebagai gantinya, Anda akan menemukan file dengan konten kueri di direktori yang ditentukan. Perhatikan bahwa setiap kali Anda menjalankan kueri dari titik ini, output ditambahkan ke file yang sama yang ditentukan setelah perintah \o. Untuk berhenti menyimpan output ke file itu, Anda dapat menentukan file baru atau memasukkan \o tanpa nama file untuk melanjutkan dengan menampilkan hasil ke layar.

Membaca dan Mengeksekusi SQL yang Disimpan dalam File

Anda dapat menjalankan SQL yang disimpan dalam file teks dengan mengeksekusi psql pada baris perintah dan memberikan nama file setelah argumen -f. Sintaks ini memungkinkan Anda dengan cepat menjalankan kueri atau pembaruan tabel dari baris perintah atau bersama dengan penjadwal sistem untuk menjalankan pekerjaan secara berkala.

Katakanlah Anda menyimpan nilai SELECT * FROM; kueri dalam file bernama display-grades.sql. Untuk menjalankan kueri yang disimpan, gunakan sintaks psql berikut di baris perintah Anda:

```
Psql –d analysis –U postgres –f display-grades.sql
```

Saat Anda menekan ENTER, psql akan diluncurkan, menjalankan kueri yang disimpan dalam file, menampilkan hasilnya, dan keluar. Untuk tugas yang berulang, alur kerja ini dapat menghemat banyak waktu karena Anda menghindari peluncuran pgAdmin atau menulis ulang kueri. Anda juga dapat menumpuk beberapa kueri dalam file sehingga kueri tersebut berjalan secara berurutan, yang, misalnya, mungkin Anda lakukan jika ingin menjalankan beberapa pembaruan pada database Anda.

Utilitas Baris Perintah Tambahan untuk Mempercepat Tugas

PostgreSQL menyertakan utilitas baris perintah tambahan yang berguna jika Anda terhubung ke server jarak jauh atau hanya ingin menghemat waktu dengan menggunakan baris perintah alih-alih meluncurkan pgAdmin atau GUI lain. Anda dapat memasukkan perintah ini di antarmuka baris perintah Anda tanpa meluncurkan psql. Daftar tersedia di https://www.postgresql.org/docs/current/static/reference-client.html , dan saya akan menjelaskan beberapa di Bab 17 yang khusus untuk pemeliharaan database. Tapi di sini saya akan membahas dua yang sangat berguna: membuat basis data pada baris perintah dengan utilitas createb dan memuat shapefile ke dalam database PostGIS melalui utilitas shp2pgsql.

Menambahkan Database dengan Createdb

Pernyataan SQL pertama yang Anda pelajari di Bab 1 adalah CREATE DATABASE, yang Anda gunakan untuk menambahkan analisis database ke server PostgreSQL Anda. Daripada meluncurkan pgAdmin dan menulis pernyataan CREATE DATABASE, Anda dapat melakukan tindakan serupa menggunakan utilitas baris perintah Createdb. Misalnya, untuk membuat database baru di server bernama box_office, jalankan perintah berikut di baris perintah Anda:

```
created –U postgres –e box_office
```

Argumen -U memberi tahu perintah untuk terhubung ke server PostgreSQL menggunakan akun postgres. Argumen -e (untuk "echo") memberitahu perintah untuk mencetak pernyataan SQL ke layar. Menjalankan perintah ini menghasilkan respons CREATE DATABASE box_office; selain membuat database. Anda kemudian dapat terhubung ke database baru melalui psql menggunakan baris berikut:

```
psql –d box_office –U postgres
```

Perintah Createdb menerima argumen untuk terhubung ke server jauh (seperti yang dilakukan psql) dan untuk mengatur opsi untuk database baru. Daftar lengkap argumen tersedia di https://www.postgresql.org/docs/current/static/ app-createdb.html. Sekali lagi, perintah Createdb adalah penghemat waktu yang berguna saat Anda tidak memiliki akses ke GUI.

Memuat Shapefile dengan shp2pgsql

Di Bab 14, Anda belajar mengimpor shapefile ke dalam database dengan Shapefile Import/Export Manager yang disertakan dalam paket PostGIS. GUI alat itu mudah dinavigasi, tetapi mengimpor shapefile menggunakan alat baris perintah PostGIS shp2pgsql memungkinkan Anda menyelesaikan hal yang sama menggunakan satu perintah teks.

Untuk mengimpor shapefile ke tabel baru dari baris perintah, gunakan sintaks berikut:

```
shp2psql –I –s SRID –W encoding shapefile_name table_name | psql –d database –U user
```

Banyak yang terjadi dalam satu baris ini. Berikut rincian argumen (jika Anda melewatkan Bab 14, Anda mungkin perlu meninjaunya sekarang):
-I Menambahkan indeks GiST pada kolom geometri tabel baru.
-s Memungkinkan Anda menentukan SRID untuk data geometris.
-W Memungkinkan Anda menentukan pengkodean. (Ingat bahwa kita menggunakan Latin1 untuk shapefile sensus.)
shapefile_name Nama (termasuk path lengkap) file yang diakhiri dengan ekstensi .shp.
table_name Nama tabel tempat shapefile diimpor.

Mengikuti argumen ini, Anda menempatkan simbol pipa (|) untuk mengarahkan output shp2pgsql ke psql, yang memiliki argumen untuk penamaan database dan pengguna. Misalnya, untuk memuat shapefile tl_2010_us_county10.shp ke dalam tabel us_counties_2010_shp di database gis_analysis, seperti yang Anda lakukan di Bab 14, Anda cukup menjalankan perintah berikut.

Perhatikan bahwa meskipun perintah ini membungkus dua baris di sini, itu harus dimasukkan sebagai satu baris di baris perintah:

```
shp2psql –I –s 4269 –W Latini tl_2010_us_county10,shp us_counties_2010_shp | psql
–d gis_analysis –U postgres
```

Server harus merespons dengan sejumlah pernyataan SQL INSERT sebelum membuat indeks dan mengembalikan Anda ke baris perintah. Mungkin perlu beberapa waktu untuk menyusun seluruh rangkaian argumen untuk pertama kalinya. Tetapi setelah Anda melakukannya, impor berikutnya akan memakan waktu lebih sedikit karena Anda cukup mengganti nama file dan tabel ke dalam sintaks yang sudah Anda tulis.

Apakah Anda merasa misterius dan kuat? Memang, ketika Anda mempelajari antarmuka baris perintah dan membuat komputer melakukan perintah Anda menggunakan perintah teks, Anda memasuki dunia komputasi yang menyerupai urutan film fiksi ilmiah. Bekerja dari baris perintah tidak hanya menghemat waktu Anda, tetapi juga membantu Anda mengatasi hambatan yang Anda temui saat bekerja di lingkungan yang tidak mendukung alat grafis. Dalam bab ini, Anda mempelajari dasar-dasar bekerja dengan baris perintah plus spesifikasi PostgreSQL. Anda menemukan aplikasi baris perintah sistem operasi Anda dan mengaturnya agar berfungsi dengan psql. Kemudian Anda menghubungkan psql ke database dan mempelajari cara menjalankan kueri SQL melalui baris perintah. Banyak pengguna komputer

berpengalaman lebih suka menggunakan baris perintah karena kesederhanaan dan kecepatannya setelah mereka terbiasa menggunakannya. Anda mungkin juga.

Di Bab 17, kita akan meninjau tugas pemeliharaan database yang umum termasuk mencadangkan data, mengubah pengaturan server, dan mengelola pertumbuhan database Anda. Tugas-tugas ini akan memberi Anda lebih banyak kontrol atas lingkungan kerja Anda dan membantu Anda mengelola proyek analisis data dengan lebih baik.

Latihan Soal

Untuk memperkuat teknik dalam bab ini, pilih contoh dari bab sebelumnya dan coba kerjakan hanya dengan menggunakan baris perintah. Bab 14 adalah pilihan yang baik karena memberi Anda kesempatan untuk bekerja dengan psql dan pemuat shapefile shp2pgsql. Tetapi pilihlah contoh apa pun yang menurut Anda akan bermanfaat jika ditinjau.

BAB XVII
MEMELIHARA BASIS DATA

Untuk menyelesaikan eksplorasi SQL kami, kami akan melihat tugas dan opsi pemeliharaan basis data utama untuk menyesuaikan PostgreSQL.

Bab ini, Anda akan mempelajari cara melacak dan menghemat ruang di database Anda, cara mengubah pengaturan sistem, dan cara mencadangkan dan memulihkan database. Seberapa sering Anda perlu melakukan tugas ini bergantung pada peran dan minat Anda saat ini. Tetapi jika Anda ingin menjadi administrator basis data atau pengembang backend, topik yang dibahas di sini sangat penting untuk kedua pekerjaan tersebut.

Perlu dicatat bahwa pemeliharaan basis data dan penyetelan kinerja sering kali menjadi pokok bahasan seluruh buku, dan bab ini terutama berfungsi sebagai pengantar untuk beberapa hal penting.Jika Anda ingin mempelajari lebih lanjut, tempat yang baik untuk memulai adalah dengan sumber daya di Lampiran.

Mari kita mulai dengan fitur PostgreSQL VACUUM, yang memungkinkan Anda mengecilkan ukuran tabel dengan menghapus baris yang tidak digunakan.

Memulihkan Ruang yang Tidak Digunakan dengan VACUUM

Untuk mencegah file database tumbuh di luar kendali, Anda dapat menggunakan perintah PostgreSQL VACUUM Dalam “Meningkatkan Performa Saat Memperbarui Tabel Besar”, Anda mengetahui bahwa ukuran tabel PostgreSQL dapat bertambah sebagai hasil dari operasi rutin. Misalnya, saat Anda memperbarui nilai dalam satu baris, database membuat versi baru dari baris tersebut yang menyertakan nilai yang diperbarui, tetapi tidak hapus baris versi lama. (Dokumentasi PostgreSQL mengacu pada baris sisa yang tidak dapat Anda lihat sebagai baris “mati”.)

Demikian pula, ketika Anda menghapus sebuah baris, meskipun baris tersebut tidak lagi terlihat, baris tersebut akan tetap hidup sebagai baris mati dalam tabel. Database menggunakan baris mati untuk menyediakan fitur tertentu di lingkungan di mana banyak transaksi terjadi dan versi lama dari baris mungkin diperlukan oleh transaksi selain yang sekarang.

Menjalankan VACUUM menunjuk ruang yang ditempati oleh baris mati sebagai tersedia untuk database untuk digunakan lagi. Tetapi VACUUM tidak mengembalikan ruang ke disk sistem Anda. Sebaliknya, itu hanya menandai ruang itu sebagai tersedia untuk basis data yang akan digunakan untuk operasi berikutnya.Untuk mengembalikan ruang yang tidak terpakai ke disk Anda, Anda harus menggunakan opsi VACUUM FULL, yang membuat versi baru dari tabel yang tidak menyertakan ruang baris mati yang kosong.

Meskipun Anda dapat menjalankan VACUUM sesuai permintaan, secara default PostgreSQL menjalankan proses latar belakang autovacuum yang memantau database dan menjalankan

VACUUM sesuai kebutuhan. Nanti di bab ini saya akan menunjukkan cara memantau vakum otomatis serta menjalankan perintah VACUUM secara manual.

Tapi pertama-tama, mari kita lihat bagaimana tabel tumbuh sebagai hasil dari pembaruan dan bagaimana Anda dapat melacak pertumbuhan ini.

Ukuran Meja Pelacakan

Kami akan membuat tabel uji kecil dan memantau pertumbuhannya saat kami mengisinya dengan data dan melakukan pembaruan. Kode untuk latihan ini, seperti semua sumber daya untuk buku ini, tersedia di https://www.nostarch. Com / praktisSQL /.

Membuat Tabel dan Memeriksa Ukurannya

Kueri dibawah ini untuk membuat tabel vacuum_test dengan satu kolom untuk menampung bilangan bulat. Jalankan kodenya, lalu kita akan mengukur ukuran tabelnya.

```
CREATE TABLE vacuum_test (
      integer_column integer );
```

Sebelum kita mengisi tabel dengan data uji, mari kita periksa berapa banyak ruang yang digunakan pada disk untuk menetapkan titik referensi. Kita dapat melakukannya dengan dua cara: memeriksa properti tabel melalui antarmuka pgAdmin, atau menjalankan kueri menggunakan Fungsi administratif PostgreSQL Di pgAdmin, klik sekali pada tabel untuk menyorotnya, lalu klik tab Statistik Ukuran tabel adalah salah satu dari sekitar dua lusin indikator dalam daftar.

Saya akan fokus menjalankan kueri di sini karena mengetahuinya akan membantu jika karena alasan tertentu pgAdmin tidak tersedia atau Anda menggunakan GUI lain. Misalnya, sintaks dibawah ini menunjukkan cara memeriksa ukuran tabel vacuum_test menggunakan fungsi PostgreSQL:

```
SELECT pg_size_pretty(
            Pg_total_relation_size('vacuum_test')
);
```

Fungsi terluar, pg_size_pretty(), mengonversi byte ke format yang lebih mudah dipahami dalam kilobyte, megabyte, atau gigabyte. Dibungkus di dalam pg_size_pretty() adalah fungsi pg_total_relation_size() , yang melaporkan berapa banyak byte sebuah tabel, indeksnya, dan data terkompresi offline menggunakan disk. Karena tabel kosong pada saat ini, menjalankan kode di pgAdmin akan mengembalikan nilai 0 byte, seperti ini:

```
pg_size_pretty
--------------
0 bytes
```

Luncurkan psql seperti yang Anda pelajari di Bab 16. Kemudian, pada prompt, masukkan perintah \ dt + vacuum_test, yang akan menampilkan informasi berikut termasuk ukuran tabel:

```
                    List of relations
 Schema    Name        Type  Owner       Size        Description
 ------    ----------- ----- ---------   --------    -----------
 public    vacuum_test table postgress   0 bytes
```

Sekali lagi, ukuran tabel vacuum_test saat ini harus menampilkan 0 byte.

Memeriksa Ukuran Tabel Setelah Menambahkan Data Baru

Mari kita tambahkan beberapa data ke tabel dan kemudian periksa ukurannya lagi Kita akan menggunakan fungsi generate_series () yang diperkenalkan di Bab 11 untuk mengisi integer_column tabel dengan 500.000 baris Jalankan kode dibawah ini untuk melakukan ini:

```
INSERT INTO vacuum_test
SELECT * FROM generate_series(1,500000);
```

Pernyataan INSERT INTO standar ini menambahkan hasil generate_series(), yang merupakan rangkaian nilai dari 1 hingga 500.000, sebagai baris ke tabel. Setelah kueri selesai, jalankan kembali kueri pada pembahasan sebelumnya untuk memeriksa ukuran tabel Anda akan melihat output berikut:

```
pg_size_pretty
--------------
17 MB
```

Kueri melaporkan bahwa tabel vacuum_test, sekarang dengan satu kolom 500.000 bilangan bulat, menggunakan ruang disk sebesar 17 MB.

Memeriksa Ukuran Tabel Setelah Pembaruan

Sekarang, mari perbarui data untuk melihat bagaimana hal itu memengaruhi ukuran tabel. Kita akan menggunakan kode di dibawah ini untuk memperbarui setiap baris dalam tabel vacuum_test dengan menambahkan 1 ke nilai integer_column, mengganti nilai yang ada dengan angka yang satu lebih besar.

```
UPDATE vacuum_test
SET integer_column = integer_column + 1;
```

Jalankan kode, lalu uji kembali ukuran tabel.

```
pg_size_pretty
--------------
35 MB
```

Ukuran tabel menjadi dua kali lipat dari 17MB menjadi 35MB! Peningkatannya tampaknya berlebihan, karena UPDATE hanya mengganti angka yang ada dengan nilai ukuran yang sama. Tetapi seperti yang Anda duga, alasan peningkatan ukuran tabel ini adalah untuk setiap update value , PostgreSQL membuat baris baru, dan baris lama (baris "mati") tetap ada di tabel. Jadi meskipun Anda hanya melihat 500.000 baris, tabel memiliki dua kali lipat jumlah baris tersebut. Ini dapat mengejutkan pemilik basis data yang tidak memantau ruang disk karena drive akhirnya terisi dan mengarah ke Kita akan melihat bagaimana menggunakan VACUUM dan VACUUM FULL mempengaruhi ukuran tabel pada disk. Tapi pertama-tama, mari kita tinjau

proses yang berjalan VACUUM secara otomatis serta cara mengecek statistik yang berhubungan dengan table vacuums.

Memantau Proses autovacuum

Proses autovacuum PostgreSQL memantau database dan meluncurkan VACUUM secara otomatis ketika mendeteksi sejumlah besar baris mati dalam sebuah tabel. Meskipun autovacuum diaktifkan secara default, Anda dapat mengaktifkan atau menonaktifkannya dan mengonfigurasinya menggunakan pengaturan yang akan saya bahas di “Mengubah Pengaturan Server ” di pembahasan sebelumnya. Karena autovacuum berjalan di latar belakang, Anda tidak akan melihat indikasi yang langsung terlihat bahwa itu berfungsi, tetapi Anda dapat memeriksa aktivitasnya dengan menjalankan kueri.

PostgreSQL memiliki pengumpul statistik sendiri yang melacak aktivitas dan penggunaan basis data. Anda dapat melihat statistik dengan menanyakan salah satu dari beberapa tampilan yang disediakan sistem. (Lihat daftar lengkap tampilan untuk memantau status sistem di https: // www. postgresql.org/docs/current/static/monitoring-stats.html). Untuk memeriksa aktivitas autovacuum, kueri tampilan yang disebut pg_stat_all_tables menggunakan kode dibawah ini:

```
SELECT relname,
       last_vacuum,
       last_autivacuum,
       vacuum_count,
       autovacuum_count
FROM pg_stat_all_table
WHERE relname = ‘vacuum_test’;
```

Tampilan pg_stat_all_tables menunjukkan relname , yang merupakan nama tabel, ditambah statistik yang terkait dengan pemindaian indeks, baris yang dimasukkan dan dihapus, dan data lainnya. Untuk kueri ini, kami tertarik pada last_vacuum dan last_autovacuum , yang berisi yang terakhir Kami juga meminta vacuum_count dan autovacuum_count , yang menunjukkan berapa kali vakum dijalankan secara manual dan otomatis.

Secara default, autovacuum memeriksa tabel setiap menit. Jadi, jika satu menit telah berlalu sejak Anda terakhir memperbarui vacuum_test, Anda akan melihat detail aktivitas vakum saat menjalankan kueri diatas. Inilah yang ditampilkan sistem saya (perhatikan bahwa saya' telah menghapus detik dari waktu untuk menghemat ruang di sini):

```
last_vacuum      last_autovacuum   vacuum_count      autovacuum_count
-----------      ---------------   ------------      ----------------
vacuum_test      2018-02-08 13:28  0                 1
```

Tabel menunjukkan tanggal dan waktu autovacuum terakhir, dan kolom autovacuum_count menunjukkan satu kejadian. Hasil ini menunjukkan bahwa autovacuum mengeksekusi perintah VACUUM pada tabel satu kali. Namun, karena kami belum menyedot secara manual, kolom last_vacuum kosong dan vakum_count adalah 0.

PostgreSQL menyimpan informasi ini dan menggunakannya untuk mengeksekusi kueri secara efisien di masa mendatang. Anda dapat menjalankan ANALYZE secara manual jika diperlukan.

Ingat bahwa VACUUM menunjuk baris mati sebagai tersedia untuk database untuk digunakan kembali tetapi tidak mengurangi ukuran tabel pada disk Anda dapat mengkonfirmasi ini dengan menjalankan kembali kode di atas, yang menunjukkan tabel tetap di 35MB bahkan setelah vakum otomatis.

Menjalankan VAKUM Secara Manual

Tergantung pada server yang Anda gunakan, Anda dapat mematikan autovacuum. Jika autovacuum dimatikan atau jika Anda hanya ingin menjalankan VACUUM secara manual, Anda dapat melakukannya dengan menggunakan satu baris kode, seperti yang ditunjukkan kueri dibawah ini:

```
VACUUM vacuum_test;
```

Setelah Anda menjalankan perintah ini, itu akan mengembalikan pesan VACUUM dari server Sekarang ketika Anda mengambil statistik lagi menggunakan kueri di pembahasan diatas, Anda akan melihat bahwa kolom last_vacuum mencerminkan tanggal dan waktu vakum manual yang baru saja Anda jalankan dan jumlah di kolom vacuum_count akan bertambah satu.

Dalam contoh ini, kami mengeksekusi VACUUM pada tabel pengujian kami. Tetapi Anda juga dapat menjalankan VACUUM di seluruh database dengan menghilangkan nama tabel. Selain itu, Anda dapat menambahkan kata kunci VERBOSE untuk memberikan informasi yang lebih detail, seperti jumlah baris yang ditemukan dalam tabel dan jumlah baris yang dihapus, di antara informasi lainnya.

Mengurangi Ukuran Meja dengan VACUUM FULL

Selanjutnya, kita akan menjalankan VACUUM dengan opsi FULL. Tidak seperti VACUUM default, yang hanya menandai ruang yang dipegang oleh baris mati sebagai tersedia untuk penggunaan di masa mendatang, opsi FULL mengembalikan ruang kembali ke disk. Seperti yang disebutkan, VACUUM FULL membuat versi baru Meskipun ini mengosongkan ruang pada disk sistem Anda, ada beberapa peringatan yang perlu diingat. Pertama, VACUUM FULL membutuhkan lebih banyak waktu untuk diselesaikan daripada VACUUM. Kedua, ia harus memiliki akses eksklusif Perintah VACUUM biasa dapat dijalankan saat pembaruan dan operasi lainnya sedang terjadi Ke tabel saat menulis ulang, yang berarti tidak ada yang dapat memperbarui data selama operasi. Untuk melihat cara kerja VACUUM FULL, jalankan perintah sintaks dibawah ini

```
VACUUM FULL vacuum_test;
```

Setelah perintah dijalankan, uji kembali ukuran tabel, seharusnya kembali ke 17MB, yang merupakan ukuran saat pertama kali kita memasukkan data.

Tidak pernah bijaksana atau aman untuk kehabisan ruang disk, jadi memperhatikan ukuran file database Anda serta ruang sistem Anda secara keseluruhan adalah rutinitas yang berharga untuk dibangun.Menggunakan VACUUM untuk mencegah file database tumbuh lebih besar dari yang seharusnya adalah hal yang baik.

Mengubah Pengaturan Server

Anda dapat mengubah lusinan pengaturan untuk server PostgreSQL Anda dengan mengedit nilai di postgresql.conf, salah satu dari beberapa file teks konfigurasi yang mengontrol pengaturan server. File lainnya termasuk pg_hba.conf, yang mengontrol koneksi ke server, dan pg_ident.conf, yang dapat digunakan administrator database untuk memetakan nama pengguna di jaringan ke nama pengguna di PostgreSQL. Lihat dokumentasi PostgreSQL pada file ini untuk detailnya.

Untuk tujuan kami, kami akan menggunakan file postgresql.conf karena berisi pengaturan yang paling kami minati. Sebagian besar nilai dalam file diatur ke default yang tidak perlu Anda sesuaikan, tetapi perlu ditelusuri dalam kasus Mari kita mulai dengan dasar-dasar Anda ingin mengubahnya sesuai dengan kebutuhan Anda.

Menemukan dan Mengedit postgresql.conf

Sebelum Anda dapat mengedit postgresql.conf, Anda harus menemukan lokasinya, yang bervariasi tergantung pada sistem operasi dan metode penginstalan Anda. Anda dapat menjalankan kueri dibawah ini untuk mencari file:

```
SHOW config_file;
```

Ketika saya menjalankan perintah di Mac, itu menunjukkan jalur ke file sebagai:

```
/Users/Anthony/Library/Application Support/Postgres/var-10/postgresql.conf
```

Untuk mengedit postgresql.conf, navigasikan ke direktori yang ditampilkan oleh SHOW config_file; di sistem Anda, dan buka file menggunakan editor teks biasa, bukan editor teks kaya seperti Microsoft Word.

Catatan : Sebaiknya simpan salinan postgresql.conf untuk referensi jika Anda membuat perubahan yang merusak sistem dan Anda perlu kembali ke versi aslinya. Saat Anda membuka file, beberapa baris pertama akan terbaca sebagai berikut:

```
# -----------------------------
# PostgreSQL configuration file
# -----------------------------
#
# This file consists of lines of the form:
#
# name = valve
```

file postgresql.conf diatur menjadi beberapa bagian yang menentukan pengaturan untuk lokasi file, keamanan, pencatatan informasi, dan proses lainnya. Banyak baris dimulai dengan tanda hash (#), yang menunjukkan bahwa baris tersebut dikomentari dan pengaturan yang ditampilkan adalah bawaan aktif.

Misalnya, di bagian file postgresql.conf "Parameter Autovacuum," defaultnya adalah autovacuum diaktifkan. Tanda hash (#) di depan baris berarti bahwa baris tersebut dikomentari dan defaultnya berlaku:

```
#autovacuum = on        # Enable autovacuum subprocess? 'on'
```

Untuk mematikan autovacuum, Anda menghapus tanda hash di awal baris dan mengubah nilainya menjadi off:

```
autovacuum = off        # Enable autovacuum subprocess? 'on'
```

Sintaks diatas menunjukkan beberapa pengaturan lain yang mungkin ingin Anda jelajahi, yang dikutip dari bagian postgresql.conf "Default Koneksi Klien." Gunakan editor teks Anda untuk mencari file untuk pengaturan berikut.

```
datestyle = 'iso, mdy'

timezone = 'US/Eastern'

default_text_search_config = 'pg_catalog.english'
```

Anda dapat menggunakan pengaturan gaya tanggal untuk menentukan bagaimana PostgreSQL menampilkan tanggal dalam hasil kueri. Pengaturan ini menggunakan dua parameter: format output dan urutan bulan, hari, dan tahun. Default untuk format output adalah format ISO (YYYY-MM-DD) yang kami gunakan di seluruh buku ini, yang saya sarankan Anda gunakan untuk portabilitas lintas negara.Namun, Anda juga dapat menggunakan format SQL tradisional (MM / DD / YYYY), format Postgres yang diperluas (Senin) 12 22:30:00 2018 EST), atau format Jerman (DD.MM.YYYY) dengan titik-titik di antara tanggal, bulan, dan tahun. Untuk menentukan format menggunakan parameter kedua, susun m, d, dan y di urutan yang Anda inginkan.

Parameter zona waktu menetapkan zona waktu server (Anda dapat menebaknya). Sintaks pada pembahasan sebelumnya menunjukkan nilai US / Eastern, yang mencerminkan zona waktu pada mesin saya ketika saya menginstal PostgreSQL. Zona waktu Anda harus bervariasi berdasarkan lokasi Anda. Saat menyiapkan PostgreSQL untuk digunakan sebagai backend ke aplikasi database atau pada jaringan, administrator sering menetapkan nilai ini ke UTC dan menggunakannya sebagai standar pada mesin di beberapa lokasi.

Nilai default_text_search_config mengatur bahasa yang digunakan oleh operasi pencarian teks lengkap. Di sini, milik saya diatur ke bahasa Inggris. Tergantung pada kebutuhan Anda, Anda dapat mengatur ini ke bahasa Spanyol, Jerman, Rusia, atau bahasa lain pilihan Anda.

Tiga contoh ini hanya mewakili segelintir pengaturan yang tersedia untuk penyesuaian. Kecuali jika Anda terjebak dalam penyetelan sistem, Anda mungkin tidak perlu mengubah banyak hal lainnya. Juga, berhati-hatilah saat mengubah pengaturan pada server jaringan yang digunakan oleh banyak orang atau aplikasi ; perubahan dapat memiliki konsekuensi yang tidak diinginkan, jadi sebaiknya komunikasikan terlebih dahulu dengan rekan kerja.

Setelah Anda membuat perubahan pada postgresql.conf, Anda harus menyimpan file dan kemudian memuat ulang pengaturan menggunakan perintah pg_ctl PostgreSQL untuk menerapkan pengaturan baru. Mari kita lihat bagaimana melakukannya selanjutnya.

Memuat Ulang Pengaturan dengan pg_ctl

Utilitas baris perintah pg_ctl memungkinkan Anda untuk melakukan tindakan pada server PostgreSQL, seperti memulai dan menghentikannya, dan memeriksa statusnya. Di sini, kami akan menggunakan utilitas untuk memuat ulang file pengaturan sehingga perubahan yang kami buat akan diterapkan. Menjalankan perintah memuat ulang semua file pengaturan sekaligus.

Anda harus membuka dan mengonfigurasi prompt baris perintah dengan cara yang sama seperti yang Anda lakukan di Bab 16 ketika Anda mempelajari cara mengatur dan menggunakan psql. Setelah Anda meluncurkan prompt perintah, gunakan salah satu dari perintah berikut untuk memuat ulang:

- Pada Windows, gunakan:

```
pg_ctl reload –D "C:\path\to\data\directory\"
```

- Di macOS atau Linux, gunakan:

```
pg_ctl reload –D '/path/to/data/directory/'
```

Untuk menemukan lokasi direktori data PostgreSQL Anda, jalankan kueri dibawah ini:

```
SHOW data_directory;
```

Anda menjalankan perintah ini pada prompt perintah sistem Anda, bukan di dalam aplikasi psql. Masukkan perintah dan tekan ENTER; itu akan merespons dengan PostgreSQL akan memperingatkan Anda jika itu masalahnya. File pengaturan akan dimuat ulang dan perubahan akan diterapkan. Beberapa pengaturan, seperti alokasi memori, memerlukan restart server.

Mencadangkan dan Memulihkan Basis Data Anda

Saat Anda membersihkan data produsen makanan USDA "kotor" di Bab 9, Anda telah mempelajari cara membuat salinan cadangan tabel. Namun, tergantung pada kebutuhan Anda, Anda mungkin ingin mencadangkan seluruh database Anda secara teratur baik untuk disimpan atau untuk PostgreSQL menawarkan alat baris perintah yang memudahkan operasi pencadangan dan pemulihan. Beberapa bagian berikutnya menunjukkan contoh cara membuat cadangan database atau tabel tunggal, serta cara memulihkannya.

Menggunakan pg_dump untuk Mencadangkan Database atau Tabel

Alat baris perintah PostgreSQL pg_dump membuat file output yang berisi semua data dari database Anda, perintah SQL untuk membuat ulang tabel, dan objek database lainnya, serta memuat data ke dalam tabel. Anda juga dapat menggunakan pg_dump untuk menyimpan Hanya tabel yang dipilih dalam database Anda. Secara default, pg_dump mengeluarkan file teks biasa; Saya akan membahas format terkompresi khusus terlebih dahulu dan kemudian membahas opsi lainnya.

Untuk mencadangkan database analisis yang telah kami gunakan untuk latihan kami, jalankan perintah di kueri dibawah ini di command prompt sistem Anda (bukan di psql):

```
pg_dump –d analysis –U user_name –Fc > analysis_backup.sql
```

Di sini, kita memulai perintah dengan pg_dump, argumen -d, dan nama database yang akan dicadangkan, diikuti dengan argumen -U dan nama pengguna Anda. Selanjutnya, kita menggunakan argumen -Fc untuk menentukan bahwa kita ingin menghasilkan Cadangan ini dalam format terkompresi PostgreSQL khusus. Kemudian kami menempatkan simbol lebih besar dari (>) untuk mengarahkan output pg_dump ke file teks bernama analysis_backup.sql. Untuk menempatkan file di direktori selain yang diminta terminal Anda saat ini terbuka, Anda dapat menentukan jalur direktori lengkap sebelum nama file.

Saat Anda menjalankan perintah dengan menekan ENTER, tergantung pada penginstalan Anda, Anda mungkin melihat prompt kata sandi. Isi kata sandi itu, jika diminta. Kemudian, tergantung pada ukuran basis data Anda, perintah bisa memakan waktu beberapa menit untuk diselesaikan. operasi tidak menampilkan pesan apa pun ke layar saat sedang bekerja, tetapi ketika selesai, itu akan mengembalikan Anda ke prompt perintah baru dan Anda akan melihat file bernama analysis_backup.sql di direktori Anda saat ini.

Untuk membatasi pencadangan ke satu atau beberapa tabel yang cocok dengan nama tertentu, gunakan argumen -t diikuti dengan nama tabel dalam tanda kutip tunggal.Misalnya, untuk mencadangkan hanya tabel train_rides, gunakan perintah berikut:

```
pg_dump –t ‘train_rides’ –d analysis –U user_name –Fc > train_backup.sql
```

Sekarang mari kita lihat bagaimana memulihkan cadangan, dan kemudian kita akan menjelajahi opsi pg_dump tambahan.

Memulihkan Cadangan Basis Data dengan pg_restore

Setelah Anda mencadangkan basis data menggunakan pg_dump, sangat mudah untuk memulihkannya menggunakan utilitas pg_restore. Anda mungkin perlu memulihkan basis data saat memigrasikan data ke server baru atau saat memutakhirkan ke versi PostgreSQL yang baru. Untuk memulihkan analisis database (dengan asumsi Anda berada di server di mana analisis tidak ada), jalankan perintah dibawah ini di command prompt:

```
pg_restore –C –d postgress –U user_name analysis_backup.sql
```

Setelah pg_restore, Anda menambahkan argumen -C, yang memberi tahu utilitas untuk membuat database analisis di server. (Ini mendapatkan nama database dari file cadangan.) Kemudian, seperti yang Anda lihat sebelumnya, argumen -d menentukan nama dari Tekan ENTER dan pemulihan akan dimulai.Setelah selesai, Anda seharusnya dapat melihat database yang dipulihkan melalui psql atau di pgAdmin.

Opsi Pencadangan dan Pemulihan Tambahan

Anda dapat mengonfigurasi pg_dump dengan beberapa opsi untuk menyertakan atau mengecualikan objek database tertentu, seperti tabel yang cocok dengan pola nama, atau untuk menentukan format output.

Juga, ketika kami mencadangkan database analisis di “Menggunakan pg_dump untuk Mencadangkan Database atau Tabel” di halaman 321, kami menetapkan opsi -Fc dengan pg_dump untuk menghasilkan format terkompresi PostgreSQL kustom. Utilitas mendukung opsi format tambahan, termasuk format biasa teks. Untuk detailnya, periksa dokumentasi pg_dump lengkap di https://www.postgresql.org/docs/current/static/app-pgdump.html. Untuk opsi pemulihan yang sesuai, periksa dokumentasi pg_restore di https://www.postgresql .org / docs / current / static / app-pgrestore.html.

Dalam bab ini, Anda telah mempelajari cara melacak dan menghemat ruang di basis data menggunakan fitur VACUUM di PostgreSQL. Anda juga mempelajari cara mengubah pengaturan sistem serta mencadangkan dan memulihkan basis data menggunakan alat baris perintah lainnya. Anda mungkin tidak memerlukan untuk melakukan tugas-tugas ini setiap hari, tetapi trik pemeliharaan yang Anda pelajari di sini dapat membantu meningkatkan kinerja basis data Anda Perhatikan bahwa ini bukan ikhtisar topik yang komprehensif; lihat Lampiran untuk sumber daya lebih lanjut tentang pemeliharaan basis data.

Di bab berikutnya dan terakhir dari buku ini, saya akan membagikan panduan untuk mengidentifikasi tren tersembunyi dan menceritakan kisah yang efektif menggunakan data Anda.

Latihan Soal

Dengan menggunakan teknik yang Anda pelajari di bab ini, buat cadangan dan pulihkan database gis_analysis yang Anda buat di Bab 14. Setelah Anda mencadangkan database lengkap, Anda harus menghapus yang asli untuk dapat memulihkannya. Anda juga dapat mencoba mencadangkan up dan memulihkan tabel individu.

Selain itu, gunakan editor teks untuk menjelajahi file cadangan yang dibuat oleh pg_dump. Periksa bagaimana ia mengatur pernyataan untuk membuat objek dan menyisipkan data.

DAFTAR PUSTAKA

Anthony Molinaro (2005), SQL Cookbook: Query Solutions and Techniques for Database Developers (1st Edition, O'Reilly Media)

Bill Karwin (2010), SQL Antipatterns: Avoiding the Pitfalls of Database Programming (1st Edition, Pragmatic Bookshelf)

David Chappell and J Harvey Trimble Jr. (2001): A Visual Introduction to SQL (2nd Edition, Wiley).

Gary W Hansen and James V Hansen (1995), Database Management and Design (2nd Edition, Prentice Hall,)

Judith S Bowman, Sandra L Emerson, Marcy Darnovsky, (2001) The Practical SQL Handbook: Using SQL Variants (4th Edition, Addison-Wesley Professional)

John L Viescas and Michael J Hernandez, (2014), SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL (3rd Edition, Addison-Wesley Professional).

Rick F van der Lans (2006), Introduction to SQL: Mastering the Relational Database Language (4th Edition, Addison-WesleyProfessional)

Stephane Faroult and Peter Robson (2006), The Art of SQL (1st Edition, O'Reilly Media)

Relational Database Management Systems

- **PostgreSQL** https://www.postgresql.org/
- **SQLite** https://sqlite.org/
- **Firebird** http://firebirdsql.org/
- **MariaDB** https://mariadb.org/
- **HyperSQL** http://hsqldb.org/
- **H2** http://www.h2database.com

SQL Development Environments

- **DBeaver** https://dbeaver.jkiss.org/
- **DB Browser for SQLite** http://sqlitebrowser.org/

PostgreSQL Utilities, Tools, and Extensions

- **Devart Excel Add-In for PostgreSQL** An add-in that lets you load and edit data from PostgreSQL directly in Excel workbooks (see *https://www.devart.com/excel-addins/postgresql.html*).
- **MADlib** A machine learning and analytics library for large data sets (see *http://madlib.apache.org/).*
- **pgAgent** A job manager that lets you run queries at scheduled times, among other tasks (see *https://www.pgadmin.org/docs/pgadmin4/dev/ pgagent.html*).

SQL

Structured Query Language

Dr. Joseph Teguh Santoso, S.Kom, M.Kom

BIODATA PENULIS

Dr. Joseph Teguh Santoso, S.Kom, M.Kom adalah Rektor dari Universitas Sains & Teknologi Komputer (Universitas STEKOM) Semarang yang memiliki banyak pengalaman praktis dalam bidang *e-commerce*sejak Tahun 2002. Beliau mempunyai 3 (tiga) toko *Official Online Store* di China untuk merek Sepeda Raleigh, dengan omzet tahunan pada Tahun 2019 mencapai lebih dari Rp. 35 Milyar rupiah dan terus meningkat. Dr. Joseph T.S memiliki lisensi tunggal sepeda merek “Raleigh” untuk penjualan *Online* di seluruh China. Di samping itu beliau juga memiliki pabrik sepeda dan sepeda listrik merek “Fengjiu”, yaitu Pabrik Sepeda Listrik yang masih tergolong kecil di China. Pengalaman beliau malang melintang di dunia *online store* di China seperti Alibaba, Tmall, Taobao, JD, Aliexpress sangat membantu mahasiswa untuk memiliki pengalaman teknis dan praktis untuk membuka toko *online* bersama beliau.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK
JL. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-6141-75-5 (PDF)

9 786236 141755