PANDUAN DEFINITIF UNTUK POWER QUERY (M)

Dr. Phil. Dony Novaliendry, S. Kom., M. Kom

Buku ini merupakan kelanjutan dari seri Power Query yang mendalami kemampuan lanjutan dalam mengelola dan mentransformasi data menggunakan bahasa M. Disusun secara sistematis dan praktis, buku ini membahas cara membuat parameter dan custom function, serta penerapannya dalam pembuatan solusi dinamis.

Anda juga akan mempelajari teknik lengkap dalam menangani tanggal, waktu, durasi, serta manipulasi data menggunakan fungsi pembanding, pengganti, penggabung, dan pemisah—semuanya disertai contoh kasus yang aplikatif.Bagian penting lainnya adalah pembahasan mendalam tentang penanganan kesalahan (error handling) dan debugging, yang akan membantu Anda membangun transformasi data yang kokoh dan bebas error.

Ditujukan bagi pengguna Excel dan Power BI tingkat menengah hingga mahir, buku ini memperkuat pemahaman teknis dengan tes formatif, daftar fungsi, dan glosarium lengkap—menjadikannya referensi ideal untuk praktisi data yang ingin menguasai Power Query secara profesional.

PENERBITAN & PERCETAKAN UNP PRESS
Jln. Prof. Dr. Hamka Air Tawar Padang
Sumatera Barat

PANDUAN DEFINITIF UNTUK POWER QUERY (M)
Dr. Phil. Dony Novaliendry, S. Kom., M. Kom

PANDUAN DEFINITIF UNTUK POWER QUERY (M)

JILID 3

Dr. Phil. Dony Novaliendry, S. Kom., M. Kom

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 3

Dr. Phil. Dony Novaliendry, S. Kom., M. Kom

DUMMY
Penerbitan & Percetakan
UNP PRESS

UNDANG-UNDANG REPUBLIK INDONESIA
NO 19 TAHUN 2002
TENTANG HAK CIPTA
PASAL 72
KETENTUAN PIDANA SANGSI PELANGGARAN

1. Barang siapa dengan sengaja dan tanpa hak mengumumkan atau memperbanyak suatu Ciptaan atau memberi izin untuk itu, dipidana dengan pidana penjara paling singkat 1 (satu) bulan dan denda paling sedikit Rp 1.000.000, 00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan denda paling banyak Rp 5.000.000.000, 00 (lima milyar rupiah)
2. Barang siapa dengan sengaja menyerahkan, menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu Ciptaan atau barang hasil pelanggaran Hak Cipta atau Hak Terkait sebagaimana dimaksud dalam ayat (1), dipidana dengan pidana penjara paling lama 5 (lima) tahun dan denda paling banyak Rp 500.000.000, 00 (lima ratus juta rupiah).

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 3

Dr. Phil. Dony Novaliendry, S. Kom., M. Kom

DUMMY
Penerbitan & Percetakan
UNP PRESS

UNP PRESS

2025

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 3

editor, Tim editor UNP Press
Penerbit UNP Press, Padang, 2025
1 (satu) jilid; 17.6 x 25 cm (B5)
Jumlah Halaman xi + 257 Halaman Buku

ISBN :

DUMMY
Penerbitan & Percetakan
UNP PRESS

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 3

Penyusun: Dr. Phil. Dony Novaliendry, S. Kom., M. Kom
Editor Substansi: Fadhillah Majid Saragih, S.Pd., M.Pd.T
Editor Bahasa: Nama Editor Bahasa
Desain Sampul & Layout: Wulan Ainiyyah Puteri & Fauzziyah Irwani Putri

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Puji syukur kehadirat Allah SWT atas berkat limpahan rahmat dan karunia-Nya sehingga Buku Panduan Definitif untuk Power Query (M) dapat di buat. Buku Panduan Definitif untuk Power Query (M) adalah buku ajar yang membahas tentang Menguasai Transformasi Data Kompleks dengan Power Query yang bisa di manfaatkan oleh peserta didik khususnya dan bagi semua pihak dari segala lapisan yang membutuhkan sebagai referensi untuk belajar mengunakan Buku ajar Panduan Definitif untuk Power Query (M).

Kami mengucapkan terima kasih kepada semua pihak yang telah membantu dalam proses penyelesain buku ajar ini.

Kami menyadari masih terdapat banyak kekurangan dalam buku ajar ini untuk itu kritik dan saran yang membangun demi penyempurnaan buku ajar ini sangat diharapkan. Dan semoga buku ini dapat memberikan maanfaat bagi para peserta didik khususnya dan bagi semua pihak dari segala lapisan yang membutuhkan.

Padang, Juli 2025

Penulis

DAFTAR ISI

DUMMY
Penerbitan & Percetakan
UNP PRESS

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DUMMY
Penerbitan & Percetakan
UNP PRESS

PENDAHULUAN

Dalam era digital yang semakin berkembang, data telah menjadi salah satu aset terpenting bagi organisasi dan individu. Kemampuan untuk mengolah, menganalisis, dan memvisualisasikan data dengan efektif menjadi kunci untuk mengambil keputusan yang tepat dan strategis. Salah satu alat yang sangat berguna dalam proses ini adalah Power Query, sebuah fitur yang tersedia dalam Microsoft Excel dan Power BI. Power Query memungkinkan pengguna untuk mengimpor, membersihkan, dan mengubah data dari berbagai sumber dengan cara yang intuitif dan efisien.

Buku ajar ini hadir sebagai panduan definitif untuk memahami dan memanfaatkan Power Query secara maksimal. Dengan pendekatan yang sistematis, buku ini dirancang untuk membantu pembaca dari berbagai latar belakang, baik pemula maupun yang sudah berpengalaman, dalam menguasai bahasa pemrograman M yang menjadi dasar dari Power Query. Melalui penjelasan yang jelas dan contoh-contoh praktis, pembaca akan diajak untuk menjelajahi berbagai fitur dan kemampuan yang ditawarkan oleh Power Query.

Dalam setiap bab, pembaca akan menemukan langkah-langkah yang terperinci untuk melakukan berbagai tugas, mulai dari pengambilan data hingga transformasi yang kompleks. Buku ini juga akan membahas berbagai teknik dan trik yang dapat meningkatkan efisiensi kerja, serta cara mengatasi tantangan yang sering dihadapi saat bekerja dengan data. Dengan demikian, pembaca tidak hanya akan belajar cara menggunakan Power Query, tetapi juga memahami prinsip-prinsip dasar yang mendasari pengolahan data.

Akhirnya, kita berharap buku ini dapat menjadi sumber referensi yang berguna dan inspiratif bagi siapa saja yang ingin meningkatkan keterampilan analisis data mereka. Dengan menguasai Power Query, pembaca akan memiliki alat yang kuat untuk mengubah data menjadi informasi yang berharga, mendukung pengambilan keputusan yang lebih baik, dan pada akhirnya, mencapai tujuan yang diinginkan dalam

dunia yang semakin didorong oleh data. Selamat membaca dan selamat berpetualang dalam dunia Power Query!

A. Unduh File Kode Contoh

Kumpulan kode untuk buku ini dihosting di GitHub di https://github.com/PacktPublishing/The-Definitive-Guide-to-Power-Query-M-/. Kita juga memiliki kumpulan kode lain dari katalog buku dan video kita yang lengkap yang tersedia di https://github.com/PacktPublishing/. Lihatlah!

B. Unduh Gambar Berwarna

Kita juga menyediakan berkas PDF yang berisi gambar berwarna dari cuplikan layar/diagram yang digunakan dalam buku ini. Anda dapat mengunduhnya di sini: https://packt.link/gbp/9781835089729.

C. Konvensi yang Digunakan

Ada sejumlah konvensi teks yang digunakan di seluruh buku ini. CodeInText: Menunjukkan kata kode dalam teks, nama tabel basis data, nama folder, nama file, ekstensi file, nama jalur, URL tiruan, input pengguna, dan akun Twitter. Misalnya: “Arahkan ke folder /ClientApp/src/app/cities.” Blok kode ditetapkan sebagai berikut:

```
#date(

  year as number, month as number, day as number,

) as date
```

Bila kita ingin menarik perhatian Anda ke bagian tertentu dari blok kode, baris atau item yang relevan disorot:

```
#date(

  year as number, month as number, day as number,

) as date
```

Tebal: Menunjukkan istilah baru, kata penting, atau kata-kata yang Anda lihat di layar. Misalnya, kata-kata dalam menu atau kotak dialog muncul dalam teks seperti ini. Misalnya: “Arahkan ke tab

Beranda pada pita, klik menu tarik-turun di bawah tombol Ubah data, dan pilih Edit parameter.

DUMMY
Penerbitan & Percetakan
UNP PRESS

BAB 1
Parameter dan Fungsi Khusus (Custom Functions)

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Parameters
- Apa itu fungsi khusus?
- Mengubah query menjadi fungsi
- Memanggil fungsi Khusus
- Ekspresi Each
- Menyempurnakan definisi fungsi
- Merujuk pada nama kolom dan bidang
- Men-debug fungsi khusus
- Ruang lingkup fungsi

A. Pendahuluan

Bahasa M adalah bahasa fungsional yang berisi ratusan fungsi yang cocok untuk sejumlah besar tugas. Pada awalnya, pustaka fungsi standar kemungkinan akan memenuhi sebagian besar kebutuhan transformasi Anda. Namun, setelah Anda menghadapi situasi yang lebih menantang dengan bahasa M, Anda akan menemukan bahwa kemampuan menulis fungsi kustom Anda sendiri membuka kemungkinan baru; hal itu dapat sangat menyederhanakan proses transformasi data dan membuatnya sehingga Anda dapat dengan mudah mengulang logika yang rumit. Hal ini terutama berlaku ketika Anda memanfaatkan fitur seperti membuat fungsi, yang memungkinkan Anda mengubah kueri yang ada menjadi fungsi yang dinamis dan dapat digunakan kembali. Manfaat terbesar di sini adalah Anda dapat mengatur logika Anda sekali dan menerapkannya dengan mudah di tempat lain. Jika Anda perlu mengubah logika Anda nanti, cukup perbarui fungsi Anda. Perubahan ini akan secara otomatis disebarkan ke semua kueri yang menggunakan fungsi tersebut. Dengan demikian, bab ini membahas hal-hal berikut: Parameter, Apa itu fungsi Khusus?, Mengubah kueri menjadi fungsi, Memanggil fungsi Khusus, Ekspresi Each, Menyempurnakan

definisi fungsi, Merujuk nama kolom dan bidang, Men-debug fungsi Khusus, Cakupan fungsi.

1. **Kasus Pemantik Berfikir Kritis: Otomatisasi Konversi Nilai Penjualan**

   Seorang analis data di sebuah toko online ingin mengonversi nilai penjualan dari mata uang Rupiah ke beberapa mata uang asing seperti Dolar AS dan Euro. Data penjualan tersimpan dalam Excel, dan analis ingin menghindari proses manual saat nilai tukar berubah setiap hari.

   Untuk mengatasi ini, ia menggunakan Power Query dan berencana membuat fungsi khusus yang bisa mengonversi nilai Rupiah ke mata uang asing berdasarkan parameter nilai tukar yang bisa diganti-ganti.

2. **Pertanyaan Pemantik**

   a. Apa manfaat menggunakan parameter dibandingkan memasukan nilai tukar secara manual dalam setiap query?
   b. Bagaimana Anda akan membuat fungsi khusus di Power Query untuk mengonversi nilai penjualan ke mata uang asing?
   c. Apa yang akan Anda lakukan agar fungsi konversi ini bisa digunakan kembali untuk data lain yang berasal dari file berbeda?
   d. Bayangkan nilai tukar berubah setiap hari, bagaimana strategi terbaik agar sistem konversi tetap akurat dan efisien tanpa mengubah banyak bagian query?
   e. Jika ada kesalahan dalam hasil konversi, bagaimana Anda akan melacak dan memperbaiki kesalahan tersebut dalam fungsi khusus?
   f. Bagaimana Anda memastikan bahwa fungsi khusus tetap bekerja meskipun struktur tabel penjualan berubah (misalnya, kolom ditambahkan atau dihapus)?
   g. Dalam skenario nyata, bagaimana cara Anda mengintegrasikan parameter nilai tukar yang bersumber dari situs atau file yang diperbarui otomatis?

h. Apakah memungkinkan membuat fungsi konversi multi-mata uang dalam satu fungsi khusus? Jelaskan bagaimana logika fungsi tersebut bisa diatur agar fleksibel untuk berbagai jenis mata uang.

B. Parameter

Parameter dalam Power Query memainkan peran penting dalam membuat kueri menjadi dinamis. Anggaplah parameter sebagai variabel yang dapat Anda sesuaikan untuk mengubah perilaku kueri Anda. Saat Anda mempelajari bab ini, Anda akan menemukan bagaimana parameter menjadi blok penyusun yang berguna untuk menyimpan dan mengelola nilai. Kita akan membahas lebih dalam tentang apa itu parameter, bagaimana Anda dapat membuatnya, dan melihat di mana dan bagaimana menggunakannya.

1. Memahami Parameter

Parameter adalah placeholder yang memudahkan Anda untuk mengadaptasi dan mengelola kueri Anda. Parameter memungkinkan input nilai skalar seperti tanggal, angka, atau teks, tanpa perlu mengodekan nilai-nilai ini secara langsung ke dalam kueri. Desain ini memungkinkan Anda untuk membuat modifikasi eksternal pada satu parameter, yang kemudian secara otomatis menyebarkan pembaruan ke beberapa kueri atau langkah dalam kueri.

Kasus penggunaan umum untuk menggunakan parameter menyediakan fleksibilitas dengan memungkinkan pengguna untuk:

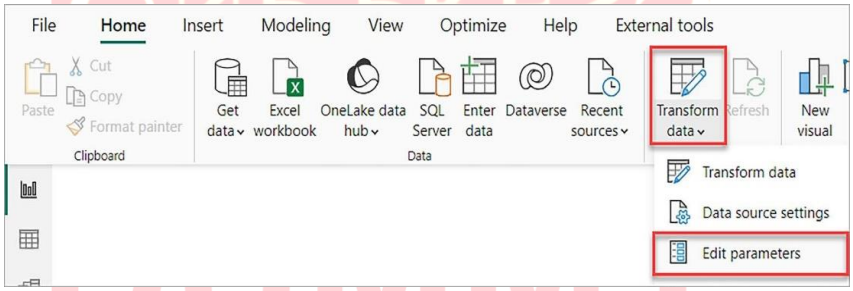
a. Memilih nama server dan basis data. Ini memungkinkan pengguna untuk beralih antara server pengembangan, pengujian, dan produksi.

b. Membatasi jumlah baris yang diimpor ke dalam kumpulan data. Selama pengembangan, pengguna kemudian dapat bekerja dengan kumpulan data kecil. Setelah mengunggah laporan, perubahan parameter yang sederhana kemudian

menyebabkan penyegaran berikutnya untuk mengimpor data yang tersisa.

Dalam skenario ini, parameter menyediakan fleksibilitas dan kemudahan penggunaan, dan membuatnya jauh lebih mudah untuk mengelola kueri Anda. Manfaat lain dari penggunaan parameter adalah Anda dapat mengubah nilainya di luar editor Power Query. Jadi, bagaimana cara kerjanya?

Bagian depan Power BI Desktop menyediakan opsi untuk mengubah parameter tanpa membuka Power Query. Untuk melakukannya, navigasikan ke tab Home pada pita, klik menu tarik-turun di bawah tombol Transform Data, lalu pilih Edit parameters, seperti yang ditunjukkan pada gambar berikut. Ingatlah bahwa Anda perlu menyegarkan kumpulan data untuk melihat perubahan pada parameter Anda berlaku.

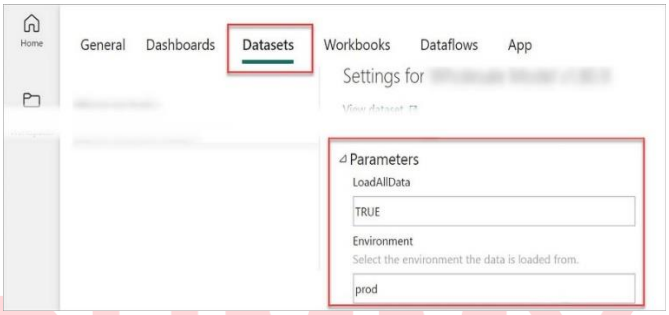


Gambar 1.1 Mengubah Parameter di Power BI Desktop

Setelah menyertakan parameter, pengguna dapat mengunggah kumpulan data mereka ke layanan Power BI. Saat kumpulan data berisi parameter, pengguna dapat mengubah parameter apa pun yang telah mereka tetapkan dengan mudah dari ruang kerja. Untuk melakukannya, pengguna dapat:

a. Buka https://app.Powerbi.com.
b. Navigasi ke **workspace** yang relevan dan pilih opsi *Settings* kumpulan data.
c. Buka bagian *Parameters*.

Layar yang dihasilkan ditampilkan dalam gambar berikut:



Gambar 1.2 Mengubah Parameter dari Pengaturan Dataset di Layanan *Power* BI

Pengguna kemudian dapat mengubah parameter yang ditentukan. Untuk parameter dalam layanan Power BI, hal yang sama berlaku di sini seperti halnya untuk file desktop; perubahan hanya akan terlihat setelah menyegarkan kumpulan data.

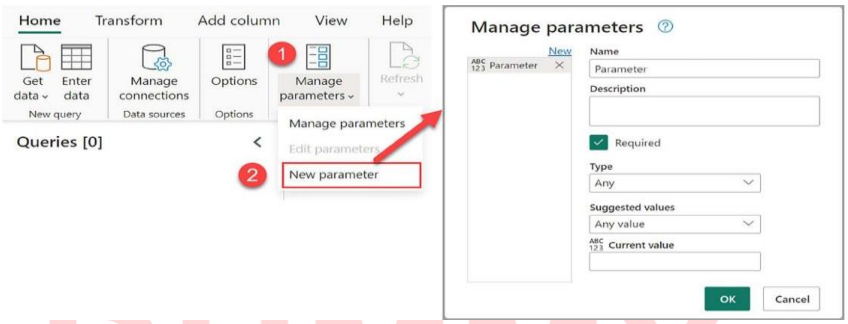
Jauh lebih mudah untuk mengubah parameter di panel kueri daripada menelusuri seluruh langkah kueri untuk menemukan kode yang relevan untuk diubah. Sekarang setelah Anda mengetahui kegunaan parameter, mari kita lihat cara membuatnya.

2. Membuat Parameter

Membuat parameter di Power BI mudah dilakukan. Pengguna dapat membuatnya dengan membuka editor Power Query dan melakukan langkah-langkah berikut:

a. Navigasi ke tab Home.
b. Pilih **Manage Parameters**.
c. Klik **New Parameters**.

Operasi ini membuka menu yang ditunjukkan pada gambar tangkapan layar berikut, tempat mereka dapat membuat atau mengubah properti parameter:



Gambar 1.3 Membuat Parameter menggunakan Kelola Parameter

Panel Manage parameters memungkinkan pengguna untuk memberikan informasi tambahan tentang parameter, tetapi juga menyediakan cara untuk menerapkan jenis nilai tertentu. Berikut ini adalah penjelasan dari masing-masing properti:

a. *Name:* Ini adalah pengenal untuk parameter Anda. Ini adalah nama yang akan Anda gunakan untuk merujuk ke parameter ini di seluruh kueri Anda. Sama seperti penamaan variabel dalam pemrograman, idealnya harus cukup deskriptif untuk memahami tujuannya secara sekilas.
b. *Deskription:* Dengan properti ini, Anda dapat menambahkan catatan singkat atau penjelasan tentang tujuan dan penggunaan parameter. Ini berfungsi sebagai pengingat atau panduan untuk memahami mengapa parameter dibuat dan untuk apa parameter itu dimaksudkan.
c. *Required:* Ini menentukan apakah suatu nilai harus diberikan untuk parameter ini agar kueri dapat berjalan.
d. *Type:* Ini memungkinkan Anda mengatur jenis data yang diharapkan untuk parameter tersebut. Dengan menentukan jenis (seperti teks, angka, atau tanggal), Anda memastikan bahwa parameter hanya menerima nilai dari jenis tertentu tersebut. Ini penting untuk integritas data dan menghindari kesalahan yang tidak terduga selama eksekusi kueri
e. *Suggested values:* Properti ini memberikan panduan tentang nilai yang dapat diterima oleh parameter. Ada tiga opsi:

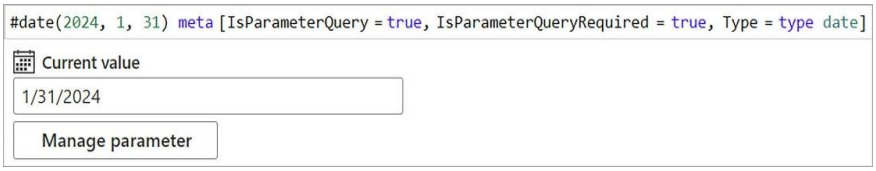
1) *Any Value*: Ini menunjukkan bahwa parameter dapat menerima nilai apa pun dari jenis yang ditentukan tanpa batasan.
2) *List of Value: Ini memungkinkan Anda menentukan serangkaian nilai yang telah ditentukan sebelumnya yang dapat diambil oleh parameter. Misalnya, jika Anda memiliki parameter yang hanya boleh mengambil nilai Low, Medium, atau High, Anda dapat menggunakan ini.*
3) *Query:* Ini memungkinkan nilai berdasarkan hasil kueri. Ini berguna ketika nilai yang diizinkan untuk parameter bersifat dinamis dan dapat berubah, berdasarkan sumber data atau logika lain. Misalnya, Anda dapat memberikan list kategori produk yang unik dari database.

f. *Current Value:* Ini adalah nilai parameter yang saat ini ditetapkan. Ini adalah nilai yang akan digunakan parameter secara default saat kueri berjalan kecuali jika diganti. Jika parameter ditandai sebagai Required tetapi tidak memiliki set Current Value, maka kueri tidak akan berjalan hingga nilai diberikan.

Setelah mengonfigurasi pengaturan, parameter Anda ditambahkan ke panel Queries. Antarmuka pengguna secara otomatis menambahkan tiga bidang metadata:

a. *IsParameterQuery*: Setel ke true untuk menunjukkan bahwa ini adalah parameter.
b. *IsParameterQueryRequired: Menunjukkan apakah parameter diperlukan.*
c. *Type: Menentukan jenis data parameter.*

Misalnya, saat memasukkan nilai tanggal yang diperlukan 1 Januari 2024, Power Query membuat parameter sebagai kueri baru. Saat Anda membuka editor Lanjutan untuk kueri tersebut, kode dan metadata berikut akan ditampilkan:



Gambar 1.4 Metadata saat Menambahkan Parameter

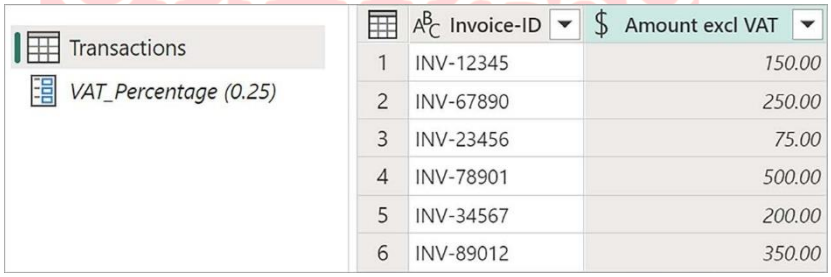
Anda dapat memeriksa kode di atas dalam berkas latihan yang disediakan dalam bab ini, yang dapat Anda temukan di GitHub. Meskipun Anda dapat menggunakan antarmuka pengguna untuk membuat dan mengedit parameter, Anda juga dapat membuat dan mengedit parameter menggunakan Advanced editor. Mengetahui sintaksis akan memungkinkan Anda untuk menambahkan parameter M secara terprogram (misalnya, dengan menggunakan Editor Tabular 3), atau untuk menempelkan kode parameter secara manual ke Advanced editor untuk kueri Anda. Anda dapat membaca lebih lanjut tentang metadata di Bab 3 Jilid 2, Gonceptualizing M.

Sekarang Anda tahu cara membuat parameter, mari kita bahas cara menggunakan parameter dalam kueri Anda.

3. Menggunakan Parameter dalam Kueri Anda

Anda dapat mengintegrasikan parameter ke dalam kode M dengan mereferensikannya, seperti halnya Anda mereferensikan tabel atau langkah tertentu dalam kueri Anda. Mari kita bahas contoh untuk memperjelas hal ini.

Kita akan bekerja dengan kumpulan data sederhana, seperti yang diilustrasikan dalam Gambar 1.5:



Gambar 1.5 Contoh Data Faktur

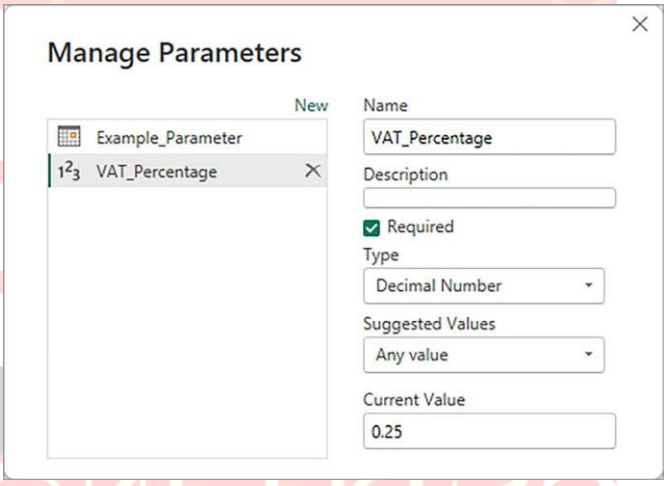
Untuk mengikuti, Anda dapat mengunduh berkas PBIX yang disediakan dari GitHub untuk bab ini.

Set data di atas memiliki kolom berisi faktur dan jumlah yang tidak termasuk PPN. Tujuan kita adalah membuat kolom baru yang berisi jumlah termasuk PPN, tetapi pengguna harus dapat dengan mudah mengubah nilai PPN menggunakan parameter.

Untuk melakukannya, pertama-tama Anda perlu membuat parameter yang diinginkan seperti yang dipelajari di bagian sebelumnya. Itu berarti Anda perlu melakukan hal berikut:

a. Buka menu **Manage Parameters**.
b. Beri nama parameter VAT_Percentage.
c. Tetapkan jenis ke Decimal Number.
d. Tetapkan nilai saat ini sebesar 0,25 (mewakili VAT 25%).

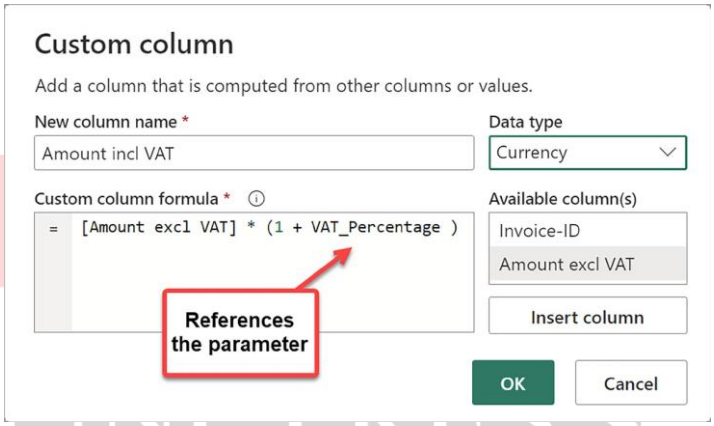
Tampilannya seperti berikut:



Gambar 1.6 Properti untuk Membuat Parameter *VAT_Percentage*

Dengan parameter yang sudah ada, Anda sekarang dapat membuat kolom kustom. Untuk melakukannya, navigasikan ke Add Column di pita, dan pilih Custom Column. Di layar pop-up, beri nama kolom Amount incl VAT.

Jadi, bagaimana Anda memasukkan parameter ke dalam ekspresi Anda? Anda dapat merujuknya seperti Anda merujuk ke tabel atau nama langkah lainnya. Proses ini diilustrasikan di sini:



Gambar 1.7 Mereferensikan Parameter dalam Rumus Kustom

Bila nama parameter Anda menyertakan karakter khusus atau spasi, berhati-hatilah. Dalam kasus tersebut, penting untuk menggunakan notasi tertentu guna memastikan bahasa M mengartikannya dengan benar. Anda akan menggunakan format #"< Your Parameter Name >". Misalnya, jika parameter Anda diberi nama VAT Rate %, Anda akan merujuknya sebagai #" VAT Rate %".

Pro Tip: Untuk menjaga kode tetap bersih, kita sarankan untuk menggunakan nama variabel yang hanya terdiri dari huruf, angka, dan garis bawah, serta memastikan nama variabel tidak dimulai dengan angka. Dengan mengikuti konvensi ini, tidak diperlukan notasi khusus.

Karena Anda sekarang tahu cara mereferensikan parameter, mari kita lihat beberapa contoh praktis yang menggunakan parameter di bagian Menyatukan Semuanya.

4. Menyatukan Semuanya

Parameter dapat berguna dalam berbagai skenario. Bagian ini membahas tiga skenario yang umum terlihat dan bagaimana mereka memanfaatkan parameter.

a. Parameterisasi Informasi Koneksi

Bekerja dengan basis data sering kali berarti harus mengatur antara lingkungan pengembangan, pengujian, dan produksi. Dan mengubah server yang berbeda dan nama masing-masing dapat menjadi tugas yang membosankan. Dalam kasus ini, parameter menawarkan solusi elegan yang menyederhanakan peralihan antara server dan basis data yang berbeda. Mari kita bahas sebuah contoh.

Contoh ini dimaksudkan untuk dibaca bersama. Namun, jika Anda memiliki akses ke basis data, jangan ragu untuk mencoba ini juga.

Koneksi ke database SQL memerlukan spesifikasi server dan database. Kode umum untuk koneksi ke database SQL terlihat seperti berikut:

```
Sql.Database(
  "localhost\sql-dev.database.windows.net",
  "db-staging-dev-westeu-001" )
```

Kita dapat menguraikannya sebagai berikut:

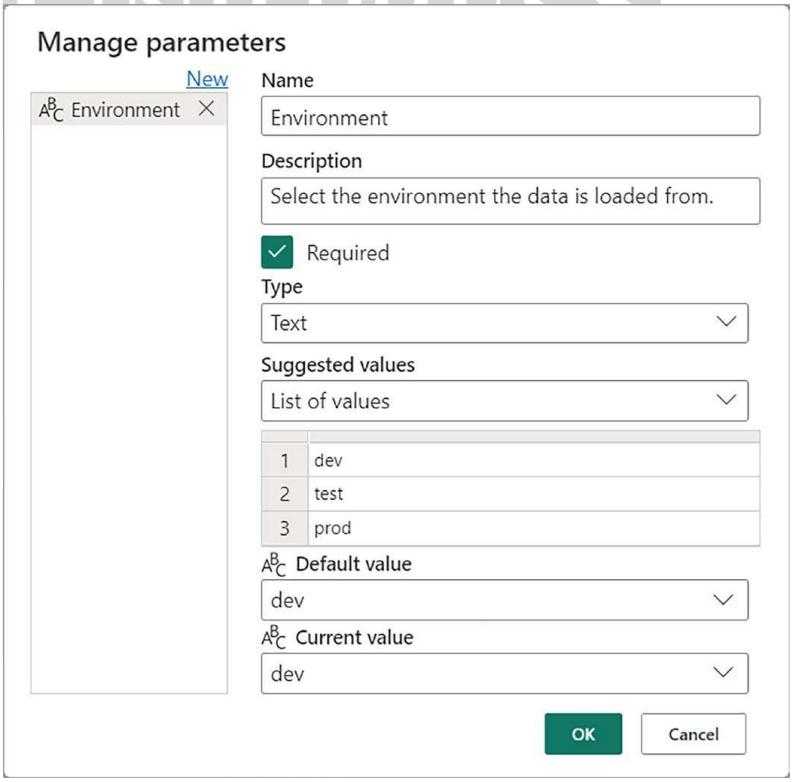
1) *Server*: *localhost\sql-dev.database.windows.net*
2) *Database: db-staging-dev-westeu-001*

Perhatikan teks dev di kedua komponen. Saat menyiapkan database, administrator mencoba menyederhanakan transisi antar lingkungan. Anda hanya perlu mengganti dev dengan test untuk lingkungan pengujian atau prod untuk lingkungan produksi.

Daripada membuat hardcoding lingkungan server, kita dapat memasukkan parameter dalam skenario ini. Cara kerjanya adalah sebagai berikut:

1) Navigasi ke **Manage Parameters** dan buat parameter baru.
2) Beri nama Environment, pastikan Required dicentang, dan tentukan Type-nya sebagai Text. Klik dropdown List of values untuk masukan yang disarankan.
3) Gunakan fitur List of values untuk mengonfigurasi opsi yang dapat dipilih di menu dropdown. Isi dengan dev, test, dan prod.

Setelah mengonfigurasi pengaturan ini, properti parameter akan terlihat sebagai berikut:



Gambar 1.8 Tentukan Parameter Koneksi Database

Ingat kode basis data yang kita lihat sebelumnya? Sekarang kita memiliki parameter untuk menyesuaikan lingkungan basis

data sesuai keinginan, apakah kita memerlukan dev, test, atau prod. Untuk melakukannya, pengguna dapat memilih langkah kueri yang berisi ekspresi koneksi basis data. Bilah rumus kemudian menunjukkan:

```
Sql.Database(
  "localhost\sql-dev.database.windows.net",
  "db-staging-dev-westeu-001" )
```

Untuk menggabungkan parameter, pengguna dapat mengganti dev dengan parameter Lingkungan:

```
Sql.Database(
 "localhost\sql-" & Environment & ".database.windows.net",
 "db-staging-" & Environment & "-westeu-001" )
```

Dalam contoh ini, parameter Environment berfungsi sebagai placeholder yang memungkinkan Anda mengubah string koneksi dengan mudah. Setelah menyimpan perubahan ini dan mengunggah file ke layanan Power BI, Anda memperoleh fleksibilitas untuk mengubah nilai parameter secara langsung dari menu pengaturan dataset. Ini berarti Anda dapat mengubah koneksi database tanpa perlu membuka Power BI Desktop. Setelah Anda melihat skenario umum pertama untuk parameter, mari kita lihat skenario kedua, jalur file dinamis.

b. Jalur File Dinamis

Saat bekerja dengan kueri yang menyertakan jalur file, penting untuk mempertimbangkan bahwa jalur ini mungkin berfungsi dengan baik untuk pembuat asli tetapi belum tentu untuk orang lain. Jika Anda mengantisipasi bahwa file Anda akan digunakan oleh banyak orang, dan berisi jalur file, desain kueri yang cermat menjadi penting. Menyediakan cara mudah bagi pengguna untuk mengubah jalur file sangatlah penting, dan parameter merupakan cara yang bagus untuk mencapai fleksibilitas ini. Mari kita bahas cara menerapkannya. Anda dapat mengikuti dengan mengunduh file Avocado Prices.csv yang disertakan dalam file latihan.

Misalkan Anda membuat materi pelatihan yang menyertakan file Avocado Price.csv yang disertakan dalam buku ini. Agar semuanya tetap teratur, Anda memutuskan untuk menyimpan file ini di drive C, dalam folder bernama data. Untuk menghubungkan ke file ini melalui Power Query, Anda:

1) Pilih **Get Data** dan pilih **Texs/CSV**.
2) Telusuri ke C:\Data dan pilih file CSV.
3) Menu impor CSV akan muncul; tekan OK.

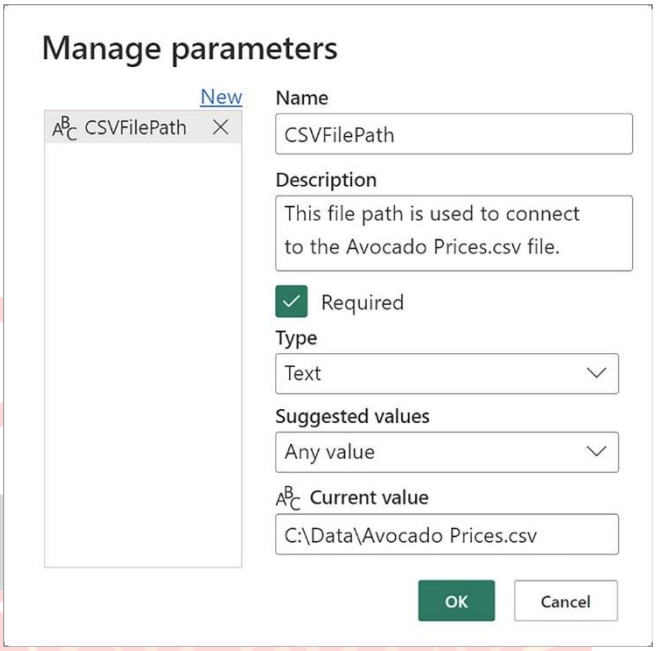
Urutan ini akan menghasilkan kode M berikut:

```
Csv.Document(
  File.Contents("C:\Data\Avocado Prices.csv"),
  [Delimiter=",", Columns=14,
   Encoding=1252, QuoteStyle=QuoteStyle.None] )
```

Perhatikan jalur berkas langsung, C:\Data\Avocado Prices.csv, yang tertanam di dalamnya. Meskipun ini mungkin mudah bagi Anda, mungkin tidak mudah bagi semua orang. Bagaimana jika peserta pelatihan Anda menyimpan berkas mereka di lokasi yang berbeda? Bagaimana jika Anda membuat kueri lain yang juga menggunakan jalur berkas ini sebagai referensi?

Hardcoding jalur file kemudian dengan cepat menjadi kendala, yang membutuhkan pembaruan manual yang membosankan oleh setiap orang. Untungnya, Anda dapat menggunakan parameter untuk memperkenalkan fleksibilitas di jalur file Anda. Berikut cara menerapkan solusi ini.

Mulailah dengan membuat parameter text type. Mari kita beri nama CSVFilePath dan tetapkan bidang Current value ke C:\Data\Avocado Prices.csv. Properti parameter ini ditampilkan dalam tangkapan layar berikut:



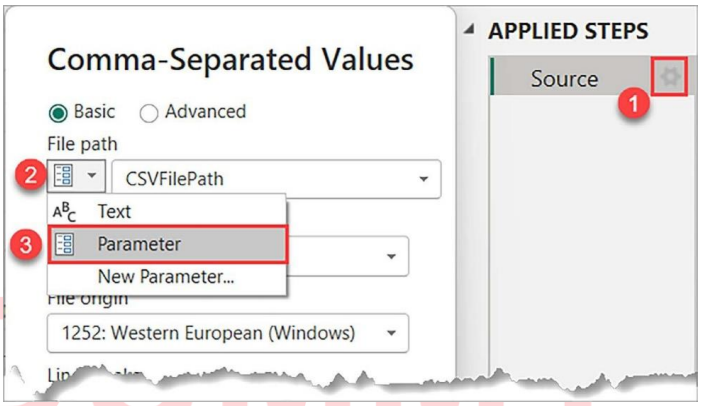
Gambar 1.9 Buat Parameter bernama CSVFilePath

Setelah parameter tersebut tersedia, sekarang saatnya untuk memasukkannya ke dalam kode. Saat Anda awalnya menautkan CSV, langkah berjudul Source dibentuk di panel Applied Steps, yang terletak di sisi kanan editor Query.

Berikut ini adalah metode mudah untuk mengintegrasikan parameter Anda:

1) Klik ikon roda gigi yang berdekatan dengan langkah **Source**.
2) Untuk File Path, buka menu drop down dan pilih Parameter.
3) Untuk menyelesaikan, pilih parameter CSVFilePath.

Langkah-langkah untuk melakukan ini ditunjukkan pada gambar berikut:



Gambar 1.10 Anda dapat Menggunakan Parameter sebagai Jalur File

Dengan modifikasi ini, kode M Anda sekarang terlihat seperti:

```
Csv.Document(
  File.Contents( CSVFilePath ),
  [Delimiter=",", Columns=14,
   Encoding=1252, QuoteStyle=QuoteStyle.None]
)
```

Anda akan melihat bahwa di tempat yang dulunya merupakan jalur file yang dikodekan secara keras, parameter CSVFilePath sekarang berada. Perubahan ini memberikan fleksibilitas pada kueri Anda.

Dengan hanya menyesuaikan nilai parameter, kueri Anda sekarang menjadi fleksibel tanpa menulis ulang atau mengubah kueri inti. Jika Anda atau orang lain perlu mengubah jalur folder di masa mendatang, mereka hanya perlu memperbarui parameter di bagian depan.

Pro Tip: Merasa percaya diri dengan kode M? Lewati antarmuka dan ubah kode secara langsung untuk merujuk ke **CSVFilePath**. Beralih dari jalur yang dikodekan secara keras ke nilai parameter sangat menyederhanakan proses kerja Anda.

Sekarang setelah Anda mengetahui cara menghubungkan ke jalur file secara dinamis, mari kita lihat skenario ketiga di mana kita memfilter rentang tanggal.

c. Memfilter Rentang Tanggal

Nilai tanggal disertakan dalam banyak model, baik saat Anda membuat laporan penjualan bulanan, menyelami pola cuaca historis, atau mengidentifikasi tren keuangan. Namun, bekerja dengan seluruh riwayat tanggal kueri sering kali membuat kueri Anda lambat. Ini adalah skenario di mana parameter dapat membantu. Parameter memberikan fleksibilitas dalam hal mengelola rentang tanggal, yang mampu menangani rentang tanggal tetap dan bergulir.

Pertimbangkan skenario di mana, untuk meningkatkan kinerja, Anda ingin bekerja dengan data hanya dari beberapa kuartal terakhir. Daripada menyesuaikan tanggal mulai dan berakhir secara manual untuk setiap kuartal, Anda dapat menggunakan parameter.

Jika Anda memiliki tabel transaksi yang dilengkapi dengan kolom tanggal, Anda dapat menggunakan ekspresi seperti:

```
Table.SelectRows(
  DataTable,
  each [Date] >= StartDate and [Date] <= EndDate )
```

Dalam kode ini, argumen pertama merujuk ke tabel DataTable untuk difilter, sementara argumen kedua memfilter kolom Tanggal.

Setiap kali fokus Anda beralih ke kuartal baru, yang perlu Anda ubah hanyalah parameter StartDate dan EndDate. Namun, ada kendala: meskipun metode di atas memungkinkan Anda menyesuaikan nilai parameter dengan mudah, Anda tetap harus mengatur StartDate dan EndDate secara manual untuk setiap kuartal. Bukankah lebih baik untuk menyediakan parameter dengan nilai dinamis?

Sayangnya, parameter bawaan pita tidak mendukung nilai dinamis berdasarkan ekspresi, tetapi itu tidak berarti Anda kehabisan pilihan. Anda dapat menyertakan output ekspresi dalam kueri Anda, meskipun tidak diberi label formal sebagai parameter.

Misalnya, pikirkan situasi saat Anda ingin memfilter data dari kuartal yang baru saja selesai. Membuat kueri kosong memungkinkan Anda untuk memasukkan ekspresi berikut, yang menentukan EndDate:

```
Date.AddDays(
  Date.StartOfQuarter( Date.From( DateTime.LocalNow() ) ),
  -1 )
```

Fungsi ini mengambil nilai datetime hari ini, mengubahnya menjadi tanggal, lalu mengembalikannya ke awal kuartal saat ini, dan mengurangi satu hari. Dengan demikian, fungsi ini secara konsisten membawa Anda ke hari terakhir kuartal sebelumnya.

Untuk StartDate, Anda cukup merujuk ke EndDate dan menggesernya ke awal kuartal:

```
Date.StartOfQuarter( EndDate )
```

Walaupun pertanyaan-pertanyaan ini bukan bagian dari layar parameter resmi dalam pita, pertanyaan-pertanyaan tersebut berfungsi serupa dan dapat dirujuk sebagaimana mestinya.

C. Fungsi Khusus (*Custom Function*)

Fungsi kustom memainkan peran penting dalam bahasa M. Fungsi kustom memungkinkan Anda untuk menggabungkan logika, menjadikan kode Anda modular, dapat digunakan kembali, dan efisien. Anggaplah fungsi kustom sebagai alat yang dirancang untuk tugas tertentu.

Dan meskipun ada ratusan fungsi pustaka standar yang tersedia, terkadang membuat fungsi Anda sendiri berguna. Fungsi kustom memungkinkan Anda untuk membuat solusi khusus untuk tantangan data Anda yang menggabungkan beberapa langkah kustom atau mengatasi masalah transformasi data yang tidak dapat ditangani oleh fungsi bawaan.

Misalnya, pertimbangkan tugas-tugas kompleks seperti membuat total berjalan, mengambil nilai baris sebelumnya, atau menghitung angka minggu ISO. Menulis kode untuk transformasi ini secara manual dapat menjadi rumit dan rawan kesalahan. Namun, dengan menyediakan fungsi kustom, Anda sekarang dapat mencapai hasil yang diinginkan dengan panggilan fungsi yang dapat digunakan kembali yang hanya perlu Anda atur sekali. Setelah dibuat, fungsi-fungsi ini dapat dengan mudah digunakan kembali, sehingga memudahkan anggota tim dengan tingkat keahlian yang lebih rendah untuk memanfaatkan solusi kustom Anda.

Halaman berikut akan memandu Anda untuk memahami apa itu fungsi, cara membuat fungsi Anda sendiri, dan cara menggunakannya secara efektif. Sepanjang perjalanan, Anda juga akan mempelajari tentang penyempurnaan definisi fungsi, debugging, dan pengelolaan cakupan fungsi.

1. Apa itu fungsi Custom

Anda dapat menganggap fungsi kustom sebagai kueri mini. Fungsi tersebut menerima input, yang kita sebut parameter, dan menjalankan tindakan berdasarkan parameter tersebut untuk mengembalikan hasil. Agar dapat membuat fungsi kustom, penting untuk mengetahui elemen apa saja yang terkandung di dalamnya.

Anda dapat menentukan fungsi kustom dengan memberikan parameter di dalam serangkaian tanda kurung, diikuti oleh =>, dan diakhiri dengan ekspresi untuk dievaluasi. Ekspresi tersebut menentukan bagaimana fungsi tersebut harus menghitung hasilnya. Mari kita lihat sebuah contoh.

Fungsi sederhana yang menambahkan 10 ke suatu nilai dapat ditulis sebagai:

```
(parameter) => parameter + 10
```

Berikut cara menafsirkannya:

a. (*parameter*): Ini menunjukkan input atau parameter yang diambil fungsi kita sebagai input. Anda dapat memberi nama parameter sesuai keinginan.
b. Simbol Goes-to: simbol => memisahkan parameter fungsi dari ekspresinya.
c. parameter + 10: Ini adalah ekspresi (atau badan fungsi) yang menginstruksikan bagaimana fungsi harus memproses input untuk menghasilkan nilai output.

Anda dapat merancang fungsi agar berfungsi dengan lebih dari satu input. Misalnya, jika Anda ingin membuat fungsi yang mengalikan dua angka, Anda dapat melakukannya sebagai berikut:

```
(a, b ) => a * b
```

Meskipun a dan b adalah nama yang dipilih untuk parameter dalam fungsi di atas, Anda bebas memberi nama sesuai keinginan. Misalnya, definisi fungsi berikut tidak berbeda dengan yang di atas:

```
(left, right ) => left * right
```

Pro Tip: Meskipun Anda bebas memilih nama parameter fungsi, sebaiknya gunakan nama yang bermakna untuk parameter, terutama jika kode Anda ditujukan untuk audiens yang lebih luas. Nama tersebut membuat tujuan parameter menjadi jelas sejak awal.

Contoh sejauh ini menuliskan seluruh ekspresi fungsi tanpa menggunakan variabel apa pun. Namun, jika Anda ingin

menambahkan kejelasan tambahan pada fungsi Anda, pertimbangkan untuk mengintegrasikan ekspresi let. Berikut cara melakukannya:

```
(MyText ) =>
let
  TextToNumber = Number.FromText( MyText ),
  MultiplyBy3  = TextToNumber * 3,
  NumberToText = Number.ToText( MultiplyBy3 ),
  OutputString = "3 times "& MyText & " is " & NumberToText
 in
  OutputString
```

Fungsi ini mengubah string teks menjadi angka, mengalikannya dengan 3, dan menyediakan string kustom sebagai output. Jadi, jika Anda memberi nama fungsi ini fxMultiplyText dan memanggilnya dengan fxMultiplyText("5"), fungsi ini akan mengembalikan string: “3 kali 5 adalah 15".

Bergantung pada cara Anda memanggil fungsi ini, fungsi ini mungkin atau mungkin tidak memunculkan kesalahan. Jika Anda mengalami kesalahan, pastikan kode yang dihasilkan dalam bilah rumus menggunakan fxMultiplyText("5"), di mana angka 5 diapit oleh sepasang tanda kurung tunggal.

Beberapa fungsi tidak memerlukan parameter input apa pun. Contoh dari pustaka fungsi standar adalah DateTime.LocalNow. Anda dapat memanggilnya dengan cukup membuka dan menutup tanda kurung di belakang nama fungsi:

```
DateTime.LocalNow()
```

Ini akan mengembalikan nilai datetime yang sesuai dengan datetime lokal di komputer Anda. Anda juga dapat membuat fungsi kustom tanpa parameter. Misalnya, Anda dapat mengembalikan nilai radius bumi dengan menulis:

```
() => 6371
```

Pernyataan ini tidak mengambil parameter apa pun dan mengembalikan nilai angka. Namun, penggunaan seperti itu jarang terjadi. Daripada membuat fungsi seperti ini, sering kali lebih praktis untuk menyimpan nilai output sebagai parameter atau variabel dalam kode Anda.

Bagian ini membahas dasar-dasar pembuatan fungsi kustom, termasuk mendefinisikan parameter dan membuat badan fungsi. Anda diminta untuk menulis definisi fungsi secara manual. Jika Anda ingin membuat kode fungsi kustom secara otomatis, Power Query juga menawarkan alternatif yang praktis. Anda dapat mengubah kueri menjadi fungsi.

2. Mengubah Kueri menjadi Fungsi

Power Query memiliki fungsi yang memungkinkan Anda mengubah kueri menjadi fungsi. Tidak seperti fungsi kustom yang ditulis secara manual, metode ini membuat semua langkah kueri tetap terlihat dan dapat diedit. Bagian berikut membahas cara menggunakan fitur ini untuk membuat fungsi yang fleksibel dan dapat digunakan kembali dari kueri Anda.

a. Apa Fungsi “Create Function”?

Di Power Query, fitur Buat Fungsi memungkinkan Anda mengonversi kueri apa pun menjadi fungsi yang dapat digunakan kembali. Jika kueri Anda tidak memiliki parameter, kueri tersebut akan membuat fungsi tanpa parameter. Untuk fungsi yang lebih canggih, fitur ini menggunakan parameter dalam kode Anda untuk membentuk argumen fungsi.

Misalnya Anda ingin menambahkan parameter ke fungsi Anda. Untuk melakukannya, Anda harus mengidentifikasi elemen variabel dalam kueri Anda dan menautkannya ke parameter terkait dalam laporan Anda. Fitur Buat Fungsi kemudian secara otomatis mengintegrasikan kode yang

diperlukan untuk parameter input ini ke dalam fungsi kustom Anda.

Jika Anda memilih untuk tidak menggunakan parameter, fungsi tersebut hanya akan mengembalikan hasil kueri asli Anda yang tidak diubah. Namun, membuat fungsi tanpa parameter kurang umum, karena Anda dapat langsung menggunakan kueri asli. Fungsionalitas Create Function ini memiliki tiga keuntungan utama:

1) Visibilitas kueri: Kueri asli tetap terlihat dalam proyek Anda. Itu berarti Anda masih dapat memeriksa, mengubah, dan menjalankan logika Anda sebagai kueri mandiri.
2) Pembaruan fungsi: Perubahan pada kueri asli secara otomatis tercermin dalam fungsi yang ditautkan. Ini berarti Anda dapat dengan mudah menyempurnakan logika tanpa harus memperbarui fungsi.
3) Kemudahan membuat fungsi: Pengguna dapat membuat fungsi kustom tanpa menulis kode M, cukup dengan menggunakan parameter.

Jadi, bagaimana kita dapat membuat fungsi menggunakan User Interface (UI)? Untuk membuat fungsi dari kueri, Anda dapat mengikuti langkah-langkah berikut:

1) Jika Anda menginginkan parameter, masukkan parameter tersebut dalam kueri Anda. Ini bisa berupa sesuatu yang sederhana seperti kriteria filter atau nilai yang berkontribusi pada perhitungan.
2) Buka panel Queries yang terletak di sebelah kiri, lalu klik kanan pada kueri Anda.
3) Klik tombol Create Function. Ini akan memunculkan kotak dialog.
4) Buat nama fungsi Anda deskriptif sehingga mudah untuk mengenali fungsinya.
5) Klik OK untuk membuat fungsi.

Setelah langkah-langkah ini, fungsi baru Anda akan muncul di panel Queries, dan Anda dapat menggunakannya seperti fungsi lainnya di Power Query. Mari kita lanjutkan dan cari tahu bagaimana kita dapat menerapkan langkah-langkah ini dengan contoh praktis.

Misalkan Anda membuat logika untuk tabel kalender. Sebelum membuat fungsi, Anda menyiapkan kueri yang disebut CalendarLogic, yang terlihat seperti ini:

```
let
  Source = List.Dates(
            #date(2023,1,1),
            Duration.Days( #date(2023,10,1) - #date(2023,1,1) ) + 1,
            #duration( 1, 0, 0, 0 )
           ),
  ToTable = Table.FromList(Source, Splitter.SplitByNothing(), type table [Date = date], null, 1),
  AddYear = Table.AddColumn(ToTable, "Year", each Date.Year([Date]), Int64.Type),
  AddMonth = Table.AddColumn(AddYear, "Month", each Date.Month([Date]), Int64.Type)
in
  AddMonth
```

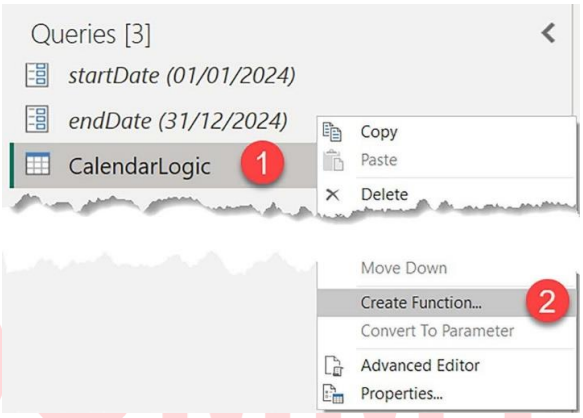
Gambar 1.11 Kueri *CalendarLogic*

Kueri ini menghasilkan kolom dengan tanggal dan menambahkan kolom Year dan Month. Anda ingin pengguna dapat menyesuaikan berapa lama kalender harus berjalan. Oleh karena itu, Anda memutuskan untuk menyertakan parameter startDate dan endDate bertipe date. Kueri CalendarLogic yang disesuaikan sekarang terlihat seperti ini:

```
let
  Source = List.Dates(
            startDate,
            Duration.Days( endDate - startDate ) + 1,
            #duration( 1, 0, 0, 0 )
           ),
  ToTable = Table.FromList(Source, Splitter.SplitByNothing(), type table [Date = date], null, 1),
  AddYear = Table.AddColumn(ToTable, "Year", each Date.Year([Date]), Int64.Type),
  AddMonth = Table.AddColumn(AddYear, "Month", each Date.Month([Date]), Int64.Type)
in
  AddMonth
```

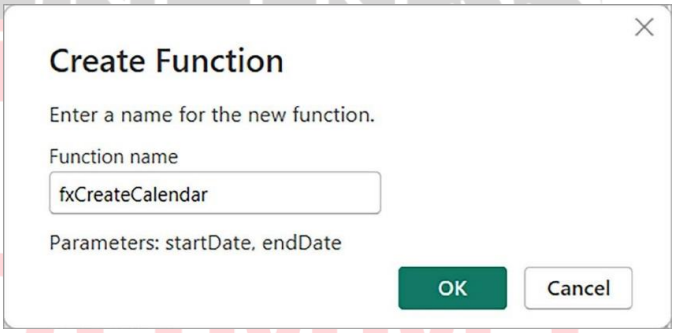
Gambar 1.12 Kueri CalendarLogic dengan Parameter

Dengan parameter ini, kita sekarang siap mengubah kueri menjadi fungsi. Untuk melakukannya, cukup klik kanan kueri Anda dan pilih Create Function, seperti yang ditunjukkan pada gambar berikut:



Gambar 1.13 Menu 'Create Function' di Panel Kueri

Anda kemudian dapat menentukan nama fungsi Anda dalam pesan pop-up yang muncul:

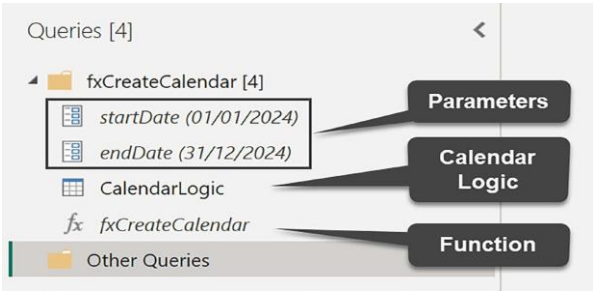


Gambar 1.14 Menu Buat Fungsi

Karena fungsi ini membuat tabel kalender, kita menyebut fungsi ini fxCreateCalendar. Setelah menekan OK, Power Query mengubah kueri Anda menjadi fungsi dengan nama yang baru saja Anda tentukan. Setelah melakukan operasi ini, Anda akan mendapatkan empat kueri:

1) Parameter *startDate* dan *endDate*
2) Kueri CalendarLogic dengan logika kalender Anda
3) Fungsi fxCreateCalendar yang terhubung dengan kueri CalendarLogic Panel kueri

Anda akan terlihat seperti gambar berikut:

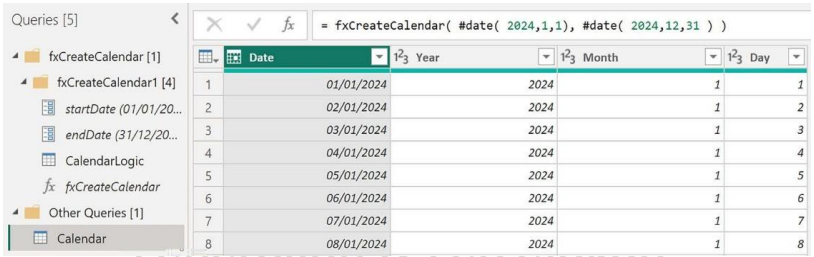


Gambar 1.15 Komponen yang digunakan dalam Operasi 'Buat Fungsi'

Sekarang setelah fungsi tersebut disiapkan, mari kita bahas cara menggunakannya secara efektif. Untuk melakukannya, buka kueri kosong baru, ganti namanya menjadi Calendar, dan masukkan kode berikut:

```
fxCreateCalendar( #date( 2024,1,1), #date( 2024,12,31 ) )
```

Ini menghasilkan tabel yang ditampilkan dalam gambar tangkapan layar berikut:



Gambar 1. 16 Gunakan Fungsi untuk membuat Tabel Kalender

Jadi dengan satu baris kode yang menggunakan fungsi kustom Anda, Anda akan mendapatkan tabel kalender yang menyertakan logika kustom Anda; bukankah itu hebat? Anda dapat menemukan solusi yang sudah jadi dalam berkas latihan yang disertakan dalam buku ini.

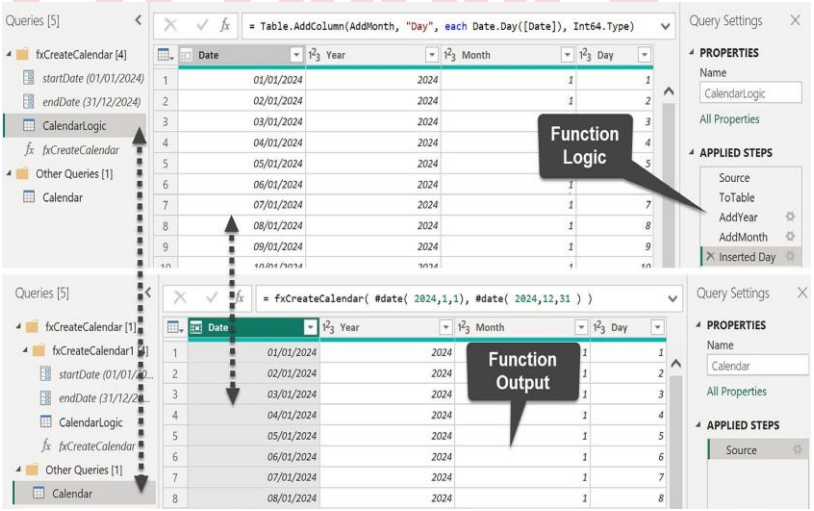
Sejauh ini, Anda mungkin berpikir keuntungan utama menggunakan fitur Create Function adalah otomatisasi pembuatan fungsi, tetapi keuntungannya lebih dari itu. Fitur

ini juga berguna dalam skenario yang memerlukan perubahan atau pemecahan masalah.

Merupakan pengalaman umum untuk mengalami kesalahan saat bekerja dengan fungsi kustom, sering kali karena kesalahan input atau kasus tepi yang tidak terduga yang tidak diperhitungkan oleh fungsi tersebut. Fungsionalitas Create Function memudahkan hidup Anda di sini, karena menawarkan cara sistematis untuk memeriksa setiap langkah fungsi Anda. Ini memungkinkan Anda untuk dengan mudah mengidentifikasi dan mengatasi penyebab kesalahan apa pun, yang akan kita tunjukkan selanjutnya.

b. Menyederhanakan Pemecahan Masalah dan Membuat Perubahan

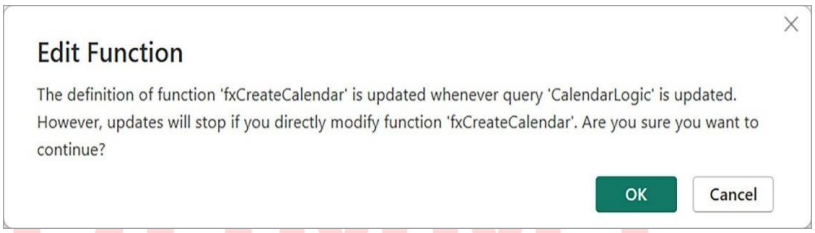
Keunggulan fitur Create Function terletak pada kemudahan Anda dalam melakukan modifikasi. Misalnya, Anda perlu menambahkan kolom yang menentukan nomor hari. Untuk melakukan perubahan tersebut, yang perlu Anda lakukan hanyalah membuka kueri CalendarLogic asli dan memasukkan kolom baru di sana. Proses ini ditunjukkan pada gambar berikut:



Gambar 1.17 Tautan Dinamis: Kueri dan Pembaruan Fungsi Kustom secara Bersamaan

Kini Anda dapat melihat langkah yang baru dibuat yang disebut Hari yang Dimasukkan dalam kueri CalendarLogic. Tautan dinamis antara fungsi fxCreateCalendar dan kueri CalendarLogic ini berarti setiap pembaruan yang Anda terapkan—baik itu menambahkan kolom atau mengubah rumus—akan langsung disinkronkan, sehingga Anda tidak perlu mengedit secara manual dan mengurangi kemungkinan terjadinya kesalahan. Hal ini tidak hanya menyederhanakan proses pemecahan masalah—memungkinkan Anda memeriksa setiap langkah kueri—tetapi juga memudahkan untuk memasukkan logika baru saat persyaratan berubah.

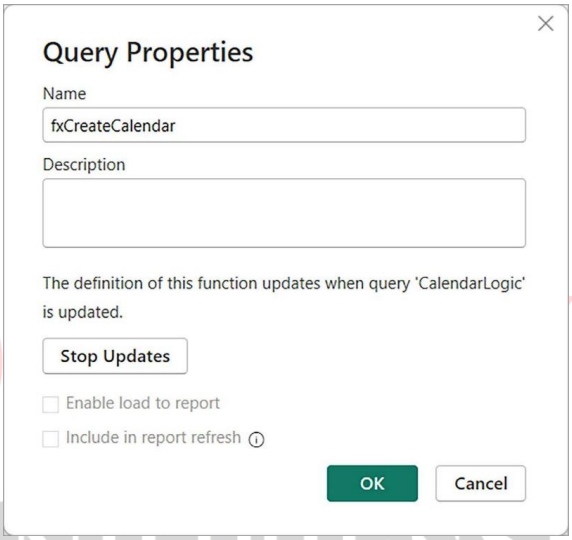
Saat Anda mencoba membuka Advanced editor dari fungsi yang ditautkan ini, sebuah pesan akan muncul. Pesan ini menunjukkan bahwa setiap modifikasi fungsi akan memutus tautan antara kueri, seperti yang ditunjukkan dalam tangkapan layar ini:



Gambar 1.18 Pesan Peringatan saat mencoba Mengedit Fungsi yang dibuat

Jika Anda hanya ingin memeriksa data, Anda dapat mengeklik OK dan melihatnya. Pastikan untuk tidak menyimpan perubahan apa pun dan tautan akan tetap ada.

Jika Anda ingin menghentikan pembaruan antara kueri dan fungsi, Anda juga dapat mengeklik kanan fungsi dan memilih Properties…. Layar ini berisi tombol Stop Updates, seperti yang ditunjukkan:



Gambar 1.19 Layar Properti suatu Fungsi yang Diperbarui dengan Perubahan Kueri

Pro Tip: Berhati-hatilah di sini. Setelah Anda mengeklik **Stop Updates**, tidak ada cara untuk memulihkan pembaruan ke fungsi saat ini. Anda harus menyiapkan fungsi baru dengan menggunakan operasi **Create Function**, seperti yang dijelaskan sebelumnya dalam bab ini.

Seperti yang telah kita lihat, fitur Create Function menawarkan cara mudah untuk mengubah kueri menjadi fungsi yang dapat digunakan kembali dan mudah disesuaikan. Namun, setelah fungsi kustom ini dibuat, bagaimana Anda benar-benar menggunakannya dalam laporan Anda? Bagian berikutnya akan membahas metode untuk memanggil fungsi kustom ini.

3. Memanggil Fungsi Custom

Setelah Anda membuat suatu fungsi, Anda dapat memanggilnya dengan berbagai cara. Anda dapat memanggilnya:

a. Secara manual di **Advanced editor** atau bilah rumus
b. Menggunakan UI

Jadi, bagaimana cara kerjanya?

a. Secara Manual di Editor Lanjutan atau Bilah Rumus

Sebagian besar pengguna tingkat lanjut memanggil suatu fungsi dengan merujuknya secara langsung. Misalkan Anda telah membuat suatu fungsi bernama *MyFunction* yang mengambil satu nilai numerik dan mengembalikan angka tersebut ditambah 10. Anda membuat kueri kosong dan menempelkan *MyFunction(20)* di bilah rumus, seperti yang diilustrasikan dalam gambar berikut:

Gambar 1.20 Memanggil Fungsi di Bilah Rumus

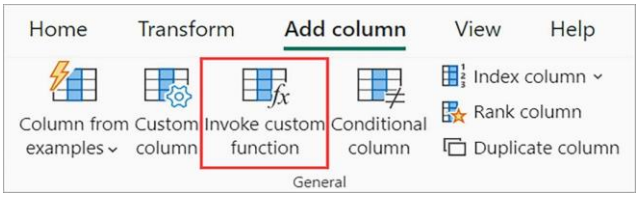
Ini memanggil fungsi dan mengembalikan 30. Bergantung pada ekspresi fungsi Anda, hasil yang merujuk pada suatu fungsi dapat berupa apa saja, baik itu list, tabel, atau kolom baru di tabel Anda—sebut saja. Jika Anda ingin memanggil fungsi kustom di dalam kolom baru, ada cara lain yang dapat dilakukan, yang akan kita bahas selanjutnya.

b. Menggunakan UI

Power Query juga menyediakan cara untuk membuat kolom yang memanggil fungsi melalui UI. Untuk melakukannya, Anda dapat:

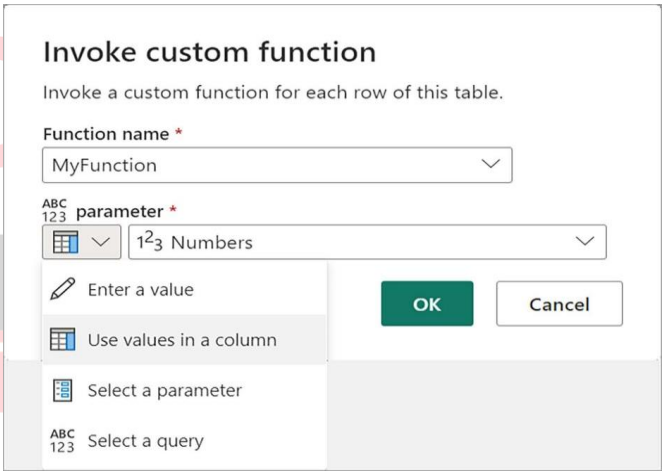
1) Buka tab **Add Column** di pita.
2) Pilih **Invoke Custom Function**.

Anda dapat menemukan opsi Invoke Custom Function pada gambar berikut:



Gambar 1.21 Tombol UI untuk Memanggil Fungsi Kustom

Ini akan meluncurkan menu pop-up yang menampilkan fungsi kustom yang tersedia di berkas Anda. Ini juga memungkinkan Anda menentukan parameter input untuk fungsi kustom Anda. Karena fungsi kustom yang digunakan sebagai berikut memiliki satu argumen, menu tersebut memerlukan informasi tentang satu parameter:



Gambar 1.22 Menu Fungsi Kustom Pemanggilan

Pada menu dropdown parameter, Anda memiliki beberapa opsi:

1) *Enter a value*: Memungkinkan Anda memasukkan nilai secara manual. Ini adalah opsi yang paling tidak dinamis.
2) *Use values in a column:* Memungkinkan Anda memilih kolom untuk menerapkan fungsi. Menu dropdown menyarankan kolom apa pun di tabel Anda.
3) *Select a parameter:* Memungkinkan Anda merujuk parameter. Parameter memungkinkan cara mudah untuk merujuk atau menyesuaikan nilai yang dapat digunakan di seluruh kueri Anda.
4) *Select a query:* Memungkinkan Anda merujuk output kueri. Perlu diingat bahwa suatu fungsi mungkin memerlukan tipe data input tertentu. Dengan merujuk kueri lain, nilai output kueri tersebut harus memiliki tipe data yang diperlukan fungsi Anda. Ini bisa berupa nilai primitif seperti teks atau

angka, tetapi juga nilai terstruktur seperti tabel, Record, atau list.

Karena metode ini memanggil fungsi menggunakan tab Add Column, hasil dari menjalankan fungsi Anda ditambahkan ke kolom baru.

Sekarang setelah Anda mengetahui dasar-dasar pemanggilan fungsi kustom, mari kita lihat ekspresi yang paling sering digunakan untuk membuat fungsi dalam bahasa M: ekspresi each. Ini adalah ekspresi yang paling sering digunakan karena Power Query secara otomatis memasukkannya ke dalam berbagai fungsi saat melakukan operasi melalui UI.

4. Ekspresi Each

Ekspresi each dalam bahasa M Power Query adalah sintaks singkat yang memungkinkan Anda menentukan fungsi unary (argumen tunggal) tanpa kerumitan deklarasi fungsi.

Jadi, mengapa ekspresi each merupakan bagian dari bahasa M, sementara kita dapat menentukan fungsi kita sendiri? Alasan terpenting adalah bahwa ekspresi ini lebih ramah pengguna dan membuat kode Anda lebih mudah dibaca.

Mari kita mulai dengan contoh dasar untuk memahami alasannya. Bayangkan Anda ingin membuat fungsi yang menerima angka dan menambahkan 5 ke dalamnya. Berikut cara penulisannya:

```
( MyValue) => MyValue + 5
```

Dalam fungsi ini, *MyValue* adalah satu-satunya parameter. Sekarang, mari kita sederhanakan:

```
(_) => _ + 5
```

Fungsi tersebut masih memiliki satu parameter, yang sekarang diwakili oleh garis bawah. Ini sangat mirip dengan apa yang dilakukan oleh konstruksi each. Konstruksi each adalah sintaks

gula untuk fungsi yang mengambil satu parameter, dengan garis bawah sebagai namanya.

Artinya, ekspresi berikut adalah versi yang lebih sederhana dari yang di atas:

```
each MyValue + 5
```

Deklarasi each yang disederhanakan digunakan untuk meningkatkan keterbacaan pemanggilan fungsi tingkat tinggi. Hal ini berguna karena mempercepat penulisan kode dan sering kali mengurangi panjang kode Anda. Untuk memahami artinya, mari kita lihat beberapa kasus penggunaan umum.

a. Kasus Penggunaan Umum

Table.SelectRows dan List.Transform. Dalam teks berikut, kita akan menggabungkan fungsi-fungsi ini dalam beberapa contoh. Kita akan melihat:

1) Penambahan otomatis *each* dalam kolom custom
2) Menggunakan masing-masing dalam *Table.TransformColumns*
3) Melewati garis bawah dalam referensi kolom atau bidang
4) Menyederhanakan pemanggilan fungsi argumen tunggal

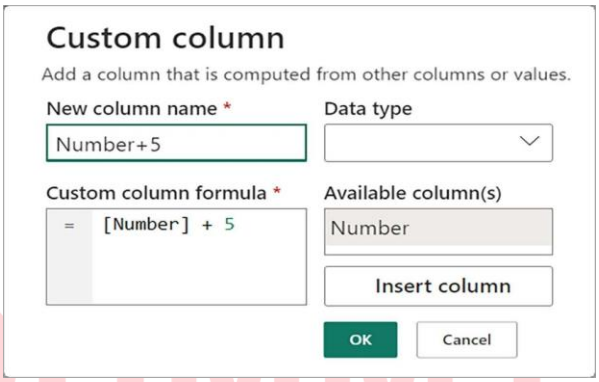
Mari kita cari tahu cara kerjanya.

b. Penambahan Otomatis Masing-Masing di Kolom Khusus

Bayangkan Anda memiliki kolom bernama Numbers dan ingin menambahkan 5 ke setiap nilai. Anda dapat membuka Add Column, pilih Custom Column, dan menambahkan rumus:

```
[Number] + 5
```

Jendela pop-up Custom column yang muncul terlihat seperti ini:



Gambar 1.23 Menyesuaikan Kolom

Setelah mengklik OK, ekspresi berikut muncul di bilah rumus:

```
Table.AddColumn(
  Source,
  "newColumn",
   each [Number] + 5 )
```

Tunggu dulu—kode di atas menyertakan kata kunci each pada baris 4. Bagaimana kode itu muncul di sana padahal kita tidak menyediakannya di kotak rumus kolom kustom?

Inilah alasannya: saat Anda menambahkan kolom kustom menggunakan UI Power Query, perangkat lunak akan mengerjakan sebagian tugas untuk Anda. Perangkat lunak tersebut secara otomatis memanggil fungsi Table.AddColumn dan menambahkan kata kunci each sebagai awalan ke rumus Anda. Anda tidak perlu memasukkannya secara manual. Ini berguna karena konsep each sering membingungkan bagi pemula. Mari kita lihat bagaimana konsep yang sama ini berlaku pada fungsi Table.TransformColumns.

c. Menggunakan Masing-Masing di Table.TransformColumns

Fungsi Table.TransformColumns digunakan untuk memanipulasi nilai kolom. Saat Anda menggunakan fungsi ini

melalui antarmuka Power Query, Anda akan sering melihat kata kunci each beraksi.

Bayangkan Anda ingin menambahkan awalan Miss ke nilai dalam kolom yang disebut Name. Untuk melakukannya, Anda pilih kolom, buka tab Transform, klik Format, pilih Add Prefix, dan ketik Miss. Kode yang dihasilkan akan muncul sebagai berikut:

```
Table.TransformColumns(
  Source, {{"Name", each "Miss " & _, type text}} )
```

Ekspresi ini secara fungsional setara dengan:

```
Table.TransformColumns(
  Source, {{"Name", (_) => "Miss " & _, type text}} )
```

Perhatikan perbedaan kedua contoh kode tersebut. Kuncinya terletak pada pemahaman tentang cara kerja fungsi Table.TransformColumns. Fungsi ini memerlukan cara khusus untuk menentukan cara mengubah setiap nilai dalam kolom. Di sinilah parameter fungsi berperan, yang mewakili setiap nilai dalam kolom yang sedang kita ubah.

Pada pendekatan pertama, kita menggunakan each, sebuah metode singkat. Pendekatan kedua menggunakan (_) =>, definisi fungsi yang lebih eksplisit. Kedua metode tersebut mencapai hasil yang sama: mereka menambahkan kata Miss di awal setiap entri dalam kolom Name.

Seperti yang telah Anda lihat dalam contoh-contoh ini, garis bawah sering digunakan untuk merujuk pada nilai yang sedang diubah. Mari kita lanjutkan dan jelajahi contoh lain untuk mengetahui kapan kita dapat menghilangkan garis bawah.

d. Melewati Garis Bawah pada Referensi Bidang atau Kolom

Dalam situasi tertentu, tidak apa-apa untuk menghilangkan karakter garis bawah saat Anda menangani data atau kolom

tabel. Misalnya, Anda ingin menambahkan kolom baru yang mengurangi 10 dari nilai di kolom Penjualan yang sudah ada. Bentuk panjang untuk menuliskannya adalah:

```
Table.AddColumn(Source, "Custom", (_) => _[Sales] - 10 )
```

Garis bawah kemudian merujuk ke seluruh baris tabel, yang direpresentasikan oleh sebuah Record.

Saat kita mengakses kolom secara langsung menggunakan tanda kurung siku—proses yang dikenal sebagai pemilihan kolom—kita tidak perlu mengawali nama kolom dengan garis bawah. Ini memungkinkan kita menulis versi yang lebih ringkas:

```
Table.AddColumn(Source, "Custom", (_) => [Sales] - 10 )
```

UI *Power Query* menyederhanakan hal ini lebih jauh dengan konstruksi *each*:

```
Table.AddColumn(Source, "Custom", each [Sales] - 10 )
```

Ketiga ekspresi tersebut mencapai hasil yang sama, yaitu mengurangi 10 dari setiap nilai di kolom [Sales].

Kita baru saja melihat penyederhanaan kode kita dengan menghilangkan garis bawah. Namun, Anda dapat menyederhanakan kode lebih jauh lagi saat memanggil fungsi yang hanya memiliki satu argumen. Mari kita cari tahu caranya.

e. Menyederhanakan Panggilan Fungsi Argumen Tunggal

Banyak fungsi yang memungkinkan Anda untuk memasukkan fungsi lain sebagai parameter. Cara Anda melakukannya bergantung pada jumlah argumen yang dibutuhkan fungsi bersarang. Saat menangani fungsi argumen tunggal, semuanya bisa jadi cukup sederhana. Anda dapat menghilangkan tanda kurung dan langsung merujuk ke nama fungsi.

Misalnya, perhatikan contoh berikut di mana kita menambahkan 10 ke setiap elemen dalam list:

```
let
    fxAddTen = (MyValue) => MyValue + 10,
    MyList = { 1 .. 5 },
    TransformList = List.Transform( MyList, fxAddTen )
  in
    TransformList
```

Dalam contoh ini, kita memanggil fungsi fxAddTen di dalam fungsi List.Transform tanpa menggunakan tanda kurung. Namun, ada cara lain yang sama validnya untuk melakukan ini. Kita dapat mengganti langkah TransformList dengan:

```
TransformList = List.Transform( MyList, each fxAddTen(_))
```

Contoh-contoh di atas mengilustrasikan bagaimana Anda dapat menyederhanakan fungsi dengan satu argumen. Contoh-contoh tersebut memiliki opsi untuk menghilangkan kata kunci each dan tanda kurung.

Namun, ketika suatu fungsi memerlukan lebih dari satu argumen, pendekatannya berubah. Fungsi dengan beberapa parameter mengharuskan Anda untuk:

1) Menggunakan konstruksi each
2) Menuliskan definisi fungsi secara formal

Berikut ini contohnya

```
TransformList = List.Transform( MyList, each fxAddTen( _, 10) )
```

Berikut ini adalah pendekatan manualnya:

```
TransformList = List.Transform( MyList, (_)=> fxAddTen( _, 10) )
```

Singkatnya, fungsi argumen tunggal dapat dipanggil hanya dengan menyebutkan nama fungsi, tanpa tanda kurung. Untuk fungsi argumen ganda, sintaksis yang lebih rinci diperlukan, yang melibatkan kata kunci each atau definisi fungsi kustom.

Sejauh ini, kita telah melihat cara membuat dan memanggil fungsi pada tingkat paling dasar. Namun, ada cara untuk menyempurnakan fungsi agar lebih tangguh dan mencegah kesalahan sejak awal. Itulah yang akan kita bahas selanjutnya.

5. Menyempurnakan Definisi Fungsi

Saat berupaya mewujudkan fungsi kustom ideal Anda, yang terpenting bukan hanya menciptakannya—tetapi menyempurnakannya. Penting untuk dicatat bahwa fungsi yang terdefinisi dengan baik memiliki tujuan yang jelas, fleksibel dalam penggunaannya, dan tangguh terhadap kesalahan.

Membangun fungsi kustom sering kali dimulai dengan logika dasar, dan melalui beberapa iterasi, lebih banyak langkah disertakan dalam fungsi tersebut hingga kita mencapai hasil yang diinginkan. Biasanya pada tahap ini kita menyempurnakan fungsi kita, sehingga menjadi lebih tangguh.

Pada bagian berikut, kita akan membahas dua metode untuk menyempurnakan fungsi kustom:

- **Menentukan tipe data**: Ini memastikan data tetap konsisten dan akurat sehingga input fungsi sesuai dengan yang diharapkan.
- **Menambahkan parameter opsional:** Ini memungkinkan pengguna memiliki fleksibilitas dalam cara menggunakan fungsi, memenuhi berbagai kebutuhan tanpa batasan yang ketat.

Strategi ini tidak hanya meningkatkan kegunaan fungsi kustom, tetapi juga membuatnya lebih tangguh dan mudah beradaptasi dengan berbagai skenario. Mari kita lihat bagaimana metode ini dapat meningkatkan kueri Anda.

a. Menyederhanakan Panggilan Fungsi Argumen Tunggal

Saat membuat fungsi kustom, tujuan kita bukan hanya membuatnya beroperasi, tetapi juga membuatnya andal dan tidak mudah mengalami kesalahan. Salah satu cara untuk melakukannya adalah dengan menyatakan dengan jelas jenis

data yang diterima setiap parameter. Dengan cara ini, kita memastikan fungsi kita hanya menerima jenis nilai tertentu, yang mencegah kejutan. Misalnya, jika kueri Anda mengharapkan tanggal tetapi Anda malah memberikan nilai list, fungsi Anda kemungkinan akan menghasilkan kesalahan.

Untuk parameter input dan output fungsi, Anda dapat menentukan tipe data. Parameter tanpa tipe data yang ditentukan disebut sebagai implicit parameter. Ini berarti parameter dapat menerima nilai dari tipe apa pun. Menentukan tipe data untuk parameter mengubahnya menjadi explicit parameter.

Setelah Anda menentukan tipe data untuk parameter input Anda, fungsi kustom hanya menerima nilai dari tipe yang ditentukan. Di Bab 5, saat membahas tipe data, kita mempelajari tentang tipe primitif dan tipe primitif yang dapat bernilai null. Sekarang kita akan mempelajari bagaimana perbedaan antara keduanya menjadi relevan saat menyempurnakan fungsi kustom.

b. Tipe Primitif

Mari kita lihat dulu bagaimana kita dapat menetapkan tipe data primitif ke suatu parameter. Data yang digunakan untuk menerapkan fungsi kustom berikut tersedia dalam berkas latihan yang disertakan dalam buku ini.

Bayangkan kita memiliki kolom berisi usia orang, dan Anda ingin membuat fungsi bernama fxValidAge. Fungsi tersebut memeriksa apakah usia tersebut realistis, yang mana kita berasumsi usia tersebut harus berada di antara 0 dan 125. Berikut cara melakukannya:

```
(age as number) as
  logical => age
```

Dalam contoh ini, usia sebagai angka adalah parameter eksplisit. Ini menunjukkan bahwa fungsi tersebut memerlukan nilai angka untuk argumen usia. Setelah tanda kurung, Anda

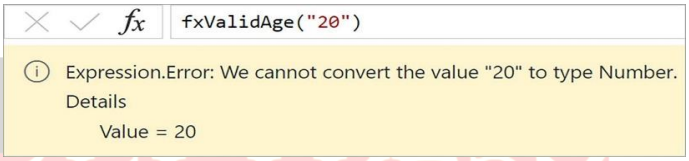
akan menemukan teks sebagai logical, yang berarti fungsi tersebut akan memberikan hasil yang benar (usia valid) atau salah (tidak valid).

Pada gambar berikut, Anda dapat melihat tabel dengan dua kolom. Kolom fxValidAge menerapkan fungsi di atas ke kolom Age dan mengembalikan hasilnya:

	1²3 Age	ABC 123 fxValidAge
1	23	TRUE
2	105	TRUE
3	5	TRUE
4	71	TRUE
5	-20	FALSE
6	195	FALSE

Gambar 1.24 Tes Fungsi Apakah Usia Valid

Kolom yang baru dibuat sekarang menampilkan nilai benar atau salah untuk menunjukkan apakah usia valid. Saat kita menentukan tipe data untuk parameter input, hal itu memaksa mesin untuk memverifikasi apakah nilai input cocok dengan tipe data. Pada titik ini, semua baris mengembalikan nilai yang valid karena usia input bertipe angka. Namun, jika Anda mencoba memberikan teks (seperti "20") atau non-angka apa pun, Power Query akan mengembalikan kesalahan berikut:



Gambar 1.25 Memberikan Angka selain Angka akan Mengakibatkan Kesalahan

Pesan galat ini muncul karena parameter eksplisit kita memerintahkan fungsi untuk hanya menerima nilai bertipe angka, tetapi menerima nilai teks. Penjelasan sejauh ini adalah tentang input fungsi; namun, kita juga dapat memerintahkan tipe data untuk output fungsi.

Sebagai output, fungsi saat ini mengharapkan nilai logika (true atau false). Ini didefinisikan sebagai logika dalam definisi fungsi. Jika Anda mengubah logika menjadi text, tetapi hasilnya tetap berupa nilai Boolean (true atau false), galat akan muncul yang mengatakan tipe data tidak seperti yang diharapkan.

Aspek penting lain dalam membangun fungsi yang tangguh adalah menangani nilai null. Sangat umum bagi kolom Anda untuk berisi nilai null dan angka. Keduanya memiliki perilaku yang sangat spesifik dan dapat dengan mudah merusak kueri Anda. Karena fungsi kustom di atas saat ini tidak menangani nilai null, menggunakannya dalam kueri Anda berisiko dan dapat menyebabkan galat saat menemukan null. Untungnya, kita dapat menambahkan dukungan untuk nilai null dengan menggunakan tipe data yang dapat berupa null saat mendefinisikan parameter fungsi Anda, yang akan kita bahas selanjutnya.

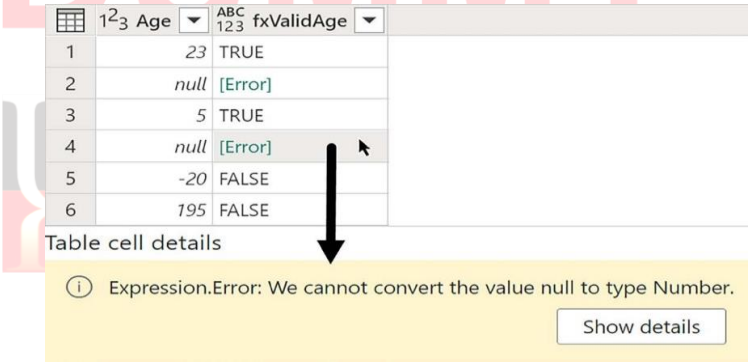
c. Tipe Primitif yang Dapat Dibatalkan

Terkadang, data Anda mungkin menyertakan nilai yang hilang atau tidak diketahui, yang disebut sebagai nilai null. Dalam skenario di mana kolom dapat berisi nilai null tersebut di samping nilai primitif, tipe primitif yang dapat berupa null menjadi tidak berharga. Tipe ini dirancang untuk menerima nilai primitif tertentu dan nilai null. Kita pertama kali membahas tipe yang dapat berupa null di Bab 1 Jilid 2, Memahami Tipe Data, yang dapat menjadi topik yang rumit. Untuk pemahaman yang lebih baik, kita sarankan untuk meninjau kembali bab tersebut sebelum melanjutkan.

Menggunakan tipe primitif yang ketat (seperti teks) untuk kolom yang mungkin menemukan nilai null dapat menyebabkan kesalahan dalam kueri Anda saat nilai null muncul. Di sisi lain, menentukan kolom sebagai tipe apa pun sering kali merupakan klasifikasi yang terlalu luas. Untuk mengatasi hal ini, sebaiknya gunakan tipe primitif yang dapat berupa null. Istilah 'dapat berupa null' di sini berarti bahwa

kolom dapat berisi nilai dari tipe primitif yang ditentukan atau nilai null. Jadi, bagaimana kita dapat menerapkannya?

Ingat fungsi pemeriksaan usia kita dari halaman sebelumnya? Misalkan Anda memiliki kumpulan data usia pelanggan, tetapi beberapa entri hilang. Mari kita lihat bagaimana Anda dapat menggunakan tipe nullable untuk menangani hal ini. Jika Anda menggunakan fungsi apa adanya pada kumpulan data ini, Anda akan mendapatkan kesalahan yang terlihat seperti gambar berikut:



Gambar 1.26 Kesalahan terjadi ketika suatu fungsi menemui nilai null yang tidak diharapkan

Fungsi Anda memeriksa apakah setiap nilai input cocok dengan tipe data yang diharapkan. Saat ini, fungsi tersebut diatur untuk menerima angka saja. Jika menemukan nilai yang tidak dikenali, seperti nilai null, fungsi tersebut akan bingung dan menghasilkan kesalahan.

Oleh karena itu, mempelajari cara mengelola situasi dengan nilai null sangatlah penting. Hal ini memungkinkan Anda untuk menangani data yang tidak sempurna yang sering kali ada di sumber data. Untuk mengatasi masalah ini, kita akan mengubah tipe parameter menjadi nullable number:

```
    (age as nullable number) as logical =>
      if age >= 0 and age <=  125 then true else false
```

Fungsi tersebut kini hanya menerima nilai usia dengan tipe angka yang dapat berupa null. Dengan modifikasi ini, fungsi

Anda akan mencoba mengembalikan null setiap kali menemukan nilai null yang tidak terhitung dalam ekspresi fungsi. Sayangnya, kode yang diubah masih mengembalikan kesalahan Expression.Error: Kita tidak dapat mengonversi nilai null ke tipe Logical.

Alasannya adalah fungsi kita juga memberlakukan nilai logika sebagai output. Dan karena fungsi tersebut mencoba mengembalikan null saat menemukan null di salah satu parameter, kita mendapatkan kesalahan.

Kita dapat memperbaikinya dengan mengubah output fungsi ke tipe yang dapat berupa null, seperti yang ditunjukkan di sini:

```
(age as nullable number) as nullable logical =>

      if age >= 0 and age <=  125 then true else false
```

Dengan perubahan yang kita buat pada kode di atas, keluaran fungsi sekarang menerima nilai Boolean dan null. Cuplikan layar berikut mengilustrasikan hasil penerapan fungsi ini pada kolom Usia:

	1²3 Age	ABC 123 fxValidAge
1	23	TRUE
2	null	null
3	5	TRUE
4	null	null
5	-20	FALSE
6	195	FALSE

Gambar 1.27 Hasil Penerapan Fungsi ini pada Kolom Usia

Ketika kita menerapkan fungsi baru ini ke kolom dengan usia (termasuk null), ia bekerja dengan lancar.

Singkatnya, dengan mendefinisikan tipe data, Anda menambahkan presisi ke fungsi Anda, memastikan input yang valid, dan memastikannya mengembalikan nilai tipe data yang diharapkan. Ia tahu apa yang diharapkan dan bagaimana meresponsnya.

Anda mungkin juga menghadapi situasi di mana Anda ingin fungsi Anda mampu menangani skenario yang berbeda. Misalkan beberapa skenario akan memerlukan lebih banyak variabel daripada yang lain. Dalam kasus tersebut, Anda dapat menggunakan beberapa argumen opsional. Dengan begitu, Anda dapat memiliki beberapa argumen yang diperlukan untuk menjaga semuanya tetap sederhana, namun menyediakan lebih banyak fungsionalitas dengan menawarkan satu atau lebih argumen opsional. Di bagian selanjutnya, kita akan membahas bagaimana Anda dapat membuat argumen opsional.

d. Menjadikan Parameter Opsional

Secara default, nilai harus diberikan untuk setiap parameter saat fungsi dipanggil. Namun, saat bekerja dengan fungsi kustom, fleksibilitas dapat menjadi fitur penting. Cara efektif untuk memperkenalkan fleksibilitas adalah dengan mengizinkan parameter tertentu menjadi opsional. Ini berarti bahwa saat memanggil fungsi, tidak selalu perlu memberikan nilai untuk setiap parameter. Misalnya, fungsi Date.StartOfWeek secara default memerlukan satu argumen. Argumen kedua yang opsional memungkinkan Anda menentukan hari mana yang Anda harapkan sebagai awal minggu. Jadi, bagaimana cara kerjanya?

Saat Anda menentukan parameter fungsi, Anda dapat menjadikannya opsional dengan menambahkan teks opsional. Parameter tersebut sering kali disertai dengan nilai default , yang memastikan bahwa fungsi tersebut tetap dapat berjalan meskipun tidak ada nilai yang diberikan untuk parameter tertentu tersebut. Mari kita lihat sebuah contoh.

Dalam contoh sebelumnya, kita menguji apakah usia valid. Bayangkan jika kita ingin memberi pengguna opsi untuk menetapkan usia valid maksimum. Kita dapat memperkenalkan parameter opsional, sebagai berikut:

```
(age as nullable number, optional maxAge as nullable
  number) as nullable logical =>
    age >= 0 and age <=  (maxAge ?? 125)
```

Kode tersebut kini menyertakan parameter maxAge opsional yang mengharapkan angka yang dapat bernilai null sebagai input. Mari kita periksa apa yang dilakukan kode ini:

1) Fungsi kustom sebelumnya memiliki dua argumen. Argumen pertama memerlukan *Age*, tetapi argumen *maxAge* bersifat opsional.
2) Argumen kedua dibuat opsional dengan mengawalinya dengan kata opsional. Dengan demikian, setiap kali pengguna memanggil fungsi tersebut, argumen kedua tidak perlu diisi.
3) Kedua argumen tersebut menggunakan tipe yang dapat bernilai null, yang menyiratkan bahwa keduanya mendukung tipe primitif dan nilai null.
4) Operator *Coalesce (??)* digunakan bersama dengan *maxAge* untuk menyediakan nilai fallback, seperti yang akan kita bahas selanjutnya.

Jadi, apa yang terjadi jika Anda memanggil fungsi tersebut tanpa menyediakan argumen opsional? Jika argumen opsional dihilangkan, fungsi tersebut secara default menggunakan nilai null. Dengan demikian, penting untuk mengantisipasi dan menangani potensi nilai null ini. Di sinilah operator Coalesce berguna. Operator ini memungkinkan Anda untuk menyediakan nilai fallback. Pertama-tama operator akan memeriksa nilai utama di sebelah kiri operator. Jika bukan null, operator akan mengembalikannya; jika null, operator akan mengembalikan nilai fallback di sebelah kanan. Dalam konteks argumen kedua fungsi tersebut, operator Coalesce akan mengembalikan 125 ketika dihadapkan dengan null.

Pro Tip: Power Query mengharuskan parameter opsional diposisikan setelah argumen yang diperlukan. Itu berarti bahwa setelah Anda menentukan parameter opsional, semua parameter yang mengikutinya juga harus opsional.

Anda sekarang dapat memanggil fungsi ini dengan satu atau dua parameter:

```
fxValidAge( 100  // Output: true
)
fxValidAge( 100, 90 ) // Output: false
```

Dalam kasus pertama, hanya parameter wajib yang disediakan. Dalam kasus kedua, parameter opsional disertakan. Parameter ini memeriksa apakah usia 100 berada dalam rentang 0 hingga 90. Karena tidak, parameter ini mengembalikan nilai false. Mengetahui hal ini akan memungkinkan pengguna untuk mengambil tindakan alternatif untuk nilai outlier. Misalnya, mereka mungkin mengembalikan kesalahan, atau menganggap input salah dan memberikan nilai fallback sebagai gantinya.

Sekarang setelah Anda mengetahui cara membuat fungsi, menentukan tipe datanya, dan bahkan cara menjadikan parameter opsional, mari kita lihat cara merujuk nama kolom dan bidang dalam fungsi. Anda mungkin ingin merujuk nama bidang menggunakan pemilihan bidang (menggunakan tanda kurung siku), tetapi anehnya, itu tidak akan berhasil. Mari kita cari tahu alasannya.

6. Referensi Nama Kolom dan Bidang

Membuat fungsi kustom di Power Query melibatkan langkah penting: mendefinisikan parameter. Parameter ini sangat penting, karena memungkinkan pengguna memengaruhi output fungsi. Namun, mereferensikan objek yang diperlukan untuk logika fungsi Anda tidak selalu mudah. Misalnya, dalam rumus standar, Anda dapat mereferensikan kolom hanya sebagai Kolom1.

Namun pendekatan langsung ini tidak berfungsi saat pengguna memasukkan nama kolom sebagai parameter dalam fungsi Anda; notasi tanda kurung siku yang umum tidak berlaku dalam konteks ini.

Bagian selanjutnya membahas skenario saat referensi objek langsung di editor Kueri menimbulkan tantangan. Bagian ini memandu Anda melalui metode alternatif, dengan fokus pada penggunaan fungsi untuk membuat referensi Anda. Pendekatan ini diperlukan saat teknik referensi standar tidak berhasil. Dengan menguasai teknik ini, Anda akan meningkatkan kemampuan Anda untuk membuat fungsi kustom yang efektif.

a. Mengacu pada Nama Bidang dalam Sebuah *Record*

Misalkan Anda ingin memfilter baris tabel. Fungsi *default* untuk memfilter baris dalam tabel adalah *Table.SelectRows*. Misalnya, untuk mengembalikan baris dengan jumlah Penjualan kurang dari 1.000, Anda menulis:

```
Table.SelectRows( Source, each [Sales] < 1000 )
```

Namun, bagaimana jika Anda menginginkan fungsi yang memungkinkan pengguna memilih kolom mana yang akan difilter? Anda dapat mencoba ini, tetapi tidak akan berhasil:

```
( table as table, columnName as text ) as table =>
  Table.SelectRows( table, each [columnName] < 1000 )
```

Ketika Anda menjalankan ini, ia memfilter tabel dalam argumen pertama dengan baik. Namun, ia tidak melakukan apa yang Anda harapkan dengan argumen kedua. Alih-alih mengambil input pengguna dari argumen columnName, mesin mencari kolom bernama columnName.

Jadi, bagaimana cara memperbaikinya? Pertama, pahami apa arti [Sales] dalam rumus awal. Itu bisa berupa nama kolom dalam tabel atau nama bidang dalam Record.

Jika Anda tidak yakin, berikut cara mudah untuk mengujinya. Coba ganti kondisi dengan garis bawah; Anda

akan mendapatkan kesalahan, tetapi kesalahan ini membantu kita. Dikatakan, Kita tidak dapat mengonversi nilai bertipe Record ke tipe Logical. Ini mengonfirmasi bahwa kita berurusan dengan Record.

Untuk merujuk ke kolom secara dinamis, Anda dapat menggunakan fungsi Record.Field. Fungsi ini mengambil dua input: Record dan nama kolom sebagai teks. Berikut cara Anda dapat menulis ulang fungsi tersebut:

```
( table as table, columnName as text ) as table =>
Table.SelectRows( table,

  each Record.Field(_, columnName) < 1000)
```

Dengan cara ini, fungsi tersebut akan memfilter baris berdasarkan nama kolom yang diberikan pengguna, sehingga menghasilkan perilaku dinamis yang kita inginkan. Solusi yang diberikan berfungsi untuk Record, tetapi bagaimana Anda dapat melanjutkan saat memilih kolom dari tabel?

b. Mengacu pada Kolom dalam Tabel

Objek lain dengan masalah serupa adalah kolom tabel. Misalkan Anda ingin memeriksa apakah suatu nilai di baris saat ini juga ada di kolom dari tabel lain. Untuk contoh ini, mari kita pertimbangkan tabel bernama Bonus dengan kolom bernama Products.

Jika Anda tidak memerlukan kolom sebagai parameter, rumusnya bisa terlihat seperti ini:

```
Table.AddColumn(

  Source,
  "IsBonusProduct",

  each if List.Contains(Bonus[Products], [Product])
    then true else false )
```

Disini, kita menambahkan kolom baru bernama IsBonusProduct ke tabel sumber. Output memberi tahu kita jika Product yang sesuai juga ada di list Bonus[Products].

Tetapi bagaimana jika Anda ingin membuatnya lebih fleksibel? Misalnya, Anda mungkin ingin membiarkan pengguna menentukan tabel pencarian dan kolom. Itu akan memungkinkan mereka untuk mempertahankan tabel sederhana dari nilai pengganti. Misalnya, mereka dapat memiliki tabel kalender dan ingin menentukan nama kolom untuk setiap bahasa yang mereka gunakan. Menyimpannya dalam tabel jauh lebih mudah daripada mengodekannya secara permanen.

Sayangnya, referensi langsung ke kolom tidak memungkinkan. Untuk mencapainya, Anda harus menggunakan fungsi Table.Column, yang mengembalikan list dari tabel dan kolom tertentu.

Berikut cara Anda dapat menulis fungsi kustom untuk menangani ini:

```
( table as table, newColumnName as text, lookupTable as table,
lookupColumnName as text ) as table =>
Table.AddColumn(
  table,
  newColumnName,
  each if List.Contains(
    Table.Column( lookupTable, lookupColumnName),
      Record.Field( table, "Product" ) )
    then true else false )
```

Dalam fungsi ini, Anda memiliki empat parameter:

1) *table*: Tabel utama tempat Anda bekerja
2) *newColumnName*: Nama kolom baru yang Anda tambahkan
3) *lookupTable*: Tabel yang berisi list yang Anda periksa
4) *lookupColumnName*: Kolom tertentu di *lookupTable* yang memiliki list

Agar ini berfungsi secara efektif, Anda perlu menggunakan dua fungsi khusus: Table.Column untuk menargetkan kolom dan Record.Field untuk menargetkan bidang.

Pertama, Table.Column mengambil list nilai dari tabel pencarian dan kolom yang ditentukan. Selanjutnya, fungsi tersebut memeriksa apakah Produk saat ini, yang diidentifikasi oleh fungsi Record.Field, ada dalam list tersebut. Kemudian, fungsi tersebut mengembalikan true atau false, berdasarkan pemeriksaan ini.

Bagian penting di sini adalah bahwa pengguna sekarang dapat memberikan nilai teks untuk lookupTable dan lookupColumnName, dan fungsi tersebut akan mengambil nilai kolom yang sebenarnya. Nanti di bab ini, kita akan menjelajahi proyek yang mengilustrasikan cara Anda dapat menggunakan ini dalam contoh praktis.

Fungsi kustom dapat menambah kompleksitas pada kode Anda. Saat Anda memperkenalkan lebih banyak iterasi, kode Anda akan memanjang, terkadang meluas hingga beberapa halaman. Kompleksitas tambahan ini dapat dikelola hingga Anda menemukan kesalahan dalam fungsi kustom Anda. Pada titik ini, pemeriksaan kode Anda langkah demi langkah menjadi berguna untuk menentukan di mana kesalahan terjadi. Tidak seperti kueri biasa, fungsi tidak secara inheren menyediakan cara sederhana untuk melakukan pemeriksaan terperinci ini. Oleh karena itu, menguasai teknik debugging untuk fungsi kustom sangatlah penting. Bagian selanjutnya akan memandu Anda melalui metode yang efektif untuk mengatasi tantangan ini.

7. Men-debug Fungsi Khusus

Proses pembuatan fungsi kustom menjadi mudah setelah Anda memahami dasar-dasarnya. Proses ini melibatkan pendefinisian parameter dan pengintegrasiannya ke dalam kode Anda, mengubah kueri Anda menjadi sebuah fungsi. Namun, apa yang terjadi jika Anda menemukan kesalahan dalam fungsi kustom?

Atau mungkin Anda mengadaptasi kode yang Anda temukan secara daring dan mengalami masalah? Kini Anda telah kehilangan langkah-langkah dari panel kueri yang biasanya membantu Anda memecahkan masalah kode. Nah, di sinilah pemahaman tentang cara men-debug fungsi kustom berperan. Misalnya, Anda bekerja dengan fungsi tertentu seperti berikut:

```
( numberList as list, minValue as number, maxValue as number ) as text =>
  let
    List        = List.Select ( numberList, each _ >= minValue and _ <= maxValue ),
    ListToText  = List.Transform ( FilteredList, each Text.From ( _ ) ),
    CombineText = Text.Combine ( ListToText, ", " )
  in
    CombineText
```

Gambar 1.28 Anda mengenali fungsi kustom berdasarkan parameter fungsi pada baris pertama

Kode di atas muncul di panel kueri sebagai fungsi. Dan setiap kali Anda menjalankan fungsi tersebut, Anda hanya akan melihat hasil akhirnya. Namun, jika Anda baru mengenal fungsi kustom atau mengalami kesalahan, Anda mungkin ingin melihat cara kerja setiap langkah.

Untungnya, ada cara mudah untuk mengembalikan fungsi ke dalam kueri. Untuk melakukannya, Anda:

a. Membuat variabel untuk setiap parameter dalam definisi kueri Anda.
b. Memberi komentar pada definisi fungsi.

Seperti inilah tampilannya:

```
// ( numberList as list, minValue as number, maxValue as number ) as text =>
  let
    numberList  = { 1 .. 10 },
    minValue    = 2,
    maxValue    = 8,
    List        = List.Select ( numberList, each _ >= minValue and _ <= maxValue ),
    ListToText  = List.Transform ( List, each Text.From ( _ ) ),
    CombineText = Text.Combine ( ListToText, ", " )
  in
    CombineText
```

Gambar 1. 29 Fungsi Kustom Berubah menjadi Kueri Biasa saat Menambahkan Parameter yang Relevan

Setiap parameter yang kita definisikan di baris 1 kini memiliki definisinya sendiri dalam ekspresi let pada baris 3–5. Ini berarti bahwa sisa kueri masih dapat merujuk ke nama parameter, tetapi kali ini, kueri mengambil nilai dari dalam kode.

Dengan melakukan ini, Anda mengubah fungsi kembali menjadi kueri. Ini memungkinkan Anda untuk memeriksa setiap langkah logika secara saksama. Setelah selesai men-debug, Anda dapat mengembalikan logika ke fungsi dengan membatalkan perubahan: komentari variabel debugging dan hapus komentar pada baris definisi fungsi.

Sampai titik ini, kita telah menyimpan fungsi sebagai kueri yang berbeda. Ini dilakukan dengan menulis fungsi secara manual atau dengan mengubah kueri yang ada menjadi kueri yang berbeda. Namun, penting untuk dicatat bahwa mendefinisikan fungsi secara langsung dalam kueri juga merupakan pendekatan yang valid. Anda kemudian dapat menggunakan fungsi ini nanti dalam kueri yang sama. Bagian selanjutnya akan membahas konsep cakupan fungsi. Kita akan membahas bagaimana memahami cakupan memungkinkan Anda memanggil fungsi secara efektif dalam satu kueri.

8. Ruang Lingkup Fungsi

Mirip dengan membuat parameter dan kueri, ada cakupan tempat Anda dapat menentukan suatu fungsi. Anda dapat menentukannya sebagai:

- Ekspresi tingkat atas
- Sebaris dalam kueri

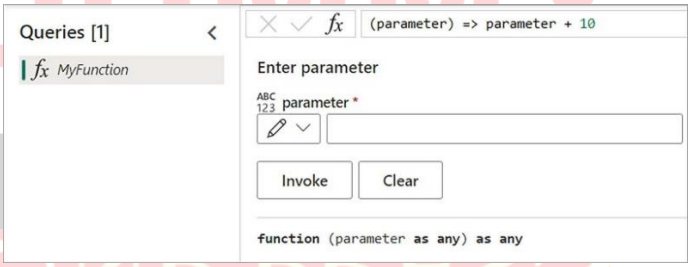
Jadi, apa perbedaan kedua cara tersebut, dan untuk apa Anda dapat menggunakannya?

a. Ekspresi Tingkat Atas

Cara kebanyakan orang menggunakan fungsi kustom adalah sebagai ekspresi tingkat atas. Untuk membuat fungsi tingkat atas, Anda dapat menempelkan kode fungsi ke Advanced editor dan menekan OK. Power Query secara

otomatis membuat fungsi dan menambahkannya di antara kueri Anda di panel Kueri. Dari sini, fungsi tersebut dapat dirujuk dari kueri atau fungsi lain.

Gambar berikut mengilustrasikan seperti apa fungsi sederhana bernama MyFunction dan bagaimana cara mendefinisikannya: Gambar berikut mengilustrasikan seperti apa fungsi sederhana bernama MyFunction dan bagaimana cara mendefinisikannya:



Gambar 1.30 Membuat fungsi khusus

Anda dapat mengenali suatu fungsi berdasarkan awalannya dengan simbol fx di panel Query. Nama apa pun yang Anda berikan pada query akan menjadi nama fungsi Anda. Misalnya, fungsi di atas disebut MyFunction. Manfaat mendefinisikan suatu fungsi pada level ini adalah Anda dapat memanggilnya dari query lain dengan merujuk pada namanya. Sejauh ini, ini juga cara kita melihat fungsi kustom di seluruh bab ini. Namun, Anda juga dapat membuat fungsi kustom dalam baris di dalam query, yang akan kita bahas sekarang.

b. Sejalan dalam Query

Menentukan fungsi dalam satu baris di dalam salah satu kueri Anda merupakan teknik yang berguna. Anggap saja seperti ini; fungsi adalah ekspresi, dan kueri Anda terdiri dari serangkaian ekspresi. Dan seperti ekspresi lainnya, Anda dapat menyertakan definisi fungsi kustom di dalam kueri Anda.

Misalnya, anggaplah Anda membuat logika berikut dalam kueri yang disebut SquareRoots:

```
let
    Value1 = 5,
    Value2 = 10,
    sqrtDivide = ( Number1,
      Number2 ) => let
        SqrtNumber1 = Number.Sqrt( Number1
        ), SqrtNumber2 = Number.Sqrt(
        Number2 ),
        Result = SqrtNumber1 / SqrtNumber2
      in
        Result,
    Applyfunction = sqrtDivide( Value1, Value2 )
in
    Applyfunction
```

Yang kita lihat di sini adalah ekspresi let seperti yang Anda ketahui dari salah satu kueri Anda. Langkah kueri ketiga mendefinisikan fungsi kustom bernama sqrtDivide. Kemudian, kita memanggil fungsi tersebut dengan memanggilnya di langkah Applyfunction. Hal ini menunjukkan bahwa Anda dapat menggunakan fungsi kustom meskipun tidak disimpan sebagai kueri tunggal.

Penting untuk diketahui bahwa fungsi yang dibuat dalam kueri hanya dapat digunakan dalam cakupan kueri tertentu. Misalnya, sqrtDivide dapat digunakan di langkah ApplyFunction karena merupakan bagian dari kueri yang sama. Jika Anda menavigasi ke kueri yang berbeda, seperti Calendar, Anda tidak akan dapat menggunakan fungsi sqrtDivide. Ini karena fungsi tersebut terbatas pada tempat pembuatannya, yaitu di luar Calendar.

Saat memutuskan cara menyiapkan fungsi Anda, pikirkan tentang cara Anda akan menggunakannya. Jika Anda perlu menggunakan fungsi kustom dalam beberapa kueri, jadikan itu ekspresi tingkat atas. Jika Anda hanya membutuhkannya dalam satu kueri, fungsi sebaris mengurangi jumlah item di panel kueri dan membuatnya tetap teratur. Pada akhirnya, keputusan ada di tangan Anda. Jadi, dengan semua pengetahuan baru yang diperoleh ini, bagaimana Anda dapat menggunakannya dalam beberapa skenario praktis? Mari kita cari tahu.

9. Menyatukan Semuanya

Sejauh ini, kita telah mempelajari berbagai cara untuk membuat fungsi kustom dan memanfaatkan parameter. Kita telah menjelajahi komponen fungsi kustom dan mempelajari bahwa ekspresi each adalah sintaksis gula untuk fungsi unary. Kita kemudian mempelajari definisi tipe data untuk parameter input dan output fungsi Anda. Dan kemudian kita melihat bagaimana membuat parameter opsional mencakup tingkat fleksibilitas tambahan.

Kita mendorong Anda untuk menerapkan konsep-konsep ini dalam proyek Anda sendiri. Setiap kali Anda membutuhkan logika di banyak tempat, cobalah untuk mencakup beberapa logika kueri Anda dalam fungsi kustom. Aplikasi praktis adalah kunci untuk memahami konsep-konsep ini. Selanjutnya, kita akan memperkuat apa yang telah kita pelajari dengan melihat beberapa contoh praktis yang menyatukan ide-ide ini.

a. Mengubah Semua Kolom menjadi Teks

Skenario pertama yang akan kita lihat adalah mengubah kolom menjadi tipe data yang diinginkan.

b. Pengantar Masalah Tipe Kolom

Untuk skenario pertama kita, kita akan melihat situasi di mana data baru mungkin masuk dari file yang tidak terstruktur. Tantangan dengan file yang tidak terstruktur seperti CSV, Excel, atau JSON adalah bahwa mereka tidak menyediakan metadata, dengan informasi tentang tipe data dalam file tersebut. Oleh karena itu, hal ini dapat menyebabkan pekerjaan berulang terkait pengaturan tipe data. Untuk mempermudah proses ini, mari kita fokus pada pembuatan fungsi kustom untuk menangani persyaratan kita. Anda dapat mengikutinya dengan menavigasi ke kumpulan data ini dalam file latihan yang menyertai buku ini.

Berikut adalah situasi yang perlu ditangani oleh fungsi kita. Anda memiliki file dengan lebih dari 20 kueri. Sebagian besar

kueri ini memiliki kolom yang berisi teks. Kolom baru ditambahkan dari waktu ke waktu, dan kolom tersebut juga berupa teks. Anda ingin menghindari pengaturan tipe kolom secara manual setiap saat. Beberapa kolom harus tetap tidak berubah dan tidak diubah menjadi teks.

Tugas Anda adalah membuat fungsi yang mengubah semua kolom dalam kueri Anda menjadi teks. Dengan fungsi tersebut, kita cukup memanggil fungsi tersebut sebagai langkah terakhir dari setiap kueri. Dan terlepas dari kolom apa pun yang ditambahkan kemudian, fungsi tersebut akan mengubah kolom Anda menjadi teks. Jadi, bagaimana kita dapat melakukannya?

c. Memahami Transformasi Tipe Data

Misalkan kita bekerja dengan tabel berikut:

Destination	Country	Popular Attractions	Best Time to Visit	Description
Paris	France	Eiffel Tower, Louvre, Notre-Dame	Spring and Fall	Known as the "City of Love," fam
Tokyo	Japan	Tokyo Disneyland, Senso-ji Temple	Spring and Autumn	A bustling metropolis with a rich
Venice	Italy	Grand Canal, St. Mark's Square	Spring and Summer	Renowned for its romantic canals
New York City	USA	Times Square, Central Park	Spring and Fall	The city that never sleeps, offerin
Sydney	Australia	Sydney Opera House, Bondi Beach	Spring and Summer	Home to iconic landmarks like th
Santorini	Greece	Oia, Fira	Summer	A stunning island in the Aegean
Cairo	Egypt	Pyramids of Giza, Egyptian Museum	Fall and Winter	Explore ancient history, includin
Rio de Janeiro	Brazil	Christ the Redeemer, Copacabana Beach	Summer	Famous for its Carnival, Christ the

Gambar 1.31 Tabel dengan Kolom Bertipe Apapun

Setiap kolom saat ini bertipe apa pun. Saat Anda mengatur dua kolom pertama untuk bertipe teks, Power Query menggunakan fungsi Table.TransformColumnTypes:

```
Table.TransformColumnTypes
  ( Source,
  {{"Destination", type text}, {"Country", type text}} )
```

Kode yang dihasilkan menunjukkan bahwa fungsi tersebut memerlukan dua hal. Pertama, fungsi tersebut memerlukan tabel sebagai titik awalnya—dalam hal ini, Sumber. Kemudian, fungsi tersebut memerlukan list luar yang memuat beberapa list dalam. Setiap list dalam berisi dua hal: nama

kolom yang akan diubah dan jenis kolom yang akan diubah. Dalam contoh ini, yaitu:

```
{{"Destination", type text}, {"Country", type text}}
```

Meskipun contoh ini menunjukkan format yang kita inginkan, formatnya tidak terlalu fleksibel. Bagaimanapun, format ini memasukkan nama kolom ke dalam ekspresi. Artinya, format ini dapat menyebabkan kesalahan saat nama kolom berubah atau dihapus, tetapi juga mengecualikan kolom baru yang mungkin ditambahkan ke data di masa mendatang.

Artinya, kode yang dibuat secara otomatis ini akan memerlukan penyesuaian manual yang konstan. Itu bukanlah pekerjaan yang ingin dilakukan oleh pengembang yang cerdas. Mari kita lihat apakah kita dapat membuat fungsi untuk ini.

d. Membangun Fungsi Awal

Untuk membuat versi pertama fungsi ini, kita akan:

1) Mengambil nama semua kolom dalam tabel dan memasukkannya ke dalam list.
2) Mengubah setiap nama kolom dalam list menjadi pasangan: nama dan tipe datanya.
3) Menggunakan list yang diperbarui ini sebagai input untuk *Table.TransformColumnTypes*.

Untuk mengambil semua nama kolom dari tabel, Anda dapat menggunakan fungsi Table.ColumnNames. Pernyataan berikut mengembalikan semua nama kolom dari tabel dalam langkah yang disebut Source:

```
Table.ColumnNames( Source )
```

Ini mengembalikan hal berikut:

```
{ "Destination", "Country", "Popular Attractions", "Best
Time to Visit", "Description" }
```

Kita akan mendapatkan list nama kolom. Idealnya, kita akan mengubah setiap nilai teks dalam list menjadi list yang berisi nama kolom dan tipe data. Untuk mencapainya, Anda dapat menggunakan fungsi List.Transform, seperti yang ditunjukkan:

```
List.Transform(
  Table.ColumnNames( Source ),
  each {_ , type text } )
```

Dalam kode ini, argumen pertama mengambil list semua nama kolom dari tabel sumber, sedangkan argumen kedua menyediakan logika untuk mengubah setiap item dalam list ke dalam format yang benar.

Garis bawah dalam argumen kedua mewakili setiap nama kolom dalam list. Ekspresi tersebut menambahkan ", ketik text" setelah nama kolom dan mengapit output dengan tanda kurung kurawal. Output dari ekspresi di atas adalah:

```
{ {"Destination", type text }
  {"Country", type text }
  {"Popular Attractions", type text }
  {"Best Time to Visit", type text }
  {"Description", type text }
}
```

Yang tersisa bagi kita adalah meneruskan argumen ini ke argumen Table.TransformColumnTypes. Sejauh ini, langkah-langkah yang kita lakukan adalah:

```
1  let
2    Source = MyTable,
3    Columns = Table.ColumnNames( Source ),
4    TransformTypes = List.Transform( Columns, each { _, type text } ),
5    TypeToText =  Table.TransformColumnTypes( Source, TransformTypes )
6  in
7    TypeToText
```

Gambar 1.32 Kueri yang mengubah Kolom menjadi Jenis Teks

Untuk mengubah kode ini menjadi fungsi, langkah pertama adalah mengidentifikasi parameter yang dibutuhkan fungsi Anda. Dalam situasi ini, fungsi tersebut bertujuan untuk mengubah semua jenis kolom menjadi teks. Oleh karena itu, Anda terutama memerlukan tabel input. Mari buat parameter untuk nama tabel dan masukkan parameter pada baris 3. Dengan melakukan hal itu, kita akan mendapatkan kode berikut:

```
1  ( InputTable as table ) as table =>
2  let
3    Source = InputTable,
4    Columns = Table.ColumnNames( Source ),
5    TransformTypes = List.Transform( Columns, each { _, type text } ),
6    TypeToText =  Table.TransformColumnTypes( Source, TransformTypes )
7  in
8    TypeToText
```

Gambar 1.33 Fungsi yang mengubah Kolom menjadi Jenis Teks

Setelah membuat versi pertama fungsi tersebut, mari kita beri nama fxToText. Sekarang, kapan pun Anda ingin mengubah semua kolom menjadi teks, yang harus Anda lakukan adalah menambahkan langkah dengan kode:

```
FxToText ( <previousStep> )
```

Menerapkan ini pada langkah Sumber mengubah semua kolom kita menjadi teks, seperti yang diilustrasikan di sini:

Gambar 1.34 Fungsi fxToText mengubah semua Kolom menjadi Teks

Ini adalah awal yang baik. Mari kita lihat apakah kita dapat menyempurnakan fungsi kita dengan fungsionalitas yang

memungkinkan Anda mengecualikan kolom dari konversi ke teks.

e. Meningkatkan Fungsi dengan Parameter Opsional

Kode di atas berfungsi, dan kini Anda dapat menggunakannya. Namun, persyaratan terakhir dari kasus ini adalah dapat mengecualikan kolom tertentu sehingga kolom tersebut tidak diubah menjadi teks. Karena fitur ini tidak akan digunakan pada setiap kueri, masuk akal untuk menjadikan parameter ini opsional. Dengan demikian, fungsi kita tidak akan memperumit penggunaan sederhana tetapi tetap dapat mencakup pengecualian bila diperlukan.

Jadi, apa yang harus kita lakukan untuk membangun persyaratan ini ke dalam fungsi?

1) Mulailah dengan menambahkan parameter opsional yang mengambil list kolom yang ingin Anda kecualikan.
2) Kemudian, ubah kode tersebut sehingga mengecualikan kolom yang ditentukan ini dari konversi.

Pertama, mari tambahkan parameter opsional bernama Exclusions. Bila suatu fungsi memerlukan beberapa item, sering kali berguna untuk memintanya dalam bentuk list. Ini memungkinkan pengguna memasukkan list secara manual atau merujuk ke kolom untuk nilai. Perbarui parameter input Anda seperti ini:

```
( InputTable as table, optional Exclusions as list ) as
table
```

Ingat apa yang terjadi jika pengguna tidak memberikan nilai untuk parameter opsional? Fungsi secara default memberikan nilai null untuk parameter ini. Agar fungsi berfungsi dengan baik, fungsi perlu menangani nilai null ini.

Untuk mengelola null, pertimbangkan bagaimana fungsi lainnya akan menggunakan Exclusions. Cara mudah untuk mengecualikan kolom tertentu adalah dengan fungsi List.RemoveItems. Anda dapat menggunakannya seperti ini:

```
List.RemoveItems( AllColumns, Exclusions )
```

Namun, jika Exclusions bernilai null, operasi List.RemoveItems akan gagal. Untuk mencegah hal ini, Anda dapat menggunakan list kosong sebagai fallback. Cara mudah untuk mencapai hal ini adalah dengan menggunakan operator Coalesce:

```
Exclusions ?? {}
```

Ekspresi ini mengembalikan list kosong setiap kali parameter Pengecualian bernilai null. Karena menghapus list kosong tidak mengubah list asli, solusi ini sesuai dengan kebutuhan kita. Menyatukan ini dalam fungsi akhir kita akan menghasilkan:

```
1  ( InputTable as table, optional Exclusions as list ) as table =>
2  let
3    Source = InputTable,
4    AllColumns = Table.ColumnNames( Source ),
5    Exclusions = Exclusions ?? {} ,
6    RelevantColumns = List.RemoveItems( AllColumns, Exclusions ),
7    TransformTypes = List.Transform( RelevantColumns, each { _, type text } ),
8    TypeToText =  Table.TransformColumnTypes( Source, TransformTypes )
9  in
10   TypeToText
```

Gambar 1.35 Menggabungkan Parameter Opsional

Karena menghapus list kosong tidak mengubah list asli, solusi ini sesuai dengan kebutuhan kita. Kita dapat menerapkannya pada tabel kita sebagai berikut:

Gambar 1.36 Fungsi yang mengubah semua Kolom kecuali Beberapa Kolom menjadi Teks

Perhatikan bagaimana kolom Destination dan Country tidak diubah menjadi teks jenis. Misi tercapai. Kita telah berhasil

membuat fungsi yang mengubah kolom Anda menjadi teks. Yang harus Anda lakukan adalah menambahkan fungsi ini sebagai langkah terakhir dari kueri Anda dan merujuk ke langkah sebelumnya. Konversi teks fungsi tersebut kemudian akan berfungsi pada kolom yang sudah ada dan yang akan datang. Dan jika Anda ingin mengecualikan kolom tertentu, Anda dapat melakukannya dengan menggunakan argumen kedua yang opsional.

Berikutnya, kita akan melihat contoh bermanfaat yang mengambil sudut pandang berbeda. Hal ini memerlukan pengetahuan tentang cara mereferensikan berbagai objek dalam tabel Anda, tetapi juga menguji pengetahuan Anda tentang cakupan.

f. Menggabungkan Tabel Berdasarkan Rentang Tanggal

Di bagian ini, kita akan mengatasi tantangan penggabungan tabel berdasarkan rentang tanggal. Kita akan menggabungkan berbagai konsep yang telah kita pelajari di seluruh bab. Anda perlu memahami setiap konstruksi dan cakupan kode yang kita gunakan. Kita akan menggunakan ini untuk membuat referensi ke berbagai nilai. Pada saat yang sama, fungsi kustom mengharuskan Anda mengetahui cara mereferensikan kolom dan bidang sehingga dapat digunakan sebagai parameter input. Data tersedia dalam file latihan yang dapat Anda temukan di halaman GitHub untuk buku tersebut.

Bayangkan Anda berurusan dengan transaksi pelanggan yang terkait dengan kontrak tertentu, tetapi tidak ada ID kontrak unik untuk menghubungkannya. Meskipun Power Query menawarkan berbagai jenis penggabungan, namun tidak cukup ketika Anda perlu menggabungkan tabel berdasarkan rentang tanggal. Kabar baiknya adalah ini adalah peluang bagus untuk mengasah keterampilan pemecahan masalah Anda, untuk mengetahui apakah Anda dapat memecahkan skenario dengan membangun fungsi kustom. Fungsi kustom kita akan memenuhi persyaratan berikut:

1) Tabel utama memiliki kolom yang berisi tanggal.
2) Tabel Kontrak mencakup tanggal mulai dan berakhir untuk setiap kontrak.
3) Tipe data semua kolom harus tetap utuh setelah penggabungan.

Kita akan mulai dengan membuat rumus untuk melakukan penggabungan khusus ini. Setelah itu, kita akan mengubahnya menjadi fungsi yang dapat digunakan kembali.

g. Melakukan Penggabungan Berdasarkan Rentang Tanggal

Untuk contoh ini, pertimbangkan kumpulan data yang disederhanakan. Kita memiliki tabel Transaksi dengan lima transaksi, masing-masing dengan tanggal yang sesuai. Ada juga tabel Kontrak, yang mencantumkan tanggal mulai dan berakhir setiap kontrak:

Table: Transactions

	Id	Date	Amount
1	SI-1401	2/7/2023	237
2	SI-1402	11/3/2023	489
3	SI-1403	9/22/2023	712
4	SI-1404	5/9/2023	56
5	SI-1405	12/29/2023	901

Table: Contract

	ContractID	Start	End
1	VI-2023-A1	1/1/2023	3/31/2023
2	VI-2023-B2	4/1/2023	6/30/2023
3	VI-2023-C3	7/1/2023	8/31/2023
4	VI-2023-D4	9/1/2023	12/31/2023

Gambar 1.37 Dataset untuk Gabungan Berbasis Rentang

Tujuan kita adalah memperkaya tabel Transaksi dengan data dari tabel Kontrak. Cara mudah untuk melakukannya adalah dengan menambahkan kolom baru dan menggunakan fungsi Table.SelectRows.

Untuk langkah pertama:

1) Buka **Add Column** di pita dan pilih **Custom Column**.
2) Beri nama kolom baru Contracts.
3) Tambahkan rumus: *Table.SelectRows(Contracts, each true)*.

Dengan pengaturan ini, seluruh tabel Contracts akan muncul di setiap baris kolom baru. Kode akan terlihat seperti ini:

```
Table.AddColumn(
  Source,
  "Contracts",
  each Table.SelectRows( Contract, each true ) )
```

Dalam kode ini, kita menambahkan kolom baru yang disebut Contracts ke tabel yang disebut Source. Kolom yang baru ditambahkan berisi salinan tabel Contracts dan menampilkan semua baris. Ini menyiapkan kita untuk menempatkan semua data kontrak dalam jangkauan untuk setiap transaksi. Karena sekarang kita memiliki akses ke data tabel Kontrak, tugas kita selanjutnya adalah memfilter tabel tersebut hingga ke tanggal tertentu.

h. Menyesuaikan Rumus untuk Tanggal Tertentu

Dalam fungsi Table.SelectRows, kondisi yang Anda tentukan hanya memengaruhi baris dalam tabel Contract. Untuk memfilter hanya kontrak yang dimulai pada 1 Juli 2023, Anda dapat menggunakan kode berikut:

```
Table.AddColumn( Source, "Contracts",
  each Table.SelectRows( Contract, each [Start] =
    #date(2023,7,1) )
)
```

Perhatikan perubahan kode pada argumen kedua fungsi Table.SelectRows. Kode tersebut menetapkan bahwa kita hanya menginginkan baris pada tanggal tertentu.

Kode di atas menunjukkan langkah yang diperlukan untuk memfilter baris, tetapi kode tersebut tetap menggunakan tanggal secara permanen. Agar fungsi kustom kita berfungsi, penting agar tanggal memfilter rentang tanggal dan melakukannya secara dinamis. Mari kita cari tahu cara melakukannya.

i. Membuat Filter Tanggal Dinamis

Berikutnya, kita bermaksud mengganti tanggal yang dikodekan secara permanen dengan referensi dinamis ke kolom Tanggal di tabel Transaksi. Ini menimbulkan masalah yang kompleks, cakupan, topik yang telah kita bahas di Bab 7, Mengkonseptualisasikan M.

Jika Anda mencoba mengganti #date(2023, 7, 1) dengan [Date], Power Query akan mengembalikan kesalahan. Ia tidak dapat menemukan bidang Tanggal dalam konteks Record saat ini.

Kuncinya terletak pada pemahaman peran kata kunci each. Contoh pertama each ada di Table.AddColumn, di mana kolom Tanggal dari tabel Transaksi masih dalam cakupan. Namun, kata kunci each lainnya muncul di Table.SelectRows. Ini menimbulkan konflik.

Ingat, each pada dasarnya adalah singkatan untuk fungsi argumen tunggal, yang direpresentasikan sebagai (_) =>. Itu berarti kode tersebut berisi dua variabel dengan garis bawah (_) sebagai namanya. Dan setiap kali Anda merujuk nama variabel yang muncul beberapa kali, variabel dalam lingkup dalam diprioritaskan daripada variabel dalam lingkup luar.

Untuk mengakses kolom internal (Start dari Contract) dan kolom eksternal (Date dari Transactions), kita perlu mengubah salah satu ekspresi each menjadi fungsi kustom kita sendiri. Ini akan memungkinkan kita untuk memberikan nama variabel yang berbeda dan, dengan itu, mengelola lingkup yang berbeda yang terlibat. Mari kita cari tahu bagaimana kita dapat memasukkan ini ke dalam kode kita.

j. Berurusan dengan Cakupan Menggunakan Fungsi Kustom

Fungsi Table.AddColumn dapat menerima parameter fungsi kustom untuk menangani masalah cakupan. Fungsi kustom ini memungkinkan akses ke tabel Transactions dan Contract:

```
Table.AddColumn( Source, "Contracts",
  (x) => Table.SelectRows( Contract, each x[Date] <=
    [Start] )
)
```

Di sini, parameter x membantu mengakses konteks luar tempat tabel Transaksi terlihat. Parameter ini tersedia dalam cakupan fungsi Table.AddColumn. Dengan menggunakan konsep yang sama, kita juga dapat menyertakan kriteria untuk tanggal akhir, yang menghasilkan:

```
Table.AddColumn( Source, "Contracts",
  (x)=> Table.SelectRows( Contract ,
    each x[Date] >= [Start] and x[Date] <= [End] )
)
```

Kondisi penyaringan sekarang merujuk ke kolom x[Tanggal], yang merujuk ke kolom Tanggal di tabel Transaksi. Bagus, jadi masalah cakupan terpecahkan di sini. Namun, saat kita memperluas kolom setelah melakukan operasi di atas, kita kehilangan tipe data. Mari kita lihat bagaimana kita dapat menjaganya tetap utuh.

k. Mengatasi Masalah Tipe Data

Sekarang kita memiliki baris yang difilter yang relevan di setiap baris kolom baru. Namun, masih ada satu masalah yang tersisa: kolom tersebut tidak memiliki tipe data yang ditentukan, seperti yang ditunjukkan pada gambar berikut:

	Id	Date	Amount	Contracts
1	SI-1401	2/7/2023	237	[Table]
2	SI-1402	11/3/2023	489	[Table]
3	SI-1403	9/22/2023	712	[Table]
4	SI-1404	5/9/2023	56	[Table]
5	SI-1405	12/29/2023	901	[Table]

Gambar 1.38 Kolom Gabungan tidak memiliki Tipe Data

Apa yang dapat kita lakukan untuk menentukan tipe data yang benar di sini? Memasukkannya secara manual akan

menghilangkan tujuan fungsi kustom. Namun, tabel Contracts sudah memiliki tipe data yang benar. Mungkin kita dapat merujuknya dan menyediakannya ke kolom baru?

Mengambil tipe data dari tabel lain menggunakan fungsi Value.Type adalah solusi paling sederhana. Anda dapat menggunakan fungsi tersebut untuk sekadar merujuk ke tabel Contracts dan mengambil definisi tipe data.

Kode, yang juga menetapkan tipe data untuk kolom baru, terlihat seperti ini:

```
Table.AddColumn( Source, "Contracts",
  (x)=> Table.SelectRows( Contract ,
    each x[Date] >= [Start] and x[Date] <= [End] ),
  Value.Type( Contract )
)
```

Contoh di atas sekarang menyertakan argumen keempat dari Table.AddColumn. Argumen ini secara dinamis mengambil tipe data dari tabel kontrak. Sekarang, kolom baru tidak hanya memfilter data dengan benar tetapi juga membawa tipe data yang sesuai dari tabel Kontrak.

Dengan logika ini, kueri kita melakukan penggabungan dengan semua persyaratan kita. Selanjutnya, Anda perlu mengubah logika ini menjadi sebuah fungsi.

l. Mengubah Logika menjadi Fungsi

Agar kode kita dapat digunakan kembali, kita akan mengubahnya menjadi fungsi kustom. Tujuannya adalah agar menyerupai fungsi Table.NestedJoin. Awalnya, kita akan memperkenalkan dua parameter:

1) Tabel untuk menambahkan kolom (Tabel)
2) Kolom yang akan digunakan untuk perbandingan tanggal (Tanggal)

Seperti yang dibahas sebelumnya dalam bab ini, dengan menggunakan fungsi Record.Field, kita dapat mengambil nilai

teks sebagai input untuk kolom tanggal dan tetap merujuk ke kolom tersebut. Menggabungkan ini ke dalam kode M kita terlihat seperti berikut:

```
( Table as table, Date as text ) as table =>
Table.AddColumn(
  Table, "Contracts",
  (x)=> Table.SelectRows( Contract,
    each Record.Field(x, Date) >= [Start] and
     Record.Field(x, Date) <= [End] ),
  Value.Type( Contract )
)
```

Meskipun kodenya menjadi lebih bertele-tele, kode tersebut berhasil merujuk ke kolom tanggal dari tabel transaksi Anda. Selanjutnya, kita tambahkan parameter untuk:

1) Tabel yang akan digabungkan (*joinTable*)
2) Nama kolom gabungan baru (*newColumnName*)
3) Tanggal mulai dan berakhir yang digunakan dalam penggabungan (*startDate* dan *endDate*)

Untuk melakukannya, kita perluas list parameter dan masukkan referensi parameter:

```
(Table as table, Date as text, joinTable as table,
startDate as text, endDate as text, newColumnName as
text) as table =>
Table.AddColumn( Table, newColumnName, (x)=>
  Table.SelectRows( joinTable, each
    Record.Field(x, Date) >= Record.Field(_, startDate)
    and Record.Field(x, Date) <= Record.Field(_, endDate)
    ),
  Value.Type( joinTable )
)
```

Anda dapat melihat bagaimana parameter joinTable menggantikan tabel Contract yang dikodekan secara permanen, dan newColumnName menggantikan nama yang dikodekan secara permanen. Terakhir, startDate dan endDate kini juga menggunakan Record.Field untuk merujuk ke kolom.

Sekarang kita memiliki fungsi kustom yang lengkap—fxDateRangeJoin. Saat Anda menerapkannya ke tabel Transactions, Anda akan berakhir dalam situasi berikut:

```
fxDateRangeJoin( Source, "Date", Contract, "Start", "End", "Contracts" )
```

	Id	Date	Amount	Contracts
1	SI-1401	2/7/2023	237	[Table]
2	SI-1402	11/3/2023	489	[Table]
3	SI-1403	9/22/2023	712	[Table]
4	SI-1404	5/9/2023	56	[Table]
5	SI-1405	12/29/2023	901	[Table]

Gambar 1.39 Fungsi Kustom untuk Gabungan Rentang Tanggal

Yang tersisa bagi kita adalah memperluas kolom Kontrak dan menyertakan tiga kolom dari tabel lainnya. Dengan begitu, kita akan mendapatkan:

	Id	Date	Amount	ContractID	Start	End
1	SI-1401	2/7/2023	237	SI-1401	1/1/2023	3/31/2023
2	SI-1402	11/3/2023	489	SI-1404	9/1/2023	12/31/2023
3	SI-1403	9/22/2023	712	SI-1404	9/1/2023	12/31/2023
4	SI-1404	5/9/2023	56	SI-1402	4/1/2023	6/30/2023
5	SI-1405	12/29/2023	901	SI-1404	9/1/2023	12/31/2023

Gambar 1.40 Memperluas Bidang Aker melakukan Penggabungan Rentang Tanggal untuk menyertakan Bidang yang Relevan

Dan selesai—kita telah mencapai tujuan kita. Mulai sekarang, kapan pun Anda perlu menggabungkan tabel berdasarkan rentang tanggal, alih-alih rumus yang rumit, Anda dapat menggunakan fungsi yang disederhanakan ini.

Seperti yang telah Anda lihat melalui kedua proyek ini, mengubah kueri menjadi fungsi kustom yang tangguh merupakan proses berulang. Anda mulai dengan membangun logika kueri Anda. Setelah puas, Anda menganalisis bagian mana yang harus dinamis. Dengan mengingat hal itu, Anda memperkenalkan parameter dan menggabungkannya di seluruh kode Anda. Proses ini mungkin menakutkan pada awalnya, tetapi dengan pengalaman, ini menjadi konsep yang sangat ampuh untuk digunakan dalam kueri Anda.

D. Ringkasan

Bab ini memperkenalkan parameter. Seperti yang telah Anda lihat, parameter lebih dari sekadar placeholder di Power Query. Parameter memberi Anda fleksibilitas dan memenuhi berbagai skenario. Saat Anda maju dalam perjalanan Power Query, Anda akan menemukan diri Anda semakin sering menggunakan parameter untuk membuat kueri Anda lebih mudah disesuaikan.

Kita juga menyelami lebih dalam dunia fungsi kustom di Power Query M. Fungsi tersebut memungkinkan Anda mengubah kueri kustom menjadi logika yang dapat digunakan kembali. Anda mempelajari cara mengubah kueri menjadi fungsi tanpa harus menuliskannya sendiri. Kita kemudian membahas seluk-beluk pemanggilan fungsi dan menyoroti peran ekspresi each dalam menyederhanakan pembuatan fungsi.

E. Daftar Pustaka

David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon (1989), Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. Operations Research, 1989, vol. 37, issue 6, 865-892 – http://dx.doi.org/10.1287/ opre.37.6.865

L. Breiman (1996), Stacked regressions. Mach Learn 24, 49–64 – https://doi.org/10.1007/ BF00117832

Mark J. van der Laan; Eric C.Polley; and Alan E.Hubbard (2007), Super Learner. U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 222: https://biostats. bepress.com/ucbbiostat/paper222

F. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan apa yang dimaksud dengan parameter dalam *Power Query*. Bagaimana parameter ini digunakan untuk membuat transformasi data menjadi lebih dinamis dan fleksibel?	10
2.	Apa yang dimaksud dengan fungsi khusus	10

No	Soal	Bobot
	(custom function) dalam *Power Query* M? Jelaskan kelebihan menggunakan fungsi khusus dibandingkan menulis ulang logika pada setiap query.	
3.	Uraikan langkah-langkah dalam mengubah query menjadi fungsi khusus. Sertakan penjelasan tentang bagaimana parameter dimasukkan ke dalam fungsi tersebut.	10
4.	Bagaimana cara memanggil fungsi khusus dalam *Power Query*? Berikan contoh penerapannya dalam sebuah skenario transformasi data.	10
5.	Jelaskan peran ekspresi each dalam *Power Query* M. Mengapa ekspresi ini sering digunakan dalam pembuatan fungsi?	10
6.	Apa saja kesalahan umum yang terjadi saat membuat atau menggunakan fungsi khusus di *Power Query*? Bagaimana cara men-debug dan memperbaiki kesalahan tersebut?	10
7.	Bagaimana menyempurnakan definisi fungsi khusus agar lebih tangguh? Diskusikan penggunaan tipe data yang dapat berupa null (nullable types) sebagai bagian dari strategi penguatan fungsi.	10
8.	Apa pentingnya memahami ruang lingkup (scope) dalam konteks fungsi khusus *Power Query*? Jelaskan dengan contoh penerapan praktis.	10
9.	Berikan contoh penerapan parameter dalam mengatur informasi koneksi atau nilai pajak (VAT) dalam transformasi data. Apa keuntungan dari pendekatan ini?	10
10.	Analisislah bagaimana penggunaan fungsi khusus dapat meningkatkan efisiensi dan keterbacaan kode dalam proyek transformasi data berskala besar.	10

BAB 2
Berurusan dengan Tanggal, Waktu, dan Durasi

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Tanggal
- Waktu
- Tanggal dan Waktu
- Zona Waktu
- Durasi

A. Pendahuluan

Mengingat pentingnya tanggal dalam pelaporan, memahami cara menangani tanggal, waktu, dan durasi di Power Query merupakan keterampilan penting yang harus dikuasai. Memang, hampir setiap model data Power BI memiliki, atau seharusnya memiliki, dimensi tabel tanggal. Hal ini karena meskipun mengetahui nilai KPI atau metrik itu penting, jauh lebih penting untuk mengetahui nilai KPI atau metrik tersebut pada tanggal atau waktu tertentu dan bahkan lebih penting lagi untuk memahami tren KPI atau metrik tersebut selama rentang hari, bulan, atau bahkan tahun.

Untungnya, bahasa M mencakup lebih dari 100 fungsi standar untuk menangani tanggal, waktu, durasi, kalkulasi dan transformasi tanggal/waktu, dan tanggal/waktu/zona waktu. Dengan fungsi-fungsi ini, relatif mudah untuk melakukan sebagian besar kalkulasi temporal standar. Dalam bab ini, kita akan menjelajahi banyak fungsi ini serta aplikasi praktis yang akan meningkatkan kedalaman pengetahuan dan pemahaman Anda tentang bahasa M. Secara khusus, bab ini membahas topik-topik berikut: Tanggal, Waktu, Tanggal dan waktu, Zona waktu, Durasi.

1. Kasus Pemantik Berfikir Kritis: Analisis Keterlambatan Pengiriman

Sebuah toko online ingin mengevaluasi efisiensi layanan pengirimannya. Data transaksi yang mereka miliki mencakup

tanggal pemesanan, tanggal pengiriman, dan jam pengiriman tiba. Namun, data tersebut berasal dari berbagai sistem dan masih dalam format teks yang berbeda-beda.

Tim analis data ditugaskan untuk menggunakan Power Query untuk membersihkan dan menganalisis data tersebut.

2. Pertanyaan Pemantik

a. Bagaimana Anda mengonversi semua data tanggal dan waktu tersebut ke dalam format yang dapat digunakan untuk perhitungan durasi?
b. Langkah apa yang akan Anda ambil untuk menghitung durasi keterlambatan antara tanggal pemesanan dan waktu pengiriman tiba?
c. Apa saja tantangan yang mungkin muncul dalam proses manipulasi data tanggal dan waktu, dan bagaimana Anda mengatasinya?
d. Jika ternyata ada pengiriman yang memiliki nilai waktu tiba lebih awal dari waktu pemesanan, apa yang mungkin menjadi penyebabnya dan bagaimana solusinya?
e. Bagaimana Anda menyajikan hasil analisis ini ke dalam visualisasi sederhana yang bisa dimengerti manajemen?
f. Dalam kasus pengolahan data absensi karyawan, bagaimana Anda menggunakan *Power Query* untuk menghitung jumlah jam kerja harian dari waktu masuk dan keluar?
g. Jika data tanggal yang Anda terima memiliki format berbeda-beda (misalnya, DD/MM/YYYY dan MM-DD-YYYY), bagaimana strategi Anda untuk menstandarkan format tersebut sebelum dianalisis?
h. Bagaimana Anda menangani data waktu yang mengandung zona waktu berbeda (misalnya UTC dan waktu lokal)? Mengapa penting untuk mempertimbangkannya dalam analisis?
i. Dalam analisis tren penjualan mingguan, bagaimana Anda bisa memanfaatkan fungsi tanggal untuk mengelompokkan data berdasarkan minggu tertentu?

j. Bagaimana Anda dapat menggunakan fungsi *Duration.Days* untuk mengidentifikasi pelanggan yang mengalami keterlambatan pengiriman lebih dari 7 hari? Apa dampak analisis ini terhadap kebijakan pelayanan?

B. Persyaratan Teknis

Editor Power Query dalam Power BI Desktop dapat digunakan untuk melengkapi contoh-contoh dalam bab ini. Untuk tinjauan penggunaan editor Power Query, lihat Bab 2.

C. Tanggal

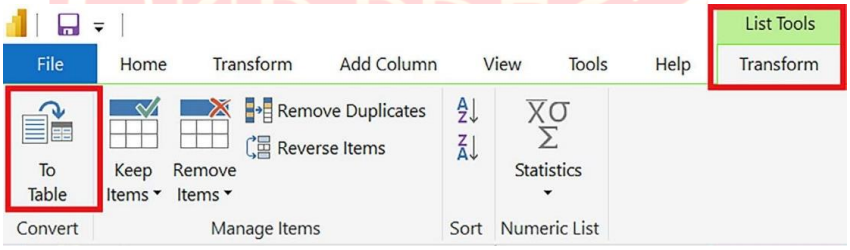
Tanggal merupakan tulang punggung analisis temporal, yang menjadi fondasi bagi kita untuk menghasilkan wawasan yang bermakna tentang data. Ada hampir 60 juta fungsi khusus untuk bekerja dengan tanggal. Namun, tanggal memiliki rahasia – tanggal sebenarnya hanyalah angka yang mewakili jumlah hari sebelum atau sesudah tanggal referensi tertentu.

Kita dapat menunjukkannya dengan contoh sederhana dengan membuat kueri kosong dan menggunakan editor tingkat lanjut dengan kode berikut:

1. Buat kueri berikut di editor *Power Query*:

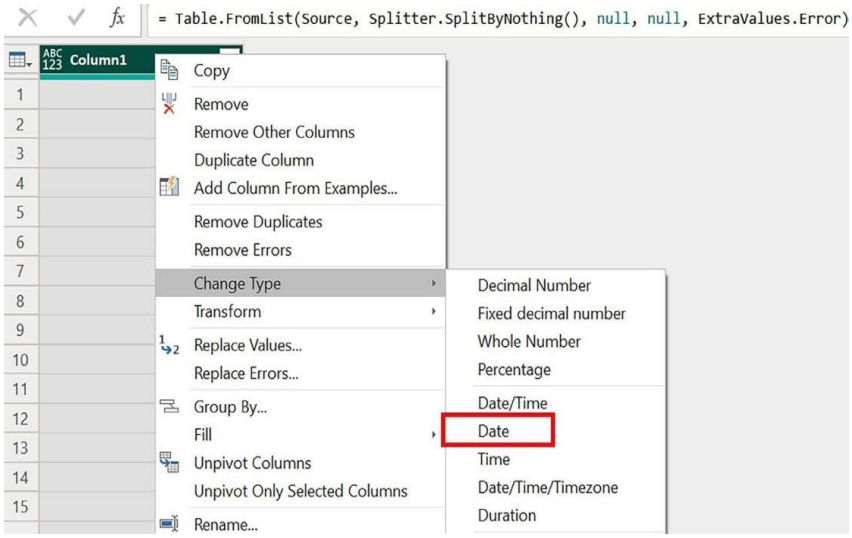
```
let
    Source = List.Generate(() => -10, each _ <= 10, each _ +
    1)
in
    Source
```

2. Pada tab List Tools | Transform pada pita, klik tombol To Table:



Gambar 2.1 Ubah List menjadi Tabel

3. Pada dialog To Table, cukup klik tombol OK.
4. Klik kanan header untuk Column1 dan pilih Duplicate Column.
5. Klik kanan header untuk Column1, pilih Change Type, lalu Date:



Gambar 2.2 Ubah Kolom menjadi Tanggal

Setelah kolom diubah menjadi tanggal, kita dapat melihat bahwa angka 0 sesuai dengan tanggal referensi 12/30/1899. Setiap penambahan positif atau negatif sebesar 1 akan menambah atau mengurangi tanggal tersebut satu hari, berturut-turut.

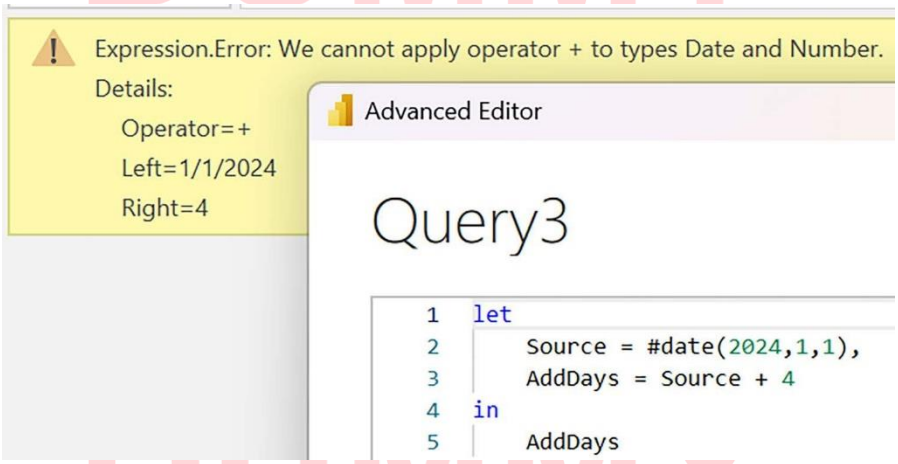
Ada batasan pada kemampuan M untuk menangani tanggal tertentu. Misalnya, tanggal sebelum 1/1/1 dan setelah 12/31/9999 tidak mungkin direpresentasikan dalam M. Mungkin anehnya, mengubah dari angka menjadi tanggal memiliki batasan yang berbeda. Sementara tanggal maksimum yang sama yaitu 12/31/9999 berlaku (2.958.465), angka minimum yang dapat diubah adalah -657.434, yang sesuai dengan tanggal 1/1/100.

Batasan ini tampak sangat sewenang-wenang, karena tidak sesuai dengan hal-hal seperti jumlah byte untuk menyimpan angka, juga tidak ada batasan yang seimbang antara hari positif dan negatif setelah atau sebelum tanggal referensi. Namun, untuk sebagian besar aplikasi praktis, batasan ini tidak menjadi masalah. Meski demikian, keterbatasan ini dapat memengaruhi penghitungan berbagai hal

seperti peristiwa astronomi masa lalu dan masa depan, seperti titik balik matahari dan ekuinoks (lihat contoh hari Julian di bagian ini).

Fakta bahwa tanggal direpresentasikan sebagai angka mungkin tidak sepenting dalam M seperti, misalnya, DAX. Dalam DAX, fakta bahwa tanggal adalah angka memungkinkan berbagai operasi aritmatika, seperti menggunakan penjumlahan atau pengurangan sederhana untuk menghitung tanggal.

Namun, di M, jika kita mencoba melakukan ini, kita akan menerima kesalahan, seperti yang ditunjukkan pada Gambar 2.3:



Gambar 2.3 Kesalahan saat menambahkan nomor ke tanggal

Sebaliknya, kita mesti mengonversi tanggal secara spesifik menjadi angka dan angka menjadi tanggal, dengan menggunakan kode berikut dalam kueri baru (kosong):

```
let
    Source = #date(2024,1,1),
    AddDays = Number.From(Source) + 4, NewDate =
    Date.From(AddDays)
in
    NewDate
```

Kode ini mengembalikan tanggal 5 Januari 2024. Tanggal ini mungkin muncul sebagai 1/5/2024 atau 5/1/2024, tergantung pada pengaturan budaya Anda.

Pertama-tama kita buat tanggal menggunakan fungsi #date, yang mengambil tahun, bulan, dan hari sebagai parameter pertama, kedua, dan ketiga. Fungsi Number.From digunakan untuk mengubah tipe data tanggal menjadi angka, lalu fungsi Date.From digunakan untuk mengubah angka kembali menjadi tipe data tanggal. Fungsi Date.From mengambil nilai text, datetime, datetimezone, atau angka apa pun sebagai parameter pertamanya dan mencoba mengonversi nilai tersebut menjadi tanggal. Jika tipe data lain disediakan atau fungsi Date.From tidak dapat mengurai nilai dan menentukan tanggal, kesalahan akan ditampilkan.

Parameter kedua opsional, Culture, memungkinkan spesifikasi budaya saat mengonversi nilai menjadi tanggal. Misalnya, jika Anda menjalankan Power BI Desktop di komputer dengan budaya lokal Bahasa Inggris Amerika Serikat, en-US, dan menerima tanggal sebagai teks dalam budaya Inggris Raya, en-GB, maka ekspresi berikut dapat digunakan untuk mengurai tanggal dengan benar dan mengembalikan 14 Februari 2024 (14/2/2024):

```
Date.From("14/2/2024", "en-GB")
```

Operasi aritmatika tertentu diperbolehkan. Misalnya, perhatikan kode berikut:

```
let
    Date1 = #date(2024, 1, 1),
    Date2 = #date(2024, 1, 5),
    Diff = Date2 - Date1
in
    Diff
```

Ini mengembalikan tipe Duration dengan nilai 4.00:00:00 atau, pada dasarnya, 4 hari.

Selanjutnya mari kita lihat beberapa fungsi yang tersedia untuk tanggal dalam M, dimulai dengan kueri sederhana untuk menampilkan list tanggal untuk tahun 2024, 2025, dan 2026:

```
let
    Source = List.Generate( () => #date(2024, 1, 1), each _ <=
#date(2026, 12, 31), each Date.AddDays( _, 1)),
    Table = Table.FromList(Source, Splitter.SplitByNothing(), null,
null, ExtraValues.Error),
RenameColumn1 = Table.RenameColumns(Table,{{"Column1", "Date"}}) in
    RenameColumn1
```

Di sini, kita melihat bahwa dalam pernyataan Source, kita menggunakan fungsi Date.AddDays dalam fungsi List.Generate untuk menambah tanggal sebesar 1 hingga mencapai tanggal 31 Desember 2026. Ada juga fungsi terkait untuk satuan temporal standar lainnya, termasuk Date.AddWeeks, Date.AddMonths, Date.AddQuarters, dan Date.AddYears.

Ada sejumlah fungsi untuk mengekstrak komponen tanggal, seperti yang ditunjukkan dalam kode berikut, yang dapat kita tambahkan ke kueri asli tanggal untuk tahun 2024–2026:

```
AddYearColumn = Table.AddColumn( RenameColumn1, "Year", each Date.
Year([Date])),
AddMonthColumn = Table.AddColumn( AddYearColumn, "Month", each
Date. Month([Date])),
AddDayColumn = Table.AddColumn( AddMonthColumn, "Day", each Date.
Day([Date]))
```

Anda dapat mengubah kode sebelumnya dan menambahkan kode ini dengan melakukan hal berikut:

1. Tambahkan koma di akhir baris terakhir di atas kata kunci in.
2. Tempel tiga baris kode setelah koma yang baru saja Anda tambahkan dan sebelum kata kunci *in*.
3. Ubah pernyataan setelah kata kunci *in* dari *RenameColumn1* menjadi *AddDayColumn*.

Di sini, fungsi Date.Year, Date.Month, dan Date.Day digunakan untuk mengekstrak komponen tanggal masing-masing. Cara lain untuk mengekstrak komponen tanggal adalah melalui fungsi Date.ToRecord. Misalnya, ekspresi berikut mengembalikan Record yang menyertakan pasangan bidang/nilai untuk tahun, bulan, dan tanggal:

```
Date.ToRecord(#date(2024, 2, 14))
```

Ada juga fungsi untuk menyimpulkan informasi tambahan dari suatu tanggal, seperti kuartal dalam setahun, minggu dalam setahun, minggu dalam sebulan, nama hari, dan seterusnya:

```
    AddQuarterColumn = Table.AddColumn( AddDayColumn, "Quarter", each
    Date.
QuarterOfYear([Date])),
    AddWeekOfYearColumn = Table.AddColumn( AddQuarterColumn, "Week of
Year", each Date.WeekOfYear([Date])),
    AddWeekOfMonthColumn = Table.AddColumn( AddWeekOfYearColumn, "Week
of Month", each Date.WeekOfMonth([Date])),
    AddDayOfWeekColumn = Table.AddColumn( AddWeekOfMonthColumn, "Day of
Week", each Date.DayOfWeek([Date])),
    AddDayOfWeekNameColumn = Table.AddColumn( AddDayOfWeekColumn, "Day
of Week Name", each Date.DayOfWeekName([Date])),
    AddMonthNameColumn = Table.AddColumn( AddDayOfWeekNameColumn, "Month
Name", each Date.MonthName([Date])),
    AddDaysInMonthColumn = Table.AddColumn( AddMonthNameColumn, "Days In
Month", each Date.DaysInMonth([Date])),
    AddLeapYearColumn = Table.AddColumn( AddDaysInMonthColumn, "Is Leap
Year", each Date.IsLeapYear([Date]))
```

Anda dapat mengubah kode sebelumnya dan menambahkan kode ini dengan melakukan hal berikut:

1. Tambahkan koma di akhir baris terakhir di atas kata kunci in. Kode ini mengasumsikan bahwa langkah terakhir sebelum kata kunci in adalah langkah yang disebut *AddDayColumn*.
2. Tempel delapan baris kode setelah koma yang baru saja Anda tambahkan dan sebelum kata kunci in.
3. Ubah pernyataan setelah kata kunci in menjadi *AddLeapYearColumn*.

Fungsi Date.StartOf dan Date.EndOf juga disertakan dalam pustaka M standar. Fungsi M tersedia untuk masing-masing keluarga ini untuk satuan temporal standar hari, minggu, bulan, kuartal, dan tahun dan mengembalikan nilai datetime, yang masing-masing menunjukkan awal atau akhir satuan temporal. Misalnya, kueri berikut mengembalikan 2/29/2024:

```
let
    Source = Date.EndOfMonth(#date(2024, 2, 14))
in
    Source
```

Di sini, fungsi Date.EndOfMonth digunakan untuk mengambil tanggal akhir bulan untuk tanggal 14 Februari 2024.

Keluarga fungsi Date.StartOf dan Date.EndOf mungkin agak aneh karena output-nya bergantung pada tipe data yang dimasukkan ke dalam fungsi tersebut. Melewatkan tipe data date, datetime, atau datetimezone akan menghasilkan tipe data yang sesuai sebagai output. Misalnya, mari kita ubah baris Source untuk kueri tersebut menjadi:

```
Source = Date.EndOfMonth(#datetime(2024, 2, 14, 0, 0, 0))
```

Ini menghasilkan output *2024-02-29T23:59:59.9999999*, tipe data *datetime*. Sekarang mari kita ubah baris *Source* menjadi:

```
Source = Date.EndOfMonth(#datetimezone(2024, 2, 14, 0, 0 , 0, -
5, 0))
```

Ini menghasilkan output 2024-02-29T23:59:59.9999999-05:00, tipe data datetimezone dengan zona waktu Eastern Standard Time (EST) atau -5 Coordinated Universal Time (UTC).

Tiga keluarga fungsi tambahan tersedia untuk tipe data tanggal: IsInCurrent, IsInNext, dan IsInPrevious. Sekali lagi, masing-masing keluarga ini menyertakan fungsi untuk satuan temporal standar hari, minggu, bulan, kuartal, dan tahun.

Fungsi-fungsi ini mengembalikan output logis tergantung pada apakah tanggal input, misalnya, berada dalam tahun berjalan untuk fungsi IsInCurrentYear. Penentuan ini didasarkan pada tanggal dan waktu saat ini pada sistem, dan dengan demikian, hasil yang berbeda dapat dikembalikan saat menjalankan Power BI Desktop pada komputer lokal dibandingkan dalam layanan Power BI, dengan penyewa yang ada di zona waktu yang berbeda.

Keluarga fungsi IsInNext dan IsInPrevious mencakup varian N tambahan, seperti IsInNextNDays, IsInNextNWeeks, IsInNextNMonths, IsInNextNQuarters, dan IsInNextNYears. Selain parameter pertama dari tipe data tanggal, fungsi-fungsi ini mencakup parameter kedua yang menentukan jumlah unit temporal (positif

atau negatif) untuk menentukan apakah tanggal berada dalam rentang tanggal yang ditentukan, seperti dalam 2 minggu ke depan: Date.IsInNextNWeeks(#date(2024, 2, 14), 2).

Fungsi IsIn tambahan untuk tahun berjalan hingga saat ini tersedia, Date.IsInYearToDate. Fungsi ini mengembalikan true atau false, tergantung pada apakah tanggal yang dimasukkan ke dalam fungsi berada dalam tahun berjalan hingga saat ini.

Terakhir, ada fungsi Date.ToText. Seperti yang mungkin diharapkan, fungsi ini mengonversi tipe data date ke tipe data text. Namun, fungsi ini menyertakan dua argumen opsional, parameter Format dan parameter Culture. Rick de Groot menyimpan list kode berbeda yang dapat digunakan dalam parameter Format di sini: https://Powerquery.how/date-totext/. Tabel 2.1 menampilkan tabel kode format untuk tanggal dari situs web Powerquery.how:

Tabel 2.1 Kode Format untuk Tipe Data Tanggal

Format	Deskripsi	December 31, 2023	February 1, 2003
%d	Hari Satu Digit (1–31)	31	1
dd	Hari Dua Digit (01–31)	31	01
ddd	Nama Hari Kerja Singkat	Sun	Sat
dddd	Nama Hari Kerja Lengkap	Sunday	Saturday
%M	Bulan Satu Digit (1–12)	12	2
MM	Bulan Dua Digit (01–12)	12	02
MMM	Nama Bulan Pendek	Dec	Feb
MMMM	Nama Bulan Lengkap	December	February
%y	Year (0–99)	23	3
yy	Year (00–99)	23	03
yyy	Tahun dengan setidaknya tiga digit	2023	2003
yyyy	Tahun Empat Digit	2023	2003
yyyyy	Tahun Lima Digit	02023	02003
m, M	Hari diikuti oleh Nama Bulan Lengkap	December 31	February 1
y, Y	Tanggal Panjang Standar	December 2023	February 2003
d	Tanggal Singkat Standar	12/31/2023	2/1/2003
D	Tanggal Panjang Penuh	Sunday, December 31, 2023	Saturday, February 1, 2003
%g, gg	Periode Sebuah Era	A.D.	A.D.

Jadi, menggunakan Date.ToText, seperti dalam ekspresi berikut:

```
Date.ToText( #date(2024, 2, 14), "dddd, MMMM %d yyyy gg")
```

Kembali pada hari Rabu, 14 Februari 2024 M. Hal ini dapat bervariasi tergantung pada pengaturan Budaya Anda. Parameter Budaya dapat memengaruhi hal ini lebih lanjut. Misalnya, ekspresi berikut:

```
Date.ToText( #date(2024, 2, 14), "dddd, MMMM %d yyyy gg", "uk")
```

Seperti yang mungkin diharapkan, fungsi Date.FromText pada dasarnya melakukan kebalikan dari fungsi Date.ToText, mengubah representasi teks suatu tanggal menjadi tipe data date.

Meskipun seseorang mungkin berpikir bahwa *Date.FromText* hanyalah sintaksis yang rumit untuk Date.From, fungsi *Date.FromText* mendukung parameter Format dan Culture yang sama seperti *Date.ToText*.

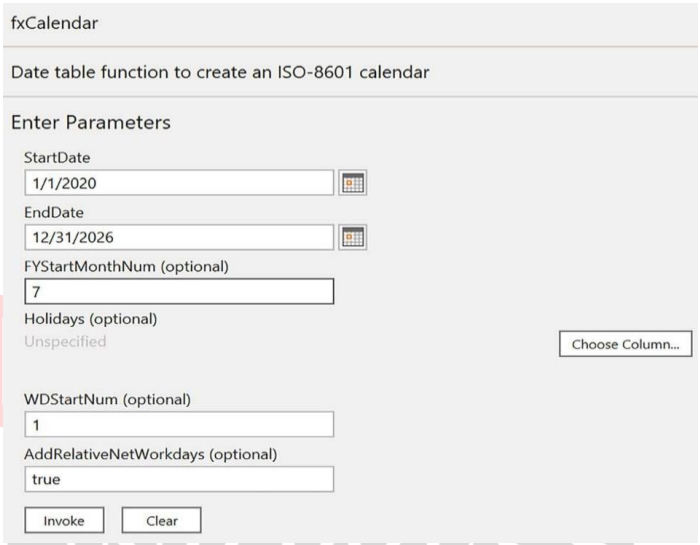
Ini melengkapi penjelajahan kita terhadap tanggal dalam *Power Query* M. Sekarang kita lanjut ke penjelajahan waktu.

1. M Tabel Kalender

Seperti yang disebutkan sebelumnya dalam bab ini, hampir semua model data *Power* BI memiliki atau seharusnya memiliki tabel dimensi tanggal. Sering kali, tabel dimensi tanggal ini berasal dari gudang data. Namun, dalam banyak keadaan, tidak ada gudang data tempat mengambil tabel dimensi tanggal.

Dalam kasus ini, fungsi **Extended Date Table M** yang terkenal dari Melissa de Korte dapat digunakan. Kode untuk fungsi ini terlalu panjang untuk disertakan sebagai teks dalam buku, tetapi disertakan dalam repositori *GitHub* untuk buku ini baik dalam file teks maupun dalam file PBIX untuk bab ini. Kode tersebut juga tersedia di tautan berikut: https://bit.ly/484rXxi.

Untuk menggunakan Tabel Tanggal Diperpanjang, cukup buat kueri kosong di editor *Power Query*. Buka kueri dengan editor tingkat lanjut, hapus semua kode yang ada, dan tempel kode untuk Tabel Tanggal Diperpanjang. Setelah dibuat, fungsi tersebut menampilkan yang berikut:



Gambar 2.4 Tabel Tanggal Diperpanjang

Fungsi ini memerlukan dua parameter, *StartDate* dan *EndDate*, seperti yang ditunjukkan pada Gambar 2.4. Selain itu, ada empat parameter opsional:

a. *FYStartMonthNum*: Nomor bulan dimulainya tahun fiskal; Januari jika dihilangkan
b. *Holidays*: Pilih kueri (dan kolom) yang berisi list tanggal hari libur
c. *WDStartNum*: Ganti penomoran hari kerja *default* dari 0–6 ke 1–7 dengan memasukkan angka 1
d. *AddRelativeNetWorkdays*: Jika benar, tambahkan kolom Relative *Networkdays* ke tabel tanggal

Jadi, Gambar 10.4 menampilkan konfigurasi untuk membuat tabel tanggal antara tahun 2020 dan 2026, di mana tahun fiskal dimulai pada bulan Juli, hari kerja diberi nomor 1–7, dan kolom Relative Networkdays akan disertakan dalam tabel tanggal. Selain itu, tidak ada kolom Holidays yang ditentukan.

Menekan Invoke pada konfigurasi ini akan menciptakan kueri Fungsi yang Dipanggil, berisi ekspresi berikut:

```
let
    Source = fxCalendar(#date(2020, 1, 1), #date(2026, 12,
31), 7, null, 1, true)
in
    Source
```

Harap perhatikan bahwa dalam kasus ini, kueri yang memuat fungsi tabel tanggal Melissa de Korte diberi nama *fxCalendar*, seperti yang ditunjukkan pada Gambar 10.4. Jika Anda memberi nama kueri dengan nama yang berbeda, ubah *fxCalendar* menjadi nama kueri Anda.

Kueri ini dapat diubah namanya menjadi Tanggal atau Kalender jika diinginkan. 61 kolom dibuat untuk tanggal yang mencakup 1/1/2020 hingga 12/31/2026. Penting juga untuk memperhatikan hal berikut:

a. Kolom *Fiscal Week* dimulai pada hari Senin dan dapat memuat kurang dari 7 hari di Minggu Pertama dan/atau Minggu Terakhir tahun anggaran.
b. Kolom *IsWeekDay* tidak memperhitungkan tanggal hari libur.
c. Kolom *IsBusinessDay* memperhitungkan tanggal hari libur opsional.
d. Kolom *IsPYTD* dan *IsPFYTD* membandingkan kolom Hari dalam Setahun sebelumnya dengan nomor kolom *Day* dalam *Year* saat ini, sehingga tanggal tidak selaras pada tahun kabisat.
e. Tidak ada kolom Fiscal yang akan ditambahkan jika parameter *FYStartMonthNum* dihilangkan.

Salah satu fitur luar biasa dari tabel tanggal ini adalah penerapan offset seperti kolom *CurrYearOffset*, *CurrQuarterOffset*, *CurrMonthOffset*, *CurrWeekOffset*, dan *CurrDayOffset*. Kolom-kolom ini membuat kalkulasi kecerdasan tanggal/waktu dalam DAX jauh lebih mudah seperti yang ditunjukkan dalam artikel blog yang luar biasa ini, Kecerdasan

Waktu dalam DAX: Cara Memilih Periode Awal Secara Dinamis, oleh Brian Julius: *https:// bit.ly/47OBQ1W*.

2. Format Tanggal Lainnya

Tanggal-tanggal yang dibahas di sini digunakan di sebagian besar belahan dunia dan didasarkan pada apa yang dikenal sebagai kalender Gregorian. Kalender Gregorian diperkenalkan pada tahun 1582 oleh Paus Gregorius XIII sebagai pengganti kalender Julian. Akan tetapi, ada kalender dan format tanggal lain yang digunakan di seluruh dunia dan dalam domain ilmiah tertentu. Kita akan membahas dua sistem tersebut, dimulai dengan hari-hari Julian.

a. Hari Julian

Hari Julian adalah hitungan berkelanjutan dari seluruh hari matahari, karena tanggal referensi 0 pada tengah hari UTC, Senin 1 Januari 4713 SM. Tanggal yang setara dalam kalender Gregorian adalah 24 November 4714 SM. **Julian Day Number (JDN)** adalah cara untuk mengidentifikasi setiap hari secara unik dengan memberinya satu nomor. JDN digunakan dalam berbagai domain ilmiah dan komputasi, termasuk:

1) **Pengukuran waktu**: Hari Julian menyediakan sistem pengukuran waktu yang berkesinambungan dan seragam yang banyak digunakan dalam astronomi dan astrofisika untuk menentukan tanggal dan waktu pengamatan, perhitungan, dan peristiwa. Sistem ini menyederhanakan perhitungan yang melibatkan interval antara dua tanggal.
2) **Efemeris**: Efemeris astronomi, yang menyediakan posisi benda langit dari waktu ke waktu, sering menggunakan Hari Julian untuk skala waktunya.
3) **Sistem penanggalan yang seragam**: Hari Julian menawarkan sistem penanggalan yang seragam dan berkesinambungan yang berguna dalam kronologi sejarah. Sistem ini memungkinkan perhitungan interval waktu yang mudah dan sangat berharga untuk peristiwa sejarah yang berlangsung dalam jangka waktu yang panjang.

4) **Data bercap waktu**: Dalam geofisika dan ilmu bumi, data yang dikumpulkan dari waktu ke waktu, seperti peristiwa seismik atau pengamatan lingkungan, sering dicatat dengan JDN. Hal ini memungkinkan pengurutan kronologis peristiwa yang tepat. • Efisiensi komputasi: Julian Days praktis untuk algoritme dan pemrograman komputer karena menyediakan angka tunggal dan berkelanjutan yang menyederhanakan kalkulasi tanggal dan waktu. Ini khususnya berguna saat menangani kumpulan data besar atau melakukan kalkulasi yang melibatkan interval waktu.
5) **Kalkulasi navigasi**: Dalam navigasi, Julian Days digunakan untuk menghitung posisi benda langit dan menentukan waktu kejadian langit. Ini menyediakan cara standar untuk merepresentasikan waktu untuk tujuan navigasi.
6) **Konsistensi dalam data**: Julian Days menyediakan cara yang konsisten dan tidak ambigu untuk merepresentasikan tanggal dan waktu di berbagai kumpulan data dan disiplin ilmu. Konsistensi ini penting untuk penelitian, analisis data, dan komunikasi antara komunitas ilmiah.
7) **Perhitungan hari dan tahun kabisat**: Hari Julian menyederhanakan perhitungan yang melibatkan hari kabisat dan tahun kabisat karena perhitungan tersebut merupakan hitungan hari yang berkelanjutan, tidak terpengaruh oleh variasi panjang bulan atau tahun.

Singkatnya, hari Julian memainkan peran penting dalam berbagai konteks ilmiah dan komputasi, menyediakan sistem pengukuran waktu yang terstandardisasi dan berkelanjutan yang sangat berguna dalam disiplin ilmu yang membutuhkan urutan kronologis yang tepat dan interval waktu yang konsisten.

Kita dapat membuat fungsi M untuk mengonversi tanggal Gregorian ke tanggal Julian sebagai berikut:

```
let fnJulianDay = ( GregorianDate as date) as number => let
          GregorianYear = Date.Year( GregorianDate ), GregorianMonth =
          Date.Month( GregorianDate ), GregorianDay = Date.Day(
          GregorianDate ),
          Y = if GregorianMonth > 2 then GregorianYear else
          GregorianYear - 1,
          M = if GregorianMonth > 2 then GregorianMonth else
          GregorianMonth + 12,
          D = GregorianDay,
          A = Number.IntegerDivide( Y, 100),
          B = 2 - A + Number.IntegerDivide( A, 4),
          JulianDay = Number.IntegerDivide( 365.25 * ( Y + 4716), 1) +
          Number.
IntegerDivide( 30.6001 * ( M + 1), 1) + D + B - 1524.5
     In
          JulianDay, Documentation = [
          Documentation.Name =  " fnJulianDay",
          Documentation.Description = " Converts Gregorian date to
          Julian day",
          Documentation.LongDescription = " Converts Gregorian date to
          Julian
day",
          Documentation.Category = " Number", Documentation.Version =
          " 1.0", Documentation.Source = " local",
          Documentation.Author = " Gregory J. Deckler",
          Documentation.Examples = { [Description =  " fnJulianDay(
#date( 1987, 1, 1 ) )",
          Code = " Based on Jean Meeus' Astrological Algorithms",
          Result = " " ] }
          ]
in
     Value.ReplaceType( fnJulianDay, Value.ReplaceMetadata(
Value.Type( fnJulianDay ), Documentation ))
```

Beri nama kueri ini *fnJulianDay*. Selanjutnya, buat kueri berikut:

```
let
     Source = List.Generate( () => #date(1987, 1, 1), each _ <= #date(1987,
12, 31), each Date.AddDays( _, 1)),
     Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null,
ExtraValues.Error),
     RenameColumns = Table.RenameColumns(Table,{{"Column1", "Date"}}),
     AddJDNColumn = Table.AddColumn(RenameColumns, "Julian Day", each
fnJulianDay([Date]))
in
     AddJDNColumn
```

Ekspresi ini mengembalikan tabel untuk tahun 1987 yang mencakup kolom untuk ekuivalen Hari Julian.

Serupa dengan itu, kita dapat membuat fungsi *fnGregorianDate* untuk mengonversi Hari Julian ke tanggal ekuivalen kalender Gregorian:

```
let fnGregorianDate = ( JulianDay as number) as date => let
          Z = Number.IntegerDivide( JulianDay + .5, 1 ), F =
          JulianDay + .5 - Z,
          alpha = Number.IntegerDivide( Z - 1867216.25, 36524.25),
          A = if Z < 2299161 then Z else Z + 1 + alpha -
Number.IntegerDivide( alpha, 4 ),
          B = A + 1524,
          C = Number.IntegerDivide( B - 122.1, 365.25 ),
          D = Number.IntegerDivide( 365.25 * C, 1),
          E = Number.IntegerDivide( B - D, 30.6001),
          Day = B - D - Number.IntegerDivide( 30.6001 * E, 1), Month
          = if E < 14 then E - 1 else E - 13,
          Year = if Month > 2 then C - 4716 else C - 4715,
          GrgorianDate = #date( Year, Month, Day )
     in
          GrgorianDate, Documentati
          Documentation.Name =  " fnGregorianDate",
          Documentation.Description = " Converts Julian day to
          Gregorian date",
          Documentation.LongDescription = " Converts Julian day to
          Gregorian date"
          Documentation.Category = " Date", Documentation.Version =
          " 1.0", Documentation.Source = " local",
          Documentation.Author = " Gregory J. Deckler",
          Documentation.Examples = { [Description =  "
          fnGregorianDate( 2446796.5
)",
          Code = " Based on Jean Meeus' Astrological Algorithms",
          Result = " " ] }
          ]
in
     Value.ReplaceType( fnGregorianDate, Value.ReplaceMetadata(
Value.Type( fnGregorianDate ), Documentation ))
```

Namai kueri ini *fnGregorianDate*. Kita kemudian dapat memodifikasi kueri contoh kita sebagai berikut untuk mengembalikan kolom *Julian Day* dan kolom *Gregorian Date*:

```
let
     Source = List.Generate( () => #date(1987, 1, 1), each _
<= #date(1987, 12, 31), each Date.AddDays( _, 1)),
     Table = Table.FromList(Source, Splitter.SplitByNothing(),
null, null, ExtraValues.Error),
     RenameColumns = Table.RenameColumns(Table,{{"Column1",
     "Date"}}),
     AddJDNColumn = Table.AddColumn(RenameColumns, "Julian Day",
each fnJulianDay([Date])),
     AddGregorianDateColumn = Table.AddColumn(AddJDNColumn,
"Gregorian Date", each fnGregorianDate([Julian Day]))
in
     AddGregorianDateColumn
```

b. Format Tanggal Alternatif

Sistem tertentu menyimpan tanggal dalam format yang berbeda. Contohnya adalah beberapa versi sistem perangkat lunak Enterprise Resource Planning (ERP) J. D. Edwards yang populer, yang sekarang dimiliki oleh Oracle. Sistem ini menyimpan tanggal menggunakan tanda abad, dengan format berikut: cyyddd. Di sini, c adalah tanda abad di mana 0 sama dengan 1900 dan 1 sama dengan 2000. Tahun dalam abad tersebut disimpan sebagai angka dua digit antara 0 dan 99. Terakhir, hari disimpan sebagai hari numerik dalam setahun.

Untuk mengonversi format tanggal ini ke tipe data tanggal di Power Query, kita dapat menggunakan fungsi berikut dalam kueri bernama fnJDEToDate:

```
let fnJDEToDate = ( JDEDate as any) as date => let
        c = Number.From(Text.Start(Text.From( JDEDate ), 1)),
        yy = Number.From(Text.Middle(Text.From( JDEDate ), 1, 2)),
        ddd = Number.From(Text.End(Text.From( JDEDate ), 3)),
        Year = 1900 + c * 100 + yy,
        tmpDateList = List.Generate( () => #date( Year, 1, 1 ),
each _ < #date( Year + 1, 1, 1), each Date.AddDays( _, 1 )),
        tmpDateTable = Table.FromList(tmpDateList,
Splitter.SplitByNothing(), null, null, ExtraValues.Error),
        AddDayOfYear = Table.AddColumn( tmpDateTable, "Day of
        Year", each Date.
DayOfYear([Column1])),
        DateTable = Table.SelectColumns( Table.SelectRows(
AddDayOfYear, each [Day of Year] = ddd), "Column1" ),
        Date = DateTable[Column1]{0}
    in
        Date, Documentation = [
        Documentation.Name = " fnJDEToDate",
        Documentation.Description = " Converts cyyddd to date",
        Documentation.LongDescription = " Converts cyyddd to
        date", Documentation.Category = " Date",
        Documentation.Version = " 1.0", Documentation.Source = "
        local", Documentation.Author = " Gregory J. Deckler",
        Documentation.Examples = { [Description = " fnJDEToDate(
        124045 )", Code = " ",
        Result = " " ] }
        ]
in
    Value.ReplaceType( fnJDEToDate, Value.ReplaceMetadata(
Value.Type( fnJDEToDate ), Documentation ))
```

Sebagai penjelasan, kita ambil komponen-komponen individual dari abad (c), tahun dua digit (yy) dan hari tiga digit (ddd), masing-masing menggunakan fungsi Text.Start,

Text.Middle dan Text.End. Kita kemudian dapat menghitung tahun dengan rumus 1900 + c * 100 + yy. Setelah kita mendapatkan tahunnya, kita dapat membuat list tanggal untuk tahun itu, mengonversi list tersebut ke dalam tabel, menambahkan kolom Hari dalam Tahun menggunakan fungsi Date.DayOfYear, dan akhirnya, memfilter ke baris yang benar dalam tabel, mengonversi tabel itu ke dalam list, dan mengekstrak nilai pertama dalam list, { 0 }.

Kita dapat menggunakan fungsi ini untuk mengonversi tabel tanggal dengan cepat dalam format *cyyddd*, seperti ekspresi berikut:

```
let
    Source = List.Generate( () => 124001, each _ < 124367, each _
    + 1), #"Converted to Table" = Table.FromList(Source,
    Splitter.SplitByNothing(),
null, null, ExtraValues.Error),
    #"Duplicated Column" = Table.DuplicateColumn(#"Converted to
Table", "Column1", "Column1 - Copy"),
    #"Removed Columns" = Table.RemoveColumns(#"Duplicated
Column",{"Column1 - Copy"}),
    #"Invoked Custom Function" = Table.AddColumn(#"Removed
Columns", "Date", each fnJDEToDate([Column1]))
in
    #"Invoked Custom Function"
```

Ekspresi ini mengembalikan tabel untuk semua tanggal di tahun 2024.

3. Fungsi Tanggal Khusus Tambahan

Membuat fungsi khusus untuk bekerja dengan tanggal dapat menghemat banyak waktu. Di sini, kita membahas sejumlah fungsi khusus yang praktis untuk tanggal.

a. Hari Kerja

Seperti yang telah ditunjukkan sejauh ini, ada banyak cara berbeda untuk menyatakan tanggal dalam masyarakat maupun dalam sistem **information technology (TI)**. Demikian pula, ada banyak cara untuk mempertimbangkan perjalanan waktu. Contohnya adalah banyak organisasi bisnis melacak perjalanan

waktu dalam hari kerja (Senin–Jumat) dan tidak menghitung hari libur.

Dua fungsi kustom dapat membantu saat menangani hari kerja. Yang pertama dapat membantu mengklasifikasikan hari kerja dan hari libur:

```
let CategorizeDay = (date as date) => let
          Category = if Date.DayOfWeek(date, Day.Monday) <= 4 then
"Weekday" else "Weekend"
     in
          Category
in
     CategorizeDay
```

Fungsi *CategorizeDay* menggunakan satu parameter tanggal sebagai input. Fungsi *DayOfWeek* digunakan untuk mengembalikan nilai numerik dari 0 hingga 6, dengan 0 sebagai hari Senin. Parameter opsional kedua dari fungsi *DayOfWeek* digunakan untuk mengubah output dari *default* 0 untuk hari Minggu menjadi 0 untuk hari Senin. *Day.Monday* adalah enumerasi bawaan. Jadi, hari kerja diberi nomor 0, 1, 2, 3, dan 4 untuk hari Senin, Selasa, Rabu, Kitas, dan Jumat. Jadi, nilai apa pun yang kurang dari 4 berarti tanggal tersebut adalah hari kerja dan nilai yang lebih besar dari 4 adalah akhir pekan.

Fungsi kustom kedua dapat membantu menentukan jumlah hari kerja antara dua tanggal, termasuk menangani hari libur:

```
let WorkingDaysBetweenDates = (startDate as date, endDate as
date, optional holidays as list) =>
     let
          holidayDates = if holidays = null then { } else
          holidays, days = Number.From(Duration.From(endDate -
          startDate)) + 1,
          allDates = List.Dates(startDate, days, #duration(1, 0,
          0, 0)), workingDays = List.Select(allDates, each
          Date.DayOfWeek(_, Day.Monday)
<= 4 and not List.Contains(holidayDates, _))
     in
          List.Count(workingDays)
in
     WorkingDaysBetweenDates
```

Di sini, fungsi kustom *WorkingDaysBetweenDates* mengambil tiga parameter: tanggal mulai, tanggal akhir, dan

list hari libur opsional. Pertama-tama kita periksa apakah parameter ketiga opsional disertakan dan, jika tidak, buat list kosong. Berikutnya, kita hitung jumlah hari yang akan disertakan dalam rentang tanggal kita, dari *startDate* hingga *endDate*, dengan mengurangi kedua tanggal tersebut, mengubah durasi yang dihasilkan menjadi angka, dan menambahkan 1. Kemudian kita gunakan fungsi *List.Dates* untuk membuat list tanggal yang disertakan dalam rentang kita. Perhatikan bahwa kita juga dapat menggunakan fungsi *List.Generate* untuk tujuan ini. Ekspresi berikutnya memilih item list yang dihasilkan, menggunakan logika yang sama seperti fungsi *CategorizeDay* kita, tetapi juga tidak memilih tanggal apa pun yang ada dalam list *holidayDates*. Terakhir, kita hanya perlu menghitung item dalam list yang difilter.

b. Rata-Rata Bergerak

Skenario umum dalam analisis data adalah menyediakan rata-rata bergerak atau bergulir dari metrik tertentu. Meskipun ini dapat dicapai dalam DAX, Power Query M juga dapat digunakan untuk menambahkan informasi ini ke tabel. Fungsi kustom berikut menambahkan kolom rata-rata bergerak (MovingAvg) ke tabel:

```
let MovingAverage = (sourceTable as table, valueColumn as text,
windowSize as number) =>
    let
        BufferedValues = List.Buffer( Table.ToColumns(
                Table.SelectColumns(sourceTable, valueColumn)
            ){0}
        ),
        addIndex = Table.AddIndexColumn(sourceTable, "Index", 1,
        1, Int64.
Type),
        movingAvg = Table.AddColumn(
        addIndex,"MovingAvg", each
                if [Index] >= windowSize then List.Average(
                        List.Range( BufferedValues,
                        _[Index] - windowSize, windowSize
                    )
                ) else null, type number
        ),
        removeIndex = Table.RemoveColumns(movingAvg, {"Index"})
    in removeIndex
in
MovingAverage
```

Fungsi ini mengambil tiga parameter untuk tabel, nama kolom untuk menghitung rata-rata bergerak, dan ukuran jendela yang ditentukan untuk rata-rata bergerak. Fungsi ini pertama-tama menyimpan kolom nilai rata-rata bergerak yang ditentukan dari tabel ke dalam memori. Kolom indeks ditambahkan, dimulai dengan nilai 1 dan setiap baris bertambah 1. Selanjutnya, kolom rata-rata bergerak, MovingAvg, ditambahkan ke tabel, dengan parameter windowSize menentukan berapa banyak baris yang akan disertakan dalam rata-rata bergerak. Terakhir, seluruh tabel asli ditambah kolom MovingAvg dikembalikan sebagai tabel.

Sebagai contoh, dengan asumsi fungsi ini digunakan pada tabel fakta yang menyertakan informasi penjualan harian dengan ukuran jendela 30, memanggil fungsi ini akan mengembalikan tabel fakta tersebut dengan rata-rata bergerak yang dihitung untuk setiap baris selama 30 hari terakhir.

Pola yang digunakan di sini dapat dengan mudah diadaptasi ke jumlah bergulir atau perhitungan serupa dengan hanya mengganti fungsi List.Average dengan fungsi List.Sum, atau fungsi serupa. Sekarang mari kita lanjutkan untuk membahas waktu.

D. Waktu

Meskipun mungkin tidak terlalu sering digunakan dalam pelaporan seperti halnya tanggal, waktu juga dapat menjadi elemen pelaporan yang penting, khususnya saat melaporkan metrik seperti kinerja karyawan atau produktivitas rantai pasokan.

Mirip dengan tanggal, waktu juga direpresentasikan dalam bentuk numerik sebagai pecahan hari. Untuk mengamati ini, buat kueri baru dengan:

```
let
    Source = List.Generate(() => 0, each _ <= 24, each _ + 1),
    Table = Table.FromList(Source, Splitter.SplitByNothing(), null,
null, ExtraValues.Error),
    AddMultiplicationColumn = Table.AddColumn(Table, "Multiplication",
each [Column1] / 24, type number),
    DuplicateColumn = Table.DuplicateColumn(AddMultiplicationColumn,
"Multiplication", "Multiplication - Copy"),
    ChangeType =
Table.TransformColumnTypes(DuplicateColumn,{{"Multiplication - Copy",
type time}}),
    RenameColumns = Table.RenameColumns(ChangeType,{{"Multiplication -
Copy", "Time"}})
in
    RenameColumns
```


Tabel yang ditunjukkan pada Gambar 2.5 ditampilkan:

	Column1	Multiplication	Time
1	0	0	12:00:00 AM
2	1	0.041666667	1:00:00 AM
3	2	0.083333333	2:00:00 AM
4	3	0.125	3:00:00 AM
5	4	0.166666667	4:00:00 AM
6	5	0.208333333	5:00:00 AM
7	6	0.25	6:00:00 AM
8	7	0.291666667	7:00:00 AM
9	8	0.333333333	8:00:00 AM
10	9	0.375	9:00:00 AM
11	10	0.416666667	10:00:00 AM
12	11	0.458333333	11:00:00 AM
13	12	0.5	12:00:00 PM
14	13	0.541666667	1:00:00 PM
15	14	0.583333333	2:00:00 PM
16	15	0.625	3:00:00 PM
17	16	0.666666667	4:00:00 PM
18	17	0.708333333	5:00:00 PM
19	18	0.75	6:00:00 PM
20	19	0.791666667	7:00:00 PM
21	20	0.833333333	8:00:00 PM
22	21	0.875	9:00:00 PM
23	22	0.916666667	10:00:00 PM
24	23	0.958333333	11:00:00 PM
25	24	1	Error

Gambar 2.5 Tabel Waktu Jam

Kita dapat melihat dari tabel bahwa nilai numerik 0 sesuai dengan tengah malam (12:00:00 AM). Di kolom Multiplication, kita telah membagi Column1 dengan 24 untuk mendapatkan pecahan jam dalam sehari. Nilai ini, ketika dikonversi ke nilai waktu, mengembalikan jam yang sesuai dalam sehari. Perhatikan bahwa nilai yang sama dengan atau lebih besar dari 1 mengembalikan kesalahan.

Nilai waktu memiliki batasan yang sama mengenai operasi aritmatika seperti tipe data tanggal. Ini berarti Anda dapat menambah atau mengurangi nilai waktu dari satu sama lain, tetapi Anda tidak dapat melakukan operasi aritmatika yang melibatkan angka tanpa secara khusus mengonversi nilai antara tipe data.

Serupa dengan tanggal, kita memiliki sejumlah fungsi untuk mengekstrak komponen waktu Hour, Minute, dan Second, seperti berikut ini:

```
    AddHourColumn = Table.AddColumn(RenameColumns, "Hour", each Time.
Hour([Time])),
    AddMinuteColumn = Table.AddColumn(AddHourColumn, "Minute", each
    Time.
Minute([Time])),
    AddSecondColumn = Table.AddCvvv(AddMinuteColumn, "Second", each
    Time.
Second([Time]))
```

Mirip dengan tanggal, ada fungsi Time.Record yang juga mengekstrak komponen waktu yang sama ke dalam Record.

Mirip dengan tipe data tanggal, tipe data time juga memiliki konstruktor, fungsi #time. Fungsi #time mengambil tiga parameter: Hour, Minute dan Second. Meskipun Anda mungkin berpikir bahwa Anda dapat menggunakan addition untuk menambahkan waktu ke tanggal dan mengembalikan tipe data datetime, tindakan tersebut akan menghasilkan kesalahan:

```
#date(2024, 2, 14) + #time(17, 0, 0)
```

```
Error: We cannot apply operator + to types Date and Time.
```

Namun, operator penggabungan berfungsi. Dengan demikian, ekspresi berikut mengembalikan tipe data *datetime* 14 Februari 2024 pukul 5 sore (14/2/2024 pukul 5:00:00 sore):

```
#date(2024, 2, 14) & #time(17, 0, 0)
```

Dua fungsi tambahan, *Time.StartOfHour* dan *Time.EndOfHour*, mengembalikan tipe data datetime yang mewakili awal dan akhir jam, dengan tipe data nilai input waktu, datetime, atau datetimezone. Misalnya, ekspresi berikut mengembalikan 13:59:59.9999999:

```
Time.EndOfHour(#time(13,10,5))
```

Dalam contoh ini, fungsi *#time* digunakan untuk membuat nilai tipe data waktu. Fungsi #time mengambil jam, menit, dan detik sebagai parameter pertama, kedua, dan ketiga.

Fungsi Time.From dan Time.FromText mengonversi nilai input ke tipe data waktu. Fungsi Time.From menerima nilai input teks, datetime, datetimezone, atau number. Kedua fungsi ini juga menyertakan parameter Culture opsional yang mirip dengan fungsi tanggal yang setara.

Terakhir, fungsi Time.ToText mengonversi tipe data waktu ke tipe data teks. Representasi waktu teks ini dapat dikontrol melalui parameter kedua opsional, parameter Format. Informasi lebih lanjut tentang berbagai opsi pemformatan dapat ditemukan di situs web Gorilla BI yang dioperasikan oleh Rick de Groot di sini: https://gorilla.bi/Power-query/custom-format-strings/. Mirip dengan fungsi data yang setara, Date.ToText, parameter ketiga opsional, Culture, juga dapat ditentukan.

1. Membuat Tabel Waktu

Aplikasi pelaporan tertentu memerlukan dimensi waktu yang tepat pada granularitas menit atau detik. Hal ini dapat dicapai melalui kode M berikut:

```
let
    Source = List.Generate( () => 0, each _ < 24 * 60, each _ + 1 ),
    Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null,
ExtraValues.Error),
    Divide = Table.TransformColumns(Table, {{"Column1", each _ / 24 / 60, type
number}}),
    ChangeType = Table.TransformColumnTypes(Divide,{{"Column1", type time}}),
    RenameColumns = Table.RenameColumns(ChangeType,{{"Column1", "Time"}}),
    AddHourColumn = Table.AddColumn(RenameColumns, "Hour", each Time.
Hour([Time])),
    AddMinuteColumn = Table.AddColumn(AddHourColumn, "Minute", each Time.
Minute([Time])),
    AddSecondColumn = Table.AddColumn(AddMinuteColumn, "Second", each Time.
Second([Time])),
    AddStartOfHourColumn = Table.AddColumn(AddSecondColumn, "Start of Hour",
each Time.StartOfHour([Time])),
    AddEndOfHourColumn = Table.AddColumn(AddStartOfHourColumn, "End of Hour",
each Time.EndOfHour([Time]))
in
    AddEndOfHourColumn
```

Ekspresi M ini mengembalikan tabel waktu pada tingkat ketelitian menit. Beberapa modifikasi kecil akan mengembalikan tabel waktu pada tingkat ketelitian kedua:

```
let
     Source = List.Generate( () => 0, each _ < 24 * 60 * 60, each _ + 1 ),
     Table = Table.FromList(Source, Splitter.SplitByNothing(), null, null,
ExtraValues.Error),
     Divide = Table.TransformColumns(Table, {{"Column1", each _ / 24 / 60 / 60,
type number}}),
     ChangeType = Table.TransformColumnTypes(Divide,{{"Column1", type time}}),
     RenameColumns = Table.RenameColumns(ChangeType,{{"Column1", "Time"}}),
     AddHourColumn = Table.AddColumn(RenameColumns, "Hour", each Time.
Hour([Time])),
     AddMinuteColumn = Table.AddColumn(AddHourColumn, "Minute", each Time.
Minute([Time])),
     AddSecondColumn = Table.AddColumn(AddMinuteColumn, "Second", each Time.
Second([Time])),
     AddStartOfHourColumn = Table.AddColumn(AddSecondColumn, "Start of Hour",
each Time.StartOfHour([Time])),
     AddEndOfHourColumn = Table.AddColumn(AddStartOfHourColumn, "End of Hour",
each Time.EndOfHour([Time]))
in
     AddEndOfHourColumn
```

2. Klasifikasi Pergeseran (*Shift Classification*)

Kita dapat membuat fungsi sederhana untuk mengklasifikasikan waktu menjadi shift pertama, kedua, dan

ketiga. Dengan asumsi bahwa shift pertama berlangsung dari pukul 8 pagi hingga 4 sore, shift kedua dari pukul 4 sore hingga tengah malam, dan shift ketiga dari tengah malam hingga pukul 8 pagi, fungsi berikut mengklasifikasikan waktu yang terlewati sebagai parameter yang termasuk dalam shift tertentu:

```
let ShiftClassification = ( actualTime as time ) => let
        hourEndShift3 = Time.Hour(#time(8, 0, 0)),
        hourEndShift1 = Time.Hour(#time(16, 0, 0)),
        hour = Time.Hour(Time.StartOfHour( actualTime )),
        shift = if hour < hourEndShift3 then "Third" else if
hour < hourEndShift1 then "First" else "Second"
    in
        shift
in
    ShiftClassification
```

Menyelesaikan eksplorasi kita tentang waktu dalam bahasa M. Sekarang kita beralih ke tanggal dan waktu.

E. Tanggal dan Waktu

Seperti yang dapat diantisipasi, representasi tanggal dan waktu dapat dilakukan dalam bahasa M. Tipe data ini adalah tipe data datetime dan menunjukkan properti yang mirip dengan tipe data tanggal dan tipe data waktu. Faktanya, karena tanggal hanyalah bilangan bulat berdasarkan tanggal referensi dan waktu adalah bilangan desimal yang mencerminkan pecahan hari, tipe data datetime dapat direpresentasikan sebagai bilangan desimal, di mana bagian bilangan bulat mewakili tanggal dan bagian desimal mewakili waktu. Misalnya, ekspresi berikut mengembalikan tengah hari pada tanggal 14 Februari (14/2/2024 12:00:00 PM):

```
DateTime.From(45336.5)
```

Komponen-komponen individual dari tanggal dan waktu dapat diekstrak dari tipe data datetime menggunakan fungsi DateTime.Date dan DateTime.Time. Atau, fungsi DateTime.ToRecord mengembalikan Record dengan pasangan bidang/nilai untuk Year, Month, Day, Hour, Minute, dan Second.

Mirip dengan tanggal dan waktu, konstruktor tersedia untuk tipe data datetime: fungsi #datetime. Berikut ini mengembalikan tanggal 14 Februari 2024 pukul 5 sore:

```
#datetime(2024, 2, 14, 17, 0, 0)
```

Dua fungsi tambahan, DateTime.LocalNow dan DateTime.FixedLocalNow, mengembalikan tanggal dan waktu sistem saat ini. Satu-satunya perbedaan antara kedua fungsi ini adalah DateTime.LocalNow dapat mengembalikan nilai yang berbeda saat dijalankan beberapa kali selama evaluasi ekspresi M. DateTime.FixedLocalNow tidak menunjukkan perilaku ini dan hanya akan mengembalikan satu nilai datetime selama beberapa eksekusi dalam ekspresi M.

Sebelum adanya peningkatan dalam layanan Power BI, kedua fungsi ini sering digunakan dalam kueri terpisah untuk mengembalikan waktu penyegaran terakhir kumpulan data. Tanggal dan waktu ini kemudian dapat ditampilkan dalam visual Kartu pada halaman laporan.

Serupa dengan tanggal, ada keluarga fungsi IsInCurrent, IsInNext, dan IsInPrevious. Setiap keluarga ini mencakup fungsi individual untuk komponen waktu Jam, Menit, dan Detik. Keluarga IsInNext dan IsInPrevious juga mencakup varian N, seperti IsInNextNHours, IsInNextNMinutes, dan IsInNextNSeconds. Semua fungsi ini mengembalikan nilai Boolean logika true atau false, tergantung pada apakah nilai datetime memenuhi kondisi yang ditetapkan atau tidak.

Fungsi DateTime.FromFileTime agak tidak jelas; fungsi ini mengonversi waktu berkas Windows ke dalam format datetime standar. Waktu berkas Windows dicatat sebagai jumlah total interval 100 nanodetik sejak 12:00 AM, 1 Januari 1601 M (juga disebut 1601 CE), dalam Coordinated Universal Time (UTC). Titik awal khusus ini digunakan sebagai referensi untuk memastikan konsistensi dalam catatan waktu berkas di seluruh sistem Windows. Fungsi DateTime.FromFileTime hanya mengonversi waktu berkas Windows ke nilai tipe data datetime. Dengan demikian, ekspresi

DateTime.FromFileTime(133524036000000000) mengembalikan nilai datetime 14 Februari (14/2/2024 12:00:00 PM) jika Anda berada di Zona Waktu Timur Amerika Serikat (Waktu Standar bukan Waktu Musim Panas). Terakhir, fungsi DateTime.AddTimeZone menambahkan informasi zona waktu ke tipe data datetime, yang mengembalikan tipe data datetimezone. Melanjutkan contoh sebelumnya, ekspresi M berikut mengembalikan tipe data datetimezone yang menambahkan informasi zona waktu untuk Eastern Time (US):

```
= DateTime.AddZone(DateTime.From(45336.5), -5, 0)
```

Nilai yang dikembalikan dari ekspresi ini adalah 2/14/2024 12:00:00 PM -05:00.

Berbicara tentang zona waktu, selanjutnya mari kita lanjutkan dengan menjelajahi fungsi yang terkait dengan tipe data datetimezone.

F. Zona Waktu

Meskipun sangat praktis dan relatif mudah dipahami, zona waktu telah menyebabkan masalah dengan sistem perangkat lunak dan pemrograman sejak awal mula komputasi. Untungnya, bahasa M menyertakan tipe data datetimezone, dengan serangkaian fungsi pendukung yang memudahkan penanganan tanggal dan waktu yang menyertakan zona waktu.

Tidak seperti tipe data tanggal, waktu, dan datetime, tipe data datetimezone tidak dapat direpresentasikan hanya sebagai angka desimal. Faktanya, membuat nilai datetimezone menggunakan DateTimeZone.From(45336.5) menambahkan informasi zona waktu tambahan dari sistem lokal, yang mengembalikan 2/14/2024 12:00:00 PM -05:00, misalnya, untuk sistem yang berjalan di zona waktu EST Amerika Serikat.

Sebenarnya ada dua elemen tambahan untuk tipe data datetimezone: ZoneHours dan ZoneMinutes. Elemen tambahan ini dapat diambil menggunakan fungsi DateTimeZone.ZoneHours dan DateTimeZone. ZoneMinutes. Selain itu, fungsi

DateTimeZone.ToRecord mengembalikan Record yang terdiri dari bidang-bidang berikut: *Year, Month, Day, Hour, Minute, Second, ZoneHours, ZoneMinutes.*

Seperti halnya tipe data tanggal, waktu, dan datetime, tipe data datetimezone mencakup sebuah konstruksi, yaitu fungsi #datetimezone. Fungsi ini menggunakan hingga delapan parameter – tahun, bulan, hari, jam, menit, detik, offset jam, dan offset menit. Misalnya, berikut ini mengembalikan tanggal 14 Februari 2024 pukul 17.00 EST:

```
#datetimezone(2024, 2, 14, 17, 0, 0, -5, 0)
```

Mirip dengan keluarga fungsi DateTime, keluarga fungsi DateTimeZone mencakup fungsi untuk mengembalikan waktu sistem lokal saat ini, DateTimeZone.LocalNow dan DateTimeZone.FixedLocalNow. Fungsi-fungsi ini bekerja sama seperti padanan DateTime-nya tetapi juga mencakup informasi zona waktu lokal.

Selain itu, dua fungsi tambahan disediakan untuk mengambil waktu UTC saat ini. Fungsi-fungsi ini adalah DateTimeZone.UtcNow dan DateTimeZone.FixedUtcNow. Sekali lagi, perbedaan antara fungsi-fungsi ini adalah bahwa beberapa panggilan dalam suatu ekspresi ke UtcNow dapat mengembalikan waktu yang berbeda, sementara beberapa panggilan ke FixedUtcNow akan mengembalikan waktu yang sama.

Fungsi tambahan, DateTimeZone.ToUtc, mengonversi nilai datetimezone ke waktu UTC. Melanjutkan contoh kita, ekspresi DateTimeZone.ToUtc(DateTimeZone.From(45336.5)) mengembalikan 2/14/2024 5:00:00 PM +00:00 ketika dievaluasi pada sistem di zona waktu EST Amerika Serikat. Di sini, nilai asli tengah hari pada tanggal 14 Februari 2024 dimajukan 5 jam menjadi 5PM. Ini masuk akal mengingat EST memiliki ZoneHours -5, yang berarti zona waktu EST tertinggal 5 jam dari waktu UTC.

Serupa dengan fungsi ToUtc, fungsi DateTimeZone.ToLocal mengonversi nilai datetimezone ke waktu sistem lokal komputer yang mengevaluasi ekspresi tersebut. Jadi, ekspresi DateTimeZone.

ToLocal(DateTimeZone.From("2/14/2024 5:00:00 PM +00:00")) mengembalikan 2/14/2024 12:00:00 PM -05:00 saat dievaluasi pada komputer yang berjalan di zona waktu EST.

Informasi zona waktu juga dapat dimanipulasi dengan fungsi DateTimeZone.RemoveZone dan DateTimeZone. SwitchZone. Fungsi DateTimeZone.RemoveZone menghapus informasi zona waktu, mengembalikan tipe data datetime. Fungsi DateTimeZone.SwitchZone mengganti informasi zona waktu dengan zona waktu yang ditentukan oleh parameter kedua dan ketiga dari fungsi tersebut.

Perlu dicatat bahwa fungsi SwitchZone tidak hanya mengganti zona waktu, tetapi juga mengonversi nilai datetimezone ke zona waktu yang baru. Dengan demikian, ekspresi Date Time Zone. Switch Zone (Date Time Zone. From(45336.5), 0, 0) setara dengan contoh kita sebelumnya yang menggunakan fungsi ToUtc dan mengembalikan nilai yang sama persis, 2/14/2024 5:00:00 PM +00:00. Dengan demikian, fungsi SwitchZone dapat dilihat sebagai fungsi penggunaan yang lebih umum daripada fungsi ToUtc dan ToLocal yang lebih spesifik.

Terakhir, DateTimeZone.FromFileTime bekerja identik dengan fungsi DateTime tetapi, seperti yang diharapkan, menambahkan informasi zona waktu yang diambil dari sistem lokal yang mengevaluasi ekspresi.

1. Memperbaiki Waktu Penyegaran Data

Skenario umum dalam pelaporan adalah menyertakan waktu terakhir data disegarkan. Namun, layanan Power BI mencatat waktu penyegaran dalam zona waktu wilayah penyewa Power BI. Ini mungkin atau mungkin tidak sesuai dengan zona waktu negara tempat pengguna melihat laporan.

Masalah ini dapat diperbaiki dengan menggunakan fungsi kustom untuk menghitung waktu penyegaran, menyimpan tanggal dan waktu sebagai tabel baris tunggal dalam model data, lalu menampilkan tanggal dan waktu ini di halaman laporan. Fungsi kustom berikut menyesuaikan waktu penyegaran untuk

memperhitungkan masalah tersebut dan juga menyertakan kemampuan untuk menangani daylight saving time:

```
(Summer_GMT_Offset as number, Winter_GMT_Offset as number) => let
     UTC_DateTimeZone = DateTimeZone.UtcNow(), UTC_Date = Date.From(
     UTC_DateTimeZone ),
     StartSummerTime = Date.StartOfWeek( #date( Date.Year( UTC_Date ) , 3 , 31
), Day.Sunday ),
     StartWinterTime = Date.StartOfWeek( #date( Date.Year( UTC_Date ) , 10, 31
), Day.Sunday ),
     UTC_Offset = if UTC_Date >= StartSummerTime and UTC_Date < StartWinterTime
then Summer_GMT_Offset else Winter_GMT_Offset,
     CET_Timezone = DateTimeZone.SwitchZone( UTC_DateTimeZone, UTC_Offset)
in
     CET_Timezone
```

Fungsi ini berasal dari Rick de Groot dan mengambil dua parameter: selisih waktu musim panas dan selisih waktu musim dingin dari waktu UTC untuk zona waktu yang diinginkan untuk menampilkan waktu yang diinginkan. Pertama, fungsi *DateTimeZone.UtcNow* digunakan untuk mendapatkan tanggal, waktu, dan zona waktu. Selanjutnya, fungsi *Date.From* digunakan untuk menyimpan hanya bagian tanggal. Dengan asumsi bahwa musim panas (waktu standar) dimulai pada hari Minggu terakhir di bulan Maret dan musim dingin (waktu musim panas) dimulai pada hari Minggu terakhir di bulan Oktober, dua baris berikutnya menentukan tanggal musim panas dan musim dingin.

Offset yang diinginkan kemudian dihitung berdasarkan tanggal saat ini dibandingkan dengan tanggal musim panas dan musim dingin. Waktu kemudian disesuaikan menggunakan fungsi DateTimeZone.SwitchZone. Sekarang mari kita bahas durasinya.

G. Durasi

Tidak seperti model data tabular Analysis Services, bahasa M menyertakan tipe data untuk durasi. Kelalaian tipe data durasi yang tidak diharapkan dalam Analysis Services berarti bahwa perhitungan yang melibatkan durasi mungkin lebih baik ditangani di Power Query daripada dengan kolom atau ukuran DAX. Namun, penting juga untuk menyadari bahwa kolom tipe data duration di

Power Query dikonversi ke angka desimal saat dimuat ke dalam model data. Pertimbangkan ekspresi berikut:

```
let
    Source = Duration.From("0.01:00:00"), #"Converted to Table" =
    #table(1, {{Source}}),
    #"Changed Type" = Table.TransformColumnTypes(#"Converted to
Table",{{"Column1", type duration}})
in
    #"Changed Type"
```

Tabel baris tunggal ini, yang berisi kolom tipe data durasi dengan nilai 1 jam, saat dimuat ke dalam model data akan menjadi nilai 0,0416666666666667, yang merupakan jumlah pecahan dari satu hari (1 jam/24 jam). Jadi, untuk menampilkan format durasi dalam laporan, Anda harus melakukan konversi pada data dan akhirnya mengonversi nilai tersebut menjadi teks, menggunakan fungsi FORMAT DAX, atau memanfaatkan String Format Kustom untuk mempertahankan nilai sebagai angka.

Tipe data durasi mencakup konstruktor, fungsi #duration. Fungsi #duration mengambil empat parameter – hari, jam, menit, dan detik. Ekspresi berikut mengembalikan durasi 05.02:15:33:

```
#duration(5, 2, 15, 33)
```

Tipe data durasi dapat ditambahkan ke tipe data tanggal, waktu, datetime, atau datetimezone menggunakan operator penambahan (+). Dengan demikian, ekspresi berikut mengembalikan tanggal 14 Februari 2024 sebagai 2/14/2024:

```
#date(2024, 2, 9) + #duration(5, 2, 15, 33)
```

Sementara ekspresi berikut mengembalikan waktu 7:15:33 PM:

```
#time( 17, 0, 0) + #duration(5, 2, 15, 33)
```

Komponen individual durasi dapat diekstrak dengan fungsi Duration. Days, Duration.Hours, Duration.Minutes, dan Duration. Seconds. Atau, fungsi Duration.ToRecord mengembalikan Record

dengan nilai yang sama, sebagai bidang berlabel Days, Hours, Minutes, dan Seconds.

Empat fungsi tambahan membuat penghitungan jumlah total hari, jam, menit, dan detik untuk durasi menjadi mudah. Fungsi-fungsi ini adalah Duration.TotalDays, Duration.TotalHours, Duration.TotalMinutes, dan Duration.TotalSeconds. Dengan demikian, ekspresi seperti Duration.TotalSeconds (Duration.From("1.2:10:12")) mengembalikan nilai 94.212, yaitu 3.600 detik dalam satu jam dikalikan 26 jam (1 hari penuh ditambah 2 jam), ditambah 10 menit dikali 60 detik per menit, ditambah 12 detik, sama dengan 94.212 detik total.

Seperti yang disebutkan, karena mesin tabular Analysis Services tidak mendukung tipe data durasi yang sebenarnya, sebaiknya lakukan penghitungan durasi dalam Power Query lalu gunakan salah satu fungsi Total dalam Power Query, seperti Duration.TotalSeconds, untuk mengonversi kolom durasi menjadi jumlah total detik numerik sebelum memuat ke dalam model data. Setelah disimpan dalam hitungan detik, ada banyak metode DAX standar untuk mengonversi nilai ke durasi, ditambah dengan penggunaan String Format Kustom untuk menampilkan angka numerik dalam format durasi. Salah satu contohnya adalah Durasi karya Chelsie Eiden di Galeri Pengukuran Cepat situs Komunitas Power BI: https://bit.ly/47JMPJR.

1. Durasi kerja

Mirip dengan menghitung hari kerja antara dua tanggal, fungsi kustom sederhana dapat dibuat untuk menghitung durasi kerja antara dua waktu:

```
( startTime as time, endTime as time, workStartTime as time,
workEndTime as time) =>
    let
        actualStartTime = if startTime < workStartTime then
workStartTime else startTime,
        actualEndTime = if endTime > workEndTime then workEndTime
        else endTime, workDuration = actualEndTime -
        actualStartTime
    in
        workDuration
```

Fungsi ini menggunakan empat parameter, yaitu waktu mulai dan berakhir yang sebenarnya serta waktu mulai dan berakhirnya hari kerja. Pernyataan if sederhana digunakan untuk menentukan apakah waktu mulai yang dimasukkan lebih kecil dari waktu mulai kerja dan apakah waktu berakhir yang dimasukkan lebih besar dari waktu berakhirnya kerja. Durasinya kemudian dihitung dengan cara mengurangi waktu mulai dan berakhir yang dimodifikasi.

Fungsi yang lebih sederhana dapat digunakan untuk mengonversi jumlah hari dan jam kerja per hari menjadi durasi:

```
(days as number, hoursPerDay as number) => days *
    hoursPerDay * #duration(0, 1, 0, 0)
```

H. Ringkasan

Dalam bab ini, kita membahas topik tentang tanggal, waktu, tanggal dan waktu, zona waktu, dan durasi. Sebagian besar fungsi yang menangani tipe data ini dibahas, dan kita mengeksplorasi beberapa aplikasi praktis penggunaan fungsi ini, termasuk Tabel Tanggal Diperpanjang Melissa de Korte yang terkenal, pembuatan fungsi untuk mengonversi antara tanggal Gregorian dan hari Julian, dan menangani format tanggal alternatif. Kita juga membahas contoh penghitungan jumlah hari kerja antara dua tanggal dan cara menghitung rata-rata bergerak. Sehubungan dengan waktu, kita menunjukkan pembuatan tabel dimensi waktu pada tingkat ketelitian menit dan kedua dan cara mengklasifikasikan waktu menjadi shift pertama, kedua, dan ketiga. Tanggal dan waktu juga dibahas dan menyertakan contoh untuk mengoreksi waktu

penyegaran data. Terakhir, durasi dibahas, dengan contoh penghitungan durasi kerja dan mengonversi hari kerja menjadi durasi.

Dalam bab berikutnya, kita melanjutkan eksplorasi M dengan membahas beberapa teknik untuk memanipulasi dan menyempurnakan data. Ini termasuk tinjauan terperinci tentang membandingkan, mengganti, menggabungkan, dan membagi nilai.

I. Daftar Pustaka

Montero-Manso, P., Hyndman, R.J. (2020), Principles and algorithms for forecasting groups of time series: Locality and globality. arXiv:2008.00444[cs.LG]: https://arxiv.org/abs/2008.00444.

Micci-Barreca, D. (2001), A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. SIGKDD Explor. Newsl. 3, 1 (July 2001), 27–32: https:// doi.org/10.1145/507533.507538.

Fisher, W. D. (1958). On Grouping for Maximum Homogeneity. Journal of the American Statistical Association, 53(284), 789–798: https://doi.org/10.2307/2281952.

Fisher, W.D. (1958), A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. SIGKDD Explor. Newsl. 3, 1 (July 2001), 27–32.

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Spaces. In Proceedings of the 8th International Conference on Database Theory (ICDT '01). Springer-Verlag, Berlin, Heidelberg, 420–434: https://dl.acm.org/doi/10.5555/645504.656414.

Oreshkin, B. N., Carpov D., Chapados N., and Bengio Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. 8th International Conference on Learning Representations, ICLR 2020: https://openreview.net/forum?id=r1ecqn4YwB.

J. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan perbedaan antara tipe data date, time, dan datetime dalam *Power Query*. Sertakan contoh penggunaannya dalam proses transformasi data.	10
2.	Mengapa penting untuk mempertimbangkan zona waktu saat memanipulasi data bertipe waktu di *Power Query*? Berikan ilustrasi nyata dari sebuah studi kasus yang mungkin mengalami kesalahan karena pengabaian zona waktu.	10
3.	Uraikan langkah-langkah untuk mengonversi nilai teks menjadi tipe data tanggal di *Power Query*. Apa tantangan yang dapat muncul saat mengonversi data tersebut?	10
4.	Bagaimana cara menghitung selisih antara dua nilai waktu dalam *Power Query* menggunakan fungsi durasi? Sertakan contoh kode dan jelaskan satu kasus bisnis yang relevan.	10
5.	Apa itu nilai duration dalam *Power Query*? Jelaskan bagaimana nilai ini dihitung dan bagaimana penggunaannya berbeda dari nilai datetime.	10
6.	Diskusikan fungsi-fungsi utama yang tersedia dalam *Power Query* untuk memanipulasi tanggal. Berikan minimal tiga fungsi dan deskripsikan penggunaannya.	10
7.	Bagaimana *Power Query* menangani nilai waktu yang berada di luar rentang valid, misalnya jam ke-25 atau tanggal 32 Januari? Jelaskan dengan contoh dan solusi penanganannya.	10
8.	Berikan penjelasan mengenai fungsi DateTimeZone dalam *Power Query*. Apa perbedaannya dengan DateTime, dan dalam situasi apa Anda lebih memilih menggunakannya?	10

No	Soal	Bobot
9.	Sebutkan dan jelaskan dua teknik yang dapat digunakan untuk mengekstrak bagian tertentu dari nilai tanggal (seperti tahun, bulan, hari) dalam *Power Query*.	10
10.	Analisis bagaimana penggunaan data waktu yang akurat dapat meningkatkan kualitas pelaporan dalam *Power* BI. Hubungkan dengan praktik penggunaan fungsi waktu di *Power Query*.	10

DUMMY
Penerbitan & Percetakan
UNP PRESS

BAB 3
Pembanding, Pengganti, Penggabung, dan Pemisah

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Konsep kunci
- Pembanding
- Kriteria perbandingan
- Kriteria persamaan
- Pengganti
- Pengganti custom
- Penggabung
- Pemisah
- Contoh praktis

A. Pendahuluan

Persiapan data dalam Power Query melibatkan beberapa teknik yang ditujukan untuk memanipulasi dan menyempurnakan data. Teknik-teknik ini meliputi pemisahan, penggabungan, pembandingan, dan penggantian nilai untuk mencapai struktur dan kualitas data yang diinginkan. Berikut ini ikhtisar singkat dari setiap teknik:

Tabel 3.1 Tinjauan singkat tentang teknik yang digunakan untuk mengelola data

Teknik	**Tujuan**	**Contoh**
Comparers	Tentukan kesetaraan dan ketertiban	Mengidentifikasi duplikat, memberi peringkat, dan menyortir data
Replacer s	Nilai pengganti	Memperbaiki kesalahan ejaan atau ketidakteraturan untuk memastikan keseragaman
Combiners	Gabungkan nilai-nilai	Menggabungkan *string*, menggabungkan kolom
Splitters	Nilai-nilai yang terpisah	Membagi *string*, membagi kolom

Bab ini memberikan gambaran umum tentang kemampuan pemisahan, penggabungan, pembandingan, dan penggantian dalam

bahasa Power Query M. Bab ini bertujuan untuk membekali Anda dengan pengetahuan yang diperlukan untuk menerapkan transformasi ini secara efektif dalam alur kerja Anda. Bab ini membahas topik-topik berikut: Konsep utama, Pembanding, Kriteria perbandingan, Kriteria persamaan, Pengganti, Pengganti kustom, Penggabung, Pemisah dan Contoh praktis.

1. Kasus Pemantik Berfikir Kritis

Seorang analis data diminta untuk membersihkan dan menyusun ulang data pelanggan dari sebuah toko online. Dalam data tersebut, terdapat kolom "Nama Lengkap", "Alamat", dan "Email", tetapi ditemukan beberapa masalah seperti:

a. Nama pelanggan ditulis dengan berbagai gaya penulisan (contoh: "andi", "Andi", "ANDI").

b. Ada nilai yang sama di kolom "Email" tetapi dengan huruf besar-kecil yang berbeda.

c. Kolom "Alamat" berisi kota dan negara yang digabung menjadi satu, seperti: "Padang, Indonesia".

d. Beberapa nilai di kolom "Nama Lengkap" mengandung kata yang salah tulis, misalnya "Jhon" yang seharusnya "John".

2. Pertanyaan Pemantik:

a. Bagaimana cara Anda membandingkan dan menyamakan data yang memiliki perbedaan huruf besar-kecil?
b. Apa langkah yang dapat Anda lakukan untuk memperbaiki kesalahan penulisan nama seperti "Jhon"?
c. Bagaimana Anda dapat memisahkan kota dan negara dalam kolom "Alamat" menjadi dua kolom yang berbeda?
d. Jika Anda ingin menggabungkan kembali kota dan negara dalam satu kolom dengan format "Indonesia - Padang", fungsi *Power Query* apa yang bisa digunakan?
e. Mengapa penting menggunakan fungsi pembanding (*comparer*) saat ingin mengidentifikasi data duplikat seperti email atau nama?

f. Dalam kasus seperti “Jhon” yang salah tulis, kapan sebaiknya Anda menggunakan *Replacer .ReplaceText* dan kapan menggunakan pendekatan manual?

B. Persyaratan Teknis

Untuk memulai, kunjungi repositori GitHub dan unduh berkas PBIX yang menyertai bab ini. Berkas tersebut disiapkan untuk Anda ikuti dan terapkan teknik yang dibahas di sini, menawarkan pendekatan langsung yang meningkatkan pemahaman Anda. Dengan mengikuti, Anda tidak hanya belajar secara teoritis; Anda juga memperoleh pengalaman praktis yang memperkuat pengetahuan dan keterampilan Anda.

C. Konsep-konsep Kunci

Sebelum kita membahas pokok bahasan utama bab ini, ada baiknya kita meninjau beberapa topik terkait secara singkat. Pertama, hampir semua pembanding, penggabung, dan pemisah menghasilkan nilai fungsi. Nilai-nilai ini kemudian digunakan sebagai argumen untuk fungsi M lainnya. Oleh karena itu, memahami konsep yang terkait dengan fungsi dan pemanggilannya sangatlah penting. Kedua, apa kesamaan antara fungsi pembanding dan enumerasi Order.Type?

1. Pemanggilan Fungsi

Untuk menggunakan fungsi secara efektif dalam kode kita, memahami ekspresi invoke adalah kuncinya. Ekspresi invoke terdiri dari serangkaian tanda kurung yang secara opsional dapat berisi list argumen. Ekspresi ini memicu eksekusi badan fungsi, yang mengembalikan nilai atau menghasilkan kesalahan. Untuk pemahaman fungsi yang lebih mendalam, silakan lihat Bab 9, Parameter dan Fungsi Gustom.

Cara paling mudah untuk memanggil fungsi adalah dengan menggunakan nama lengkapnya. Setelah nama, Anda menyertakan ekspresi invoke yang mencantumkan argumen

untuk semua parameter yang diperlukan. Misalnya, dengan fungsi Text.Contains, Anda dapat memeriksa apakah hello adalah bagian dari Hello World. Kedua masukan teks ini memenuhi parameter yang diperlukan fungsi:

```
let
    Result = Text.Contains("Hello World", "hello")
in
    Result
```

Ketika pemanggilan fungsi menghasilkan fungsi baru, proses kualifikasi penuh dapat dilanjutkan dengan menambahkan ekspresi pemanggilan yang mencantumkan semua argumen yang diperlukan, sebagaimana diperlukan. Misalnya, discountFunction ini akan menghasilkan fungsi baru yang selanjutnya memerlukan argumen numerik sebagai input. Pemanggilan yang sepenuhnya memenuhi syarat ditunjukkan pada langkah Result:

```
let
    discountFunction = (discountRate as number) as
        function => (sales as number) as number => sales
        * (1 - discountRate),
    Result = discountFunction(0.1)(100)
in
```

2. Beberapa kesalahan umum

Berikut ini adalah beberapa kesalahan umum yang perlu Anda waspadai:

a. **Tanda kurung penutup hilang?**: Sebagian besar IDE mendukung penyorotan tanda kurung yang cocok. Gunakan fitur ini untuk mengidentifikasi lokasi tanda kurung yang hilang dengan cepat.
b. **Jumlah argumen salah?**: Lihat IntelliSense atau dokumentasi fungsi untuk memastikan Anda memberikan jumlah argumen yang benar.
c. **Jenis argumen tidak cocok?**: Jenis argumen harus kompatibel dengan jenis parameter yang sesuai. Lihat IntelliSense atau dokumentasi untuk memastikan Anda memberikan jenis yang benar untuk setiap argumen.

d. **Salah mengeja nama fungsi atau salah menggunakan huruf besar?**: Gunakan IntelliSense dan pilih fungsi dari list saran untuk menghindari kesalahan tersebut.

3. Penutupan (Closures)

Penutupan mungkin terdengar teknis atau seperti istilah yang diperuntukkan bagi para ahli dan pengembang, tetapi kita jamin, penutupan tidak sesulit yang terlihat. Mari kita tunjukkan kepada Anda bagaimana penutupan dapat membuat kode Anda lebih cerdas. Penutupan adalah fungsi yang mampu mempertahankan nilai argumen untuk parameter dari cakupan luarnya (induk), bahkan setelah fungsi luar tersebut telah menyelesaikan eksekusi. Kemampuan untuk "mengingat" ini membedakan penutupan dari fungsi biasa, mempertahankan nilai untuk pemanggilan di masa mendatang, terlepas dari lingkungan tempat penutupan tersebut dipanggil. Lihat contoh kode berikut, serta Gambar 3.1. Anda dapat memperoleh pemahaman yang lebih mendalam tentang penutupan, cakupan, konteks, dan lingkungan di Bab 7, Gonceptualizing M.

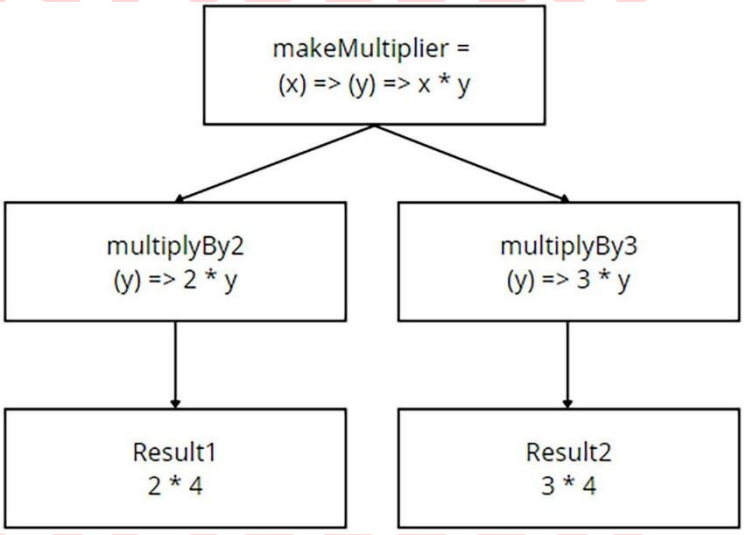
```
  let
  makeMultiplier = (x) => (y) => x * y, multiplyBy2 =
makeMultiplier(2), multiplyBy3 = makeMultiplier(3),
  Result1 = multiplyBy2(4),  // Result will be 8
  Result2 = multiplyBy3(4)      // Result will be 12
  in
  [Result1=Result1, Result2=Result2]
```

Mari kita uraikan contoh kode ini untuk memperoleh pemahaman yang lebih baik tentang apa artinya:

a. *makeMultiplier* adalah fungsi yang menerima satu argumen, x. Fungsi ini, *makeMultiplier*, mengembalikan fungsi lain. Saat diinisialisasi, fungsi ini mengembalikan fungsi baru ini: (y) => x * y.
b. *MultiplyBy2* dan *multiplyBy3* adalah closure yang dihasilkan dengan memanggil fungsi makeMultiplier dengan argumen 2 dan 3, berturut-turut. Masing-masing closure ini adalah fungsi yang "mengingat" nilai x sejak pembuatannya, 2 untuk

multiplyBy2 dan 3 untuk *multiplyBy3*. Kemampuan untuk mengingat variabel dari lingkungan tempat variabel tersebut dibuat adalah konsep dasar closure.

c. Terakhir, penutupan ini dapat dipanggil. Misalnya, ketika *multiplyBy2*(4) dipanggil, fungsi tersebut menggunakan nilai x yang "diingat" (2) dan nilai baru untuk y (4) untuk mengembalikan 8. Demikian pula, fungsi *multiplyBy3*(4) mengembalikan 12:



Gambar 3.1 Tampilan Diagram Kode Kita

Dengan memahami closure, Anda dapat membuat kode yang lebih modular, dapat digunakan kembali, dan dinamis. Hal ini memungkinkan Anda untuk merancang solusi dan fungsi kustom yang disertai dengan tingkat adaptasi tertentu, yang mampu menangani berbagai persyaratan yang lebih luas.

4. Fungsi Tingkat Tinggi

Jika kita menyelami lebih dalam kemampuan bahasa M, kita akan menemukan konsep fungsi tingkat tinggi, yang banyak digunakan di berbagai bahasa pemrograman. M digambarkan sebagai bahasa fungsional dan menunjukkan ciri-ciri tertentu; misalnya, fungsi dianggap sebagai warga negara kelas satu, yang berarti fungsi tersebut dapat ditetapkan ke variabel, diteruskan sebagai argumen ke fungsi lain, dan dikembalikan sebagai hasil.

Hal ini memungkinkan fungsi tersebut untuk dimanipulasi dan diedarkan seperti nilai lainnya. Istilah "higher-order function" digunakan untuk menggambarkan fungsi yang dapat menerima fungsi lain sebagai argumen atau mengembalikan fungsi sebagai hasil, seperti makeMultiplier dari contoh sebelumnya.

Banyak fungsi pustaka standar memiliki parameter tipe fungsi, yang mengkualifikasinya sebagai fungsi tingkat tinggi, seperti List.Transform, misalnya, yang mengambil list dan fungsi transformasi sebagai parameternya. List.Transform menerapkan fungsi transformasi ke setiap elemen list, yang menghasilkan list dengan elemen yang baru dibentuk:

```
let
    defaultDiscount = 0.1,
    d
    nction = (discountRate as number) as function => (sales
        as number) as number => sales * (1 - discountRate),
    applyDefaultDiscount = discountFunction(defaultDiscount),
    myList = {100, 900, 200, 500},
    Result = List.Transform( myList, applyDefaultDiscount )
in
    Result
```

Bagaimana cara kerja kode ini? Mari kita uraikan:

a. Variabel *default Discount* ditetapkan ke 0,1 atau 10%.
b. *discountFunction* menerima satu argumen, *discountRate*. Ini adalah fungsi yang mengembalikan fungsi (penjualan sebagai angka) sebagai angka => penjualan * (1 – *discountRate*).
c. Variabel *applyDefault Discount* adalah penutupan yang dihasilkan dengan memanggil fungsi *discountFunction* dengan nilai argumen, *default Discount*. Fungsi ini akan menerapkan nilai untuk *default Discount* sejak pembuatannya.
d. Variabel *myList* adalah list yang berisi empat nilai: 100, 900, 200, dan 500.
e. Terakhir, fungsi *List.Transform* memanggil penutupan, *applyDefault Discount*, ke setiap elemen di myList. Ini mengembalikan list baru dengan nilai yang diubah.

Fungsi dapat mengubah elemen individual dari suatu list atau tabel, dan fungsi tingkat tinggi memungkinkan transformasi ini diterapkan ke semua elemen secara efisien dan ringkas. Dalam kasus ini, List.Transform secara otomatis meneruskan argumen yang diperlukan ke penutupan applyDefault Discount pada saat pemanggilan. Dalam terminologi ilmu komputer, perilaku ini sering disebut sebagai "callback function invocation" atau sekadar "callback invocation". Namun, dokumentasi bahasa M Power Query tidak secara resmi menetapkan proses ini dengan istilah tertentu.

Berikut ini adalah contoh yang menunjukkan bagaimana fungsi tingkat tinggi dapat membuat kode lebih mudah dibaca dengan menyederhanakan cara nilai argumen diteruskan, bahkan saat ada beberapa argumen:

```
let
    Result = Text.Contains(
              "Hello World", "hello", Comparer.OrdinalIgnoreCase
         )
in
    Result
```

Pembanding digunakan sebagai argumen untuk fungsi Text.Contains tingkat tinggi. Fungsi ini secara otomatis menyediakan dua parameter inputnya ke pembanding, sehingga tidak perlu lagi mengkualifikasinya sebagai argumen secara eksplisit. Kita akan menjelajahi berbagai pembanding yang tersedia dalam bahasa M secara lebih mendalam. Sebelum melakukannya, mari kita jelajahi fungsi anonim atau fungsi kustom yang ditentukan pengguna terlebih dahulu. Fungsi ini bukan asli bahasa M tetapi dapat dibuat oleh pengguna dengan cepat

5. Fungsi nonim

Fungsi anonim adalah fungsi tanpa nama yang dapat didefinisikan secara inline dalam fungsi lain. Fungsi ini berguna ketika logika fungsi kustom diperlukan untuk tujuan tertentu dan tidak perlu digunakan kembali di tempat lain. Berikut ini contohnya:

```
let
    Result = ((x, y) => x + y)(3, 4)
in
    Result
```

Fungsi anonim dapat terlihat menakutkan, terutama jika parameternya memiliki nama yang tidak deskriptif. Mari kita uraikan contoh sebelumnya:

a. Fungsi *(x, y) => x + y* memerlukan dua nilai *(x, y)* dan akan menerapkan operator + pada keduanya, untuk menghasilkan satu nilai atau menimbulkan kesalahan.
b. Fungsi diapit oleh serangkaian tanda kurung, untuk urutan prioritas, dan akan mengevaluasi ekspresi fungsi untuk menghasilkan nilai fungsi.
c. Operator pemanggilan yang berisi list argumen, (3, 4), ditambahkan. Operator ini mengevaluasi isi fungsi untuk menghasilkan nilai 7.

Ketahui bahwa dalam kebanyakan kasus, evaluasi ekspresi fungsi dipisahkan dari pemanggilan fungsi. Anda dapat mempelajari semua yang perlu diketahui tentang fungsi di Bab 9, Parameter dan Fungsi Gustom.

Sejauh ini, kita telah menyentuh konsep-konsep seperti pemanggilan fungsi, penutupan, dan fungsi tingkat tinggi. Sekarang saatnya untuk melihat apa yang sama antara fungsi enumerasi dan pembanding Order.Type.

6. Nilai Pemesanan

Apakah Anda mengurutkan buku di rak buku Anda berdasarkan satu atau beberapa kriteria, seperti judul, penulis, genre, dan mungkin ukuran atau warna? Demikian pula, dalam Power Query, fungsi enumerasi Order.Type dan pembanding memungkinkan kita untuk mengurutkan data berdasarkan kriteria tertentu.

Dalam bahasa M, fungsi pembanding dan enumerasi Order.Type digunakan untuk mengurutkan nilai. Namun, keduanya tidak sama:

a. Enumerasi menentukan serangkaian nilai bernama, mengatur konstanta terkait untuk kode yang lebih mudah dibaca dan dipelihara. Nilai bernama ini memperjelas tujuannya, menyederhanakan proses pengaturan opsi atau status dalam kode Anda. Enumerasi diperkenalkan di Bab 4 buku ini. Enumerasi *Order.Type* adalah serangkaian nilai yang telah ditentukan sebelumnya yang mewakili berbagai jenis pengurutan. Metode yang mudah untuk menentukan urutan nilai yang diinginkan adalah sebagai berikut:

Tabel 3. 2 Menentukan Urutan Nilai

Order.Type	Value	Direction
`Order.Ascending`	`0`	Secara numerik dari rendah ke tinggi, atau secara alfabetis menaik
`Order. Descending`	`1`	Secara numerik dari tinggi ke rendah, atau secara alfabetis menurun

b. Fungsi pembanding, di sisi lain, digunakan untuk mengevaluasi dua nilai dan mengembalikan bilangan bulat yang menunjukkan urutan relatifnya. Fungsi-fungsi ini memberi Anda kendali atas cara item dibandingkan saat mengurutkan tabel atau list. Fitur ini sangat dapat disesuaikan, memungkinkan Anda untuk menetapkan aturan dan kriteria pengurutan tertentu.

Singkatnya, fungsi pembanding menawarkan metode yang komprehensif dan dapat disesuaikan untuk membandingkan dan mengurutkan nilai, sementara enumerasi Order.Type memudahkan untuk menetapkan arah pengurutan. Meskipun keduanya digunakan untuk mengatur nilai, keduanya memiliki peran yang berbeda dan tidak dapat digunakan secara bergantian dalam bahasa M.

D. Pembanding

Fungsi pembanding memainkan peran penting dalam pemrosesan data, menyediakan sarana untuk mengevaluasi dan menetapkan urutan relatif atau menentukan kesetaraan. Ambil contoh, string Hello, hello, dan HELLO. Anda mungkin mempertanyakan apakah

keduanya identik. Namun, fungsi pembandinglah yang dapat menentukan aturan – seperti kepekaan huruf besar/kecil – tentang cara string ditafsirkan dan dibandingkan.

Peran fungsi pembanding adalah menyediakan metode untuk membandingkan nilai dan menentukan urutan relatif atau kesetaraannya. Fungsi pembanding digunakan dalam berbagai skenario, seperti mengurutkan atau melakukan perbandingan, misalnya, dalam ekspresi bersyarat.

Di Power Query, fungsi-fungsi ini sebagian besar digunakan sebagai argumen opsional untuk fungsi tingkat tinggi yang memerlukan operasi perbandingan atau persamaan. Beberapa di antaranya tercantum di sini, tetapi ada lebih banyak fungsi yang dilengkapi dengan parameter pembanding atau or equationCriteria atau comparisonCriteria:

Tabel 3.3 Fungsi Pembanding

Text functions	**List functions**	**Table functions**
`Text.Contains`	`List.Sort`	`Table.Sort`
`Text.StartsWith` or `EndsWith`	`List.Max` or `Min`	`Table.Max` or `Min`
`Text.PositionOf`	`List.RemoveMatchingItems`	`Table.Group`

Dalam bahasa M, terdapat empat fungsi pembanding bawaan. Kecuali Comparer.Equals, semuanya mengembalikan nilai fungsi.

1. *Comparer.Equals*

Comparer.Equals mengembalikan nilai logika setelah mengevaluasi kesetaraan antara dua nilai. Fungsi ini memerlukan tiga argumen: pembanding untuk memandu evaluasi dan dua nilai yang akan dibandingkan. Anda dapat menentukan pembanding bawaan, seperti *Comparer.Ordinal*, *Comparer.OrdinalIgnoreCase*, atau *Comparer.FromCulture*:

```
let
    myTable = Table.FromRows(
        {
            { "APPLE", "apple" },
            { "apple", "apple" },
            { "42", 42.0 },
            { 42, 42.0 }
        }
    ),
    equivalenceChecks = Table.AddColumn(myTable, "equivalenceCheck",
        each Comparer.Equals(Comparer.FromCulture("en-US"),
            [Column1],
            [Column2]
        ), type logical
    )
in
    equivalenceChecks
```

Berikut adalah rincian kode yang hasilnya digambarkan pada Gambar 11.2:

a. Variabel *myTable* menghasilkan tabel dari list dengan list nilai baris.
b. Fungsi *Table.AddColumn* digunakan untuk menambahkan kolom baru ke *myTable*.
c. Kolom baru tersebut diberi nama *equivalenceCheck*.
d. Kata kunci each memberi tahu *Power Query* untuk menerapkan fungsi berikut ke setiap baris di *myTable*.
e. Untuk menentukan kesetaraan nilai, kita menyediakan *Comparer.FromCulture("en-US")*, yang akan membandingkan nilai di Kolom1 dengan nilai di Kolom2 menurut aturan budaya Inggris (Amerika Serikat) dan menghasilkan nilai Boolean:

	Column1	Column2	equivalenceCheck
1	APPLE	apple	FALSE
2	apple	apple	TRUE
3	42	42	FALSE
4	42	42	TRUE

Gambar 3.2 Mengilustrasikan Pemeriksaan Kesetaraan Pembanding

2. *Comparer.Ordinal*

Comparer.Ordinal mengembalikan fungsi pembanding yang mengambil dua nilai argumen. Fungsi ini kemudian menerapkan aturan ordinal untuk perbandingan, yang bergantung pada nilai kode Unicode dari karakter yang terlibat. Sebagai ilustrasi, kueri ini menghasilkan tabel selebar dua kolom:

```
let
    myTable = Table.FromRows(
        {
            { "A" },
            { "APPLE" },
            { "Apple" },
            { "B" },
            { "a" },
            { "apple" }
        },
        type table[ Value1=text]
    ),
    unicodeValue = Table.AddColumn( myTable, "Unicode code value",
        each Text.Combine(
            List.Transform( Text.ToList( [Value1] ), (v)=>
                Text.From( Character.ToNumber(v)) ), ", "
     ), type text)
in
    unicodeValue
```

Berikut adalah rincian kode yang hasilnya digambarkan pada Gambar 11.3:

a. Variabel *myTable* menghasilkan tabel dari list dengan list nilai baris.
b. Fungsi *Table.AddColumn* digunakan untuk menambahkan kolom baru ke *myTable*. Kolom baru ini diberi nama nilai kode Unicode.
c. *Text.ToList([Value1])* mengambil teks dari kolom *Value1* dan mengubah *string* tersebut menjadi list yang setiap item listnya adalah satu karakter dari teks asli.
d. *List karakter* ini dimasukkan ke dalam fungsi List.Transform, yang memungkinkan kita menerapkan operasi tertentu ke setiap karakter dalam list.
e. *(v)=> Text.From(Character.ToNumber(v))* adalah tempat keajaiban terjadi. Setiap karakter (v) dalam list diteruskan ke Character.ToNumber(v), yang menghasilkan nilai kode

Unicode-nya, yaitu angka. Kemudian, *Text.From* mengubah angka tersebut kembali menjadi nilai teks.

f. Setelah mengubah setiap karakter, kita akan mendapatkan list nilai teks. *Text.Combine* mengambil bagian-bagian ini dan menyusunnya menjadi satu *string*, memisahkan setiap nilai dengan koma dan spasi:

	Value1	Unicode code value
1	A	65
2	APPLE	65, 80, 80, 76, 69
3	Apple	65, 112, 112, 108, 101
4	B	66
5	a	97
6	apple	97, 112, 112, 108, 101

Gambar 3.3 Nilai kode *Unicode* untuk setiap karakter di *Value1*

Kolom pertama berisi nilai tekstual dan kolom kedua menggambarkan nilai kode Unicode yang menyusun karakter dalam string. Melalui ini, peringkat relatif antara nilai teks dapat ditetapkan.

Kueri ini menggambarkan bagaimana proses pengurutan nilai yang diberi peringkat dilakukan, menempatkannya dalam urutan menaik (dari terendah ke tertinggi):

Berikut adalah rincian kode yang hasilnya ditunjukkan pada Gambar 11.4:

```
let
    myTable = Table.FromRows(
        {
            { "APPLE", "apple" }, { "apple", "apple" }, { "apple",
            "APPLE" }
        },
        type table[ Value1=text, Value2=text]
    ),
    myComparer = Comparer.Ordinal,
    sortValue2 = List.Sort( myTable[Value2], Comparer.Ordinal ),
    equivalenceChecks = Table.AddColumn( myTable,
        "equivalenceCheck", each myComparer(
            [Value1], [Value2]
        ), Int64.Type
    )
in
    equivalenceChecks
```

a. Variabel *myTable* menghasilkan tabel dari list dengan list nilai baris.
b. *Comparer.Ordinal* ditetapkan ke variabel *myComparer*.
c. *sortValue2* menghasilkan list yang diurutkan dari nilai kolom *Value2* setelah menerapkan aturan ordinal untuk memberi peringkat dan mengurutkannya. Untuk memahami urutan item list di *sortValue2*, kita akan membandingkan dua nilai pada setiap baris dan menunjukkan nilai peringkat yang ditetapkan di kolom baru: *equivalenceCheck*.
d. setiap *myComparer([Value1], [Value2])* akan mengevaluasi dua nilai pada setiap baris, menetapkan peringkat relatif saat menerapkan aturan ordinal. Ini mengembalikan bilangan bulat yang menunjukkan urutan relatif di antara keduanya, yang memungkinkan pengurutan dari terendah ke tertinggi:

	Value1	Value2	equivalenceCheck
1	APPLE	apple	-1
2	apple	apple	0
3	apple	APPLE	1

Gambar 3.4 Perbandingan dengan Aturan Ordinal

Fungsi myComparer mengembalikan nilai integer -1, 0, atau 1.

a. Jika nilai yang dikembalikan negatif (-1), artinya nilai pertama harus muncul sebelum nilai kedua dalam hasil yang diurutkan karena memiliki peringkat atau posisi yang lebih rendah, dibandingkan dengan nilai kedua.
b. Jika nilai yang dikembalikan nol (0), artinya nilai pertama dianggap sama dengan nilai kedua dalam hasil yang diurutkan karena memiliki peringkat atau posisi yang sama, dibandingkan dengan nilai kedua. Oleh karena itu, urutannya tidak diatur ulang.
c. Jika nilai yang dikembalikan positif (1), artinya nilai pertama harus muncul setelah nilai kedua dalam hasil yang diurutkan karena memiliki peringkat atau posisi yang lebih tinggi, dibandingkan dengan nilai kedua.

Di panel **Query Settings**, buka bagian **Applied Steps** dan pilih langkah **sortValue2.** Ini akan menampilkan list nilai yang diurutkan berdasarkan input dari kolom kedua tabel. Output mempertahankan urutan yang sama dengan kolom pertama tabel,

selaras dengan peringkat masing-masing seperti yang ditunjukkan di kolom **equivalenceCheck.**

3. *Comparer.OrdinalIgnoreCase*

Comparer.OrdinalIgnoreCase mengembalikan fungsi pembanding yang mengambil dua nilai argumen. Fungsi ini menerapkan aturan perbandingan ordinal tetapi mengabaikan karakter huruf besar-kecil. Fungsi ini memberi peringkat pada nilai-nilai ini, menentukan posisi relatifnya dalam urutan menaik. Item dengan peringkat yang sama mempertahankan urutan aslinya dari input:

```
let
    myTable = Table.FromRows(
        {
            { "APPLE", "apple" }, { "apple", "Apple" }, { "apple",
            "APPLE" }
        },
        type table[ Value1=text, Value2=text]
    ),
    myComparer = Comparer.OrdinalIgnoreCase, sortValue2 = List.Sort(
    myTable[Value2],Comparer.OrdinalIgnoreCase ), equivalenceChecks
    = Table.AddColumn( myTable,
        "equivalenceCheck", each myComparer(
            [Value1], [Value2]
        ), Int64.Type
    )
in
    equivalenceChecks
```

Proses dan struktur kode setara dengan contoh sebelumnya, dengan pengecualian bahwa kita sekarang menerapkan aturan ordinal sambil mengabaikan huruf besar pada karakter. Oleh karena itu, perincian hanya menyoroti perbedaan berikut:

Proses dan struktur kode setara dengan contoh sebelumnya, dengan pengecualian bahwa kita sekarang menerapkan aturan ordinal sambil mengabaikan huruf besar pada karakter. Oleh karena itu, perincian hanya menyoroti perbedaan berikut:

a. *Comparer.Ordinal Ignore Case* ditetapkan ke variabel *my Comparer*.
b. *sortValue2* menghasilkan list yang diurutkan dari nilai kolom *Value2* setelah menerapkan aturan ordinal dan mengabaikan huruf besar untuk memberi peringkat dan mengurutkannya:

	Value1	Value2	equivalenceCheck
1	APPLE	apple	0
2	apple	Apple	0
3	apple	APPLE	0

Gambar 3.5 Perbandingan dengan Aturan Ordinal sambil mengabaikan Kasus

Dalam pengaturan yang tidak peka huruf besar/kecil, kapitalisasi yang berbeda dari kata yang sama, seperti "*apple,*" *"Apple,"* dan *"APPLE,"* dianggap sama. Akibatnya, urutannya tetap sama dalam list yang diurutkan, seperti yang ditunjukkan pada langkah *sortValue2* yang diilustrasikan pada Gambar 3.5.

4. *Comparer.FromCulture*

Comparer.FromCulture mengembalikan fungsi pembanding yang mengambil dua argumen. Yang pertama adalah budaya, yang diwakili oleh tag bahasa yang dikenal luas seperti *en-US* untuk bahasa Inggris di Amerika Serikat. Tag ini digunakan untuk menentukan pengaturan lokal dalam kerangka kerja .NET. Argumen kedua adalah *ignoreCase* Boolean opsional, nilai *true* atau *false* untuk menentukan apakah huruf besar karakter harus diabaikan; jika dihilangkan, nilai *default* nya adalah *false*:

```
let
    myTable = Table.FromRows(
        {
            { "APPLE", "apple" },
            { "apple", "apple" },
            { "apple", "APPLE" }
        },
        type table[ Value1=text, Value2=text]
    ),
    myComparer = Comparer.FromCulture("en-US"), equivalenceChecks =
    Table.AddColumn( myTable,
        "equivalenceCheck", each myComparer(
            [Value1], [Value2]
        ), Int64.Type
    )
in
    equivalenceChecks
```

Untuk menyorot perbedaan antara contoh kode sebelumnya, di sini *Comparer.FromCulture* ditetapkan ke variabel *myComparer*. Gambar 11.6 menunjukkan hasilnya:

	Value1	Value2	equivalenceCheck
1	APPLE	apple	1
2	apple	apple	0
3	apple	APPLE	-1

Gambar 3.6 Perbandingan dengan Aturan yang Peka terhadap Budaya

Ini melakukan perbandingan dua nilai yang peka terhadap budaya berdasarkan aturan yang ditetapkan oleh budaya yang ditentukan. Ini memperhitungkan konvensi budaya seperti aturan pengurutan khusus bahasa dan kepekaan huruf besar/kecil. Khususnya, dalam contoh ini, kata dengan huruf kecil diberi peringkat dan diurutkan sebelum huruf besar, berbeda dengan perbandingan ordinal yang terlihat sebelumnya. Itu karena cara kerja pembanding yang peka terhadap budaya .NET.

Dampak penerapan aturan khusus bahasa menjadi lebih jelas saat membandingkan kata Denmark *Færdig* dengan *Faerdig*, misalnya:

```
let
    Danish = Comparer.FromCulture("da-DK")( "Færdig", "Faerdig"),
    English =  Comparer.FromCulture("en-US")( "Færdig", "Faerdig")
in
    [DK = Danish, US = English]
```

Catatan ini menunjukkan bahwa dalam bahasa Inggris, karakter æ dan ae dianggap identik, sehingga menghasilkan nilai bidang AS sebesar 0. Namun, dalam bahasa Denmark (*da-DK*), karakter-karakter ini dianggap berbeda, sehingga menghasilkan nilai bidang DK sebesar 1:

DK	1
US	0

Gambar 3.7 Mengilustrasikan Perbedaan Budaya dengan Membandingkan Karakter “ae” dan “ae”

E. Kriteria Perbandingan

Sekarang periksa buku-buku di rak Anda; apakah Anda melihat ada yang tidak pada tempatnya? Ambillah. Menentukan lokasi yang tepat memerlukan perbandingan. Anda akan membandingkan buku di tangan Anda dengan buku lain di rak dengan menerapkan kriteria tertentu, mengulangi proses tersebut hingga Anda menemukan tempat yang tepat. Hal yang sama berlaku untuk data. Dengan menetapkan kriteria yang relevan, Anda dapat mengurutkannya dengan cara yang bermakna, seperti mengurutkan nama bulan dari Januari hingga Desember, hari kerja dari Senin hingga Minggu, dan seterusnya.

Kriteria perbandingan dapat digunakan untuk mengatur data menurut logika tertentu. Di bagian ini, kita akan menjelajahi berbagai metode yang disediakan oleh bahasa Power Query M untuk menentukan kriteria perbandingan guna mengurutkan nilai. Kita akan menggunakan fungsi List.Sort dan Table.Sort sebagai contoh untuk menunjukkan bagaimana Anda dapat membuat aturan khusus dan memperoleh kontrol yang tepat atas pengurutan data.

1. Nilai Numerik

Saat mengurutkan data menggunakan *Table.Sort*, Anda dapat menetapkan nilai numerik apa pun untuk menunjukkan urutan pengurutan yang diinginkan. Nilai yang lebih rendah didahulukan daripada nilai yang lebih tinggi, dengan data disusun dalam urutan menaik. Metode ini mudah dan sesuai dengan kasus di mana urutan pengurutan dapat langsung dikaitkan dengan nilai numerik. Berikut ini adalah contohnya:

```
let
    myTable = Table.FromRecords(
        {
            [Label="Moderate", Order=1], [Label="Low", Order=2],
            [Label="High", Order=0]
        },
        type table[Label=text, Order=Int64.Type]
    ),
    sortData = Table.Sort( myTable, each [Order] )
in
    sortData
```

Table.Sort(myTable, each [Order]) akan mengurutkan baris berdasarkan nilai di kolom Order. Secara default, pengurutan ini dilakukan dalam urutan menaik. Ini berarti baris akan disusun dari nilai terendah di kolom Order ke nilai tertinggi, seperti yang diilustrasikan di sini:

	Label	Order
1	High	0
2	Moderate	1
3	Low	2

Gambar 3.8 Hasil dari langkah sortData

2. Menghitung Kunci Sortir

Untuk skenario yang lebih kompleks, Anda dapat menggunakan fungsi untuk menghitung kunci untuk setiap elemen. Kunci tersebut menjadi dasar untuk pengurutan. Metode ini khususnya bermanfaat ketika kunci pengurutan yang diinginkan tidak dapat secara langsung diturunkan dari nilai asli dan memerlukan semacam perhitungan. Pertimbangkan ekspresi ini:

```
List.Sort(
    { "April", "February", "January", "March" }, each
    Date.From( _ & " 2020" ))
```

Ada list dengan nama bulan dalam urutan abjad: April, February, January, dan March. Untuk menempatkannya dalam urutan kronologis, kita dapat memperoleh nilai tipe tanggal. Di sini, kunci pengurutan dihitung dengan menambahkan tahun ke setiap nama bulan dan memanfaatkan fungsi Date.From untuk mengubah string tersebut menjadi nilai tipe tanggal. List.Sort kemudian akan mengurutkan nama bulan dalam urutan kronologis, bukan urutan abjad, seperti yang ditunjukkan pada Gambar 3.9:

	List
1	January
2	February
3	March
4	April

Gambar 3.9 Mengurutkan Nama Bulan secara Kronologis

Anda mungkin menemukan situasi di mana nilai ordinal tidak dapat dihitung secara langsung. Dalam kasus seperti itu, pencarian mungkin diperlukan untuk menetapkan urutan tertentu. Pertimbangkan kueri ini, yang mengembalikan tabel selebar satu kolom, myTable, yang menunjukkan nilai Moderate, Low, dan High:

```
let
    myTable = Table.FromColumns({
        { "Moderate", "Low", "High" }},
  type table[Label=text]), sortData = Table.Sort( myTable,
  each Record.FieldOrDefault([High=0, Moderate=1, Low=2],
  [Label]))
in
    sortData
```

Bahkan jika kolom ini merupakan bagian dari tabel yang sangat besar, varians dalam kolom tersebut terbatas pada tiga nilai ini. Itu berarti kita dapat dengan mudah membuat Record dengan cepat untuk menentukan urutan yang diinginkan:

a. Pastikan setiap label cocok dengan nama kolom dalam *Record*.
b. Tetapkan setiap kolom nilai numerik; perlu diingat bahwa angka diurutkan dari rendah ke tinggi. Kita akan menggunakan 0, 1, 2, tetapi 100, 200, 300 akan mencapai hasil yang sama.
c. Gunakan inisialisasi *Record*, serangkaian tanda kurung siku, untuk membuat *Record* dan pisahkan setiap kolom dengan koma. Ekspresi *Record* akan terlihat seperti ini:

```
[High=0, Moderate=1, Low=2]
```

Perhatikan bahwa jika Anda ingin menggunakan ekspresi Record ini di lebih dari satu kueri, Anda harus menyimpannya sebagai kueri terpisah dalam berkas Anda. Itu akan

memungkinkan Anda untuk menggunakannya berulang kali tanpa menduplikasi kodenya. Namun, dalam contoh ini, kita akan memanfaatkannya hanya sekali; oleh karena itu, kita akan memasukkan Record langsung ke dalam operasi pengurutan:

a. Panggil *Table.Sort* untuk mengubah urutan baris dalam *myTable*.
b. Terapkan fungsi berikut:

```
each Record.FieldOrDefault( [High=0, Moderate=1, Low=2],
[Label] )
```

Fungsi Record ini akan mencoba memetakan nama bidang dalam Record, yang diberikan sebagai argumen pertamanya, ke label (teks) yang diberikan sebagai argumen kedua, untuk memperoleh nilai bidangnya. Karena itu adalah angka, maka ia akan menentukan urutannya, seperti yang diilustrasikan pada Gambar 3.10.

	Label
1	High
2	Moderate
3	Low

Gambar 3.10 Mengurutkan Nilai dengan Bantuan Catatan Pencarian

Sekarang setelah Anda memahami cara menghitung kunci sortir, mari kita jelajahi cara menerapkan kunci ini beserta arah sortir yang memungkinkan kita menyempurnakan penataan data untuk memenuhi persyaratan yang sangat spesifik.

3. List dengan Kunci dan Urutan

Untuk memilih kunci tertentu secara bersamaan untuk menyortir dan mengontrol urutan penyortiran, Anda dapat menggunakan list list sebagai kriteria penyortiran. Setiap list bertingkat harus berisi dua elemen: kunci untuk mengurutkan dan arah penyortiran yang diinginkan, yang ditentukan menggunakan *Order.Type (Order.Ascending = 0* atau *Order.Descending = 1)*.

Metode ini memberikan fleksibilitas saat logika penyortiran khusus diperlukan. Mari kita lihat contohnya:

```
let
  myTable = Table.FromRecords({
    [Category = "Category A", SubCategory = "SubCategory 11"],
    [Category = "Category C", SubCategory = "SubCategory 20"],
    [Category = "Category A", SubCategory = "SubCategory 1"],
    [Category = "Category B", SubCategory = "SubCategory 2"],
    [Category = "Category C", SubCategory = "SubCategory 9"],
    [Category = "Category B", SubCategory = "SubCategory 5"],
    [Category = "Category A", SubCategory = "SubCategory 10"]
  }, type table [Category=text,
  SubCategory=text]), sortedTable =
  Table.Sort( myTable,
    {
      {each Text.Lower( Text.AfterDelimiter([Category], " ")), 1},
      {each Number.From(Text.Select([SubCategory], {"0".."9"})), 0}
    })
in
  sortedTable
```

Mari kita uraikan isi dari setiap list penyortiran kolom bersarang:

```
{each Text.Lower( Text.AfterDelimiter([Category], " ")), 1}
```

a. *Text.AfterDelimiter([Category], " ")*:
   Gunakan Text.AfterDelimiter untuk mengekstrak teks dari kolom Category setelah karakter spasi pertama.
b. Gunakan *Text.Lower* untuk mengubah *string* Category menjadi huruf kecil.
c. Urutkan sebagai *descending*, dengan memberikan 1 atau *Order.Descending*:

```
{each Number.From(Text.Select([SubCategory], {"0".."9"})), 0}
```

d. Gunakan *Text.Select* untuk mengekstrak angka dari kolom SubCategory.
e. Gunakan *Number.From* untuk mengubah *string* tersebut menjadi angka.
f. Urutkan sebagai menaik, dengan memberikan 0 atau *Order.Ascending*.

Tangkapan layar berikut menunjukkan hasilnya:

	Category	SubCategory
1	Category C	SubCategory 9
2	Category C	SubCategory 20
3	Category B	SubCategory 2
4	Category B	SubCategory 5
5	Category A	SubCategory 1
6	Category A	SubCategory 10
7	Category A	SubCategory 11

Gambar 3.11 Mengurutkan Nama dalam Urutan Menurun dan Nomor Subkategori dalam Urutan Menaik

4. Pembanding Custom dengan Logika Kondisional

Untuk mengendalikan proses perbandingan secara penuh, Anda dapat membuat fungsi pembanding kustom yang menggunakan dua argumen. Fungsi ini akan mengembalikan -1, 0, atau 1, tergantung pada hubungan antara input kiri dan kanan. Meskipun contoh berikut disederhanakan, contoh ini menyoroti bagaimana Anda dapat menggunakan logika kondisional untuk menangani skenario pengurutan yang lebih kompleks:

```
let
    customComparer = (x, y) =>
        if Number.IsEven( x ) and x > y then -1 else if x < y
        then 1
        else 0,
    listNumbers = {1..10},
    sortedList = List.Sort( listNumbers, customComparer
    )
in
    sortedList
```

Ekspresi customComparer dimulai dengan (x, y) =>, yang menunjukkan bahwa itu adalah fungsi dengan dua parameter, x dan y, yang akan mengembalikan nilai fungsi.

Isi fungsi hanya akan dievaluasi saat dipanggil. Isi fungsi tersebut berisi pernyataan kondisional yang memprioritaskan angka genap daripada angka ganjil, di mana x lebih besar dari y, dalam urutan item list yang ditemukan. Kasus di mana x lebih kecil dari y ditempatkan kemudian dalam urutan penyortiran.

Saat diterapkan ke list dengan nomor berurutan yang berjalan dari 1 hingga 10, ia mengembalikan hasil yang digambarkan dalam Gambar 11.12:

	List
1	10
2	8
3	6
4	4
5	2
6	1
7	3
8	5
9	7
10	9

Gambar 3.12 Pembanding Khusus dengan Logika Kondisional untuk Mengurutkan Nilai

5. Pembanding Custom dengan Value.Compare

Namun, Anda juga dapat membuat fungsi pembanding kustom dengan dua parameter untuk mengendalikan proses perbandingan sepenuhnya. Gunakan fungsi Value.Compare untuk mendelegasikan logika perbandingan kustom guna menghasilkan hasil -1, 0, atau 1 sesuai kebutuhan. Metode ini sangat bermanfaat saat aturan perbandingan yang kompleks diperlukan. Pertimbangkan contoh berikut:

```
let
  weekdayComparer = (a, b) =>
    let weekstartDate = #date(2024, 1, 1), weekdayCulture = "en-US",
        weekdayRecord = Record.Combine(
          List.Transform(
            List.Dates( weekstartDate, 7, Duration.From(1)), each
            Record.FromList(
            {Date.DayOfWeek( _, Date.DayOfWeek( weekstartDate, 0
            ))},
            {Date.DayOfWeekName( _, try weekdayCulture ?? "en-US"
                otherwise "en-US" )}
    ))) in Value.Compare(
      Record.Field( weekdayRecord, a[colName] ), Record.Field(
      weekdayRecord, b[colName] )
  `
```

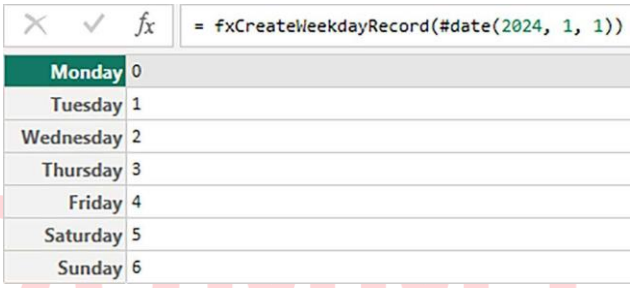
```
        weekdayRecord, b[colName] )
      ),
    myData = Table.FromColumns({
      {"Thursday", "Sunday", "Saturday", "Monday", "Tuesday",
"Friday", "Wednesday"}
      }, type table [colName=text]
    ),
    sortedData = Table.Sort(myData, weekdayComparer)
in
    sortedData
```

Dalam contoh ini, weekdayComparer menyertakan fungsi bersarang yang menghasilkan weekdayRecord, yang peka terhadap budaya dan menunjukkan hari-hari dalam seminggu beserta jumlahnya. Budaya default -nya adalah en-US. Jika diperlukan eksplorasi lebih lanjut, kode M untuk bagian fungsi tersebut dicantumkan secara terpisah di sini sebagai fxCreateWeekdayRecord:

```
  let
    fxCreateWeekdayRecord = ( weekstartDate as date, optional
        weekdayCulture as text
      ) as record => Record.Combine(
        List.Transform(
          List.Dates( weekstartDate, 7, Duration.From(1)), each
          Record.FromList(
          { Date.DayOfWeek( _, Date.DayOfWeek( weekstartDate,
              Day.Sunday )) },
          { Date.DayOfWeekName( _, try weekdayCulture ?? "en-US"
              otherwise "en-US" ) }
        )
)),
    fxDocumentation = [
        Documentation.Name = " fxCreateWeekdayRecord ",
        Documentation.Description = " Returns a record where Fieldnames
        correspont to the weekday name, Fieldvalues correspont to the
        weekday number.
        'weekstartDate' is considered to be the first day of the week
        'weekdayCulture' is optional and defaults to 'en-US'. ",
        Documentation.Author = " Melissa de Korte ",
        Documentation.Version = " 1.0 "
]
In
    Value.ReplaceType( fxCreateWeekdayRecord, Value.ReplaceMetadata(
        Value.Type( fxCreateWeekdayRecord ),
        fxDocumentation
      )
    )
```

Seperti yang terlihat pada Gambar 3.13, semua fungsi fxCreateWeekdayRecord ini hanya memerlukan nilai tanggal

tunggal yang menandai hari pertama dalam seminggu dan secara opsional sebuah budaya:



Gambar 3.13 Output *fxCreateWeekdayRecord*, sebuah *Record*

Saat bekerja dengan kriteria perbandingan untuk mengatur data, penting untuk memahami kemampuan adaptasinya terhadap berbagai situasi. Teknik yang dibahas di sini, seperti hanya menggunakan nilai numerik untuk mengurutkan atau memanfaatkan fungsi yang lebih canggih untuk menghitung kunci pengurutan, menyoroti beragam kotak peralatan yang tersedia.

Kemampuan untuk membuat fungsi pembanding khusus, untuk menyediakan metode untuk membandingkan nilai dan menentukan urutan atau kesetaraan relatifnya, memungkinkan perbandingan yang rumit dan menambahkan lapisan kecanggihan lain pada proses tersebut.

F. Kriteria Perbandingan

Dalam dunia data, menentukan kesetaraan tidak selalu mudah. Kriteria persamaan memungkinkan kita mengidentifikasi perbedaan atau kesamaan antara dua nilai.

Kriteria persamaan digunakan untuk menentukan kesetaraan dan mencocokkan data dalam tabel atau list menurut logika tertentu. Di bagian ini, kita akan mengeksplorasi berbagai cara untuk menentukan kriteria persamaan dalam bahasa M. Kita akan menggunakan fungsi List.Contains untuk mengilustrasikan bagaimana metode ini membantu menetapkan kontrol atas pengujian kesetaraan.

1. Pembanding Default

Dalam bab ini, kita telah membahas peran fungsi pembanding secara panjang lebar, yaitu menyediakan metode untuk membandingkan nilai, menentukan urutan relatif dan kesetaraannya. Oleh karena itu, Anda cukup memberikan pembanding default sebagai equationCriteria. Di sini, pembanding yang dipilih bertindak seperti buku aturan bawaan yang diikuti List.Contains saat membandingkan nilai. Dengan mengabaikan sensitivitas huruf besar/kecil, New York dan New York dianggap sama:

```
List.Contains(
    {"New York", "London", "Tokyo"}, "new york",
    Comparer.OrdinalIgnoreCase
)
//Result: TRUE
```

2. Pembanding Custom

Untuk skenario yang lebih rumit, Anda dapat membuat pembanding khusus dengan dua parameter. Hal ini memungkinkan kontrol penuh atas proses perbandingan dengan menetapkan aturan kesetaraan Anda sendiri. Misalnya, fungsi ini memverifikasi apakah item list (x) berisi substring tertentu (y):

```
List.Contains(
    {"New York", "London", "Tokyo"}, "yO",
    (x, y)=> Text.Contains( x, y, Comparer.OrdinalIgnoreCase)
)
//Result: TRUE
```

3. Pemilih Kunci

Pemilih kunci memungkinkan Anda untuk fokus pada bidang atau pilihan bidang tertentu, mengarahkan perhatian fungsi ke area yang ditentukan ini. Misalnya, jika Anda bekerja dengan list Record dan hanya tertarik untuk mencocokkan bidang 'ID', pemilih kunci akan memungkinkan fokus hanya pada bidang tersebut, mengabaikan semua nilai bidang lainnya:

```
List.Contains(
{ [ID=1, Name="John", Role="Manager"], [ID=2, Name="Jane",
   Role="Developer"], [ID=3, Name="Sam", Role="Designer"] },
 [ID=1, Name="Anne", Role="Analyst"], each [ID]
)
//Result: TRUE
```

4. Menggabungkan Pemilih Kunci dan Pembanding

Saat Anda menggabungkan pemilih kunci dan pembanding, Anda dapat fokus pada bidang tertentu (berkat pemilih kunci) dan menerapkan aturan perbandingan tertentu (berkat pembanding). Di sini, kita melakukan pencocokan tanpa memperhatikan huruf besar/kecil pada dua bidang Record:

```
List.Contains(
    {
        [ID=1, Name="John", Role="Manager"], [ID=2, Name="Jane",
        Role="Developer"], [ID=3, Name="Sam", Role="Designer"]
    },
    [ID=1, Name="Anne", Role="manager"],
    {
        each Record.SelectFields(_, {"ID", "Role"}),
        Comparer.OrdinalIgnoreCase
    }
)
//Result: TRUE
```

Kriteria persamaan berguna untuk mencocokkan data dalam tabel atau list. Kita telah memamerkan beberapa teknik yang memungkinkan Anda mengelola pengujian kesetaraan dalam fungsi M. Mirip dengan comparisonCriteria, kemampuan untuk menentukan dan meneruskan pembanding kustom sebagai equalityCriteria menawarkan fleksibilitas yang signifikan dalam menentukan kesetaraan.

Pernahkah Anda menginginkan kemampuan untuk mengoreksi kesalahan dalam sekejap? Replacer menyediakan cara itu untuk data Anda, yang memungkinkan Anda mengganti nilai sebagian atau seluruhnya.

G. Pengganti

Fungsi Replacer digunakan oleh fungsi lain dalam bahasa Power Query M. Ada dua Replacer default , Replacer.ReplaceText dan Replacer.ReplaceValue, yang digunakan sebagai argumen untuk

List.ReplaceValue atau Table.ReplaceValue untuk mengganti nilai. Hal ini memungkinkan pengguna untuk mengganti substring atau nilai tertentu dengan string, nilai, atau ekspresi baru. Fungsi ini umumnya digunakan dalam skenario transformasi data yang mengharuskan modifikasi dan pembersihan data yang tidak konsisten untuk memperoleh keseragaman.

Misalnya, mari kita lihat tabel ini (Gambar 3.14) tempat kita ingin mengganti HR dengan Human Resources di kolom Department:

	Employee name	Department
1	John	Sales
2	Lisa	Marketing
3	David	Finance
4	Sarah	HR
5	Alex	Operations

Gambar 3.14 Sample Data

Berikut cara melakukannya menggunakan **User Interface (UI)**:

1. Pilih kolom Departemen pada tabel.
2. Arahkan ke tab *Transform* pada pita dan pilih **Replace values**.
3. Kotak dialog akan muncul. Masukkan *HR* di kolom untuk nilai yang akan dicari.
4. Masukkan *Human Resources* di kolom untuk nilai yang akan diganti.
5. Di bawah **Advanced Options**, atur **Match entire cell contents** dan konfirmasi.

Langkah Replaced Value ditambahkan ke kueri, sehingga menghasilkan tabel yang dimodifikasi (Gambar 3.15) di mana HR telah diganti dengan Human Resources di kolom Department:

	Employee name	Department
1	John	Sales
2	Lisa	Marketing
3	David	Finance
4	Sarah	Human Resources
5	Alex	Operations

Gambar 3.15 Hasil dari Operasi Nilai yang diganti

Saat Anda melihat kode M di bilah rumus, Anda akan melihat fungsi tingkat tinggi Table.ReplaceValue dipanggil. Fungsi ini menyediakan semua argumen yang diperlukan untuk fungsi dalam, Replacer .ReplaceValue:

Sekarang saatnya untuk memeriksa Replacer secara lebih rinci. Ada dua fungsi Replacer default dalam bahasa M.

```
let
    Example = Table.FromColumns(
        {
            {"John", "Lisa", "David", "Sarah", "Alex"},
            {"Sales", "Marketing", "Finance", "HR", "Operations"}
        },  type table [Employee name=text, Department=text]
    ),
    #"Replaced Value" = Table.ReplaceValue(Example,"HR", "Human
        Resources",Replacer.ReplaceValue,{"Department"})
in
    #"Replaced Value"
```

1. *Replacer.ReplaceText*

Fungsi Replacer.ReplaceText memfasilitasi penggantian pola teks tertentu dengan pola lain dalam string yang disediakan. Fungsi ini memiliki tiga parameter tipe teks: input atau curText tempat penggantian akan terjadi, oldText – pola yang akan diidentifikasi dan diganti – dan terakhir, newText yang akan menggantikannya.

Fungsi ini memindai semua kemunculan oldText dalam curText dan menggantinya dengan newText. Berikut adalah contoh kueri:

```
let
    String1 = "The sun shines, shines brightly.", String2 = "A sudden
    rattle noise startled birds.",
    Example = Table.FromRecords({
      [ curText=String2, oldText="t", newText="f",
        ReplaceText=Replacer.ReplaceText(curText, oldText, newText) ],
      [ curText=String2, oldText="tt", newText="ff",
        ReplaceText=Replacer.ReplaceText(curText, oldText, newText) ],
      [ curText=String1, oldText="shines", newText="radiates",
        ReplaceText=Replacer.ReplaceText(curText, oldText, newText) ],
      [ curText=String1, oldText="shines,", newText="radiates",
        ReplaceText=Replacer.ReplaceText(curText, oldText, newText) ]
      }, let t = type text in type table
       [curText=t, oldText=t, newText=t, ReplaceText=t]
    ),
    ReplaceValue = Table.ReplaceValue( Example, ".", "!",
      Replacer.ReplaceText, {"curText", "ReplaceText"}
)
in
    ReplaceValue
```

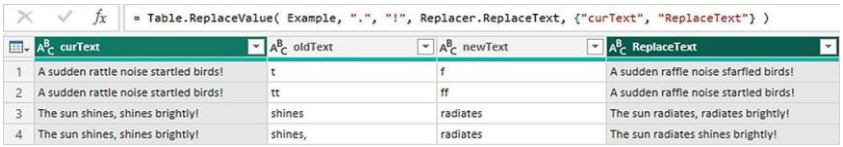
Fungsi Replacer.ReplaceText mengganti setiap kemunculan oldText dengan newText dalam string teks yang disediakan atau curText, yang mendukung penggantian teks parsial. Sintaksnya adalah sebagai berikut: Replacer.ReplaceText (curText, oldText, newText).

Proses ini diilustrasikan dalam langkah contoh kueri, seperti yang ditunjukkan pada Gambar 3.16. Tiga kolom pertama tabel menyajikan nilai input, dan kolom terakhir menampilkan output setelah fungsi Replacer.ReplaceText yang memenuhi syarat dijalankan:

	curText	oldText	newText	ReplaceText
1	A sudden rattle noise startled birds.	t	f	A sudden raffle noise sfarfled birds.
2	A sudden rattle noise startled birds.	tt	ff	A sudden raffle noise startled birds.
3	The sun shines, shines brightly.	shines	radiates	The sun radiates, radiates brightly.
4	The sun shines, shines brightly.	shines,	radiates	The sun radiates shines brightly.

Gambar 3.16 Ganti Contoh Teks

Bila Anda memilih Replace Values dari tab Transform di UI, transformasi diterapkan ke semua kolom yang dipilih. Nama-nama kolom tersebut tercantum dalam argumen terakhir Tabel.ReplaceValue, seperti yang digambarkan di sini pada Gambar 3.17:



= Table.ReplaceValue(Example, ".", "!", Replacer.ReplaceText, {"curText", "ReplaceText"})

	curText	oldText	newText	ReplaceText
1	A sudden rattle noise startled birds!	t	f	A sudden raffle noise sfarfled birds!
2	A sudden rattle noise startled birds!	tt	ff	A sudden raffle noise startled birds!
3	The sun shines, shines brightly!	shines	radiates	The sun radiates, radiates brightly!
4	The sun shines, shines brightly!	shines,	radiates	The sun radiates shines brightly!

Gambar 3.17 Pemanggilan *Implicit Replacer.ReplaceText* dalam *Table.ReplaceValue*

Di kolom curText dan ReplaceText pada tabel, titik telah diganti dengan tanda seru.

2. *Replacer.ReplaceValue*

Fungsi Replacer.ReplaceValue, di sisi lain, digunakan untuk mengganti nilai tertentu dengan nilai atau ekspresi lain. Fungsi ini memiliki tiga parameter bertipe any: input asli (atau curVal) tempat penggantian akan terjadi; oldVal, yang merupakan nilai yang akan diidentifikasi dan diganti; dan newVal yang akan menggantikannya. Fungsi ini membandingkan oldVal dengan curVal dan, jika cocok, mengganti curVal dengan newVal. Berikut ini contoh kueri untuk pemahaman yang lebih baik:

```
let
  Value1 = "Make some noise.", Example = Table.FromRecords({
    [ curVal=Value1, oldVal="noise", newVal="love",
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)],
    [ curVal=Value1, oldVal=Value1, newVal="You rock!",
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)],
    [ curVal=10.1, oldVal=10, newVal=11,
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)],
    [ curVal=10, oldVal=10, newVal=11,
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)],
    [ curVal={1..3}, oldVal={3..5}, newVal=List.First({1..3}),
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)],
    [ curVal={1..3}, oldVal={1..3}, newVal=List.First({1..3}),
      ReplaceValue=Replacer.ReplaceValue(curVal, oldVal, newVal)]
    },  let t = type text in
  type table [curVal=t, oldVal=t, newVal=t, ReplaceValue=any]) in
    Example
```

Fungsi Replacer.ReplaceValue mengganti curVal dengan newVal, jika curVal cocok dengan oldVal. Hal ini diilustrasikan dalam contoh langkah kueri (dilihat pada Gambar 3.18). Tiga kolom pertama tabel menyajikan nilai input, dan kolom terakhir menampilkan output setelah fungsi Replacer.ReplaceValue yang memenuhi syarat telah dipanggil.

	curVal	oldVal	newVal	ReplaceValue
1	Make some noise.	noise	love	Make some noise.
2	Make some noise.	Make some noise.	You rock!	You rock!
3	10,1	10	11	10,1
4	10	10	11	11
5	List	List	1	List
6	List	List	1	1

Gambar 3.18 Ganti Contoh Nilai

Penting untuk menekankan bahwa Replacer.ReplaceValue memerlukan kecocokan yang tepat. Operasi ini dapat dilakukan pada semua jenis nilai, baik primitif maupun terstruktur.

Sebagai ilustrasi, berikut adalah contoh yang melakukan penggantian bersyarat. Pertimbangkan skenario umum seperti ini, tetapi perlu diingat bahwa ada beberapa solusi untuk tantangan tersebut. Anda memiliki tabel dimensi yang berubah perlahan seiring waktu. Contoh kita menampilkan karyawan dan departemennya masing-masing selama periode kerja tertentu:

	Emp	StartDate	EndDate	Dept
1	John	1-1-2020	5-11-2023	Operations
2	Lisa	15-9-2015	10-12-2023	Sales
3	David	1-3-2019	null	Finance
4	John	6-11-2023	null	Sales
5	Lisa	11-12-2023	null	Marketing

Gambar 3.19 Perlahan-lahan Mengubah Dimensi

Salin dan tempel kode ini ke kueri kosong baru untuk mengikuti:

```
let
  SCD = Table.FromColumns(
    {
      {"John", "Lisa", "David", "John", "Lisa"},
      {#date(2020, 1, 1), #date(2015, 9, 15), #date(2019, 3, 1),
      #date(2023,
11, 6), #date(2023, 12, 11) },
      {#date(2023, 11, 5), #date(2023, 12, 10), null, null, null },
      {"Operations", "Sales", "Finance", "Sales", "Marketing" }
    }, type table [Emp=text, StartDate=date, EndDate=date,
    Dept=text]
  ),
  MergedExample = Table.NestedJoin( Table.FromColumns(
      {
        List.Repeat( {#date(2023, 11, 1), #date(2023, 12, 1),
        #date(2024, 1, 1)
}, 3 ),
```

```
            List.Repeat( {"John"}, 3 ) & List.Repeat( {"Lisa"}, 3 ) &
            List.Repeat(
  {"David"}, 3 )
        },  type table [Date=date, Employee=text]
      ), {"Employee"},
      SCD, {"Emp"},
      "Department", JoinKind.LeftOuter
)
in
    MergedExample
```

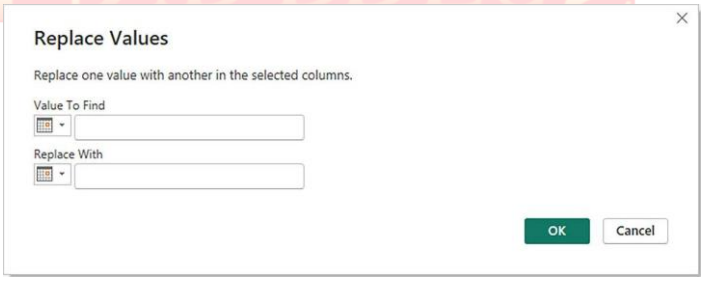
Dimensi ini telah digabungkan dengan tabel fakta untuk menghasilkan tabel dengan semua departemen tempat seorang karyawan aktif karena ada satu kunci gabungan, yaitu karyawan (Gambar 3.20). Untuk catatan awal John pada tanggal 1 November 2023, kita bermaksud mengembalikan Operasional, dan untuk catatan berikutnya, kita bermaksud mengembalikan Penjualan dari tabel bertingkat ini:

	Date	Employee	Department
1	1-11-2023	John	Table
2	1-12-2023	John	Table
3	1-1-2024	John	Table
4	1-11-2023	Lisa	Table
5	1-12-2023	Lisa	Table

Emp	StartDate	EndDate	Dept
John	1-1-2020	5-11-2023	Operations
John	6-11-2023	null	Sales

Gambar 3.20 Penggabungan tersebut Memperoleh semua Riwayat Departemen untuk Setiap Karyawan

Untuk membantu membuat kode, pilih kolom Department, buka tab Transform pada pita, lalu pilih Replace Values. Di kotak dialog, biarkan semuanya apa adanya dan konfirmasikan saja, seperti yang ditunjukkan pada Gambar 3.21:



Gambar 3.21 Biarkan Kotak Dialog Ganti Nilai Kosong

Di dalam bilah rumus, Anda dapat melihat kode yang dibuat:

```
Table.ReplaceValue(MergedExample,"","",Replacer.ReplaceValue,
    {"Department"})
```

Fungsi tingkat tinggi, Table.ReplaceValue, telah memanggil fungsi Replacer.ReplaceValue. Tugas yang tersisa adalah menentukan oldValue, yang merupakan nilai yang kita cari, dan newValue, yang merupakan nilai yang ingin kita ganti dalam ekspresi induk. Ini akan memungkinkan nilai-nilai ini diteruskan ke fungsi Replacer, alih-alih string teks kosong yang disediakan di sana.

Bagian pertama sederhana; kita berada dalam sebuah tabel, dan untuk setiap baris tabel tersebut, kita ingin mengganti nilai yang saat ini ada di sel tersebut. Oleh karena itu, kita perlu mengganti string teks kosong pertama, yang merupakan serangkaian tanda kutip ganda, dengan: each [Department]

```
Table.ReplaceValue(MergedExample, each [Department],"",
    Replacer.ReplaceValue,{"Department"})
```

Kumpulan tanda kutip ganda kedua adalah untuk nilai pengganti. Di sini kita ingin menggunakan Table.SelectRows untuk memilih satu baris dari tabel bersarang, untuk mengekstrak departemen:

```
Table.ReplaceValue( MergedExample, each [Department],
    each Table.SelectRows( [Department], (row) =>
        row[StartDate] <= [Date] and
        (row[EndDate] >= [Date] or row[EndDate] = null)
    )[Dept]?{0}?,
  Replacer.ReplaceValue, {"Department"})
```

Gambar 3.22 menunjukkan nilai yang diperoleh dari struktur bersarang:

	Date	Employee	Department
1	1-11-2023	John	Operations
2	1-12-2023	John	Sales
3	1-1-2024	John	Sales
4	1-11-2023	Lisa	Sales
5	1-12-2023	Lisa	Sales
6	1-1-2024	Lisa	Marketing
7	1-11-2023	David	Finance
8	1-12-2023	David	Finance
9	1-1-2024	David	Finance

Gambar 3.22 Hasil yang Diharapkan

Pada titik ini, jika Anda belum sepenuhnya memahami kode M, pertimbangkan untuk meninjau Bab 4 Jilid 2, yang berfokus pada pengerjaan struktur bersarang dan akan membahas jenis skenario ini secara lebih terperinci.

Kita telah membahas "pengganti" default dalam bahasa M Power Query, yang khususnya berguna untuk tugas modifikasi data dan pembersihan data. Tugas ini dapat mencakup penghapusan karakter yang tidak diinginkan, mengoreksi kesalahan ejaan, atau menstandardisasi nilai.

Replacer.ReplaceText berfokus pada substitusi substring dalam teks, sementara Replacer.ReplaceValue digunakan untuk mengganti konten sel. Keduanya digunakan oleh fungsi tingkat tinggi seperti Table.ReplaceValue dan List.ReplaceValue untuk memanipulasi dan mengubah data dengan mudah guna memenuhi persyaratan tertentu. Contoh yang sempurna adalah operasi Replace Values.

3. Pengganti Khusus

Pengganti kustom memungkinkan pengguna untuk melampaui batasan opsi default, menawarkan cara yang fleksibel dan ampuh untuk mengatasi masalah dalam data. Tidak seperti pendekatan pengganti default yang cocok untuk semua orang, pengganti kustom memungkinkan Anda untuk mengatasi anomali dengan presisi bedah, mengubah kumpulan data Anda yang hampir sempurna menjadi sempurna. Baik itu mengganti nilai yang

dimasukkan sebagai placeholder atau mengoreksi konvensi penamaan yang tidak konsisten, Anda dapat mengatasi semuanya.

Fungsi pengganti dalam Table.ReplaceValue atau List.ReplaceValue memainkan peran penting dalam transformasi dan manipulasi data. Fungsi tersebut memungkinkan pengguna untuk mengganti nilai tertentu dalam kolom tabel atau list, masing-masing, dengan nilai baru, berdasarkan aturan atau kondisi yang ditentukan. Merupakan hal yang umum untuk menemukan inkonsistensi atau kesalahan saat bekerja dengan data. Fungsi pengganti kustom menawarkan cara untuk mengatasi masalah ini dengan memungkinkan pengguna untuk menentukan logika penggantian mereka sendiri.

Untuk mengelola proses penggantian secara efektif, Anda dapat membuat fungsi pengganti anonim sebaris dengan cepat yang mengambil tiga parameter, seperti yang diilustrasikan di sini:

```
Table.ReplaceValue(
    Source, //table to transform
    each true, // oldValue, logical condition
    null, // newValue
    (x, y, z) => if y then z else x, // custom replacer logic
    colList // columns as a list
)
```

Logika pengganti kustom, mirip dengan pengganti default , memerlukan tiga parameter: currentValue (x), oldValue (y), dan newValue (z). Alih-alih fungsi inline (x, y, z) => if y then z else x, Anda juga dapat menentukan fungsi kustom seperti ekspresi customReplacer ini, yang ekuivalen:

```
let
    customReplacer = (currentValue, oldValue, newValue) => if
        oldValue then newValue else currentValue
in
    customReplacer
```

Dalam contoh ini, fungsi customReplacer didefinisikan tanpa tipe dan mengambil currentValue, oldValue, dan newValue sebagai parameter. Fungsi ini mengevaluasi apakah oldValue mengembalikan true; jika demikian, fungsi ini mengganti

currentValue dengan newValue. Jika tidak, fungsi ini mempertahankan currentValue tidak berubah.

Fungsi Replacer kustom dapat digunakan dalam fungsi Table.ReplaceValue dan List.ReplaceValue tingkat tinggi untuk menangani persyaratan penggantian tertentu. Penting untuk dicatat bahwa fungsi Replacer kustom memberikan fleksibilitas tingkat tinggi.

Perhatikan tabel pada Gambar 3.23. Tabel tersebut berisi empat kolom tanggal. Kolom-kolom ini berisi tiga nilai tanggal tertentu yang memerlukan penggantian dengan null. Untuk menghindari beberapa transformasi ReplaceValue, kita akan merancang strategi untuk mengganti semuanya dalam satu langkah.

	Label	Date Column1	Date Column2	Date Column3	Date Column4
1	Entry 1	1-11-2023	9-12-2023	21-1-2024	1-1-0001
2	Entry 2	9-9-2023	1-1-1900	1-1-0001	9-3-2024
3	Entry 3	2-2-2024	11-12-2023	9-9-9999	3-9-2023
4	Entry 4	25-5-2021	1-1-0001	1-1-1900	11-1-2024
5	Entry 5	9-9-9999	5-1-2024	9-10-2023	7-3-2024

Gambar 3.23 Beberapa Nilai Tanggal perlu diganti

Faktanya, kode M ini mengilustrasikan tiga metode yang sedikit berbeda untuk Anda jelajahi. Yang pertama adalah Anonymous, sebuah fungsi inline yang tidak diketik. Yang kedua adalah Typed, sebuah fungsi inline yang diketik. Terakhir, kita memiliki CF, sebuah fungsi kustom yang diketik yang ditetapkan ke variabel dateReplacer:

```
let
  datesToReplace = {#date(1, 1, 1),#date(1900, 1, 1),#date(9999,
9, 9)}, dateReplacer = (x as nullable date, y as nullable logical,
z as nullable
date) as nullable date =>
  if List.Contains( datesToReplace, x) then z else x, colsList =
List.Select( Table.ColumnNames(Source), each
  Text.StartsWith(_, "Date")), Source = Table.FromColumns(
    {
      {"Entry 1", "Entry 2", "Entry 3", "Entry 4", "Entry 5"},
      {#date(2023, 11, 1), #date(2023, 9, 9), #date(2024, 2, 2),
    #date(2021,5,
25), #date(9999, 9, 9)},
      {#date(2023, 12, 9), #date(1900, 1, 1), #date(2023, 12,
    11), #date(1, 1,
```

```
   1), #date(2024, 1, 5)},
         {#date(2024, 1, 21), #date(1, 1, 1), #date(9999, 9, 9),
       #date(1900, 1,
   1), #date(2023, 10, 9)},
         {#date(1, 1, 1), #date(2024, 3, 9), #date(2023, 9, 3),
       #date(2024, 1,
   11), #date(2024, 3, 7)}
   }, type table[Label=text, Date Column1=date, Date Column2=date,
Date Column3=date, Date Column4=date]
    ),
      Anonymous = Table.ReplaceValue( Source, null, null, (x, y, z)
    => if List.Contains( datesToReplace, x) then z else x, colsList
    ),
   Typed = Table.ReplaceValue( Source, null, null, (x as nullable
date, y as nullable logical, z as nullable date) as nullable date =>
       if List.Contains( datesToReplace, x) then z else x, colsList
    ),
    CF = Table.ReplaceValue( Source, null, null, dateReplacer,
  colsList )
  in
    CF
```

Berikut adalah penjelasan dari setiap variabel dalam kode M:

a. *datesToReplace*: List dengan tiga nilai tanggal untuk dicari, yang perlu diganti dalam beberapa kolom dalam tabel.
b. *dateReplacer* : Fungsi kustom yang diketik yang memeriksa apakah *currentValue* (x) ada dalam list tanggal yang akan diganti. Jika ada, fungsi akan mengembalikan *newValue* (z); jika tidak, fungsi akan mengembalikan *currentValue* (x) tanpa perubahan.
c. *colsList*: Ini hanya memilih kolom-kolom dari tabel sumber yang namanya dimulai dengan Tanggal. Ini adalah cara dinamis untuk mengidentifikasi semua kolom tanggal dalam tabel ini.
d. *Source: Menentukan tabel data yang telah diberikan kepada kita.*
e. *Anonymous: Mengilustrasikan cara menerapkan pengganti kustom sebaris yang tidak diketik.*
f. *Typed: Mengilustrasikan cara menerapkan pengganti kustom sebaris yang diketik.*
g. *CF: Mengilustrasikan cara menerapkan pengganti kustom yang ditetapkan ke variabel dateReplacer* .

Pengganti default memiliki kemampuan untuk mengganti nilai di beberapa kolom, tetapi mereka bergantung pada nilai yang

ditentukan untuk penggantian dan karenanya lebih terbatas. Namun, fungsi pengganti kustom menawarkan solusi yang lebih serbaguna. Mereka memungkinkan Anda untuk menentukan kondisi Anda sendiri, secara efektif menghilangkan batasan apa pun dan menyediakan metode yang tepat yang disesuaikan dengan kebutuhan unik Anda.

Kita telah memperkenalkan konsep pengganti kustom, yang menampilkan tiga metode berbeda: fungsi sebaris tak bertipe, fungsi sebaris bertipe, dan fungsi bertipe yang ditetapkan ke variabel. Metode ini memungkinkan penggantian nilai dinamis berdasarkan kondisi yang ditentukan. Pengganti kustom harus didefinisikan sebagai fungsi tiga parameter yang masing-masing mewakili currentValue, oldValue, dan newValue. Jika Anda ingin mempelajari lebih lanjut tentang fungsi kustom, lihat Bab 9, Parameter dan Fungsi Custom.

H. Penggabung

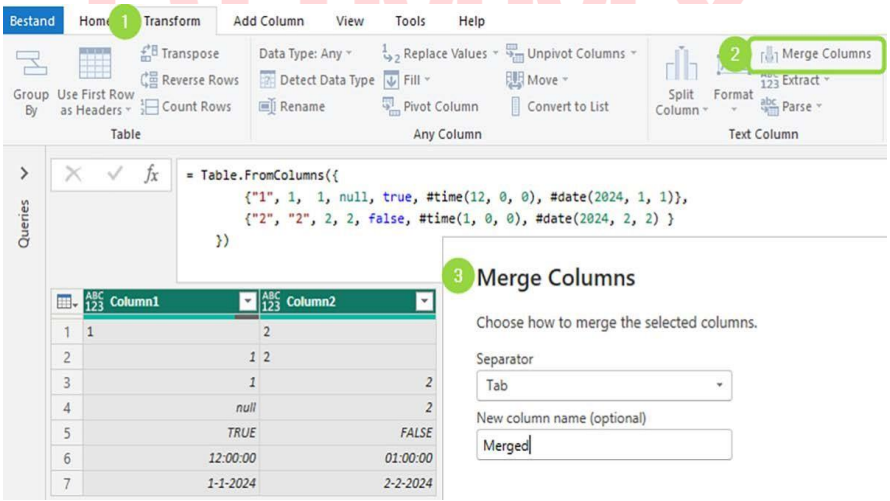
Penggabung menggabungkan nilai teks dari list atau kolom dalam tabel untuk membentuk nilai teks tunggal. Baik menggabungkan nama depan dan belakang untuk membuat nama lengkap atau menggabungkan komponen alamat menjadi alamat lengkap, penggabung sangat penting dalam skenario sehari-hari di mana elemen terpisah perlu disatukan untuk menciptakan sesuatu yang lebih bermakna. Prinsip yang sama berlaku untuk skenario transformasi, di mana penggabungan kolom sebelum transformasi tambahan, seperti unpivoting, sangat penting untuk memastikan bahwa nilai terkait tetap bersama selama proses. Ini menyediakan sarana untuk memisahkan nilai gabungan ini lagi, pada tahap selanjutnya dalam proses transformasi.

Dalam bahasa Power Query M, penggabung menggabungkan list nilai teks menjadi satu teks. Mereka digunakan oleh fungsi pustaka standar tingkat tinggi seperti Table.ToList dan Table.CombineColumns, yang menggunakan fungsi penggabung untuk memproses setiap baris dalam tabel dan menghasilkan satu nilai per baris.

Operasi umum yang dilakukan melalui UI adalah Merge Columns. Terletak pada tab Transform pada pita (lihat Gambar 3.24), ia menggunakan Table.CombineColumns dan Combiner.CombineTextByDelimiter. Untuk memastikan operasi berhasil, Table.TransformColumnTypes yang bersarang dimasukkan ke dalam transformasi saat tipe selain teks terdeteksi. Ini menjamin bahwa nilai teks tunggal atau kesalahan tingkat sel dihasilkan. Ini memungkinkan pengguna untuk menggabungkan berbagai nilai primitif, termasuk teks, angka, tanggal, dan banyak lagi. Pertimbangkan tabel yang dihasilkan kode ini:

```
let
    Source = Table.FromColumns({
        {"1", 1,  1, null, true, #time(12, 0, 0), #date(2024, 1, 1)},
        {"2", "2", 2, 2, false, #time(1, 0, 0), #date(2024, 2, 2) }
    })
in
    Source
```

Bila Anda memilih lebih dari satu kolom, dalam hal ini semua (seperti yang terlihat pada Gambar 3.24), opsi **Merge Columns** (2) diaktifkan pada tab **Transform** (1). Tindakan ini akan memunculkan kotak dialog terkait (3):



Gambar 3.24 Gabungkan Kolom melalui UI grafis

Kita telah memilih Tab sebagai pemisah dari menu drop-down, tetapi pemisah apa pun yang tidak digunakan sebagai karakter dalam kumpulan data dapat digunakan. Setelah operasi dikonfirmasi, kolom baru bernama Merged akan menggantikan kolom input. Ini menghasilkan kode M berikut. Perhatikan bahwa budaya dapat bervariasi berdasarkan lokasi Anda:

```
let
    Source = Table.FromColumns({
        {"1", 1,  1, null, true, #time(12, 0, 0), #date(2024, 1, 1)},
        {"2", "2", 2, 2, false, #time(1, 0, 0), #date(2024, 2, 2) }
    }),
    MergedColumns = Table.CombineColumns( Table.TransformColumnTypes(
        Source,
            {{"Column1", type text}, {"Column2", type text}}, "en-US"
        ),  {"Column1", "Column2"},
        Combiner.CombineTextByDelimiter( "#(tab)", QuoteStyle.None ),
        "Merged"
    )
in
    MergedColumns
```

Ini menghasilkan hasil yang ditunjukkan pada Gambar 3.25:

	Merged
1	1 2
2	1 2
3	1 2
4	2
5	true false
6	12:00 PM 1:00 AM
7	1/1/2024 2/2/2024

Gambar 3.25 Hasil Penggabungan Kolom-Kolom ini

Harap perhatikan, jika Anda ingin menyimpan kolom input, pilih **Merge Columns** dari tab **Add Column** pada pita. Namun, ini akan menghasilkan kode M yang menggunakan fungsi *Text.Combine* alih-alih penggabung untuk menghasilkan satu nilai.

Bahasa M memiliki lima fungsi penggabung *default* yang memungkinkan penggabungan nilai teks dengan berbagai cara.

1. *Combiner.CombineTextByDelimiter*

Mengembalikan fungsi yang menggabungkan list nilai teks menjadi satu teks menggunakan pemisah yang ditentukan. Parameternya meliputi:

- *Delimiter (text)*: Menentukan karakter yang akan digunakan sebagai pemisah
- *QuoteStyle (Number)*: Mengatur cara kutipan diperlakukan

a. Fungsionalitas

Pembatas memerlukan nilai jenis teks yang menentukan karakter atau karakter yang digunakan sebagai pemisah antara nilai teks gabungan. *quoteStyle* adalah enumerasi yang mengontrol bagaimana tanda kutip diperlakukan dalam *string* teks selama operasi, seperti menggabungkan teks.

b. Contoh

Di sini, pemisah --- diberikan ke fungsi penggabung. Pemisah tersebut kemudian dipanggil pada list *string* teks ini: {"a", "b", "c"}. Ini menghasilkan hasil a---b---c, di mana setiap *string* teks dipisahkan oleh pemisah yang diberikan ke fungsi penggabung:

```
Combiner.CombineTextByDelimiter("---")(
    {"a", "b", "c"}
)
//Result: a---b---c
```

Saat memasukkan *string* yang dikutip ke dalam list dengan nilai teks, *QuoteStyle.None* menandakan bahwa karakter kutipan dalam *string* tidak memiliki arti khusus. Karakter tersebut diperlakukan seperti karakter lain dalam teks, seperti yang diilustrasikan di sini:

```
Combiner.CombineTextByDelimiter("_", QuoteStyle.None)(
    {"This ", "is","a ""string in quotes"""}
)
//Result: This _is_a "string in quotes"
```

Namun, *QuoteStyle.Csv* menandakan bahwa tanda kutip harus diperlakukan sebagai karakter khusus untuk

menunjukkan awal dan akhir *string* yang dikutip. Kutipan yang bersarang di dalam *string* ditunjukkan oleh dua karakter kutipan yang berurutan:

```
Combiner.CombineTextByDelimiter("_", QuoteStyle.Csv)(
    {"This ", "is","a ""string in quotes"""}
)
//Result: This _is_"a ""string in quotes"""
```

Combiner.CombineTextByDelimiter mengembalikan fungsi yang dapat diterapkan pada list nilai teks, menggabungkannya menjadi satu teks menggunakan argumen yang ditentukan. Jika Anda menghilangkan argumen *quoteStyle* opsional atau memberikan null, *QuoteStyle.Csv* akan diterapkan secara *default*.

2. *Combiner.CombineTextByEachDelimiter*

Mengembalikan fungsi yang menggabungkan list nilai teks menjadi satu teks menggunakan setiap pemisah yang ditentukan secara berurutan. Fungsi ini mencakup parameter berikut:

- *Delimiters (list)*: Menentukan karakter yang akan digunakan sebagai *delimiters*
- *QuoteStyle (Number)*: Mengatur cara kutipan diperlakukan

a. Fungsionalitas

Urutan pemisah ditentukan oleh nomor dan urutannya, yang diberikan sebagai argumen pertamanya. Jika jumlah nilai teks yang akan digabungkan melebihi jumlah *delimiters*, nilai teks yang tersisa digabungkan tanpa pemisah. Perilaku *quoteStyle* setara dengan fungsi *Combiner.CombineTextByDelimiter*.

b. Contoh

Untuk menghindari kehabisan list dengan pembatas yang tersedia, Anda dapat menghitung jumlah pembatas minimum yang diperlukan dan menyimpannya dalam variabel. Ini memerlukan pendekatan modular, membuat fungsi, penutupan, dan memanggilnya pada *inputList*:

```
let
    inputList = {"a", "b", "c", "d", "e", "f"}, delimiterPattern = {"---",
    "_"},
    delimitersList = List.Repeat( delimiterPattern, Number.RoundUp(
              (List.Count(inputList)-1) / List.Count(delimiterPattern)
         )
    ),
    closure = Combiner.CombineTextByEachDelimiter(delimitersList), Result
    = closure(inputList)
in
    Result     //Output: a---b_c---d_e---f
```

Berikut ini adalah rincian kodenya:

1) *inputList* adalah list yang berisi enam nilai teks.
2) *delimiterPattern* adalah list yang berisi dua *delimiters*.
3) *delimitersList* mengulang *delimiterPattern* beberapa kali untuk memastikan ada cukup elemen untuk memisahkan setiap nilai teks dalam *inputList*.
4) *closure* adalah fungsi yang dihasilkan dengan memanggil fungsi combiner dengan *delimiterList* yang disediakan.
5) Hasilnya memanggil closure tersebut pada inputList, untuk menghasilkan a---b_c---d_e---f.

Fungsi *Combiner.CombineTextByEachDelimiter* mengembalikan fungsi yang menggabungkan list nilai teks menjadi satu teks, menggunakan jumlah dan urutan *delimiters* yang ditentukan.

3. *Combiner.CombineTextByLengths*

Mengembalikan fungsi yang membuat *string* teks tunggal dengan menggabungkan list nilai teks menjadi teks tunggal menggunakan panjang yang ditentukan. Parameternya meliputi:

- *Lengths (list)*: Menentukan panjang setiap bagian teks.
- *Templat (text)*: Urutan karakter awal.

a. Fungsionalitas

List panjang menunjukkan jumlah karakter yang akan diekstrak dari setiap nilai teks yang sesuai. Argumen template opsional menetapkan karakter awal untuk keluaran gabungan. Jika dihilangkan atau ditetapkan ke null, fungsi tersebut menggunakan *string* spasi yang jumlahnya sesuai dengan

jumlah panjang yang ditentukan. Jika Anda memberikan *string* kustom, *string* tersebut akan menggantikan *template default* ini.

b. Contoh

Templat akan ditimpa oleh teks yang diekstrak selama proses penggabungan. Jika templat kustom tidak memenuhi jumlah karakter yang dibutuhkan, templat akan diberi spasi tambahan. Namun, karakter berlebih dalam templat akan muncul dalam hasil akhir teks gabungan:

```
Combiner.CombineTextByLengths(
    { 4, 4, 4, 4 }, "\\\\^^^^////****")(
    {"98", "DELTA", "19990110", "NO", "¯\_(  )_/¯"}
) //Result: 98\\DELT1999NO**
```

Berikut ini adalah rincian kodenya:

1) Penggabung diberikan list panjang, { 4, 4, 4, 4 }, dan teks templat yang terdiri dari 4x4=16 karakter, "\\\\^^^^////****".
2) Selanjutnya, penggabung dipanggil pada list *string* ini: *{"98", "DELTA", "19990110", "NO", "¯_( )_/¯"}*. List ini berisi 5 *string*.
3) Berdasarkan list panjang, penggabung ini akan memperoleh 4 karakter dari 4 *string* untuk menimpa templat 16 karakter. Ketika *string* memiliki kurang dari 4 karakter, nilai templat ditampilkan, menghasilkan hasil ini: 98//DELT1999NO**.

Karakter yang berlebih diabaikan, begitu pula nilai teks tambahan yang tidak memiliki panjang yang sesuai. Namun, jika panjangnya melebihi jumlah karakter, nilai teks lengkap diekstraksi dan templat tidak sepenuhnya ditimpa.

4. *Combiner.CombineTextByPositions*

Mengembalikan fungsi yang menggabungkan teks dari posisi tertentu dari list nilai teks. Parameternya meliputi:

- *Positions (list)*: Menentukan posisi yang akan diekstrak
- *Templat (texk)*: Urutan karakter awal

a. Fungsionalitas

List *positions* menunjukkan jumlah karakter kumulatif yang akan diekstrak dari nilai teks terkait. Setiap posisi berikutnya harus sama atau melebihi posisi sebelumnya.

b. Contoh

Template opsional adalah *string* yang menentukan pola dasar yang akan ditimpa selama proses penggabungan. Bila jumlah karakter yang akan diekstrak melebihi panjang nilai teks yang sesuai, sebagian dari template tersebut tidak akan ditimpa dan menjadi bagian dari output. Setiap karakter yang berlebih akan diabaikan. Namun, jika ada teks tambahan yang tidak memiliki posisi yang diberikan, hanya teks lengkap pertama yang disertakan, sedangkan nilai teks berikutnya akan dikecualikan:

```
Combiner.CombineTextByPositions(
  { 0, 4, 6, 10, 12, 12 }, Text.Repeat( "*", 14))(
  {"98", "DELTA", "19990110", "NO", "¯\_( )_/¯", "Wait", "What"}
)
//Result: 98**DE1999NOWait
```

Gambar 3.26 mengilustrasikan cara kerja proses ini:

Texts	98	DELTA	19990110	NO	¯_(ツ)_/¯	Wait	What
Positions	0	4	6	10	12	12	
Chars to extact	4	2	4	2	0		
explanation	*=4-0*	*=6-4*	*=10-6*	*=12-10*	*=12-12*		
yield	*98***	*DE*	*1999*	*NO*		Wait	

Gambar 3.26 Proses *Combiner.CombineTextByPositions*

5. *Combiner.CombineTextByRanges*

Mengembalikan fungsi yang menggabungkan segmen teks menggunakan posisi dan panjang yang ditentukan. Berikut ini adalah parameter yang digunakan:

- *Ranges (list)*: Menentukan posisi dan jumlah karakter yang akan diekstrak
- *Templat (text)*: Urutan karakter awal

a. Fungsionalitas

List rentang berisi list bertingkat dengan dua nilai: posisi awal, yang menunjukkan di mana karakter akan ditempatkan dalam *string* keluaran dan jumlah karakter yang akan diekstrak dari teks terkait, atau null yang menunjukkan seluruh *string* harus disertakan. Penting untuk dicatat bahwa jika posisi ekstraksi selanjutnya tumpang tindih dengan yang sebelumnya, ekstraksi selanjutnya akan menimpa yang sebelumnya. Item berlebih apa pun dalam list masukan yang tidak memiliki rentang yang ditentukan akan diabaikan.

b. Contoh

Template opsional menyediakan pola *string* awal yang ditimpa. Bila posisi awal melampaui jumlah karakter yang diekstrak pada titik tertentu atau jumlah karakter dalam template kustom melampaui jumlah karakter yang disediakan, karakter template menjadi bagian dari output:

```
Combiner.CombineTextByRanges(
    {{2, 2 }, {6, 4}, {10, 4}, {0, null}}, Text.Repeat( "*",
    16))(
    {"98", "DELTA", "19990110", "NO", "¯\_(  )_/¯"}
)
//Result: NO98**DELT1999**
```

Kita telah membahas penggabung dalam bahasa *Power Query* M, yang penting untuk menggabungkan nilai teks menjadi satu string. Penggabung ini digunakan oleh fungsi tingkat tinggi seperti *Table.ToList* dan *Table.CombineColumns*, yang memanfaatkan penggabung untuk memproses baris tabel, menghasilkan satu nilai per baris. Contoh sempurna dari hal ini adalah operasi **Merge Columns** yang ditawarkan oleh UI. Lebih jauh, kita telah menyediakan cuplikan kode praktis untuk menggabungkan nilai teks secara efektif untuk setiap fungsi penggabung *default* dalam bahasa M. Contoh-contoh ini informatif dan dapat dieksekusi, menyediakan referensi berharga untuk mengeksplorasi kemampuannya.

I. Pemisah

Di sisi lain, pemisah melakukan tugas membagi nilai teks tunggal menjadi beberapa komponen berbeda. Baik memisahkan nama lengkap menjadi nama depan dan belakang atau membedah alamat lengkap menjadi elemen individual, pemisah dapat memecah *string* menjadi bagian yang lebih rinci. Prinsip ini juga relevan dalam skenario transformasi di mana nilai kolom yang sebelumnya digabungkan perlu dipisahkan. Pemisah menyediakan sarana untuk membalikkan proses penggabungan, memisahkan nilai kembali ke komponen asli yang berbeda pada tahap berikutnya dalam proses transformasi data.

Dalam bahasa *Power Query* M, pemisah digunakan untuk memisahkan *string* menjadi list nilai teks berdasarkan pembatas atau pola tertentu. Pemisah ini sering digunakan oleh fungsi pustaka standar tingkat tinggi seperti *Table.SplitColumn*, yang dirancang untuk membagi satu kolom menjadi beberapa kolom atau baris. Ini adalah operasi umum yang sering digunakan saat menangani *string* yang memerlukan penguraian atau transformasi.

Saat Anda memilih kolom yang berisi nilai teks, tombol **Split Column** diaktifkan di UI. Anda dapat menemukannya di tab **Home** dan **Transform** pada pita. Ini memanggil fungsi *Table.SplitColumn*, yang menggunakan salah satu pemisah *default* berdasarkan pilihan pengguna, dalam kasus ini **By Delimiter**.

Misalnya, mari kita lihat tabel berikut tempat kita ingin membagi *Column1* menjadi kolom-kolom:

	ABC 123 Column1
1	Hello, World
2	Power Query

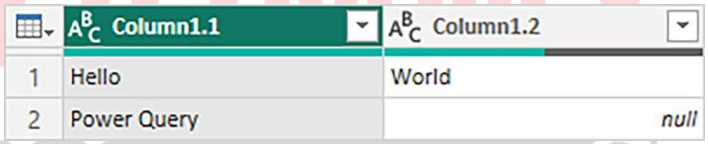
Gambar 3.27 Sample Data

Berikut cara melakukannya menggunakan UI.

- Pilih kolom *Column1* pada tabel.
- Navigasi ke tab **Home** atau **Transform** pada pita dan pilih **Split Column**.

- Kotak dialog akan muncul. Pilih **Custom**, lalu masukkan koma diikuti spasi.
- Buka **Advanced Options** dan pastikan split into **Columns** dipilih dan jumlah kolom yang akan dipisah adalah 2. Konfirmasi.

Langkah *Split Column by Delimiter* ditambahkan ke kueri, yang menghasilkan tabel yang dimodifikasi (Gambar 3.28) di mana Kolom1.2 telah ditambahkan dengan Dunia di sel pertama kolom baru ini:



	Column1.1	Column1.2
1	Hello	World
2	Power Query	*null*

Gambar 3.28 Hasil dari Operasi Split Column ini

Setelah memeriksa kode M dalam bilah rumus, Anda akan menemukan ekspresi yang ditetapkan ke *langkah #"Split Column by Delimiter*". Fungsi Table.SplitColumn tingkat tinggi dipanggil dan menyediakan semua argumen teks yang diperlukan ke fungsi *Splitter.SplitTextByDelimiter*(", ", QuoteStyle.Csv) bagian dalam:

```
let
    Source = Table.FromList(
        {"Hello, World", "Power Query"},
        Splitter.SplitByNothing()
    ),
    #"Split Column by Delimiter" =
        Table.SplitColumn(Source, "Column1",
        Splitter.SplitTextByDelimiter(", ", QuoteStyle.Csv),
        {"Column1.1", "Column1.2"}
    )
in
    #"Split Column by Delimiter"
```

Fungsi pemisah bersifat serbaguna, menjadikannya alat yang ampuh untuk manipulasi teks. Masing-masing menghasilkan nilai fungsi saat dipanggil. Untuk menggunakan pemisah secara efektif, sebaiknya definisikan dulu dengan menentukan argumen, lalu panggil fungsi yang dikembalikan, yang juga dikenal sebagai closure, dengan meneruskan string teks yang akan dipisah.

Fungsi yang dikembalikan, closure, melakukan operasi pemisahan yang sebenarnya. Proses dua langkah ini memungkinkan

fleksibilitas yang lebih besar karena Anda dapat mendefinisikan pemisah sekali dengan argumen tertentu dan menggunakannya kembali beberapa kali dengan input string teks yang berbeda, sehingga efisien untuk pemrosesan teks batch. Sekarang, mari kita jelajahi ciri-ciri khusus setiap pemisah. Bahasa M memiliki 2 fungsi pemisah default. Empat di antaranya menerima parameter quoteStyle opsional. Yaitu:

- *Splitter.SplitTextByDelimiter*
- *Splitter.SplitTextByAnyDelimiter*
- *Splitter.SplitTextByEachDelimiter*
- *Splitter.SplitTextByWhitespace*

Parameter quoteStyle adalah enumerasi yang mengatur penanganan tanda kutip dalam string selama operasi pemisahan:

- *quoteStyle.None* menunjukkan bahwa tanda kutip dalam *string* tidak memiliki arti khusus dan diperlakukan seperti karakter lain dalam teks.
- *quoteStyle.Csv* menyiratkan bahwa tanda kutip harus dianggap sebagai karakter khusus untuk menunjukkan awal dan akhir *string* yang dikutip. Kutipan bersarang dalam *string* diwakili oleh dua karakter kutipan berurutan. Jika Anda menghilangkan argumen *quoteStyle* opsional atau memberikan null, *QuoteStyle.Csv* akan digunakan secara *default* .

1. *Splitter.SplitByNothing*

Mengembalikan fungsi yang mencegah pemisahan dengan merangkum argumennya dalam sebuah list. Perlu dicatat, fungsi ini tidak memiliki parameter.

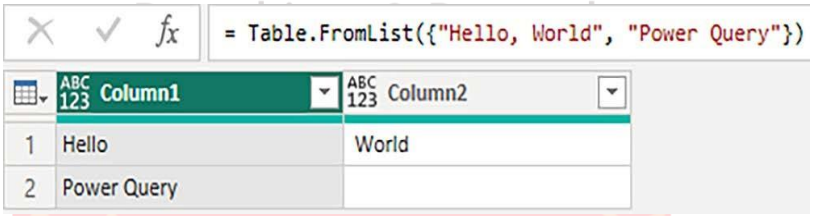
a. Fungsionalitas

Saat dipanggil, *Splitter.SplitByNothing* mengembalikan fungsi yang secara khusus dirancang untuk mengelola perilaku pemisahan dalam operasi *Power Query*, sekaligus mempertahankan struktur data yang dimaksud. Hal ini dicapai dengan merangkum argumennya dalam list, sehingga membuat list elemen tunggal yang mencegah operasi

pemisahan. Hal ini memastikan data tetap tidak terpengaruh oleh perilaku pemisahan dari fungsi tingkat tinggi.

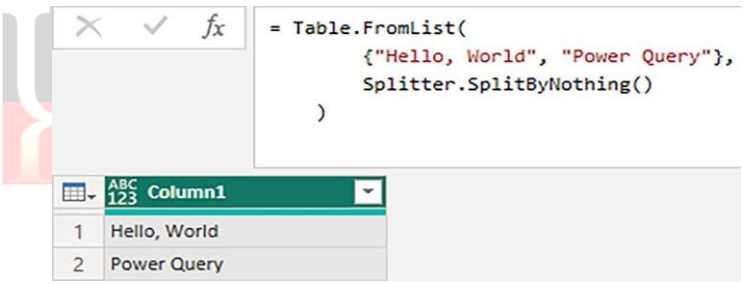
b. Contoh

Power Query, *secara default* , menggunakan koma sebagai pemisah. Misalnya, jika kita memiliki dua elemen dalam list dan ingin mengubahnya menjadi tabel satu kolom, kita dapat memanfaatkan fungsi Table.FromList. Namun, menjalankan kode yang ditunjukkan pada Gambar 3.29 mengakibatkan setiap string dibagi menjadi kolom, sehingga menghasilkan tabel selebar dua kolom.



Gambar 3.29 Pemisahan *Default* berdasarkan *Table.FromList*

Untuk menyesuaikan perilaku pemisahan, Splitter.SplitByNothing dapat diberikan sebagai nilai argumen kedua. Ini akan merangkum setiap nilai dalam list, memastikan setiap string diperlakukan sebagai elemen individual, dan mengembalikan tabel satu kolom, seperti yang terlihat pada Gambar 3.30:



Gambar 3. 30 Mengganti Perilaku Pemisahan *Default*

Berikut adalah kode untuk contoh-contoh tersebut, beserta kode tambahan yang menunjukkan proses enkapsulasi dengan menempatkan argumennya di dalam sebuah list, sehingga

menciptakan list berelemen tunggal yang mencegah operasi pemisahan:

```
Let
       defaultBehaviour = Table.FromList(
          {"Hello, World", "Power Query"}
     ),
     modifiedBehaviour = Table.FromList(
          {"Hello, World", "Power Query"},
          Splitter.SplitByNothing()
     ),
     encapsulatingElements= List.Transform(
          {"Hello, World", "Power Query"},
          Splitter.SplitByNothing()
     )
in
     encapsulatingElements
```

2. *Splitter.SplitTextByAnyDelimiter*

Mengembalikan fungsi yang membagi teks ke dalam list nilai teks menggunakan salah satu *delimiters* yang ditentukan. Fungsi ini menggunakan parameter berikut:

- *delimiters* (*list*): Menentukan karakter yang akan digunakan sebagai pembatas
- *quoteStyle* (*number*): Mengatur bagaimana kutipan diperlakukan
- *startAtEnd* (*logical*): Menentukan arah pemisahan

a. Fungsionalitas

Mengembalikan fungsi yang dapat diterapkan pada nilai teks, membaginya ke dalam list nilai teks berdasarkan argumen yang ditentukan. delimiters adalah argumen yang diperlukan dan mengambil list nilai teks yang masing-masing bertindak sebagai pembatas untuk operasi pemisahan. Pembatas ini dapat berupa karakter tunggal atau beberapa karakter, dan fungsi akan membagi teks pada setiap kemunculan pembatas.

quoteStyle adalah argumen opsional dan menentukan bagaimana fungsi harus menangani kutipan dalam teks. Ini adalah enumerasi dan menerima dua nilai, *QuoteStyle.None* dan *QuoteStyle.Csv*, atau nilai numerik yang setara. Yang

pertama menyertakan kutipan sebagai bagian dari output, sedangkan yang kedua mengabaikannya.

startAtEnd juga merupakan argumen opsional dan mengambil nilai *Boolean*. Ketika diatur ke *true*, fungsi membalikkan arah pemisahan dan memulai operasi pemisahan dari akhir *string* teks, yang dapat menghasilkan hasil yang berbeda dalam kasus tertentu.

b. Contoh

Di sini, argumen *startAtEnd* berguna saat *string* teks tidak simetris atau berisi tanda kutip yang membuat *string* tidak seragam. Dalam kasus seperti itu, memulai operasi pemisahan dari akhir dapat menghasilkan hasil yang berbeda, yang memberikan lapisan kontrol tambahan atas proses manipulasi teks, seperti yang diilustrasikan dalam contoh ini:

```
let
    string = "apple,""orange,banana;grape"",mango""",
    splitFunction = Splitter.SplitTextByAnyDelimiter(
        {",", ";"},
        QuoteStyle.Cs
        v, true
    ),
    result = splitFunction(string)
in
    result
// startAtEnd true: {"apple,orange", "banana", "grape,mango"}
// startAtEnd false: {"apple", "orange,banana,grape, "mango"}
```

3. *Splitter.SplitTextByCharacterTransition*

Mengembalikan fungsi yang membagi teks menjadi list nilai teks pada titik-titik di mana transisi karakter memenuhi semua kondisi. Parameter yang digunakan adalah:

- *before (list* atau *function)*: Mengevaluasi karakter untuk menghasilkan *true* atau *false*
- *after (list* atau *function)*: Mengevaluasi karakter untuk menghasilkan *true* atau *false*

a. Fungsionalitas

Seperti semua pemisah lainnya, saat dipanggil, ia akan menghasilkan fungsi baru, dalam hal ini, yang dirancang untuk membagi teks ke dalam list berdasarkan transisi antar karakter. Ini memungkinkan Anda untuk membagi teks setiap kali terjadi pergeseran dari karakter 'sebelum' ke karakter 'sesudah', yang berarti bahwa kedua kondisi tersebut harus terpenuhi.

b. Contoh

Parameter *before* diatur untuk mengembalikan *true* untuk karakter apa pun. Parameter *after* menggunakan fungsi isEven untuk memverifikasi apakah karakter mewakili angka genap. Ketika kedua kondisi terpenuhi, *string* dibagi, menjamin bahwa *string* teks input dipisahkan sebelum setiap karakter yang mewakili angka genap. Jelas, fungsi *isEven* dapat diganti dengan list yang berisi angka genap, seperti {"0", "2", "4", "6", "8"}, untuk mencapai hasil yang sama:

```
let
    string = "Abc1234Abc1234", isEven = (char) =>
        try Number.IsEven( Number.From(char)) otherwise
    false, splitFunction =
    Splitter.SplitTextByCharacterTransition(
        each true, isEven
    ),
    result = splitFunction(string)
in
    result
//Output: {"Abc1", "23", "4Abc1", "23", "4"}
```

Saat menggunakan fungsi sebagai argumen, penting untuk menangani pengecualian. Dalam contoh ini, kata kunci try dan otherwise digunakan untuk mencegah fungsi gagal saat mencoba mengubah karakter non-numerik menjadi angka.

Kemampuan menggunakan fungsi sebagai parameter dalam Splitter.SplitTextByCharacterTransition memperluas kegunaannya secara signifikan. Hal ini memungkinkan

skenario pemisahan teks yang lebih rumit yang disesuaikan untuk memenuhi persyaratan tertentu.

4. *Splitter.SplitTextByDelimiter*

Ini mengembalikan fungsi yang membagi teks ke dalam list nilai teks menggunakan pemisah yang ditentukan. Berikut parameternya:

- *delimiter (text):* Menentukan karakter yang akan digunakan sebagai pemisah
- *quoteStyle (number):* Mengatur bagaimana kutipan diperlakukan

a. Fungsionalitas

Pembatas memerlukan nilai jenis teks yang menentukan karakter atau karakter yang digunakan sebagai pemisah untuk membagi *string* teks. *quoteStyle* adalah enumerasi yang menentukan bagaimana tanda kutip dalam teks ditangani selama proses pemisahan.

b. Contoh

Tidak seperti fungsi Splitter.SplitTextByAnyDelimiter, fungsi ini tidak menerima list pembatas atau menyertakan parameter startAtEnd, sehingga membuatnya kurang fleksibel tetapi lebih mudah dipahami. Jika dibandingkan dengan fungsi Text.Split yang terkait erat, fitur pembeda dari Splitter.SplitTextByDelimiter adalah kemampuannya untuk mengelola gaya kutipan. Namun, jika Anda tidak memerlukan QuoteStyle.Csv, kesederhanaan dan kemudahan penggunaan Text.Split menjadikannya pilihan ideal untuk tugas pemisahan teks dasar:

```
Let
    string = "apple,""orange,banana;grape"",mango""",
    splitFunction = Splitter.SplitTextByDelimiter(
        ",",
        QuoteStyle.None
    ),
    result = splitFunction(string)
in
    result
//Output: {"apple", """orange", "banana;grape""", "mango"""}
```

Dalam kasus ini, proses pemisahan menggunakan koma sebagai pemisah. Enumerasi QuoteStyle.None digunakan untuk menunjukkan bahwa tanda kutip dalam teks tidak boleh dianggap sebagai karakter khusus selama pemisahan. Akibatnya, string input dibagi pada setiap koma, mengabaikan tanda kutip ganda. Namun, jika Anda mengubah quoteStyle menjadi QuoteStyle.Csv, teks apa pun yang diapit tanda kutip ganda dianggap sebagai satu kesatuan; meskipun menyertakan pemisah yang dipilih, string tersebut tidak akan terpengaruh oleh proses pemisahan.

5. *Splitter.SplitTextByEachDelimiter*

Mengembalikan fungsi yang membagi teks menjadi list nilai teks secara berurutan, dalam urutan pemisah yang ditentukan. Parameternya adalah sebagai berikut:

- *delimiters (list):* Menentukan karakter yang akan digunakan sebagai pemisah
- *quoteStyle (number):* Mengatur bagaimana kutipan diperlakukan
- *startAtEnd (logical):* Menentukan arah pemisahan

a. Fungsionalitas

Urutan dan jumlah pemisah yang diberikan sebagai argumen pertama akan menentukan di mana dan berapa kali *string* teks dipisah. Perilaku *quoteStyle* dan *startAtEnd* setara dengan yang dijelaskan sebelumnya.

b. Contoh

Dalam kasus ini, *startAtEnd* ditetapkan ke true, yang menunjukkan bahwa fungsi tersebut memulai operasi pemisahan dari akhir *string* teks. Kemudian, fungsi tersebut mengikuti urutan dan jumlah pembatas yang tepat yang disediakan untuk mengarahkan proses pemisahan:

```
let
    string = "apple,""orange,banana;grape"",mango""",
    splitFunction = Splitter.SplitTextByEachDelimiter(
        {",", ";"},
        QuoteStyle.None, true
    ),
    result = splitFunction(string)
in
    result
//Output: {"apple,""orange,banana", "grape""", "mango"""}
```

6. *Splitter.SplitTextByLengths*

Mengembalikan fungsi yang memecah teks menjadi list nilai teks berdasarkan panjang yang ditentukan. Parameternya meliputi:

- *Lengths (list):* Menentukan panjang setiap bagian teks
- *StartAtEnd (logical):* Menentukan arah pemisahan

a. Fungsionalitas

Lengths adalah list yang berisi angka positif yang mewakili panjang setiap segmen untuk membagi teks. Parameter *startAtEnd* adalah nilai Boolean opsional yang, jika disetel ke *true*, memungkinkan Anda untuk membalik arah pemisahan untuk memulai dari akhir. Fungsi yang dikembalikan dapat diterapkan ke nilai teks. Ini membagi inputnya ke arah dan panjang yang ditentukan untuk menghasilkan list dengan nilai teks.

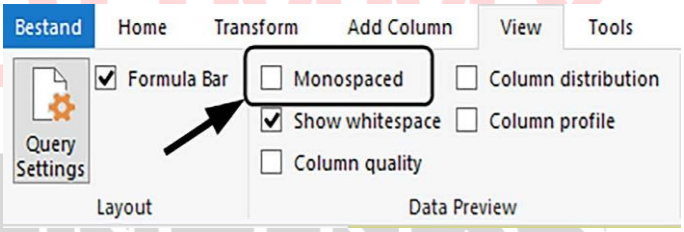
Karakter yang tidak memiliki panjang yang sesuai akan diabaikan. Namun, jika panjangnya melebihi jumlah karakter total, item list dengan nol karakter akan ditambahkan untuk setiap item dengan panjang berlebih, tanpa menimbulkan kesalahan.

b. Contoh

Fungsi *Splitter.SplitTextByLengths* sangat berguna saat menangani *string* dari basis data atau file datar dengan panjang lebar tetap yang telah ditentukan sebelumnya untuk setiap bidang atau kolom. Ini menjamin setiap bidang menempati jumlah karakter yang konsisten, sehingga menyederhanakan proses penguraian. Mari kita lihat contoh berikut:

```
let
     string = "apple,""orange,banana;grape"",mango""", lengths =
     {5, 1, 1, 6, 1, 6, 1, 5, 1},
     splitFunction = Splitter.SplitTextByLengths(
          lengths, false
     ),
     result = splitFunction(string)
in
     result
//Output: {"apple",",","""","orange",",","banana",";","grape","""}
```

Mengidentifikasi data dengan lebar tetap mudah dilakukan. Navigasi ke tab View dan aktifkan Monospaced; lihat Gambar 3.31. Pengaturan ini mengubah jenis huruf menjadi jenis huruf yang setiap karakternya menempati ruang horizontal dalam jumlah yang sama. Jika ruang yang sama telah ditetapkan untuk bidang atau kolom, sekarang seharusnya terlihat jelas.



Gambar 3.31 Opsi *Monospaced* terletak pada Tab Tampilan

7. *Splitter.SplitTextByPositions*

Ini mengembalikan fungsi yang membagi teks pada posisi tertentu untuk menghasilkan list nilai teks. Parameter untuk fungsi ini adalah:

- *positions (list):* Menentukan posisi untuk dibagi
- *startAtEnd (logical):* Menentukan arah pemisahan

a. Fungsionalitas

List posisi menunjukkan jumlah karakter kumulatif untuk pemisahan. Setiap item dalam list posisi harus sesuai atau melebihi yang sebelumnya dan tidak boleh negatif.

b. Contoh

Bila item terakhir dalam list posisi kurang dari jumlah karakter total, maka item tersebut akan menyertakan semua karakter yang tersisa dari string tersebut. Namun, untuk setiap item dalam list posisi yang melebihi jumlah karakter total, item dengan karakter nol akan ditambahkan ke list hasil, tanpa menimbulkan kesalahan:

```
let
    string = "apple,""orange,banana;grape"",mango""", positions =
    {0, 5, 5, 6, 7, 13},
    splitFunction = Splitter.SplitTextByPositions( positions,
        false
    ),
    result = splitFunction(string)
in
    result
//Output: {"apple", "", ",","""","orange",
",banana;grape"",mango"""}
```


8. *Splitter.SplitTextByRanges*

Ini mengembalikan fungsi yang membagi teks menurut rentang yang ditentukan, yang ditentukan oleh posisi awal dan panjang. Fungsi ini menggunakan parameter berikut:

- *ranges (list)*: Menentukan posisi dan jumlah karakter yang akan diambil
- *startAtEnd (logical)*: Menentukan arah pemisahan

a. Fungsionalitas

List *ranges*, yang berisi list bertingkat dengan dua nilai, menentukan urutan keluaran. Nilai-nilai ini menunjukkan posisi awal dan jumlah karakter yang akan diambil, atau *null* untuk menyertakan semua karakter yang tersisa dalam *string*. Jika posisi pemisahan tumpang tindih dengan yang

sebelumnya, fungsi akan mengambil kembali segmen teks tersebut.

b. Contoh

Dalam contoh berikut, untuk setiap item dalam list rentang yang posisinya melebihi jumlah karakter total, item dengan karakter nol ditambahkan ke list *result*, tanpa menimbulkan kesalahan:

```
let
    string = "apple,""orange,banana;grape"",mango""", ranges =
    {{14, null}, {0, 5}, {3, 2}, {7, 6}},
    splitFunction = Splitter.SplitTextByRanges( ranges,
        false
    ),
    result = splitFunction(string)
in
    result
//Output: {"banana;grape"",mango""", "apple", "le", "orange"}
```

9. *Splitter.SplitTextByRepeatedLengths*

Mengembalikan fungsi yang membagi teks menjadi list nilai teks dengan panjang tertentu, berulang kali. Parameternya adalah sebagai berikut:

- *lengths (number):* Menentukan panjang setiap bagian teks
- *startAtEnd (logical):* Menentukan arah pemisahan

a. Fungsionalitas

Panjang adalah angka positif yang mewakili panjang segmen untuk membagi teks. Parameter *startAtEnd* adalah nilai Boolean opsional yang, jika ditetapkan ke true, memungkinkan Anda untuk membalik arah pembagian untuk memulai dari akhir.

b. Contoh

Fungsi yang dikembalikan dapat diterapkan pada nilai teks, membaginya berdasarkan arah dan panjang yang ditentukan, untuk menghasilkan list dengan nilai teks. Tidak seperti *Splitter.SplitTextByLengths*, fungsi ini tidak mengambil list panjang, sehingga kurang fleksibel tetapi lebih mudah

digunakan untuk membagi tugas dengan panjang yang konsisten:

```
let
     string = "apple,""orange,banana;grape"",mango""",
     splitFunction = Splitter.SplitTextByRepeatedLengths(
          10,
          false
     ),
     result = splitFunction(string)
in
     result
//Output: {"apple,""ora", "nge,banana", ";grape""""",ma", "ngo"""}
```

Tidak ada kemampuan pengisian; oleh karena itu, item list akhir dari list result mungkin terdiri dari lebih sedikit karakter daripada panjang yang ditentukan.

10. *Splitter.SplitTextByWhitespace*

Mengembalikan fungsi yang memisahkan teks ke dalam list nilai teks berdasarkan keberadaan karakter spasi, yang menawarkan metode langsung untuk membagi teks. Fungsi ini mencakup parameter berikut:

• *quoteStyle (Number):* Mengatur cara memperlakukan kutipan

a. Fungsionalitas

Mengembalikan fungsi yang membagi teks menjadi list teks di setiap spasi. Spasi dalam bahasa *Power Query* M mencakup berbagai karakter, termasuk spasi dan karakter kontrol seperti carriage return, line feed, dan tab. *quoteStyle* bersifat opsional dan menentukan bagaimana kutipan dalam teks ditangani. *QuoteStyle.None* menyertakan kutipan dan memperlakukannya seperti karakter lainnya, sementara *QuoteStyle.Csv* mengabaikannya. Bagian yang dikutip tidak akan dipisahkan.

b. Contoh

Saat bekerja dengan nilai teks, Anda perlu menggunakan urutan karakter escape untuk menanamkan karakter kontrol. Secara khusus, jika Anda ingin menyisipkan carriage return, Anda memiliki tiga alternatif: Anda dapat menggunakan format heksadesimal 4 digit pendek *#(000D)*, memilih

representasi Unicode heksadesimal 8 digit yang lebih panjang *#(0000000D)*, atau menggunakan singkatan escape yang lebih sederhana *#(cr)*.

Secara umum, singkatan escape lebih disukai. Ini berarti bahwa untuk tab, kita menggunakan *#(tab)*, dan untuk baris umpan, *#(lf)*. Beberapa kode escape dapat disertakan dalam satu urutan escape, dipisahkan dengan koma, seperti dalam *#(cr,lf)*, yang setara dengan *#(cr)#(lf)*:

```
let
    string = "A#(000D) B#(tab) C#(lf)D#(cr,lf)
       E#(cr)F#(lf)1#(0000000D)",
    splitFunction = Splitter.SplitTextByWhitespace(), result =
    splitFunction(string)
in
    result
//Output: {"A", "B","C", "D","E", "F","1", ""}
```

Pemisah berguna saat Anda perlu memecah *string* menjadi bagian-bagian yang lebih kecil berdasarkan pola, pembatas, panjang, atau posisi tertentu. Pemisah umumnya digunakan dalam skenario yang melibatkan penguraian *string* yang berisi nilai gabungan, seperti dalam CSV, dalam file teks yang dibatasi, atau dalam *string* yang berisi nilai lain yang menarik, seperti tanggal atau angka, misalnya. Cuplikan kode yang disediakan dimaksudkan untuk digunakan sebagai referensi dan untuk mengeksplorasi fungsinya.

J. Contoh Praktis

Sejauh ini, Anda telah memperoleh pemahaman tentang berbagai pembanding, pengganti, penggabung, dan pemisah yang tersedia dalam bahasa M. Sekarang, dengan pengetahuan dasar ini, saatnya untuk memperkuat pembelajaran Anda melalui contoh-contoh praktis yang menunjukkan penggunaannya dalam skenario dunia nyata. Contoh-contoh ini akan meningkatkan pemahaman Anda tentang cara mengintegrasikan fungsi-fungsi ini ke dalam alur kerja Anda sendiri.

Jangan ragu untuk merancang strategi Anda sendiri untuk skenario-skenario ini sebelum mempelajarinya. Jika Anda tidak

nyaman menerjemahkannya ke dalam kode M saat ini, tidak apa-apa. Anda masih dapat merumuskan kerangka dan mempertimbangkan kendala-kendala potensial untuk diatasi. Ini akan memungkinkan Anda untuk membandingkan pendekatan Anda dengan yang disarankan di sini. Namun, perlu diingat bahwa ada banyak solusi untuk setiap masalah yang diberikan; buku ini hanya menyajikan satu metode yang mungkin.

1. Menghapus Karakter Kontrol dan Spasi Berlebih

Tugas umum dalam membersihkan dan mengubah *string* teks adalah menghapus karakter kontrol dan spasi berlebih. Tidak seperti fungsi trim Excel, padanan *Power Query*-nya, *Text.Trim*, hanya menghapus spasi di awal dan akhir dari nilai teks, meskipun Anda dapat menentukan karakter lain untuk dipangkas juga. Berikut contoh tabelnya:

```
let
  Source = Table.FromColumns(
    {
      {
        "A#(000D) B#(tab) C#(lf)D#(cr,lf) E#(cr)F#(lf)1#(0000000D)",
        " A   #(000D)  B#(tab)  C#(lf)D  "
      }
    }
                                                                )
in
  Source
```

a. *Goals*

1) Pastikan masukan pengguna sesuai dengan format yang diharapkan
2) Pastikan konsistensi dengan menghilangkan variasi spasi dan menghilangkan karakter yang tidak dapat dicetak

b. Fungsi *cleanTrim*

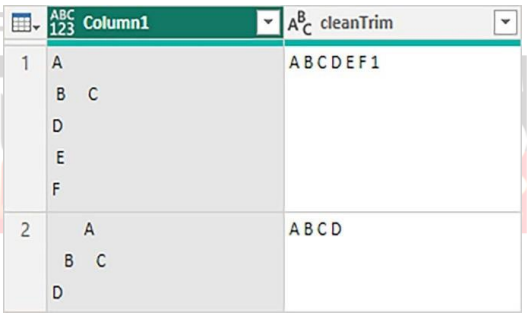
Mari kita buat fungsi kita sendiri, *cleanTrim*, yang dapat melakukan semuanya. Sempurnakan *Text.Trim* untuk menghapus karakter kontrol dan semua spasi berlebih dalam *string* juga. Ini adalah domain fungsi *Splitter.SplitTextByWhitespace*, yang membagi teks pada setiap spasi, dan mengembalikan list nilai teks:

```
(string as text) as text => Text.Trim(
        Text.Combine(
            Splitter.SplitTextByWhitespace()( string
            ), " "
        )
    )
```

Whitespace dalam M mencakup spasi dan karakter kontrol. Saat splitter dipanggil, ia mengembalikan list dengan nilai teks, yang dapat digabungkan dengan satu spasi, untuk diubah kembali menjadi *string*. Kita dapat menggunakan fungsi seperti *Combiner.CombineTextByDelimiter* atau *Text.Combine*, misalnya; keduanya menghasilkan hasil yang sama. Namun, penting untuk dipahami bahwa jika list berisi *string* teks kosong sebagai item awal atau akhir, operasi gabungan akan menyisipkan spasi sebelum atau sesudahnya.

Hal ini dapat diatasi dengan *Text.Trim*. Perlu disebutkan bahwa, berdasarkan pengujian terbatas, operasi trim pada input '*string*' bagian dalam tidak menunjukkan dampak kinerja yang signifikan jika dibandingkan dengan penerapannya pada output gabungan, seperti yang ditunjukkan di sini.

Ketika fungsi cleanTrim dipanggil pada *Column1*, menambahkan kolom ke tabel, ia menghasilkan hasil yang ditunjukkan pada Gambar3.32:



Gambar 3.32 Hasil Pemanggilan *cleanTrim* pada Data Sampel

2. Ekstrak Alamat Email dari *String*

Selain membersihkan dan mengubah *string* teks, kemampuan untuk mengekstrak bagian tertentu dari *string* juga sama pentingnya. Meskipun ada banyak jenis skenario dan tidak ada solusi yang "cocok untuk semua", memahami pemisah dapat bermanfaat secara strategis. Tugas kita berikutnya adalah mengekstrak apa yang tampak sebagai alamat email yang valid dari contoh data ini:

```
let
    Source = Table.FromRows(
        {
            {"Wendy Darling, UK, wendy.darling@nevergrowup.co.uk"},
            {"john.darling@nevergrowup.co.uk,
            UK;john.boy@abc.com"},
            {"michael; michael.darling@nevergrowup"},
            {"Peter Pan peter.pan@neverland.com"},
            {"tink@pixie@fairy.org Tinker Bell"},
            {"Captain Hook: captain.hook@piratescove.org"}
        }, type table [Contact info=text]
    )
in
    Source
```

a. *Goals*

1) Mengisolasi dan mengambil alamat email dari data teks campuran
2) Meningkatkan akurasi dengan mengidentifikasi dan menganalisis pola

b. Mengembangkan Fungsi *getEmail*

Dengan memeriksa contoh ini, jelas bahwa baris tiga dan lima tidak valid secara struktural karena domain yang hilang atau menyertakan lebih dari satu simbol @ di dalam alamat. Hal penting lainnya yang perlu diperhatikan adalah kita memerlukan lebih dari satu pemisah untuk mengelompokkan setiap *string* secara efektif.

Pada akhirnya, kita bertujuan untuk membuat fungsi *getEmail* kustom. Namun, pertama-tama kita perlu mengembangkan logika yang memungkinkan kita untuk mengecualikan dan mengekstrak alamat email dari entri ini.

Klik ikon **mini table** yang terletak di sebelah kiri tajuk kolom Info Kontak dan pilih **Add Custom Column**. Kita akan secara bertahap membangun solusi sambil meninjau hasil antara:

1) Di kotak dialog, Anda dapat secara opsional menetapkan nama lain untuk kolom ini. Karena ini hanyalah kolom sementara, kita akan membiarkannya apa adanya.
2) Masukkan *Splitter.SplitTextByAnyDelimiter()* di bagian rumus karena beberapa pemisah diperlukan.
3) Di dalam tanda kurung, sertakan list dengan semua pemisah yang dapat dibedakan: {" ", ",", ";", ":"}.
4) Untuk menjalankan fungsi ini, tambahkan satu set tanda kurung lagi setelah tanda kurung penutup dan pilih kolom Info Kontak dari list **Available columns**.
5) Penting untuk dicatat di sini bahwa kode tersebut telah diformat agar mudah dibaca, tetapi di bilah rumus, kode yang dihasilkan oleh tindakan ini akan terlihat seperti ini:

```
Table.AddColumn(Source, "Custom", each
        Splitter.SplitTextByAnyDelimiter(
            {" ", ",", ";", ":"}
        )([Contact info])
    )
```

6) Pratinjau konten list keluaran dengan mengeklik spasi di sebelah nilai list bersarang ini. Misalnya, nilai list baris kedua adalah *{ "john.darling@nevergrowup.co.uk", "", "UK", "john.boy@abc.com"}*.
7) List ini memerlukan pemfilteran. Untuk melakukannya, kembali ke kotak dialog **Custom Column** dengan mengeklik ikon **gear wheel** yang terletak di sebelah kanan nama langkah di bagian **Applied Steps** pada panel **Query Settings**.
8) Di depan rumus, tambahkan *List.Select(*. Lalu, lanjutkan ke akhir dan masukkan koma diikuti tanda kurung tutup ')'. Anda akan melihat peringatan yang menunjukkan bahwa koma tidak dapat mendahului tanda kurung tutup. Selanjutnya, tentukan fungsi yang akan menentukan kriteria pemilihan sebagai argumen.

9) Alamat email yang valid secara struktural harus menyertakan simbol at (@), diikuti tanda titik (.), dalam urutan tersebut. Dengan demikian, kita dapat mencoba memisahkan setiap item list lagi menggunakan *Splitter.SplitTextByEachDelimite-r* dengan argumen ini: {"@", "."}. Untuk memanggilnya, tambahkan serangkaian tanda kurung dengan garis bawah (_).

10) Meskipun ini mengatasi peringatan kesalahan, ekspresi kita belum selesai. Pemisah ini akan menghasilkan list baru, tetapi kita hanya tertarik pada list yang berisi tiga elemen. Oleh karena itu, tambahkan *List.Count(...) =3*. Kode dalam kotak rumus akan terlihat seperti ini:

```
List.Select(
    Splitter.SplitTextByAnyDelimiter(
        {" ", ",", ";", ":"}
    )([Contact info]), each List.Count(
        Splitter.SplitTextByEachDelimiter(
            {"@", "."}
        )(_)
    ) =3
)
```

11) Pratinjau konten sekali lagi dengan mengeklik spasi di sebelah nilai list bersarang. Kriteria tambahan diperlukan untuk mengecualikan alamat email yang tidak valid secara struktural. Misalnya, kasus yang menyertakan lebih dari satu simbol @, atau contoh di mana simbol @ langsung diikuti oleh titik (.) masih perlu ditangani.

12) Untuk melakukannya, kembali ke kotak dialog **Custom Column** dengan mengeklik ikon **gear wheel**. Perluas pilihan sebagai argumen fungsi dengan memasukkan kata kunci **and** diikuti oleh ekspresi ini: *Text.Length(Text.Select(_, "@")) =1*. Ini menyelesaikan satu masalah potensial. Untuk mengatasi yang lain, masukkan kata kunci and lain diikuti oleh *not Text.Contains(_, "@.")*, seperti yang ditunjukkan di sini:

```
  List.Select(
      Splitter.SplitTextByAnyDelimiter(
          {" ", ",", ";", ":"})(
      [Contact info]),
      each List.Count(
          Splitter.SplitTextByEachDelimiter(
              {"@", "."}
    )(_)
) =3 and
      Text.Length( Text.Select(_, "@")) =1 and
      not Text.Contains(_, "@.")
  )
```

13) Langkah terakhir melibatkan ekstraksi dan penggabungan item list yang tersisa. Buka kotak dialog dan bungkus *Text.Combine(..., ", ")* di sekitarnya. Setelah Anda puas dengan hasilnya, Anda dapat menyalin kode M dari sini secara keseluruhan.

14) Pilih **New Source**, lalu **Blank Query**. Hapus ekspresi *let default* dan ganti dengan kode yang disalin. Untuk mengubah kode ini menjadi kueri fungsi kustom, kita perlu memulai ekspresi ini dengan inisialisasi fungsi. Posisikan kursor Anda sebelum kode dan tambahkan *(string as text) as text =>*.

15) Fungsi ini mengambil input tipe teks dan akan menghasilkan nilai tipe teks: (*string* sebagai teks) sebagai teks. Parameter input diberi nama *string*, bukan [*Contact info*]. Ganti itu, lalu tutup kueri ini dan beri nama yang sesuai seperti *getEmail*:

```
(string as text) as text => Text.Combine(
          List.Select(
              Splitter.SplitTextByAnyDelimiter(
                  {" ", ",", ";", ":"}
              )(string),
              each List.Count(
                  Splitter.SplitTextByEachDelimiter(
                      {"@", "."})(_)) =3 and
              Text.Length( Text.Select(_, "@")) =1 and not
              Text.Contains(_, "@.")
          ), ", "
      )
```

Verifikasi fungsionalitas fungsi kustom ini dengan menerapkannya pada data sampel dan membandingkan hasilnya dengan output dari **Custom Function**. Cukup pilih

Panggil **Add Column** pada tab Tambahkan Kolom pada pita dan tetapkan argumen untuk setiap input. Gambar 3.33 menunjukkan hasilnya:

	Contact info	getEmail
1	Wendy Darling, UK, wendy.darling@nevergrowup.co.uk	wendy.darling@nevergrowup.co.uk
2	john.darling@nevergrowup.co.uk, UK;john.boy@abc.com	john.darling@nevergrowup.co.uk, john.boy@abc.com
3	michael; michael.darling@nevergrowup	
4	Peter Pan peter.pan@neverland.com	peter.pan@neverland.com
5	tink@pixie@fairy.org Tinker Bell	
6	Captain Hook: captain.hook@piratescove.org	captain.hook@piratescove.org

Gambar 3.33 Hasil Pemanggilan *getEmail* pada Data Sampel

3. Membagi Nilai Sel Gabungan menjadi Baris

Memisahkan nilai sel gabungan merupakan tugas umum saat menangani data yang disimpan dalam satu sel yang perlu dipisahkan. Proses ini mudah dan dapat dicapai dengan memilih **Split Column** pada pita di UI. Dengan menyesuaikan pengaturan yang diperlukan dalam dialog, Anda dapat dengan mudah memisahkan nilai sel gabungan menjadi beberapa kolom atau baris (tersedia di bawah **Advanced options**).

Namun, jika Anda menemukan beberapa kolom dengan nilai sel gabungan yang perlu dipisahkan menjadi baris secara khusus, menerapkan proses ini secara berulang mungkin tidak menghasilkan hasil yang diharapkan. Pertimbangkan contoh data berikut ini:

```
let
  Data = [
    p = Lines.ToText( {"1".."3"} ) & "4", c = "W01",
    s1 = Text.Combine( List.Repeat( {"24x7 #(cr,lf)"}, 3) &
      {"""8x5 (9:00 - 17:00)"""} ),
    s2 = Text.Combine( List.Repeat( {"""9x5 (9:00 - 18:00)"""} &
      {"#(lf,cr)"}, 3) & {"""9x5 (9:00 - 18:00)"""} ),
    s3 = Text.Combine( List.Repeat( {""" """} & {"#(lf)"}, 2) &
     {"""9x5 (9:00 - 18:00)"""} & {"#(cr)"} & {"""9x5 (9:00 -
     18:00)"""})
  ],
  Source = Table.FromRecords(
    {
      [ID = "A", Code=Data[c], Priority = Data[p], Schedule =
      Data[s1]], [ID = "B", Code=Data[c], Priority = Data[p],
      Schedule = Data[s2]], [ID = "C", Code=Data[c], Priority =
```

```
          Data[p], Schedule = Data[s3]]
        },
        type table [ID = text, Code = text, Priority = text, Schedule =
        text]
                                                                    )
in
     Source
```

Data contoh digambarkan pada Gambar 3.34. Gambar tersebut dengan jelas memperlihatkan semua sel di kolom *Priority* dan *Schedule* berisi beberapa nilai yang perlu dipisahkan menjadi beberapa baris:

	ID	Code	Priority	Schedule
1	A	W01	1	24x7
			2	24x7
			3	24x7
			4	"8x5 (9:00 - 17:00)"
2	B	W01	1	"9x5 (9:00 - 18:00)"
			2	
			3	"9x5 (9:00 - 18:00)"
			4	
				"9x5 (9:00 - 18:00)"
3	C	W01	1	" "
			2	" "
			3	"9x5 (9:00 - 18:00)"
			4	"9x5 (9:00 - 18:00)"

Gambar 3.34 Contoh Data Gabungan Nilai Sel

Untuk mengikuti langkah-langkah yang dijelaskan pada bagian berikutnya, silakan pindahkan kode yang diberikan untuk contoh ini ke kueri baru yang kosong.

a. *Goals*

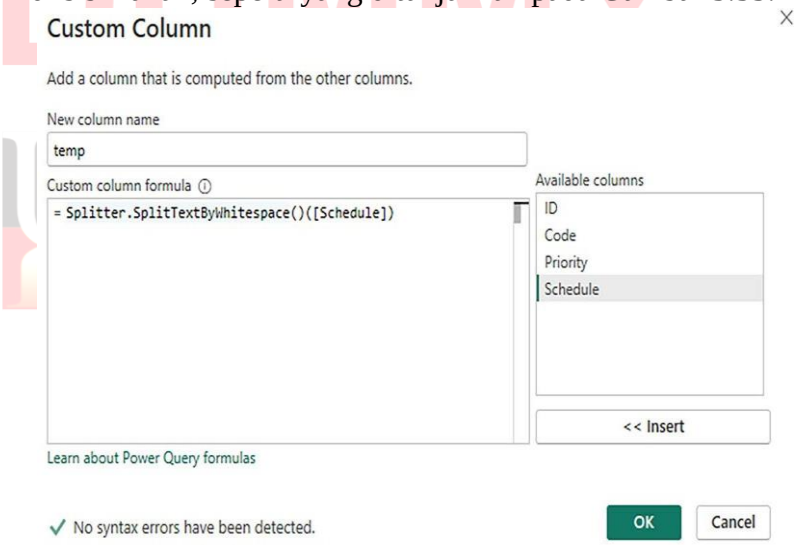
Mempersiapkan data untuk analisis dengan memastikan setiap catatan menempati satu baris.

b. Mengubah Tabel

Jelas, kita berurusan dengan karakter spasi, yang sering kali tidak terlihat (*nongrafis*). Ini termasuk spasi (bagian dari kelas Z *Unicode*), tab horizontal dan vertikal, umpan formulir, dan urutan karakter baris baru. Urutan baris baru mencakup carriage return, line feed, kombinasi carriage return dan line feed, baris berikutnya, dan karakter pemisah paragraf. Ini bisa jadi artefak dari ekstraksi data atau masukan pengguna.

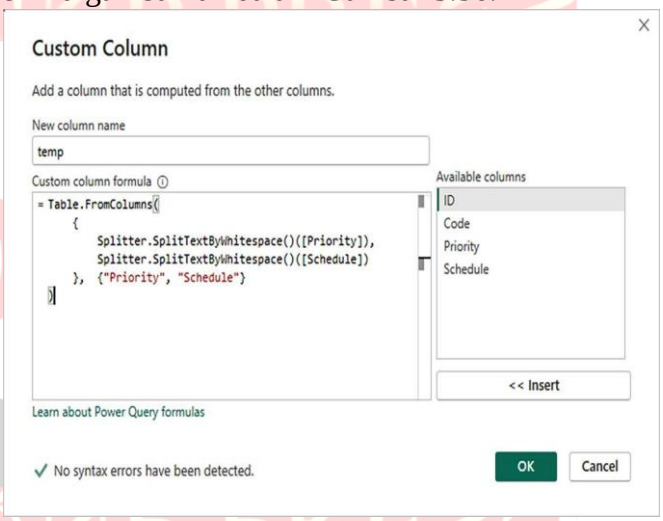
Di bagian sebelumnya dari bab ini, kita mengeksplorasi bagaimana pemisah menghasilkan fungsi untuk membagi *string* eks menjadi list nilai teks. Fungsi *Splitter.SplitTextByWhitespa-ce* dapat digunakan untuk memisahkan dan dengan demikian menghilangkan spasi dari *string*. Parameter *quoteStyle* opsionalnya menentukan bagaimana kutipan dalam teks ditangani. Secara *default* , *QuoteStyle.Csv* diterapkan, yang mengabaikan *string* yang dikutip sehingga tidak terpengaruh oleh proses pemisahan. Oleh karena itu, mari kita lihat bagaimana kita dapat mengubah tabel:

1) Pada tab **Add Column** di pita, pilih **Custom Column**.
2) Di kotak dialog, masukkan temp sebagai nama kolom baru.
3) Ingat, ekspresi pemanggilan adalah serangkaian tanda kurung yang dapat menyertakan list argumen opsional. Ini memulai pemanggilan fungsi. Untuk rumus kolom kustom, kita dapat memasukkan *Splitter.SplitTextByWhitespace()* untuk memulai pemisah dan menerima nilai pengembaliannya, yang merupakan fungsi baru.
4) Kita kemudian dapat menambahkan ekspresi pemanggilan lain dan memilih **Schedule** dari bagian **Available columns** di sisi kanan, seperti yang ditunjukkan pada Gambar 3.35.



Gambar 3.35 Kotak Dialog Kolom Kustom

5) Kolom *temp* baru berisi list nilai teks. Dengan mengklik spasi di samping nilai list, pratinjau sekunder akan muncul, yang menyediakan tampilan terbatas konten dalam list.
   Proses ini dapat diulang untuk kolom *Priority* juga, tetapi mengelola beberapa kolom list dapat menjadi rumit, karena memperluas masing-masing kolom secara terpisah dapat mengakibatkan duplikasi baris yang tidak diinginkan dalam tabel. Dengan menggabungkan semua list ke dalam satu tabel, kita dapat membuat satu struktur yang menyederhanakan dan mengendalikan proses perluasan.
6) Di bagian **Applied Steps** dari pengaturan kueri, klik ikon **gear wheel** yang terlihat pada langkah **Added Custom** untuk membuka kembali kotak dialog **Custom Column**.
7) Di sini, kita akan mengubah kode untuk membuat tabel yang menyertakan nilai dari kolom *Priority* dan *Schedule*. Kode M digambarkan dalam Gambar 3.36:



Gambar 3. 36 Memperbarui Kode di Kotak Dialog Kolom Kustom

8) Untuk membuat tabel dari beberapa list, gunakan fungsi Table.FromColumns:
   a) Untuk argumen pertamanya, tentukan list list yang dipisahkan oleh koma yang mewakili nilai kolom.
   b) Untuk argumen kedua yang opsional, berikan list nama kolom.

9) Dengan mengikuti langkah-langkah ini, tabel akan terbentuk.
10) Kita juga dapat menetapkan tipe kolom ke tabel bersarang ini, sehingga tidak perlu lagi melakukan langkah *Change Type* nanti. Di dalam bilah rumus, di antara tanda kurung tutup fungsi *Table.FromColumns* dan *Table.AddColumn*, lakukan hal berikut:
    a) Pertama, masukkan koma.
    b) Kemudian tambahkan tabel tipe dan *Record* pengaturan tipe yang mencantumkan setiap kolom berdasarkan nama dan tipenya, seperti ini: tabel tipe *[Priority = text, Schedule = text]*.
11) Sebelum memperluas tabel bersarang di kolom temp, mari bersihkan kolom *Priority* dan Jadwal asli. Ada beberapa metode untuk melakukannya, seperti menghapus atau memilih kolom. Namun, karena kita hanya tertarik pada tiga kolom, menerapkan proyeksi akan menjadi metode yang paling ringkas. Konsep ini dibahas secara menyeluruh di Bab 6, Nilai Terstruktur. Di bilah rumus, setelah tanda kurung tutup fungsi *Table.AddColumn*, masukkan satu set tanda kurung siku. Di dalam tanda kurung ini, pilih setiap kolom yang ingin Anda pertahankan dengan merujuk namanya di dalam set tanda kurung siku lainnya, misalnya, *[[ID], [Code], [temp]]*.
12) Terakhir, ekstrak kolom dengan memperluas tabel bersarang di kolom *temp*, dengan panah samping tersedia di header.

    Berikut langkah-langkah yang diterjemahkan ke dalam kode M. Untuk menggabungkan ini ke dalam kode contoh yang diberikan, koma harus ditempatkan tepat setelah tanda kurung tutup fungsi *Table.FromRecords* di langkah *Source*, yang ada di baris pertama. Klausa in berikutnya dan variabel Source harus dihilangkan saat menggabungkan kode ini:

```
,   #"Added Custom" = Table.AddColumn( Source, "temp", each
      Table.FromColumns(
         {     Splitter.SplitTextByWhitespace()([Priority]),
               Splitter.SplitTextByWhitespace()([Schedule])
         },  {"Priority", "Schedule"}
    ),
    type table [Priority = text, Schedule = text] )[[ID], [Code],
    [temp]], #"Expanded temp" = Table.ExpandTableColumn(#"Added Custom",
    "temp",
         {"Priority", "Schedule"}, {"Priority", "Schedule"}
                                                                    )
    in
    #"Expanded temp"
```

Hasilnya adalah tabel transformasi yang terlihat seperti ini:

	ID	Code	Priority	Schedule
1	A	W01	1	24x7
2	A	W01	2	24x7
3	A	W01	3	24x7
4	A	W01	4	8x5 (9:00 - 17:00)
5	B	W01	1	9x5 (9:00 - 18:00)
6	B	W01	2	9x5 (9:00 - 18:00)
7	B	W01	3	9x5 (9:00 - 18:00)
8	B	W01	4	9x5 (9:00 - 18:00)
9	C	W01	1	
10	C	W01	2	
11	C	W01	3	9x5 (9:00 - 18:00)
12	C	W01	4	9x5 (9:00 - 18:00)

Gambar 3.37 Nilai Sel Gabungan dibagi menjadi Baris

4. Mengganti Beberapa Nilai

Pertanyaan yang sering diajukan di banyak forum kurang lebih seperti ini: “Dalam proyek saya yang sedang berlangsung, saya menangani kumpulan data yang memerlukan penggantian nilai yang ekstensif di banyak kolom. Saya telah menggunakan langkah-langkah *ReplaceValue* individual, yang mengakibatkan kueri yang lambat dan sulit diatur. Apakah ada metode yang lebih efisien untuk mengelola penggantian massal?”

Ya, ada. Berikut adalah kumpulan data contoh kita. Anda dapat mentransfer kode ini ke **Advanced Editor** dan mengganti nama kueri *rawData*:

```
let
   rawData = Table.FromColumns(
      {
         {1..3},
         {"Prdct A, 5pcs", "Product B, Qty: 10", "Prdct C; Quantity: 2"},
         {"Prdct A, 5pcs", "Product B, Qty: 10", "Prdct C; Quantity: 2"},
         {"Prdct A, 5pcs", "Product B, Qty: 10", "Prdct C; Quantity: 2"},
         {"Prdct A, 5pcs", "Product B, Qty: 10", "Prdct C; Quantity: 2"},
         {"Prdct A, 5pcs", "Product B, Qty: 10", "Prdct C; Quantity: 2"}
      }, type table [
         ID=Int64.Type, Description=text, Description2=text, Description3=text,
         Description4=text, Description5=text
      ]
   ),
   reSized = Table.Repeat( rawData, 100 )
in
   reSized
```

Gambar 3.38 menunjukkan sebagian kecil tabel contoh:

	ID	Description	Description2
1	1	Prdct A, 5pcs	Prdct A, 5pcs
2	2	Product B, Qty: 10	Product B, Qty: 10
3	3	Prdct C; Quantity: 2	Prdct C; Quantity: 2
4	1	Prdct A, 5pcs	Prdct A, 5pcs
5	2	Product B, Qty: 10	Product B, Qty: 10

Gambar 3.38 Tampilan Terbatas dari Tabel Sampel

Kueri berikut akan menghasilkan tabel *Replacements*, yang membantu dalam memvisualisasikan proses penggantian. Kolom Lama berisi *oldValue*, sub-*string* yang ingin kita ganti dengan *newValue* yang ditemukan di kolom Baru yang berdekatan:

```
let
    Source = Table.FromRows(
        {{"Prdct", "Product"}, {"pcs", " pieces"}, {"Qty",
        "Quantity"}}, type table [Old=text, New=text]
    )
in
    Source
```

Gambar 3.39 menunjukkan tabel *Replacements* secara lengkap:

	Old	New
1	Prdct	Product
2	pcs	pieces
3	Qty	Quantity

Gambar 3. 39 Meja pengganti

a. *Goals*

1) Memperoleh keseragaman (misalnya, mengganti N/A, na, dan none dengan satu nilai yang konsisten).
2) Memperbaiki kesalahan tipografi, kesalahan ejaan, atau ketidakakuratan.

b. Mengumpulkan Hasil

Dalam kueri solusi kita, kita tidak akan memanfaatkan tabel Replacements aktual yang dibagikan sebelumnya; sebagai gantinya, kita akan menggunakan dua list yang mewakili setiap kolom tabel tersebut. Ini berarti *oldValue* dan *newValue* harus selaras dan berbagi indeks posisi yang sama dalam list masing-masing. Kode berikut menampilkan fungsi *List.Accumulate* yang lebih canggih, yang akan dibahas lebih rinci di Bab 1 Jilid 4, Iterasi dan Rekursi. Mari kita uraikan solusi ini:

```
let
    Old = List.Buffer({"Prdct",         "pcs",      "Qty" }),
    New = List.Buffer({"Product", " pieces", "Quantity"}),
    Iterations = List.Buffer({0..List.Count(Old)-1}), Replacer =
        List.Accumulate( Iterations, rawData,
            (s, a) => Table.ReplaceValue( s,
                Old{a},
                New{a}, Replacer.ReplaceText, List.Select(
                    Table.ColumnNames(rawData), each
                    Text.StartsWith(_, "Description")
                )
            )
        )
in
    Replacer
```

Tujuan dari kueri ini adalah untuk mengganti sub*string* tertentu di kolom yang namanya dimulai dengan Deskripsi. Kita akan mengganti elemen dari list Lama dengan elemen yang sesuai dari list Baru di dalam kolom yang dipilih. Berikut cara kerja kueri:

1) *Old = List.Buffer({"Prdct", "pcs", "Qty" })*
   List Lama terdiri dari sub*string* yang dimaksudkan untuk penggantian. Menggunakan List.Buffer memastikan list dievaluasi sekali dan disimpan dalam memori untuk

meningkatkan kinerja. Namun, buffering nilai ke dalam memori tidak menjamin peningkatan kinerja dan perlu diuji.

2) *New = List.Buffer({"Product", "pieces", "Quantity"})*
   List Baru berisi sub*string* yang ditetapkan untuk mengganti sub*string* yang sesuai di list Lama. List.Buffer disertakan untuk meningkatkan kinerja, meskipun tidak dijamin.
3) *Iterations = List.Buffer({0..List.Count(Old)-1})*
   List *Iterations* menghasilkan list indeks dan memuatnya ke dalam memori.
4) *Replacer = List.Accumulate(...)*
   Fungsi ini mengulangi list Iterasi, yang pada dasarnya menyediakan indeks untuk list Lama dan Baru. Fungsi ini menerapkan transformasi ke tabel *rawData*, yang merujuk ke kueri yang berisi kumpulan data sampel. Jika Anda memberi nama kueri tersebut dengan data sampel yang berbeda, Anda perlu memperbarui referensi ini sebagaimana mestinya.
5) *(s, a) => Table.ReplaceValue(...)*
   Benih, dilambangkan sebagai s, adalah nilai awal, yang dalam kasus ini adalah tabel rawData. Akumulator, dilambangkan sebagai a, akan mengasumsikan setiap nilai dalam list *{0..List.Count(Old)-1}*, yang pada dasarnya mewakili indeks untuk meneruskan semua elemen dalam list *Old* dan *New*.
6) *Table.ReplaceValue(s,Old{a},New{a},Replacer .ReplaceText, ...)*
   Fungsi ini mengganti semua contoh sub*string* *Old{a}* dengan *New{a}* dalam kolom tabel yang ditentukan.
7) *List.Select(Table.ColumnNames(rawData), eachText.StartsWith(_, "Description"))*
   Ekspresi ini memilih kolom mana yang akan diganti. Ekspresi ini memilih semua kolom yang namanya dimulai dengan *Description*.

Gambar 3.40 menunjukkan sebagian kecil tabel contoh yang telah diubah:

	ID	Description	Description2
1	1	Product A, 5 pieces	Product A, 5 pieces
2	2	Product B, Quantity: 10	Product B, Quantity: 10
3	3	Product C; Quantity: 2	Product C; Quantity: 2
4	1	Product A, 5 pieces	Product A, 5 pieces
5	2	Product B, Quantity: 10	Product B, Quantity: 10

Gambar 3.40 Pandangan terbatas terhadap hasil

Solusi ini akan terbukti lebih efisien dan dinamis saat mengganti beberapa nilai teks dalam kolom tertentu pada tabel. *List.Accumulate* memungkinkan penggantian nilai per baris dan kolom per kolom tanpa hardcoding, sehingga kueri lebih mudah dikelola. Buffering list Lama dan Baru merupakan langkah pengoptimalan yang memastikan list ini hanya dievaluasi satu kali, sehingga meningkatkan kinerja, secara umum, meskipun hal ini akan selalu memerlukan pengujian.

5. Menggabungkan Baris secara Kondisional

Berikut persyaratan umum lainnya: mengubah *Records* multi-baris ini, dalam kasus ini dari contoh laporan bank (Gambar 3.41), menjadi satu baris.

Untuk mencapainya, kita harus mengumpulkan dan menggabungkan semua detail dari baris kotak dan mengubahnya menjadi catatan tunggal, untuk menghasilkan tabel enam baris, yang tampilan sebagiannya dapat dilihat di sini:

	Date	Details	Debit
1	05-02-24	CloudBliss Shopping	-30,99
2	05-02-24	Grocery Store	-15,50
3	null	Nature's Pantry	null
4	null	Card number 564	null
5	05-02-24	PowerPro Utilities	-90,00
6	null	Period: January 2024	null
7	null	Account: 123456789	null
8	null	Reference Number: PPU-7890	null
9	06-02-24	Gourmet Bistro	-75,50
10	null	Date: February 6, 2024	null
11	null	123 Main Street, Anytown	null
12	null	Card number 564	null

Gambar 3.41 Contoh Data Laporan Bank

Berikut ini adalah contoh data yang dapat Anda jelajahi dan coba:

```
let
    Source = Table.FromRows(
        {
            {#date(2024, 2, 5), "CloudBliss Shopping", -30.99},
            {#date(2024, 2, 5), "Grocery Store",-15.5},
            {null, "Nature's Pantry", null},
            {null, "Card number 564", null},
            {#date(2024, 2, 5), "PowerPro Utilities", -90.0},
            {null, "Period: January 2024", null},
            {null, "Account: 123456789", null},
            {null, "Reference Number: PPU-7890", null},
            {#date(2024, 2, 6), "Gourmet Bistro", -75.50},
            {null, "Date: February 6, 2024", null},
            {null, "123 Main Street, Anytown", null},
            {null, "Card number 564", null},
            {#date(2024, 2, 6), "TrustWise Bank", -150},
            {null, "ATM cash withdrawl", null},
            {null, "Location: 456 Oak Street, Anytown", null},
            {null, "Card number 843", null},
            {#date(2024, 2, 6), "HappyTimes: PQE-789012", 5}
        }, type table [Date=date, Details=text, Debit=Currency.Type]
    )
in
    Source
```

a. *Goals*

Mempersiapkan data untuk analisis dengan memastikan setiap *Record* menempati satu baris.

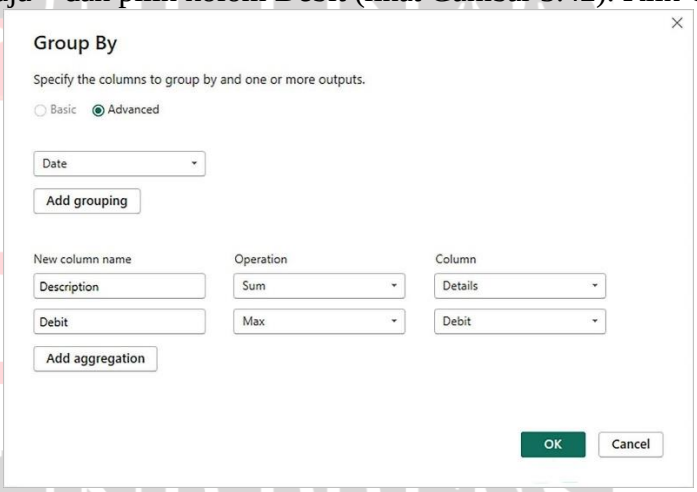
b. Pembanding *Group By* dengan penyelamatan

Beberapa strategi dapat digunakan dalam skenario seperti ini, dengan fungsi Table.Group yang serbaguna sering kali terbukti paling efektif, terutama dalam kasus di mana semua catatan terkait berbagi kunci yang sama. Namun, kesulitan muncul ketika tidak ada kunci bersama dan pembuatannya tidak mudah.

Dalam kasus ini, misalnya, jumlah baris yang saling terkait berfluktuasi; tidak ada pola tetap yang dapat diandalkan. Dari perspektif lain, menggunakan kolom tanggal sebagai kunci tidak cukup untuk memisahkan dan menggabungkan catatan ini. Itu karena kemungkinan ada beberapa transaksi pada setiap tanggal tertentu. Tanpa penanda tambahan untuk

membedakann-ya, itu bukan pilihan yang layak. Namun, *Table.Group* menyedi-akan pembanding opsional yang dapat kita manfaatkan. Mari kita bahas kemungkinan itu:

1) Pertama, kita menggunakan UI untuk membuat sebagian besar kode M. Pilih kolom *Date* dan pilih **Group By** pada tab **Home** atau Transformasi pada pita.
2) Atur ke **Advanced**, masukkan nama kolom baru, Deskripsi, dan pilih operasi. Operasi spesifik tidak penting pada tahap ini karena kita akan memperbaruinya nanti. Pilih kolom *Details*.
3) Karena kita telah memilih **Advanced**, sekarang kita dapat menambahkan agregasi lain. Kali ini, untuk jumlah Debit, tetapkan **Min** atau **Max** sebagai operasi – keduanya boleh saja – dan pilih kolom Debit (lihat Gambar 3.42). Klik **OK**.



Gambar 3.42 Kotak Dialog Group By

4) Sekarang kita dapat mengubah kode di dalam bilah rumus. Jelas bahwa nilai teks tidak dapat dijumlahkan, jadi kita akan mengganti fungsi *List.Sum* dengan *Text.Combine* dan memberikan spasi sebagai argumen keduanya, pemisah.
5) Karena selalu hanya ada satu jumlah debit per transaksi, tidak masalah apakah Anda memilih *List.Min* atau *List.Max*; keduanya akan menghasilkan hasil yang sama.
6) *Table.Group* mengambil argumen keempat opsional, *groupKind*. Kita dapat meneruskan *GroupKind.Local* untuk

menunjukkan bahwa data kita diurutkan berdasarkan kunci dan urutan tersebut harus dipatuhi.

Sekarang mari kita fokus pada argumen kelimanya, pembanding opsional. Sangat penting untuk dipahami bahwa ketika fungsi pembanding kustom diteruskan, fungsi tersebut akan diutamakan daripada kunci yang ditentukan dan bertanggung jawab untuk membuat grup baru. Ini berarti baris mungkin ditempatkan dalam grup yang kuncinya berbeda dari miliknya sendiri.

Berikut fungsi pembanding kustom kita: *(x, y)=> Number.From(y[Date] <> null)*. Dua nilai dilewatkan sekaligus, nilai saat ini (x) dan baris berikutnya (y), dan membuat grup baru jika Tanggal pada baris berikutnya bukan null. Dengan kata lain, jika ini dievaluasi menjadi *false*, *Number.From* mengembalikan 0 (nol) dan menetapkan baris ke grup saat ini. Namun, ketika dievaluasi menjadi *true*, *Number.From* mengembalikan nilai 1 dan membuat grup baru. Berikut data dan solusi lengkap yang baru saja kita buat:

```
let
    Source = Table.FromRows(
      {
        {#date(2024, 2, 5), "CloudBliss Shopping", -30.99},
        {#date(2024, 2, 5), "Grocery Store",-15.5},
        {null, "Nature's Pantry", null},
        {null, "Card number 564", null},
        {#date(2024, 2, 5), "PowerPro Utilities", -90.0},
        {null, "Period: January 2024", null},
        {null, "Account: 123456789", null},
        {null, "Reference Number: PPU-7890", null},
        {#date(2024, 2, 6), "Gourmet Bistro", -75.50},
        {null, "Date: February 6, 2024", null},
        {null, "123 Main Street, Anytown", null},
        {null, "Card number 564", null},
        {#date(2024, 2, 6), "TrustWise Bank", -150},
        {null, "ATM cash withdrawl", null},
        {null, "Location: 456 Oak Street, Anytown", null},
        {null, "Card number 843", null},
        {#date(2024, 2, 6), "HappyTimes: PQE-789012", 5}
      }, type table [Date=date, Details=text,
      Debit=Currency.Type]
    ),
```

```
        #"Grouped Rows" = Table.Group( Source, {"Date"},
        {
          {"Description", each Text.Combine([Details], " "), type
          text},
          {"Debit", each List.Max([Debit]), type number}
        },
        GroupKind.Local,
         (x, y)=> Number.From(y[Date] <> null)
    )
in
    #"Grouped Rows"
```

Gambar 3.43 menunjukkan hasil setelah menerapkan transformasi ini:

	Date	Description	Debit
1	05-02-24	CloudBliss Shopping	-30,99
2	05-02-24	Grocery Store Nature's Pantry Card number 564	-15,5
3	05-02-24	PowerPro Utilities Period: January 2024 Account: 123456789 Referenc...	-90
4	06-02-24	Gourmet Bistro Date: February 6, 2024 123 Main Street, Anytown Car...	-75,5
5	06-02-24	TrustWise Bank ATM cash withdrawl Location: 456 Oak Street, Anytow...	-150
6	06-02-24	HappyTimes: PQE-789012	5

Gambar 3.43 Data Sampel yang Diubah

K. Ringkasan

Bab ini dimulai dengan meletakkan dasar untuk konsep-konsep utama dalam bahasa M, yang sangat penting untuk memahami beberapa seluk-beluk yang dibahas nanti. Kita mengeksplorasi teknik-teknik seperti pemisahan, penggabungan, pembandingan, dan penggantian nilai, antara lain, menggunakan contoh-contoh ilustrasi untuk pemahaman yang lebih baik. Teknik-teknik ini akan terbukti sangat berharga dalam meningkatkan kualitas data. Mencapai kemahiran akan membutuhkan dedikasi dan penerapan praktis yang teratur dalam rutinitas harian Anda – sebuah investasi yang sepadan dengan usaha yang dikeluarkan untuk memperoleh hasilnya.

Di bab berikutnya, Anda akan mempelajari tentang apa itu kesalahan dan bagaimana kesalahan dapat dimunculkan, diatasi, dan dideteksi, serta alat-alat dalam perangkat penanganan kesalahan Anda.

L. Daftar Pustaka

Kyoung-Su Oh and Keechul Jung. (2004), GPU implementation of neural networks. Pattern Recognition, Volume 37, Issue 6, 2004: https://doi.org/10.1016/j. patcog.2004.01.013.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. (2009), Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09): https://doi.org/10.1145/1553374.1553486.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. (2012), ImageNet Classification with Deep Convolutional Neural Networks. Commun. ACM 60, 6 (June 2017), 84–90: https:// doi.org/10.1145/3065386.

Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. (2020). The Computational Limits of Deep Learning. arXiv:2007.05558v1 [cs.LG]: https://arxiv. org/abs/2007.05558v1.

Frank Rosenblatt. (1957), The perceptron – A perceiving and recognizing automaton, Technical Report 85-460-1, Cornell Aeronautical Laboratory.

Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. (2001). On the Surprising Behavior of Distance Metrics in High Dimensional Spaces. In Proceedings of the 8th International Conference on Database Theory (ICDT '01). Springer-Verlag, Berlin, Heidelberg, 420–434: https://dl.acm.org/doi/10.5555/645504.656414.

Nair, V., and Hinton, G.E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. ICML: https://icml.cc/Conferences/2010/papers/432.pdf.

Andrew L. Maas and Awni Y. Hannun and Andrew Y. Ng. (2013). Rectifier nonlinearities improve neural network acoustic models. ICML Workshop on Deep Learning for Audio, Speech, and Language Processing: https://ai.stanford.edu/~amaas/papers/ relu_hybrid_icml2013_final.pdf.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human- Level Performance on ImageNet Classification. 2015 IEEE International Conference on Computer Vision (ICCV), 1026-1034: https://ieeexplore.ieee.org/document/741048.0

Sara Hooker. (2021). The hardware lottery. Commun. ACM, Volume 64: https://doi. org/10.1145/3467017.

M. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan peran fungsi pembanding (comparers) dalam *Power Query* M. Sertakan contoh penggunaannya dalam mengurutkan data atau mengidentifikasi duplikasi.	10
2.	Apa perbedaan antara kriteria perbandingan dan kriteria persamaan dalam konteks fungsi pembanding di *Power Query*? Jelaskan dengan ilustrasi penggunaan dalam sebuah transformasi data.	10
3.	Diskusikan bagaimana Anda dapat membuat pembanding kustom dalam *Power Query* M. Berikan contoh kode dan jelaskan skenario bisnis yang sesuai untuk penggunaannya.	10
4.	Apa fungsi dari pengganti (*Replacer* s) dalam *Power Query* M? Jelaskan bagaimana pengganti dapat membantu membersihkan data dan berikan satu contoh praktis penggunaannya.	10
5.	Bandingkan penggunaan *Replacer* standar dengan *Replacer* kustom dalam *Power Query*. Dalam situasi seperti apa kita lebih baik menggunakan *Replacer* kustom?	10
6.	Bagaimana penggabung (combiners) bekerja dalam *Power Query* M? Berikan contoh penggunaan fungsi combiner untuk menggabungkan nilai dari beberapa kolom menjadi satu *string*.	10
7.	Jelaskan cara kerja splitters dalam *Power Query*. Berikan contoh situasi di mana pemisahan data penting untuk analisis yang lebih lanjut.	10
8.	Apa perbedaan antara fungsi Splitter.SplitTextByDelimiter dan Splitter.SplitTextByWhitespace? Jelaskan dengan contoh praktis penggunaannya masing-masing.	10
9.	Mengapa penting memahami jenis karakter	10

	pemisah saat memproses data teks di *Power Query*? Diskusikan konsekuensi jika karakter pemisah diabaikan.	
10.	Analisis bagaimana penggabungan, pemisahan, perbandingan, dan penggantian nilai dapat meningkatkan kualitas data dan efisiensi pelaporan. Hubungkan jawaban Anda dengan praktik yang relevan dalam pengolahan data *Power* BI.	10

DUMMY
Penerbitan & Percetakan
UNP PRESS

BAB 4
Penanganan Kesalahan (Error) dan Debugging

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Apa itu kesalahan (*Error*)?
- Penahanan kesalahan
- Deteksi kesalahan
- Menimbulkan kesalahan
- Penanganan kesalahan
- Strategi untuk debugging
- Kesalahan umum
- Menyatukan semuanya

A. Pendahuluan

Penanganan kesalahan dalam bahasa M *Power Query* melibatkan teknik untuk mengelola dan mengatasi kesalahan yang mungkin muncul selama evaluasi suatu ekspresi. Dengan memanfaatkan ekspresi dan fungsi seperti try, otherwise, dan catch, Anda dapat menangani potensi kesalahan dengan baik dan mengendalikan alur eksekusi secara efektif. Selain itu, editor *Power Query* menawarkan fitur bawaan yang praktis yang disebut panel Applied Steps, yang memungkinkan Anda menelusuri kode, sehingga memudahkan identifikasi masalah dalam kueri Anda.

Bab ini memberi Anda gambaran umum tentang kapabilitas penanganan kesalahan dan strategi *debugging* dalam *Power Query*. Tujuannya adalah untuk membekali Anda dengan pengetahuan yang diperlukan untuk mengatasi kesalahan secara efektif dan memastikan kueri Anda lebih tahan lama. Dengan memahami kesalahan umum yang dapat terjadi dalam bahasa M dan menerapkan teknik yang dibahas di sini, Anda akan membangun dasar yang kuat untuk menangani kesalahan dan membuat kueri yang lebih tangguh. Bab ini membahas topik-topik berikut: Apa itu kesalahan (*Error*)?, Pengendalian kesalahan, Deteksi kesalahan,

Menimbulkan kesalahan, Penanganan kesalahan, Strategi untuk *debugging*, Kesalahan umum, Menyatukan semuanya.

1. Kasus Pemantik Berfikir Kritis: Masalah Kesalahan Data Impor

Seorang analis data sedang mengimpor data pelanggan dari berbagai file Excel menggunakan *Power Query*. Saat melakukan transformasi, muncul error pada salah satu kolom yang berisi tanggal lahir pelanggan. Setelah diperiksa, ternyata ada beberapa sel kosong dan format tanggal yang tidak konsisten.

Analis tersebut harus segera memperbaiki kesalahan ini agar data dapat diproses untuk keperluan laporan manajemen.

2. Pertanyaan Pemantik

a. Bagaimana langkah-langkah yang harus dilakukan untuk mendeteksi dan memahami penyebab kesalahan tersebut?
b. Strategi penanganan kesalahan apa yang bisa digunakan untuk memperbaiki masalah ini tanpa harus memperbaiki data sumber satu per satu?
c. Bagaimana penerapan fungsi *try ... otherwise* dapat membantu dalam situasi ini?
d. Mengapa penting untuk melakukan *debugging* setelah perbaikan error diterapkan
e. Apa risiko yang mungkin terjadi jika kesalahan data seperti ini tidak segera ditangani dalam laporan manajemen?
f. Bagaimana Anda menentukan apakah error harus diperbaiki di level *Power Query* atau di file sumber datanya?
g. Jika terdapat lebih dari satu jenis error (contohnya: error teks dan error tanggal), bagaimana Anda mengelola dan memprioritaskan penanganannya?
h. Apa kelebihan menggunakan strategi try ... otherwise dibandingkan langsung menghapus baris error?
i. Bagaimana cara memastikan bahwa setelah proses debugging, hasil transformasi data sudah bebas dari kesalahan?
j. Dalam situasi apa Anda memilih untuk membiarkan error tetap ada dalam dataset? Berikan contoh alasannya.

B. Persyaratan Teknis

Untuk memanfaatkan bab ini sebaik-baiknya, kita menganjurkan Anda untuk membuka editor Power Query favorit Anda dan mencoba contoh yang disediakan. Dengan menjalankan dan menjelajahi skrip ini, Anda akan memperoleh pemahaman yang lebih mendalam tentang cara menangani kesalahan dan men-debug.

C. Apa itu Kesalahan (*Error*)?

Dalam bahasa M Power Query, salah satu blok penyusun fundamental adalah ekspresi, yang bertanggung jawab untuk menghasilkan nilai setelah evaluasi. Ketika terjadi kesalahan, ini menunjukkan bahwa evaluasi ekspresi gagal dan tidak dapat diselesaikan dengan sukses. Kesalahan dapat disebabkan oleh berbagai hal, seperti pengidentifikasi atau operasi yang tidak valid, tipe data yang tidak kompatibel, dan banyak lagi. Memahami, mencegah, dan menangani kesalahan secara efektif sangat penting untuk membuat kueri yang lebih tangguh dan andal. Saat mengevaluasi ekspresi M, ada dua kemungkinan hasil:

1. Menghasilkan Satu Nilai

Ini menunjukkan bahwa proses evaluasi berhasil, dan ekspresi mampu menghasilkan hasil.

```
let
    result = 1 / 0
in
    result
```

Pembagian dengan nol tidak menimbulkan kesalahan tetapi menghasilkan tak terhingga.

2. Menimbulkan Kesalahan

Saat kesalahan timbul, ini menunjukkan bahwa evaluasi ekspresi gagal menghasilkan nilai. Kesalahan itu sendiri tidak dianggap sebagai nilai dalam bahasa Power Query M. Catatan kesalahan berisi detail tentang penyebab masalah selama evaluasi. Ini membantu mengidentifikasi operasi yang

bertanggung jawab atas kesalahan dan membantu dalam pemecahan masalah.

```
let
    result = 1 / "0"
in
```

Pembagian dengan operan tipe teks mengembalikan *Expression.Error*.

Sebelum mempelajari penanganan kesalahan, penting untuk memahami bagaimana kesalahan dapat diatasi, dideteksi, dan dimunculkan, yang akan menjadi dasar penanganan kesalahan yang efektif. Menggabungkan teknik pencegahan dan penanganan kesalahan memastikan Anda dapat membangun kueri yang lebih andal dan tangguh yang memberikan hasil yang diharapkan.

D. Penahanan Kesalahan

Bahasa M *Power Query* mengintegrasikan prinsip pemrograman berbasis penahanan. Mari kita bahas apa artinya bagi evaluasi ekspresi.

Penahanan dan penyebaran berarti bahwa meskipun terjadi kesalahan saat mengevaluasi nilai atau ekspresi anggota tertentu, hal itu tidak serta merta menghentikan seluruh proses evaluasi. Sebaliknya, kesalahan dapat "ditahan" tanpa menyebar secara otomatis ke ekspresi tingkat atas.

Bila terjadi kesalahan, evaluasi ekspresi anggota saat ini akan berhenti; secara efektif akan membatalkan, atau membalikkan, bagian ekspresi yang dievaluasi sebelumnya. Namun, selama kesalahan dapat diatasi, kesalahan tersebut tidak akan menyebar ke ekspresi tingkat atas dan suatu nilai mungkin masih dihasilkan.

Bayangkan Anda memiliki tabel yang berisi sel dengan kesalahan; kesalahan ini dapat diatasi dalam bidang catatan. Entri akan ditandai sebagai memiliki kesalahan, dan catatan kesalahan akan disimpan sehingga dapat disebarkan sesuai kebutuhan. Ini

memastikan bahwa dengan setiap upaya berikutnya untuk mengakses sel tersebut, kesalahan yang sama persis akan muncul. Misalnya, perhatikan cuplikan kode berikut yang memiliki kesalahan:

```
let
    A = 1 / "0",
    B = [a = A]
in
    B
```

Kesalahan yang terkandung dalam nilai bidang Record hanya akan menyebar saat diakses.

Ekspresi anggota List, Record, dan Tabel, serta ekspresi let, dievaluasi menggunakan evaluasi lazy. Ini berarti bahwa jika tidak ada yang mencoba mengakses nilai yang salah, ekspresinya mungkin tidak akan pernah dievaluasi, dan bahkan mungkin tidak menimbulkan kesalahan.

Mekanisme utama Power Query adalah penahanan dan penyebaran. Ini berarti bahwa meskipun ada kesalahan di beberapa bagian data, hal itu tidak serta merta menyebabkan seluruh proses evaluasi berakhir sepenuhnya. Sebaliknya, jika kesalahan dapat dibatasi, evaluasi ekspresi dilanjutkan. Namun, kesalahan akan tetap menyebar melalui ekspresi yang mengaksesnya.

E. Deteksi Kesalahan

Saat bekerja di Power Query, menemukan kesalahan adalah bagian alami dari proses pengembangan, terlepas dari apakah Anda mendesain kueri melalui User Interface (UI), memodifikasi kode yang ada, atau menulis kode M dari awal.

Saat Anda menambahkan data "dunia nyata" ke dalam campuran, kesalahan menjadi bagian dari kehidupan yang harus Anda tangani dan kelola, sebuah proses yang dimulai dengan deteksi kesalahan. Kesalahan diklasifikasikan menjadi dua jenis:

1. Kesalahan tingkat langkah
2. Kesalahan tingkat sel

Yang membedakan keduanya adalah penahanannya. Kesalahan tingkat langkah menyebar ke ekspresi tingkat atas dan menyebabkan evaluasi seluruh kueri berakhir, yang menyebabkan kueri gagal. Untuk mengatasi kesalahan tingkat langkah, penting untuk menentukan di mana kesalahan muncul. Bagian Applied Steps dalam panel Query Settings adalah tempat Anda perlu melakukan evaluasi kueri langkah demi langkah. Dengan memilih setiap langkah dan mengevaluasinya, mulai dari atas dan bergerak ke bawah, Anda dapat menentukan lokasi (langkah) yang tepat di mana kesalahan terjadi dan menganalisis masalah yang mendasarinya. Pendekatan ini membantu mengidentifikasi jenis kesalahan ini secara efisien.

Kesalahan tingkat langkah yang paling umum disebabkan oleh hal-hal berikut:

1. Kesalahan sumber data
2. Pengidentifikasi yang tidak diketahui atau hilang
3. Kesalahan sintaksis
4. Kesalahan referensi siklik (atau referensi melingkar)

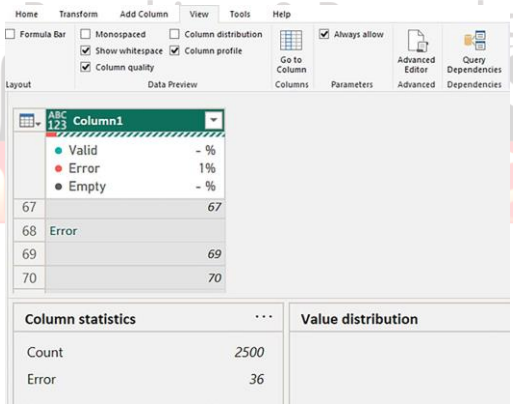
> Mengidentifikasi kesalahan pada tingkat sel adalah masalah yang sama sekali berbeda. Secara default, Anda dapat melihat garis tipis tepat di bawah tajuk kolom, yang merupakan bagian dari opsi pembuatan profil kolom (lihat Gambar 12.1). Jika seluruh garis menunjukkan warna hijau, berarti tidak ada kesalahan yang terdeteksi. Namun, inilah kendalanya: rentang pemindaian pembuatan profil kolom standar dibatasi hingga 1000 baris teratas. Namun, Anda dapat dengan mudah beralih untuk memindai seluruh kumpulan data di bilah status.

Saat ini, Anda tidak perlu khawatir untuk memahami bagian kode M berikut. Skrip ini hanya dimaksudkan untuk mengilustrasikan efek rentang pemindaian profil kolom. Masukkan kode ini ke dalam kueri kosong baru untuk melihat dan mencobanya sendiri:

```
let
    t = Table.FromColumns(
        {
            List.Transform( {1..2500},
                each if Number.Mod(_, 68) =0 then ... else _
        }
    )
i
    t
```

Tidak berubah, pemindaian 1000 baris teratas menunjukkan 14 kesalahan saat mengarahkan kursor ke baris profil kolom kecil tersebut. Namun, jangan ragu untuk memilih tampilan **Data Preview** yang berbeda pada tab **View** seperti yang terlihat pada Gambar 12.1 seperti **Column quality, Column distribution,** atau **Column profile**.

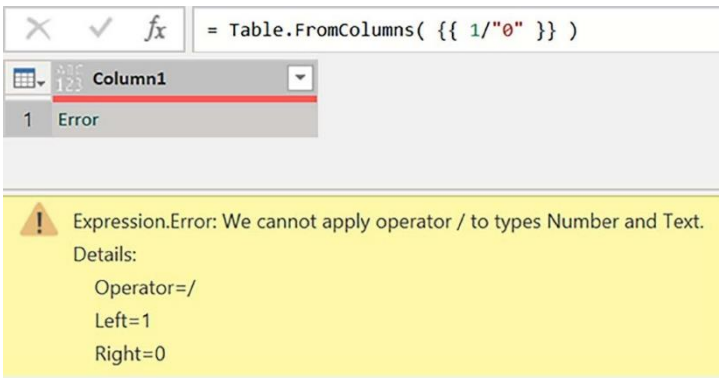
Mengubah rentang pemindaian profil kolom di bilah status dari 1000 baris teratas ke pemindaian seluruh kumpulan data dapat berimplikasi pada deteksi dan kinerja kesalahan. Dalam contoh ini, jumlah kesalahan meningkat dari 14 menjadi 36 saat rentang pemindaian diperluas. Hal ini menyoroti pentingnya kewaspadaan saat memvalidasi data. Secara sepintas, data mungkin tampak bebas kesalahan hingga waktu proses kueri saat Anda dihadapkan pada kegagalan yang tidak terduga. Penting untuk menyadari hal ini karena kesan awal data bebas kesalahan mungkin menipu.



Gambar 4.1 Tab Tampilan dengan Opsi Pratinjau Data dan Kualitas erta Profil Kolom yang Diaktifkan

Setelah keberadaan kesalahan tingkat sel dikonfirmasi, Anda dapat mengambil tindakan yang diperlukan dengan memilih Keep errors di bagian **Reduce Rows** di bawah **Keep rows** pada **tab Home.** Ini memanggil fungsi M yang disebut Table.SelectRowsWithErrors dari UI, menghasilkan tabel baru yang hanya menyertakan baris yang berisi setidaknya satu kesalahan di salah satu selnya. Jika Anda, secara opsional, memberinya list dengan nama kolom, fungsi tersebut akan fokus pada pemeriksaan kesalahan hanya di dalam sel-sel kolom yang ditentukan tersebut. Dengan memanfaatkan pendekatan ini, Anda dapat mengisolasi baris yang bermasalah dengan cepat untuk penyelidikan lebih lanjut.

Untuk melihat detail tentang kesalahan yang timbul, Anda cukup mengeklik spasi putih di sel yang berisi nilai kesalahan. Tindakan ini akan menampilkan pratinjau sekunder di panel pratinjau, Gambar 4.2, yang menunjukkan informasi dari Record kesalahan:



Gambar 4.2 Tampilan Detail Kesalahan

Informasi tersebut mencakup penyebab masalah selama evaluasi, pesan kesalahan, operator, dan nilai yang terlibat. Rincian ini membantu mengidentifikasi operasi yang bertanggung jawab atas kesalahan dan membantu menyelesaikan masalah. Informasi lebih lanjut tentang penyelesaian kesalahan ada di bagian mendatang dalam bab ini.

Dengan mengeksplorasi berbagai teknik dan memanfaatkan kreativitas, menjadi mungkin untuk merancang tabel pelaporan

kesalahan tingkat sel. Namun, sebelum mempelajari topik yang lebih lanjut seperti itu, ada konsep penting lainnya yang perlu dibahas. Yakinlah, contoh tabel pelaporan kesalahan tingkat sel akan dibahas di bagian Menyatukan semuanya.

Kesalahan dapat muncul selama evaluasi bagian mana pun dari ekspresi. Namun, kesalahan juga dapat ditangani dari dalam ekspresi. Kesalahan tingkat langkah terjadi selama evaluasi langkah kueri individual, dan, jika tidak dikelola dengan tepat, kesalahan tersebut menyebar ke ekspresi tingkat atas, menghentikan seluruh eksekusi kueri. Di sisi lain, kesalahan tingkat sel berkaitan dengan sel tertentu dalam tabel, yang menunjukkan masalah dengan kualitas data atau transformasi pada tingkat yang lebih terperinci. Dengan mendeteksi kesalahan secara efektif, Anda dapat menyederhanakan proses *debugging* dan fokus pada penyelesaian masalah secara efisien.

F. Meningkatkan Kesalahan

Dalam bahasa M Power Query, Anda dapat menentukan kesalahan kustom. Ini berguna saat Anda ingin memberikan pesan kesalahan yang lebih spesifik dan bermakna, mampu menangani kasus luar biasa, menerapkan aturan validasi data kustom, atau meningkatkan pengalaman pengguna secara keseluruhan misalnya.

1. Ekspresi Kesalahan

Untuk memunculkan kesalahan, cukup panggil ekspresi yang memunculkan kesalahan error dan gunakan nilai teks untuk memberikan pesan kesalahan yang ingin Anda tampilkan. Misalnya, ekspresi ini akan memunculkan kesalahan berikut:

```
error "Invalid data, you did not provide a table."
```

Kesalahan menyediakan catatan kesalahan; oleh karena itu, Anda juga dapat menentukan catatan untuk menyertakan informasi lebih lanjut tentang kesalahan tersebut. Catatan kesalahan ini harus berisi bidang-bidang seperti Alasan, Pesan,

dan Detail. Berikut ini adalah contoh cara memunculkan kesalahan dengan catatan kesalahan kustom:

```
error [
    Reason = "Invalid data, you did not provide a table.",
    Message = "The data provided is not valid.",
    Detail = [Operator = "&", Left = "123", Right = "321"]
]
```

Namun, Anda juga dapat membuat Record kesalahan dari nilai teks untuk *Reason*, *Message*, dan *Detail* seperti apa pun, dengan memanfaatkan pustaka standar, fungsi Error.Record:

```
error Error.Record(
    "Invalid data, you did not provide a table.",
    "The data provided is not valid.",
    [Operator = "&", Left = "123", Right = "321"]
)
```

Ini mengembalikan *Record* kesalahan yang sama seperti yang diilustrasikan dalam contoh sebelumnya.

Penetapan kesalahan khusus dalam bahasa M telah disempurnakan lebih lanjut dengan diperkenalkannya pesan kesalahan terstruktur pada pertengahan tahun 2022. Pembaruan ini mencakup penambahan dua bidang baru ke Record kesalahan: *Message.Format* dan *Message.Parameters.*

Message.Format menyediakan struktur untuk menghasilkan pesan kesalahan yang dirangkai untuk ditampilkan kepada pengguna. Dalam sebuah string, sepasang tanda kutip tunggal dapat digunakan untuk mengakses item dari list *Message.Parameters*, dalam bentuk ini: *#{x}*, di mana x adalah nomor indeks berbasis nol untuk nilai parameter yang akan dimasukkan. Misalnya, *#{0}* mewakili item pertama dari list *Message.Parameters*.

Message.Parameters adalah list yang berisi nilai-nilai dari mana item dapat diminta dan dimasukkan ke dalam string pesan kesalahan pada posisi yang ditentukan. Jangan khawatir jika saat ini Anda tidak sepenuhnya memahami kode berikut. Masukkan kode M ini ke dalam kueri kosong baru dan ganti nama kueri tersebut menjadi fxDivision untuk mencobanya:

```
(dividend as number, divisor as number) as number => let
      IsZero = divisor = 0,
      Division = if IsZero then ( error [
            Message.Format = "Encountered: '#{0}', resolution:
            '#{1}'", Message.Parameters = { "division by zero
            error",
                "only numbers <> 0 are allowed divisors.",
                dividend, divisor}
          ] ) [Error]
        else dividend / divisor
  in
      Division
```

Melewati nilai argumen 0 ke pembagi fungsi kustom ini, misalnya, *fxDivision(3, 0)*, akan menyebabkan kesalahan berikut: Terjadi kesalahan pada kueri ". Expression.Error: Ditemukan: An error occurred in the " query. Expression.Error: Encountered: 'division by *zero error'*, resolution: *'only numbers <> 0 are allowed divisors*.' Hal ini dapat dilihat pada gambar berikut:

Gambar 4.3 Pesan Kesalahan Khusus

Lebih jauh lagi, bidang pesan kesalahan terstruktur ini juga menyediakan sarana untuk memperoleh semua informasi tentang kesalahan dengan cepat dan mudah tanpa perlu penguraian teks khusus. Nilai masukan dapat diperoleh dari bidang kesalahan Detail dan salah satu masukan *string* dalam pesan kesalahan dari list *Message.Parameters*, yang dapat berguna untuk pelaporan dan analisis kesalahan. Berikut ini adalah sebuah contoh:

```
let
    myErr = error [
        Message.Format = "Encountered: '#{0}', resolution: '#{1}'",
        Message.Parameters = { "division by zero error",
            "only numbers <> 0 are allowed divisors" }
        ],
    CatchErr = try myErr,
    Detail = CatchErr[Error][Message.Parameters]{1}
in
    Detail
```

Ini hanya mengembalikan angka <> 0 adalah pembagi yang diizinkan tetapi Anda dapat mengembalikan item apa pun dengan

posisi indeks berbasis nol. Misalnya, saat membuat laporan penanganan kesalahan yang lebih kompleks, Anda dapat mengekstrak satu atau beberapa item. Saat menampilkan nilai input dalam pesan kesalahan, semua nilai primitif akan ditampilkan sebagai teks. Setiap nilai terstruktur ditampilkan menggunakan nama tipe literalnya (seperti Table, Record, atau List).

Saat kedua bidang kesalahan Message.Parameters dan Message.Format ditentukan, Message.Format diutamakan dan digunakan untuk menghasilkan string yang akan menimpa Message.Parameters dalam kesalahan yang disebarkan.

Menentukan Message.Format tanpa Message.Parameters berlaku selama tidak ada input parameter yang diharapkan dalam string format yang diberikan.

Menentukan Message.Parameters tanpa Message.Format akan menghasilkan nilai null, bukan list parameter dalam kesalahan yang disebarkan.

Terakhir, Message.Format dan Message.Parameters tidak berpengaruh pada bidang kesalahan Reason dan Detail.

Secara keseluruhan, pesan kesalahan terstruktur dalam bahasa M Power Query menyederhanakan proses pencatatan dan pelaporan kesalahan secara signifikan, karena pesan tersebut menawarkan cara yang sistematis dan telah ditetapkan sebelumnya untuk mendapatkan detail kesalahan penting tanpa perlu penguraian teks. Dengan menggunakan fitur ini, Anda dapat membuat pesan kesalahan yang bermakna bagi pengguna dan memfasilitasi analisis masalah data yang lebih baik dalam alur kerja Anda.

2. Operator … (*ellipsis*)

```
Besides the error expression, the M language also includes the
ellipsis operator (three dots); it's important to understand that
this will not only raise an error but also provide a predefined
error message.
```

```
Although, depending on the version of your M (mashup) engine, that
message could be something like Expression.Error: Not Implemented or
Expression.Error: Value was not specified. Here is an example of how
the ellipsis operator can be used:
let
     complexCode =

          if 1 + 1 = 2
          then ...
          else "¯\_(  )_/¯ That's not right."
in
     complexCode
```

Pintasan ini dapat sangat membantu saat mengembangkan ekspresi yang kompleks. Tetapkan elipsis ke cabang kode yang masih dalam tahap pengembangan; Anda dapat menganggapnya sebagai tempat penampung untuk bagian yang masih Anda kerjakan. Ini akan memungkinkan Anda untuk menjalankan dan menguji cabang lain dalam kode Anda, sehingga proses pengembangan terus berjalan.

Membuat kesalahan khusus menawarkan sejumlah kemungkinan seperti menyediakan pesan kesalahan yang lebih informatif, menangani kasus luar biasa, atau bahkan menegakkan aturan validasi data perusahaan, dan meningkatkan pengalaman pengguna secara keseluruhan.

G. Penanganan Kesalahan

Menerapkan strategi penanganan kesalahan tidak hanya terbatas pada penanganan kesalahan yang muncul. Ini juga tentang membuat pilihan dan menggunakan teknik yang tersedia dalam bahasa M untuk mengurangi kesalahan secara efektif dan memastikan kekokohan alur kerja transformasi data Anda. Saat bekerja dengan data, kesalahan seperti nilai yang hilang, tipe data yang tidak kompatibel, dan kalkulasi yang tidak diharapkan atau tidak valid adalah hal yang umum terjadi. Untuk mengatasi tantangan ini, setiap teknik harus dipertimbangkan. Tujuannya adalah untuk menerapkan strategi penanganan kesalahan yang akan membantu menghindari, mengelola, dan menyelesaikan kesalahan di tempat terjadinya, sehingga memastikan hasil yang dapat diprediksi. Berikut ini adalah hal-hal yang ada dalam perangkat Anda:

1. *Coalesce*, Menangani Null

Bahasa M *Power Query* mendukung propagasi null, yang berarti bahwa jika suatu nilai null dalam serangkaian operasi, hasilnya akan null. Kolom a dalam my*Record* menggambarkan hal ini. Perilaku inilah yang membantu mencegah kesalahan yang dapat muncul akibat melakukan operasi pada nilai null. Berikut ini adalah contohnya:

```
let
   myRecord = [
              a = 1 + null,
              b = List.Sum( {1, null} ),
              c = List.Count( null ),
              d = List.Count( x ?? y ?? {} ), x = null,
              y = null
          ]
in
     myRecord
```

Namun, ketika null dilewatkan sebagai argumen ke fungsi yang mengharapkan tipe lain, hal itu dapat menjadi akar penyebab kesalahan, seperti yang akan ditunjukkan oleh kolom c di myRecord. Logika kondisional dapat menjadi solusi, tetapi di Bab 4, Memahami Nilai dan Ekspresi, Anda telah mempelajari bahwa bahasa M juga menyertakan operator penggabungan, yang merupakan tanda tanya ganda (??).

Coalesce beroperasi pada dua nilai, mengembalikan nilai kiri jika bukan null, jika tidak, mengambil nilai kanan. Lebih banyak argumen dapat disertakan untuk membentuk rantai nilai atau ekspresi, memisahkannya dengan operator penggabungan. Jika semua nilai dalam rantai bernilai null, null akan dikembalikan. Menerapkan Coalesce membuat kode lebih ringkas dan mudah dibaca, seperti yang ditunjukkan dalam kode berikut:

```
let
     c = List.Count( x ?? y ?? {} ), x = null,
     y = null
in
     c
```

Di sini, dalam x ?? y ?? {}, hanya tiga nilai yang membentuk rantai, x, y, dan list kosong. Karena list kosong adalah nilai bukan

nol pertama, nilai tersebut dikembalikan sebagai argumen ke fungsi List.Count.

2. Akses Item atau Bidang Opsional

Akses item dan bidang wajib default dalam bahasa M, artinya saat merujuk ke sesuatu yang tidak ada, kesalahan akan muncul. Namun, Anda dapat dengan cepat beralih dari akses item atau bidang wajib ke opsional dengan menambahkan tanda tanya (tunggal) di bagian akhir. Berikut ini adalah contohnya:

```
let
    itemThree = {1..3}{3}?
in
    itemThree
```

List yang kita akses berisi 3 nilai; oleh karena itu, indeks berbasis nol untuk item list terakhir adalah 2. Karena kita menerapkan akses item opsional, ekspresi ini tidak akan menimbulkan kesalahan tetapi malah mengembalikan nilai nol.

Hal yang sama berlaku untuk akses bidang, yang diilustrasikan dalam cuplikan kode berikut:

```
let
    fieldThree = [One = 1, Two = 2][Three]?
in
    fieldThree
```

Record yang kita akses berisi 2 bidang; tidak ada satu pun bidang yang diberi nama Tiga yang berarti akses bidang yang diperlukan akan menimbulkan kesalahan. Karena kita menerapkan akses bidang opsional, ekspresi ini tidak akan menimbulkan kesalahan tetapi malah mengembalikan nilai null.

3. *Enumerator MissingField.Type*

Saat Anda melihat missingField opsional sebagai angka yang dapat dibatalkan dalam list parameter fungsi M pustaka standar, artinya fungsi tersebut memungkinkan Anda untuk mengontrol perilakunya saat menemukan bidang Record atau kolom tabel yang hilang. Secara default, semua bidang atau kolom yang ditentukan wajib diisi, dan yang hilang akan menimbulkan

kesalahan. Namun, fungsi dengan parameter MissingField.Type menawarkan solusi fleksibel untuk menangani situasi ini. Ada tiga opsi yang tersedia:

a. *MissingField.Error* (*0* atau *null*): Ini adalah opsi *default* dan menunjukkan bahwa jika ada kolom atau bidang yang hilang, kesalahan akan muncul.
b. *MissingField.Ignore (1):* Saat opsi ini dipilih, kolom atau bidang yang hilang diabaikan begitu saja. Kolom atau bidang tersebut tidak akan muncul kesalahan dan tidak akan ditampilkan dalam output.
c. *MissingField.UseNull (2):* Jika opsi ini dipilih, bidang atau bidang yang hilang akan disertakan dalam output, tetapi nilainya akan ditetapkan menjadi null. Dengan demikian, Anda dapat menyimpan seluruh data sekaligus menunjukkan tidak adanya informasi tertentu.

```
Various Record and Table functions provide this optional
MissingField. Type parameter, allowing you to tailor the behavior
of these functions to your specific needs, such as
Record.RemoveFields, Table.RemoveColumns, Record.RenameFields,
Table.RenameColumns, Record.ReorderFields, Table.
ReorderColumns, Record.SelectFields, Table.SelectColumns, Record.
TransformFields, and Table.FromRecords.
```

Mengetahui *MissingField.Type* memberdayakan Anda untuk menangani potensi masalah secara efektif dengan mengesampingkan perilaku yang menimbulkan kesalahan default dan memilih untuk mengabaikan bidang yang hilang atau menggunakan nilai null sebagai gantinya.

4. Ekspresi *If*, Logika Kondisional

Anda dapat menggunakan pernyataan kondisional if-then-else untuk memeriksa kondisi tertentu. Misalnya, Anda dapat memeriksa apakah satu atau kombinasi nilai memenuhi kriteria tertentu sebelum melakukan perhitungan apa pun pada nilai tersebut, untuk mencegah potensi kesalahan:

```
let
    a = null ?? {},
    b = if ( a <> null and a <> {} )
        then List.Count( a )
        else 1
in
    b
```

Logika kondisional memungkinkan Anda membuat cabang dalam kode Anda.

5. Ekspresi *Try*

Ekspresi try mencoba mengevaluasi ekspresi dan membuat *Record*. Bergantung pada apakah ekspresi try menangani kesalahan atau tidak, ekspresi tersebut menghasilkan nilai keluaran untuk ekspresi atau kesalahan dari *Record* yang dihasilkan:

```
let
    triedExpressions = [
          validExpression = try 1/0, invalidExpression = try 1/"0"
      ]
in
    triedExpressions
```

Penting untuk dipahami bahwa cakupan ekspresi try terbatas pada ekspresi yang mengikutinya secara langsung.

6. Try dan Otherwise

Konstruksi *try-otherwise* memungkinkan Anda mencoba operasi dan memberikan ekspresi *default* jika terjadi kesalahan. Pertimbangkan ini:

```
let
    a = "24-2-2024",
    b = try Date.FromText( a, [Culture="en-us"])
        otherwise "no valid date"
in
    b
```

Hanya ketika ekspresi *try* mengembalikan kesalahan, ekspresi default dievaluasi. Ini membantu menangani kesalahan dengan cara yang ringkas dan mudah dibaca.

7. *Try* dan *Catch*

Ini menyederhanakan penerapan penanganan kesalahan adaptif, alasannya adalah *otherwise* seperti yang lain, fungsi *catch* satu parameter memiliki akses ke *Record* kesalahan yang dikembalikan *try*. Ini menghilangkan kebutuhan akan pendekatan fungsi kustom yang lebih bertele-tele. Mari kita periksa *Record* ini:

```
[
  myInput = error "A", newValue = myInput + 1,
  catchMyError = try newValue catch (e)=> if e[Message] = "A" then
    "Oh no you've caught myInput error!"
    else e[Reason] & " message: " & e[Message], somethingElse = 1 +
  1
]
```

Ketika kesalahan dimunculkan oleh myInput, kesalahan tersebut akan menyebar ke newValue dan, selanjutnya, ke catchMyError, dan sekarang, tergantung pada pesan kesalahan yang dikembalikan, skenario penanganan kesalahan tertentu akan mengikuti dalam klausa catch ini.

Namun, fungsi catch dengan parameter nol tidak menerima detail Record kesalahan dari try dan, dalam hal itu, setara dengan konstruksi try-otherwise. Penting untuk menyadari bahwa pendekatan ini hanya cocok ketika semua kesalahan harus ditangani dan diatasi dengan cara yang persis sama.

8. Try dan Fungsi *Custom*

Ini menjadi teknik lama dengan diperkenalkannya kata kunci catch ke dalam bahasa M pada bulan Juni 2022. Meskipun Anda mungkin masih menemukannya digunakan dalam skrip kode M sebelum pengenalan catch, fungsi catch dengan satu parameter sekarang menjadi metode standar untuk penanganan kesalahan adaptif.

9. Fungsi Pustaka Standar

Bahasa M juga menyediakan dua fungsi pustaka standar yang dapat membantu dalam penanganan

kesalahan:Table.ReplaceErrorValues dan Table.RemoveRowsWithErrors memungkinkan Anda mengganti atau menghapus nilai kesalahan dari tabel. Memahami cakupan dan batasan fungsi ini akan membantu Anda membuat keputusan yang tepat saat menerapkannya dalam alur kerja transformasi data Anda:

a. *Table.ReplaceErrorValues*: Mengganti nilai kesalahan dalam kolom tabel yang ditentukan dengan nilai baru dari list dengan list errorReplacement. Format list yang berisi list ini adalah sebagai berikut: *{ {column1, value1}, ... }*.
   Hanya satu nilai penggantian per kolom yang diizinkan, dan menentukan kolom lebih dari satu kali dalam list dengan list *errorReplacement* akan menimbulkan kesalahan.
b. *Table.RemoveRowsWithErrors*: Fungsi ini membuat tabel dengan menghapus baris dari tabel yang memiliki setidaknya satu nilai kesalahan di salah satu selnya. Jika Anda memberikan fungsi ini argumen untuk parameter opsional kedua, yang mengambil list dengan nama kolom, fungsi ini hanya akan memeriksa sel-sel di kolom yang ditentukan tersebut untuk kesalahan dan menghapus baris yang sesuai dari tabel.

Singkatnya, bahasa M Power Query menawarkan berbagai metode pencegahan kesalahan serta teknik penanganan kesalahan. Penting untuk dicatat bahwa pilihannya bergantung pada kasus penggunaan dan persyaratan khusus dari proses transformasi data. Biasanya, kombinasi metode dan teknik diperlukan untuk memastikan integritas dan keandalan data.

H. Strategi untuk *Debugging*

Debugging merupakan bagian integral dari proses pengembangan, dan menghadapi situasi di mana kode berperilaku tidak terduga tidak dapat dihindari. Strategi *debugging* yang efektif memainkan peran penting dalam mengidentifikasi dan memperbaiki kesalahan sekaligus meningkatkan pemahaman kode. Praktik terbaik umum untuk men-debug kode M meliputi penggunaan komentar,

mengganti nama langkah agar mudah dibaca, memecah ekspresi kompleks, dan pengujian. Mari kita bahas lebih rinci:

1. Komentar bukan sekadar tempat penampung pemikiran atau deskripsi kode; komentar berfungsi sebagai petunjuk penting yang memandu proses *debugging*. Komentar dapat membantu Anda dan orang lain memahami tujuan dan fungsi berbagai bagian kode, sehingga memudahkan untuk menemukan dan memperbaiki masalah. Selain itu, komentar dapat digunakan untuk menandai area kode yang memerlukan peninjauan lebih lanjut atau rentan terhadap kesalahan.
2. Secara *default*, *Power Query* menetapkan nama generik untuk setiap langkah. Nama-nama ini tidak memberikan banyak wawasan tentang apa yang dilakukan langkah tersebut. Dengan mengganti nama langkah menjadi sesuatu yang lebih deskriptif, Anda membuat kode lebih mudah dibaca dan dipahami. Ini dapat sangat membantu saat Anda atau orang lain mencoba men-debug kode, karena memberikan pemahaman yang lebih baik tentang apa yang dimaksudkan untuk dilakukan setiap langkah. Nama langkah yang deskriptif juga membuat kode lebih mudah dipelihara dan diperbarui. Jika Anda perlu mengubah kueri di masa mendatang, akan jauh lebih mudah untuk menemukan langkah jika langkah tersebut diberi nama yang tepat.
3. Pecah ekspresi kompleks menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola. Bahasa M, dengan sifat fungsionalnya, memungkinkan terciptanya ekspresi bertingkat yang kompleks. Meskipun ekspresi ini dapat menjadi hebat, ekspresi ini juga dapat menjadi tantangan untuk di-debug ketika terjadi kesalahan. Dengan memecah ekspresi yang kompleks, Anda dapat mengisolasi dan memeriksa setiap bagian secara individual, sehingga lebih mudah untuk mengidentifikasi akar penyebab suatu masalah. Pendekatan ini tidak hanya menyederhanakan *debugging* tetapi juga meningkatkan keterbacaan dan pemeliharaan kode ketika dikaitkan dengan variabel yang diberi nama dengan baik.
4. Menguji kode M Anda adalah pendekatan *debugging* proaktif yang membantu mengidentifikasi kesalahan sebelum menjadi masalah, memastikan bahwa kode Anda berfungsi dengan benar.

Selain itu, pengujian akan diperlukan setiap kali perubahan dilakukan pada kode.

5. Di samping praktik terbaik umum ini, ada strategi *debugging* yang hebat yang patut diperhatikan. Anda sudah familier dengan konsep ekspresi let dan *Record*; keduanya digunakan untuk membuat ekspresi yang lebih kompleks dalam bahasa M. Namun, dalam hal pemecahan masalah dan *debugging*, ekspresi *Record* sangatlah hebat.
6. Ekspresi *Record* memungkinkan nama bidang ditulis tanpa notasi kutipan (dalam kebanyakan kasus), yang praktis dan meningkatkan keterbacaan kode. Namun yang lebih penting, ekspresi *Record* sangat fleksibel. Itu karena Anda dapat dengan cepat mengubah nilai pengembaliannya, misalnya, Anda dapat mengembalikan seluruh *Record*, yang memungkinkan Anda melihat semua bidang dan nilai secara sekilas. Jika salah satunya menunjukkan kesalahan, Anda akan segera tahu di mana harus memulai proses *debugging*. Mari kita lihat sebuah contoh:

```
[
    NumE = Number.E,
    RoundUp = Number.RoundUp( NumE, 0), Times Two = RoundUp *2
]
```

Sebagai alternatif, Anda dapat mengembalikan pilihan satu atau beberapa bidang dengan pilihan dan proyeksi:

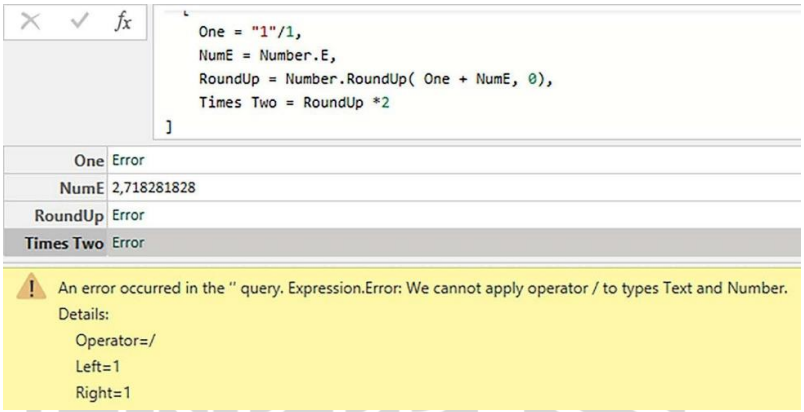
```
[
    NumE = Number.E,
    RoundUp = Number.RoundUp( NumE, 0), Times Two = RoundUp *2
] [ [RoundUp], [Times Two] ]
```

Terakhir, Anda dapat mengembalikan nilai tunggal dengan menerapkan akses bidang:

```
[
    NumE = Number.E,
    RoundUp = Number.RoundUp( NumE, 0), Times Two = RoundUp *2
] [Times Two]
```

Ekspresi *Record* menawarkan kemampuan debugging yang lebih baik dibandingkan dengan ekspresi let. Sementara ekspresi let hanya mengembalikan satu nilai—bahkan jika terstruktur—ekspresi Record memungkinkan Anda untuk melihat semua

kolom dan nilai pada saat yang bersamaan. Fitur ini sangat berguna saat memecahkan masalah ekspresi yang rumit atau bertingkat:



Gambar 4.4 Meninjau Semua Bidang *Record* untuk Memudahkan *Debugging*

Dengan sekali lihat, Anda dapat melihat semua kolom dalam *Record.* Kolom tersebut menunjukkan tiga kesalahan, dan kolom *RoundUp dan Times Two* berisi kesalahan yang disebarkan oleh One saat membagi teks dengan tipe angka, sehingga kesalahan mudah dilacak dan diperbaiki.

Debugging yang efektif dicapai melalui desain query yang cermat, menerapkan gabungan berbagai tindakan seperti menambahkan komentar, mengganti nama langkah, memecah ekspresi yang rumit, memanfaatkan ekspresi *Record*, dan menggabungkan penanganan kesalahan. Semua ini jika digabungkan tidak hanya menghasilkan kode yang lebih tangguh tetapi juga lebih mudah dipelihara.

I. Kesalahan Umum

Saat Anda menemukan dan menangani kesalahan dalam alur kerja *Power Query*, penting untuk meninjau pola dan belajar darinya. Menganalisis kesalahan umum dan memahami akar penyebabnya dapat memandu Anda dalam menyempurnakan pendekatan penanganan kesalahan. Oleh karena itu, bagian ini berfokus pada beberapa kesalahan paling umum yang mungkin

Anda temukan. Lebih jauh, penting untuk memahami bahwa beberapa kesalahan dapat ditemukan selama *debugging*, dan kesalahan sintaksis akan selalu muncul terlebih dahulu karena kesalahan tersebut membuat mesin M (*mashup*) tidak dapat mengurai kode yang diberikan.

1. Kesalahan Sintaksis

Kesalahan sintaksis dalam suatu ekspresi biasanya teridentifikasi selama fase penulisan kode. Kesalahan ini menunjukkan bahwa ada masalah dengan pola atau struktur ekspresi yang diberikan.

Koma yang hilang pada baris 3 mengembalikan *Expression.SyntaxError*: *Token ',' expected*:

```
let
    A = 1,
    B = 0
    result = A / B
in
    result
```

Tanda koma tambahan di dalam list pada baris 2 mengembalikan *Expression.SyntaxError: A ',' cannot precede a '}'*:

```
let
    maxValue = List.Max( {1, 4, 6, 8, } )
in
    maxValue
```

Namun, perlu diingat bahwa pesan kesalahan sintaksis terkadang dapat menyesatkan.

Tanda kurung penutup yang hilang akan mengembalikan *Expression.SyntaxError: Token ',' expected*:

```
let
    maxValue = List.Max( {1, 4, 6, 8 }
in
    maxValue
```

Tanda kurung penutup tambahan mengembalikan *Expression.SyntaxError: Token ',' expected*:

```
let
    maxValue = List.Max( {1, 4, 6, 8 }))
in
    maxValue
```

Namun, saat menambahkan koma di akhir baris 2, tanda kurung tutup tetap hilang:

```
let
    maxValue = List.Max( {1, 4, 6, 8 },
in
    maxValue
```

Ini akan mengembalikan *Expression.SyntaxError: Token Literal expected*, bukan Expression yang lebih jelas. *SyntaxError: A ',' tidak dapat mendahului 'in'.*

Penyebab paling umum untuk *Expression.SyntaxError* adalah koma, tanda kurung, dan tanda kurung tambahan atau hilang, antara lain. Oleh karena itu, cara untuk mengatasinya adalah dengan memeriksa ekspresi secara cermat untuk setiap koma, tanda kurung, tanda kurung, dan sejenisnya yang tambahan atau hilang.

2. Menangani Kesalahan – Prioritas Utama

Setiap kali suatu langkah menyebabkan kesalahan dalam kueri Anda, kita sarankan untuk mengatasinya terlebih dahulu sebelum melakukan tindakan lebih lanjut. Membiarkan kesalahan tidak ditangani dapat memengaruhi kueri Anda. Cara *Power Query* diimplementasikan dapat menyebabkan kesalahan diabaikan oleh fungsi seperti *Table.RemoveRowsWithErrors*. Misalnya, ini adalah kueri tempat kesalahan yang disengaja disebabkan oleh konversi tipe kolom tipe campuran.

```
let
  Source = Table.FromColumns(
    {
      {123, 34, 12, "txt", null},
      {123, "txt", 34, 12, null},
      {"txt", 123, 34, 12, null},
      {123, "txt", "34", 12, null}
                                }
),
  ChType = Table.TransformColumnTypes( Source,
    {
      {"Column1", Int64.Type},
      {"Column2", Int64.Type},
      {"Column3", Int64.Type},
      {"Column4", Int64.Type}
    }),
  RemoveBlanks = Table.SelectRows( ChType, each not List.IsEmpty(
      List.RemoveMatchingItems( Record.FieldValues(_), {"", null})
  )),
  RemoveErrors = Table.RemoveRowsWithErrors( RemoveBlanks )
in
  RemoveErrors
```

Pada baris 17, baris kosong dihapus dari tabel menggunakan *Table.SelectRows*. Namun, operasi ini tidak mengatasi kesalahan tingkat sel apa pun yang terjadi karena konversi tipe yang tidak kompatibel. Ketika fungsi *Table.RemoveRowsWithErrors* diterapk-an pada baris 21 untuk menghapus baris dengan kesalahan, kesalahan tetap ada, dan tabel yang dihasilkan akan menyertakan baris dengan kesalahan:Kunci untuk menyelesaikan ini terletak pada urutan operasi; praktik terbaik adalah menangani kesalahan segera. Dengan mengklik kanan langkah bernama *RemoveErrors* di bagian **Applied Steps** pada panel **Query Settings** dan memilih **Move before**, Anda mengubah urutan operasi. Penyesuaian ini memastikan bahwa *Table.RemoveRowsWithErrors* dijalankan sebelum *Table.SelectRows* untuk menghapus baris kosong. Hasilnya, kesalahan yang disebabkan oleh konversi tipe dihilangkan dengan benar, dan kueri mengembalikan satu baris data yang valid tanpa kesalahan yang terus-menerus.

Kunci untuk menyelesaikan ini terletak pada urutan operasi; praktik terbaik adalah menangani kesalahan segera. Dengan mengklik kanan langkah bernama *RemoveErrors* di bagian **Applied Steps** pada panel **Query Settings** dan memilih **Move**

before, Anda mengubah urutan operasi. Penyesuaian ini memastikan bahwa *Table.RemoveRowsWithErrors* dijalankan sebelum *Table.SelectRows* untuk menghapus baris kosong. Hasilnya, kesalahan yang disebabkan oleh konversi tipe dihilangkan dengan benar, dan kueri mengembalikan satu baris data yang valid tanpa kesalahan yang terus-menerus.

3. *DataSource.Error*, Tidak dapat Menemukan Sumbernya

Kesalahan ini terjadi saat sumber data tidak dapat diakses. Penyebab umumnya termasuk pengguna tidak memiliki akses ke sumber data atau sumber telah dipindahkan:

```
let
    Source = Excel.Workbook(
        File.Contents( "C:\ThereIsNoSuchFile.xlsx"), null,
        true
    )
in
    Source
```

```
DataSource.Error: Could not find file 'C:\ ThereIsNoSuchFile.xlsx'.
```

4. Pengidentifikasi yang Tidak Diketahui atau Hilang

Pengenal berfungsi sebagai nama yang digunakan untuk merujuk ke nilai tertentu, seperti kueri, variabel (langkah), atau nama bidang *Record*. Pengenal dapat dikategorikan menjadi pengenal biasa dan pengenal yang dikutip. Pengenal yang dikutip memungkinkan penggunaan urutan karakter apa pun, termasuk kata kunci, spasi, operator, dan tanda baca, misalnya, *#"error logic"*. Pengenal biasa, di sisi lain, digunakan untuk memberi nama dan mengakses bidang dan juga dikenal sebagai pengenal umum, yang tidak memerlukan notasi kutipan ini.

Jika Anda menemukan pesan kesalahan yang mirip dengan *name 'xxxxx' wasn't recognized. Make sure it's spelled correctly.*, ini menunjukkan bahwa Anda telah merujuk ke pengenal yang tidak dapat ditemukan. Untuk mengatasi masalah ini, Anda perlu menyelidiki di mana pengenal yang tidak dikenal itu digunakan

dan menentukan nilai mana yang harus dirujuknya, untuk memperbaruinya secara akurat. Atau, Anda mungkin perlu membuat variabel dengan nama itu jika tidak ada dan menetapkan nilai yang sesuai untuk itu. Jika secara khusus merujuk ke nama '_', penyebabnya adalah hilangnya kata kunci *each*.

5. Fungsi yang Tidak Diketahui

Kesalahan ini memunculkan pesan yang sama: *Expression.Error: The name 'list.Max' wasn't recognized*. Pastikan ejaannya benar, karena terdapat pengenal yang tidak diketahui atau hilang. Penyebab umumnya adalah kesalahan ejaan dan penggunaan huruf besar yang salah:

```
let
    maxValue = list.Max( {1, 4, 6, 8 } )
in
    maxValue
```

Nama fungsi M mengikuti pola yang konsisten, yang terdiri dari kategori fungsi yang diikuti oleh titik dan nama. Bila nama terdiri dari kata majemuk, konvensinya adalah mengapitalkan huruf pertama setiap kata yang ditambahkan. Misalnya, perhatikan fungsi untuk memilih baris dalam tabel, yang merupakan bagian dari kategori fungsi tabel dan memiliki nama *SelectRows*. Oleh karena itu, fungsi yang memenuhi syarat dilambangkan sebagai *Table.SelectRows*. Mengetahui konvensi ini membantu menemukan kesalahan dalam nama fungsi lebih cepat.

Pengenalan *Intellisense* di *Power Query* menandai peningkatan signifikan dalam pengalaman pengembangan bagi pengguna Power BI Desktop (rilis September 2018), Excel 365, dan versi Excel 2019. Dengan *Intellisense* diaktifkan, pengembang dapat mulai mengetik fungsi atau nama variabel dan akan disajikan dengan list turun bawah berisi kemungkinan pelengkapan. Pengguna dapat dengan cepat memilih fungsi atau variabel yang diinginkan dari saran tanpa harus mengetiknya sepenuhnya, sehingga mengurangi kemungkinan kesalahan dan kesalahan ketik.

Namun, pengguna perlu menyadari masalah yang ada pada *Intellisense*. Masalah tersebut dapat menyebabkan istilah terduplikasi, misalnya, saat mengetik *list.* untuk mencari *List.Max*, Intellisense dapat melengkapi pilihan pengguna secara otomatis ke *listList.Max*. Jika Anda mengalami hal ini, berikut adalah dua cara untuk menghindarinya:

1) Hilangkan titik saat mengetik fungsi, seperti: *TableSelectR*
2) Misalnya, masukkan hanya huruf kapital dari nama fungsi. Jika diinginkan, ketik lebih banyak, seperti tsrow untuk *Table.SelectRows*

Kedua metode akan menghasilkan fungsi yang diinginkan tanpa duplikasi istilah.

6. Referensi Kolom yang Tidak Diketahui

Bila Anda menemukan pesan galat seperti *Expression.Error*: *The column 'xxxxx' of the table wasn't found*, ini menunjukkan bahwa Anda memiliki referensi kolom yang dikodekan secara kaku yang tidak cocok dengan salah satu nama kolom yang ada pada tabel. Ada beberapa alasan terjadinya galat ini:

a. Mungkin ada galat masukan pengguna, seperti nama kolom yang salah eja saat menentukan referensi.
b. Mungkin ada perubahan yang dilakukan pada nama kolom di sumber data setelah desain kueri awal, yang menyebabkan referensi tidak cocok.
c. Kolom mungkin telah dihilangkan dari sumber data, dan akibatnya, referensi ke kolom yang hilang tersebut menyebabkan kesalahan.
d. Pengguna mungkin telah melakukan modifikasi pada langkah kueri sebelumnya, yang menyebabkan referensi kolom menjadi salah, dan seterusnya.

Cuplikan kode berikut adalah contoh kejadian di mana kesalahan tersebut mungkin terjadi:

```
let
    Source = #table(2, {{1, 2}}),
    RenameCols = Table.RenameColumns( Source,
        {
            {"Column 1", "ColumnA"},
            {"Column 2", "ColumnB"}
        } )
in
    RenameCols
```

Untuk mengatasi galat ini, Anda harus meninjau kode dengan saksama dan memeriksa ulang referensi kolom terhadap nama kolom yang sebenarnya dalam tabel. Pastikan nama kolom ada, dieja dengan benar, dan memiliki huruf besar yang benar. Saat membuat perubahan pada sumber data atau langkah sebelumnya, pastikan untuk memperbarui semua referensi kolom yang terpengaruh sebagaimana mestinya, lebih jauh di hilir dalam kode. Galat ini akan terus berlanjut hingga semua masalah nama kolom telah teratasi.

Untuk mengatasi galat ini, Anda harus meninjau kode dengan saksama dan memeriksa ulang referensi kolom terhadap nama kolom yang sebenarnya dalam tabel. Pastikan nama kolom ada, dieja dengan benar, dan memiliki huruf besar yang benar. Saat membuat perubahan pada sumber data atau langkah sebelumnya, pastikan untuk memperbarui semua referensi kolom yang terpengaruh sebagaimana mestinya, lebih jauh di hilir dalam kode. Galat ini akan terus berlanjut hingga semua masalah nama kolom telah teratasi.

Atau, Anda dapat menggunakan strategi yang berbeda untuk mengurangi galat ini. Jika memungkinkan, pertimbangkan untuk menggunakan fungsi M yang memungkinkan parameter *missingField* opsional. Parameter ini memungkinkan Anda untuk mengontrol perilaku fungsi saat bidang yang hilang ditemukan, sehingga memberikan fleksibilitas lebih dalam menangani situasi seperti itu. Selain itu, Anda dapat mengeksplorasi opsi untuk menggunakan referensi kolom dinamis alih-alih nama hardcoding. Dengan memanfaatkan referensi kolom dinamis, kueri Anda dapat beradaptasi dengan perubahan yang terjadi.

7. Referensi Lapangan yang Tidak Diketahui

Hal ini sangat mirip dengan kesalahan referensi kolom yang tidak diketahui tetapi terjadi saat mencoba mengakses, memilih, atau mencari bidang, dan nama bidang yang ditentukan tidak cocok dengan salah satu nama bidang yang sebenarnya dalam *Record* (atau baris tabel).

Alasan dan penyelesaian untuk kesalahan ini mirip dengan penanganan kesalahan referensi kolom yang tidak diketahui dengan tambahan penting bahwa jika kesalahan ini muncul saat melakukan akses atau pencarian bidang (yang diperlukan), nama bidang yang ditentukan diharapkan ada:

```
let
    GetName = [Initial = "M", Name="Mashup"][Name2] // required,
    error
in
    GetName
```

Anda dapat beralih ke akses bidang opsional dengan menambahkan satu tanda tanya setelah akses bidang atau operator pencarian, seperti ini: *[Name2]?*. Ini akan mengembalikan null alih-alih memunculkan kesalahan saat bidang tidak ditemukan dalam *Record*:

```
let
    GetName = [Initial = "M", Name="Mashup"][Name2]? // optional,
    null
in
```

Untuk mengatasi galat ini, Anda harus meninjau kode dengan saksama dan memeriksa ulang referensi bidang terhadap nama bidang yang sebenarnya dalam *Record*. Pastikan nama bidang ada, dieja dengan benar, dan memiliki huruf besar yang benar.

Jika memungkinkan, pertimbangkan untuk menggunakan fungsi M yang memungkinkan parameter *missingField* opsional. Parameter ini memungkinkan Anda untuk mengontrol perilaku fungsi saat bidang yang hilang ditemukan, sehingga memberikan fleksibilitas lebih dalam menangani situasi seperti itu. Selain itu, Anda dapat mengeksplorasi opsi untuk menggunakan referensi bidang dinamis alih-alih nama yang dikodekan secara kaku. Dengan memanfaatkan referensi bidang dinamis, Anda dapat beradaptasi dengan perubahan nama bidang.

8. Tidak Cukup Elemen dalam Enumerasi

Bahasa M menggunakan indeks berbasis nol untuk tabel dan list. Pelajari semua tentang list dan akses item, yang memungkinkan Anda mengekstrak elemen list berdasarkan

posisinya, di Bab 6, Nilai Terstruktur. Seperti akses bidang atau pencarian, akses item juga dilakukan secara wajib secara *default* , yang berarti posisi yang ditentukan diharapkan ada.

Pertimbangkan kueri ini: fungsi *List.Numbers* menghasilkan list dengan 3 elemen atau item list { 1, 2, 3 }; menerapkan akses item dan meneruskan nilai 3 sebagai nomor indeks akan mencoba mengekstrak item keempat dari list nilai, yang memunculkan *Expression.Error*: *There weren't enough elements in the enumeration to complete the operation*:

```
let
    ItemAccess = List.Numbers( 1, 3, 1 ){3}  // required, error
in
    ItemAccess
```

Anda dapat beralih ke akses item opsional dengan menambahkan satu tanda tanya setelah operator akses item, seperti berikut: *{3}?*:

```
let
    ItemAccess = List.Numbers( 1, 3, 1 ){3}? // optional, null
in
    ItemAccess
```

Ini akan mengembalikan null alih-alih memunculkan error saat posisi indeks yang diminta tidak ada dalam list. Atau, Anda dapat menggunakan strategi yang berbeda. Pertimbangkan untuk menggunakan fungsi pustaka standar seperti *List.Range*:

```
let
    ItemAccess = List.Range( List.Numbers( 1, 3, 1 ), 3, 1 ) // {}
in
    ItemAccess
```

Fungsi ini akan mengembalikan list kosong dan tidak akan memunculkan kesalahan saat nomor offset melebihi jumlah elemen dalam list.

9. *Formula.Firewall Error*

Berikut adalah hasil dari Firewall Privasi Data *Power Query*. Tujuannya sederhana: untuk mencegah *Power Query*

membocorkan data antar sumber secara tidak sengaja. Firewall ini menimbulkan kesalahan seperti:

Formula.Firewall: Query 'Query1' (step 'Source') references other queries or steps, so it may not directly access a data source. Please rebuild this data combination.

Atau:

Formula.Firewall: Query 'Query1' (step 'Source') is accessing data sources that have privacy levels which cannot be used together. Please rebuild this data combination.

Sering kali, Anda akan menemukan dua metode untuk mengatasi kesalahan ini tanpa menonaktifkan Firewall. Anda dapat menggabungkan *Query1* dan *Query2* menjadi satu kueri atau mengubah kueri ini menjadi fungsi parameter nol untuk menghindari kesalahan *Formula.Firewall*. Meskipun yang terakhir berfungsi di *Power* BI Desktop dan Excel, fungsi ini tidak berfungsi di layanan *Power* BI.

Pelajari selengkapnya tentang kesalahan Formula.Firewall di Gbapter 15, Mengoptimalkan Performa.

10. *Expression.Error*: Kunci Tidak Cocok dengan Baris Manapun di Tabel

Kesalahan ini terjadi saat *Power Query* tidak dapat menemukan nama yang dicarinya. Penyebab umumnya meliputi nama yang salah eja atau diubah, hak istimewa akun yang digunakan untuk mengakses tabel tidak mencukupi, atau beberapa kredensial untuk satu sumber data, yang tidak didukung dalam layanan *Power* BI. *Expression.Error* dalam kode berikut disebabkan oleh nilai kunci tabel yang tidak cocok dalam ekspresi pencarian:

```
let
    Source = Table.FromRows(
        {
            {"YourTable", "Table", #table(2, {{1, 2}}) },
            {"YourTable2", "Table", #table(2, {{1, 2}}) }
        }, {"Item", "Kind", "Data"}
    ),
    Navigation = Source{[Item= "myTable", Kind="Table"]}[Data]
in
    Navigation
```

Untuk mengatasi galat dalam contoh ini, Anda perlu mengubah *myTable* menjadi *YourTable* pada baris ke-8 kode M.

Galat ini sering ditemui saat menggunakan konektor *Folder* untuk menggabungkan berkas. Saat menggunakan konektor *Folder*, semua berkas diproses berdasarkan struktur berkas contoh yang dipilih. Misalnya, jika Anda memilih berkas Excel dengan tabel bernama *myData*, semua berkas lain dalam folder akan diperlakukan sebagai berkas Excel dengan tabel bernama *myData*. Jika terdapat variasi dalam struktur berkas, seperti tabel yang salah eja atau diberi nama berbeda, *Expression.Error* dapat terjadi selama penggabungan data. Untuk menghindari munculnya galat atau kehilangan data, penting untuk meninjau dan memverifikasi struktur berkas dengan saksama serta memastikan konsistensi sebelum menggabungkannya dengan teknik khusus ini.

11. *Expression.Error*: Kunci Cocok dengan Lebih dari Satu Baris dalam Tabel

Bahasa Indonesia: Bila Anda menemukan pesan galat seperti *Expression.Error*: *The key matched more than one row in the table*, ini menunjukkan bahwa operasi pencarian mengembalikan lebih dari satu baris. Galat ini muncul saat menggunakan operator akses item { } untuk mencari baris dalam tabel, dengan bidang dan nilai kunci yang disediakan dalam tanda kurung siku, [ ]. Mesin *Power Query* M (*Mashup*) menggunakan sintaksis ini khususnya untuk tabel dengan kunci utama dalam kasus navigasi atau *drill-down*; semua bidang kunci tabel dan nilai bidang yang cocok dicantumkan untuk mengidentifikasi baris yang diinginkan secara unik. *Expression.Error* di sini disebabkan oleh kegagalan

untuk menentukan nilai untuk semua kunci dalam ekspresi pencarian:

```
let
    Source = Table.FromRows(
        {
            {"YourTable", "Table", #table(2, {{1, 2}}) },
            {"YourTable2", "Table", #table(2, {{1, 2}}) }
        }, {"Item", "Kind", "Data"}
    ),
    Navigation = Source{[Kind="Table"]}[Data]
in
    Navigation
```

Untuk mengatasi galat dari contoh tersebut, Anda perlu menyertakan kolom *Item* dan menetapkan nilai yang cocok pada baris 8 kode M.

Galat ini muncul saat modifikasi manual dilakukan pada kode M yang dihasilkan UI. Galat ini juga dapat terjadi saat metode pencarian diterapkan secara manual pada tabel yang tidak memiliki nilai pencocokan kunci yang cukup atau saat tabel mungkin berisi baris duplikat. Untuk mencegah galat ini, gunakan fungsi *Table.Keys* untuk memastikan apakah tabel memiliki kunci utama. Fungsi ini akan menampilkan list dengan *Record* yang berisi dua kolom: *Columns* sebagai list nama kolom yang membentuk kunci utama untuk tabel, jika ada, dan Utama sebagai nilai Boolean. Memastikan nilai yang cocok diberikan untuk setiap kolom kunci akan menghindari munculnya galat ini.

12. *Expression.Error*: Evaluasi Mengakibatkan Tumpukan Berlebih dan Tidak dapat Dilanjutkan

Kesalahan stack *overflow* terjadi saat evaluasi kode M Anda menghabiskan memori yang tersedia di tumpukan panggilan. Hal ini dapat terjadi karena bug dalam kode Anda, seperti mengevaluasi fungsi rekursif tanpa kondisi akhir apa pun. Fungsi rekursif dapat menyebabkan stack *overflow* saat fungsi tersebut berulang kali memanggil kembali dirinya sendiri tanpa kondisi penghentian yang tepat. Cara umum untuk mengatasi stack *overflow* meliputi penerapan struktur kontrol:

```
let
    NoEnd = (x) => @NoEnd (x + 1)
in
    NoEnd(0)
```

Untuk mengilustrasikan pernyataan *if-then-else* pada baris 2, periksa apakah x kurang dari 100. Jika kondisinya benar, fungsi tersebut memanggil dirinya sendiri secara rekursif dengan operasi *x + 1* sebagai input baru. Jika tidak, jika x sama dengan atau lebih besar dari 100, fungsi tersebut mengembalikan nilai x, yang mengakhiri panggilan rekursif:

```
let
    NoEnd = (x) => if x <100 then @NoEnd(x + 1) else x
in
    NoEnd(0)
```

Kuncinya adalah memastikan bahwa kondisi penghentian pada akhirnya akan terpenuhi selama eksekusi fungsi. Tanpa kondisi penghentian yang tepat, fungsi akan terus memanggil dirinya sendiri tanpa henti, yang menyebabkan kesalahan *stack overflow*.

Lebih jauh, Anda dapat mempertimbangkan untuk mengganti rekursi dengan iterasi menggunakan fungsi seperti *List.Transform*, *List.Generate*, atau *List.Accumulate*. Fungsi iteratif ini dapat melakukan tugas yang sama seperti fungsi rekursif tetapi dengan cara yang lebih hemat memori, sehingga mengurangi risiko kesalahan stack *overflow*. Ingatlah untuk memverifikasi kondisi penghentian dan mengoptimalkan kode M untuk kinerja dan keandalan yang lebih baik.

J. Menyatukan Semuanya

Sepanjang bab ini, Anda telah memperoleh wawasan tentang berbagai aspek penanganan kesalahan di *Power Query*. Anda telah mempelajari tentang sifat kesalahan, cara memunculkannya secara sengaja, serta cara mengatasi, mendeteksi, dan menangani kesalahan. Dengan pengetahuan dasar ini, sekarang saatnya untuk mengonsolidasikan pembelajaran Anda dengan mempelajari dua contoh praktis yang menunjukkan skenario nyata penanganan kesalahan dalam tindakan. Contoh-contoh ini akan memberikan pemahaman yang lebih mendalam tentang cara mencegah kesalahan,

membuat pilihan yang disengaja, dan mengatasi kesalahan saat muncul.

Jangan ragu untuk merancang pendekatan Anda sendiri terhadap skenario ini sebelum mendalaminya. Jika Anda tidak nyaman menerjemahkannya ke dalam kode M, tidak masalah. Anda masih dapat membuat kerangka dan memikirkan kendala yang harus diatasi. Ini akan memungkinkan Anda untuk membandingkan strategi Anda dengan yang disarankan di sini. Namun ingat, ada banyak solusi untuk setiap masalah, dan buku ini hanya membahas satu pendekatan yang mungkin.

1. **Pemilihan Kolom**

   Tugas yang sangat umum adalah memilih kolom dari tabel. Ini dapat dilakukan dengan mudah melalui UI. Namun, dapatkah Anda memikirkan metode untuk memfasilitasi pemilihan kolom dengan tujuan membuat proses ini lebih mudah digunakan, tangguh, dan andal?

2. **Membangun Solusi Khusus**

   Kita akan memandu Anda melalui proses dan pengembangan kode, menjelaskan setiap bagian langkah demi langkah. Setelah kita membahas semuanya, kita akan membagikan contoh kode lengkap sehingga Anda dapat melihat bagaimana semuanya berjalan sesuai rencana.

   Mari kita fokus pada kemudahan penggunaan terlebih dahulu. Karena bahasa M peka huruf besar-kecil, dan kesalahan ejaan mudah terjadi, mari kita bahas cara menyediakan list dengan nomor kolom, bukan list dengan nama kolom. Namun, berikut pertimbangan penting: nomor kolom tidak boleh berbasis nol. Dari sudut pandang intuitif, kita terbiasa menghitung dari angka satu.

   Akan lebih mudah juga jika menyediakan tombol untuk memilih atau menghapus kolom. Bayangkan skenario saat pengguna ingin menyimpan semua kolom kecuali satu atau beberapa. Meminta pengguna untuk menyediakan list semua

posisi kolom lainnya akan merepotkan dan rawan kesalahan. Untuk menyederhanakan proses, menyertakan opsi untuk beralih antara menyimpan atau menghapus kolom yang ditentukan akan sangat meningkatkan kegunaan dan mengurangi potensi kesalahan.

Dengan mempertimbangkan pertimbangan ini, kita telah mengidentifikasi tiga parameter masukan utama untuk fungsi kustom kita. Pertama, kita memerlukan tabel untuk melakukan operasi; kedua, list dengan posisi kolom (dalam format berbasis satu) akan menunjukkan kolom tertentu yang memerlukan tindakan; dan terakhir, Boolean yang berfungsi sebagai opsi sakelar untuk menentukan apakah akan menyimpan atau menghapus kolom yang disediakan akan menambah tingkat kemudahan penggunaan bagi pengguna. Mari kita lihat sebuah contoh:

```
2    ( myTable as table, optional colPositions, optional keepOrDelete
as logical ) as table =>
```

Parameter eksplisit menentukan input fungsi dan tipe data yang dibutuhkan. Oleh karena itu, menggunakan parameter eksplisit dalam fungsi kustom merupakan praktik terbaik yang berfungsi sebagai garis pertahanan pertama dalam memastikan fungsi hanya menerima argumen dari tipe yang ditentukan. Namun, untuk parameter *colPositions*, pengetikan eksplisit sengaja dihilangkan. Meskipun menyertakan dokumentasi parameter merupakan pendekatan umum untuk memberi tahu pengguna tentang tipe argumen yang diharapkan, di sini, logika kesalahan kustom telah diterapkan untuk memberikan informasi khusus kepada pengguna saat argumen yang dilewatkan bukan dari tipe yang diharapkan, khususnya, sebuah list. Dengan menyesuaikan pesan kesalahan, kita dapat menawarkan umpan balik yang lebih berarti kepada pengguna, membimbing mereka tentang cara memenuhi persyaratan nilai argumen:

```
4       // Custom error, provide more specific information to the
        user.
5       l = if not Value.Is( colPositions, type nullable list )
            then error "Pass a list with column positions,
6           starting number is 1.
7           For example: { 1, 3, 5 } will keep or delete columns
1, 3 and 5 from the input table."




8           else colPositions??{},,,
```

Penanganan parameter opsional dalam fungsi kustom sangat penting untuk mencegah nilai *null* menjadi akar penyebab kesalahan. Ketika parameter opsional dihilangkan, nilai *default* - nya adalah *null*. Untuk melindungi dari potensi masalah, *Coalesce* digunakan pada baris 8 untuk memastikan bahwa ketika parameter *colPositions* adalah null, nilai tersebut akan digantikan oleh list kosong {}. Dengan cara ini, variabel l selalu dijamin memiliki nilai yang valid ketika diteruskan ke *List.Transform*, baik list yang disediakan pengguna atau list kosong jika tidak ada list tersebut yang disediakan.

Selain itu, *Coalesce* digunakan untuk menetapkan nilai *default* untuk parameter *keepOrDelete* opsional. Ini memastikan bahwa jika pengguna tidak meneruskan argumen untuk *keepOrDelete*, nilai *default* nya adalah Keep. Dengan demikian, fungsi tersebut dirancang agar lebih mudah digunakan dan dapat diprediksi, karena menghilangkan kebutuhan pengguna untuk secara eksplisit memberikan nilai jika mereka ingin menyimpan kolom yang ditentukan:

```
9       // Coalesce set the default to selecting, not deleting
        columns.
10       inclOrExclCols = keepOrDelete ?? true,
```

Untuk menangani nama kolom tabel secara efisien, kita akan menetapkan semua nama kolom ke variabel. *Power Query* M menggunakan indeks berbasis nol, yang berarti kita harus mengonversi input pengguna karena mereka diminta untuk memberikan indeks berbasis satu. Menentukan posisi kolom mulai dari satu, bukan nol, membuatnya lebih intuitif bagi pengguna.

Sebelum melanjutkan pengurangan, kita perlu memastikan bahwa tipe data item list dalam list *colPositions* akan sesuai dengan angka. Untuk mencapai hal ini, kita akan menyertakan penanganan kesalahan untuk kasus-kasus ketika suatu nilai tidak dapat dikonversi ke tipe angka. Dengan menggunakan konstruksi *try-otherwise*, kita dapat mencoba mengonversi setiap item list ke angka. Jika berhasil, kita melakukan pengurangan dengan satu untuk menyesuaikan dengan indeks berbasis nol. Jika ada nilai dalam list yang tidak dapat dikonversi ke angka, klausa otherwise berlaku yang mana kita memberikan nilai null; demi kesederhanaan, input yang salah diabaikan:

```
11     allCols = Table.ColumnNames( myTable ),
12       // try-otherwise protects the evaluation of the second
argument, returning a default null value for non-conforming value
types.
13       baseZero = List.Transform( l, each try Number.From(_)-1
         otherwise null
),
```

Untuk membuat list nama kolom aktual dari input pengguna dan sekaligus memastikan akses item opsional tidak menyebabkan kesalahan, kita akan menumpuk tiga fungsi. Pertama-tama, membangun logika dari dalam ke luar, memfilter list *baseZero*, dan hanya menyimpan nilai yang lebih besar dari atau sama dengan nol. Tindakan pencegahan ini menjamin bahwa akses item opsional tidak akan pernah menimbulkan kesalahan yang disebabkan oleh indeks negatif. Selanjutnya, di dalam *List.Transform*, terapkan akses item opsional untuk mendapatkan nama kolom yang sesuai dari list *allCols*. Terakhir, menghapus nilai null. Dengan menghapus null apa pun dari list, kita dapat menentukan jumlah nama kolom valid yang tersisa:

```
14     colNames = List.RemoveNulls( List.Transform( List.Select(
baseZero, each
_ >=0 ),      each allCols{_}? ) ),
```

Dengan langkah-langkah ini, kita dapat dengan yakin menghasilkan nilai keluaran. Untuk mencapainya, kita akan:

a. Memanfaatkan pernyataan *if-then-else* ganda untuk mengendalikan aliran fungsi. Kondisi if terluar memeriksa apakah ada nama kolom yang valid yang tersisa dalam list *colNames*.
b. Jika tidak ada nama kolom yang valid, fungsi mengembalikan tabel masukan apa adanya, tanpa modifikasi apa pun. Sebaliknya, jika ada nama kolom yang valid, pernyataan if-then-else kedua, berdasarkan nilai inclOrExclCols, menentukan apakah kolom yang ditentukan harus disimpan atau dihapus dari tabel masukan.

Dengan menggunakan struktur kontrol ini, kita memastikan bahwa fungsi tersebut dapat menangani berbagai skenario secara efektif. Jika pengguna memberikan posisi kolom yang tidak valid atau tidak menentukan kolom apa pun untuk disertakan atau dikecualikan, fungsi tersebut tetap memberikan hasil yang konsisten, mencegah kesalahan yang tidak terduga dan menawarkan pengalaman pengguna yang lancar:

```
newTbl = if List.Count( colNames ) >0
    then Table.SelectColumns( myTable, if
      inclOrExclCols
      then colNames
      else List.RemoveMatchingItems( allCols, colNames )
  ) else myTable
```

Berikut skrip kode M lengkap termasuk dokumentasinya.

```
let
    fxSelectColumns = ( myTable as table, optional colPositions,
optional keepOrDelete as logical ) as table =>
    let
     // Custom error logic to provide more specific information to
     the user.
     cols = if not Value.Is( colPositions, type nullable list )
     then error "Pass a list with column positions, starting number
     is 1.
```

```
      For example: { 1, 3, 5 } will keep or delete columns 1, 3 and 5
      from the input table."
              else colPositions ?? {},
      /* Coalesce to set the default value to selecting columns, not
deleting them. */
      inclOrExclCols = keepOrDelete ?? true, allCols =
      Table.ColumnNames( myTable ),
      /* try-otherwise protects the evaluation of the second
argument, returning a null for non-conforming values. */
      baseZero = List.Transform( cols, each try Number.From(_)-1
      otherwise
null ),
          colNames = List.RemoveNulls( List.Transform( List.Select(
baseZero, each _ >=0 ), each allCols{_}? )),
      /* a double if-then-else statement controls the flow for the
      table
output expression */
          newTbl = if List.Count( colNames ) >0 then
              Table.SelectColumns( myTable,
                  if inclOrExclCols then colNames
                  else List.RemoveMatchingItems( allCols, colNames )
          ) else myTable
    in
          newTbl, fnDocumentation = [
              Documentation.Name = " Select Columns by position ",
              Documentation.Description = " Selects or Removes
              columns from the input table.
              'colPositions' takes a list with one-based column
              position numbers. 'keepOrDelete' takes a boolean,
              'true'= keep and 'false' = remove columns. Returns a
              table with fewer columns or as is when no valid values
              have been passed. ",
              Documentation.Author = " Melissa de Korte ",
              Documentation.Version = " 1.0 "
          ]
in
    Value.ReplaceType( fxSelectColumns, Value.ReplaceMetadata(
              Value.Type( fxSelectColumns ),
              fnDocumentation
          )
    )
```

Contoh ini menunjukkan berbagai teknik pencegahan dan penanganan kesalahan, yang mengilustrasikan perannya dalam tugas transformasi data. Dengan menggabungkan pilihan pengembang yang jelas, kesalahan khusus, penggabungan, akses item opsional, pernyataan *try-otherwise*, dan *if-then-else*, kita telah menyusun metode yang kuat dan mudah digunakan untuk memilih kolom dalam tabel. Masing-masing teknik ini memiliki tujuan tertentu, yang berkontribusi pada ketahanan dan keandalan fungsi secara keseluruhan.

3. Melaporkan Kesalahan Tingkat Sel

Memastikan kualitas dan validitas data merupakan langkah penting dalam proses penyiapan data; menangani kesalahan tingkat sel sangat penting untuk menjaga integritas kumpulan data Anda. Meskipun *Table.RemoveRowsWithErrors* mungkin tampak seperti opsi yang mudah untuk menghapus baris dengan kesalahan, hal itu bukan tanpa konsekuensi, terutama jika Anda tidak yakin baris mana yang telah dihapus dan alasan di balik penghapusannya. Untuk menjaga transparansi data, memberdayakan Anda untuk membuat keputusan yang tepat, dan bertindak ketika masalah muncul, mari kita lihat cara membuat tabel pelaporan kesalahan untuk kueri Anda.

4. Membangun Solusi Khusus

Seperti sebelumnya, kita akan memandu Anda melalui proses dan menguraikan setiap bagian untuk Anda. Setelah kita membahas setiap bagian, kita akan menyajikan seluruh kode sehingga Anda dapat melihat bagaimana semuanya menyatu dengan mulus.

Sebagian besar tabel berisi satu atau beberapa kolom kunci, yang berfungsi sebagai pengenal unik untuk entri tertentu dalam tabel. Oleh karena itu, penting untuk memberi pengguna opsi untuk mengidentifikasi kolom kunci dalam data mereka dan menyimpannya sebagai pengenal baris dalam tabel keluaran. Selain itu, kita dapat mengizinkan pengguna untuk membatasi deteksi kesalahan pada pilihan kolom tertentu, memastikan pendekatan yang lebih terfokus dan efisien.

Dengan mempertimbangkan hal ini, kita telah mengidentifikasi tiga parameter input untuk fungsi kustom kita. Parameter ini mencakup tabel input tempat operasi akan dilakukan, list opsional yang berisi nama kolom kunci, dan list opsional yang menentukan nama kolom yang akan dipindai untuk menemukan kesalahan. Dengan menyediakan parameter ini, pengguna memiliki kontrol terperinci atas proses deteksi kesalahan.

Untuk memastikan bahwa hanya argumen yang valid yang diterima, parameter eksplisit telah dideklarasikan. Menentukan secara eksplisit tipe data yang diharapkan untuk setiap parameter menambahkan lapisan perlindungan, mencegah fungsi menerima tipe data yang tidak kompatibel sebagai input:

```
2 (myTable as table, optional keyCols as list, optional scanCols as
list ) as record =>
```

Dengan mengizinkan argumen *keyCols* menjadi opsional, pengguna akan memperoleh fleksibilitas, tetapi juga memerlukan penanganan yang tepat saat nilai tidak diberikan karena saat *keyCols* dihilangkan, nilainya adalah null. Untuk menghindari kesalahan ketidakcocokan tipe saat diberikan sebagai argumen ke fungsi lain yang mengharapkan tipe list, operator *Coalesce* dimanfaatkan untuk mengganti null dengan list kosong.

Dengan mempertimbangkan kemungkinan kesalahan ejaan pada *keyCols* yang diberikan pengguna, kita dapat memvalidasinya terhadap nama kolom di tabel input (*colNames*). Dengan demikian, kita dapat mengidentifikasi nama kolom kunci yang valid dan menyimpannya dalam variabel baru yang disebut *validKeys*. Langkah validasi ini tidak hanya membantu mencegah potensi kesalahan, tetapi juga memungkinkan kita untuk memberikan list dengan nama kolom kunci yang salah eja atau tidak ada, kembali ke pengguna dalam output. Terakhir, variabel *hasKeys* mengembalikan true atau false tergantung pada apakah kolom kunci yang valid ada:

```
colNames = Table.ColumnNames( myTable ),
validKeys = List.Intersect( { colNames, keyCols ?? {} } ),
hasKeys = List.Count( validKeys ) >0,
```

Metode *Coalesce* dan validasi yang sama digunakan untuk argumen *scanCols* opsional yang disediakan pengguna. Jika *scanCols* dihilangkan, nilai *default* nya adalah null, dan *Coalesce* mengganti null dengan list kosong, memastikan jenis list tersedia untuk pemrosesan lebih lanjut.

Untuk mengoptimalkan kinerja fungsi, kita mengambil langkah proaktif dalam persiapan data. Kolom apa pun yang tidak diperlukan segera dihapus dari tabel input dan disimpan dalam variabel *newTable*.

Dengan menghilangkan kolom yang tidak diperlukan lebih awal, kita mengurangi beban komputasi, sehingga menghasilkan eksekusi yang lebih efisien:

```
selCols = List.Intersect( { colNames, scanCols ?? {} } ),
newTable = Table.SelectColumns( myTable, if List.Count( selCols
) >0 then validKeys & selCols else colNames ),
```

Selanjutnya, kita membuat pilihan pengembangan: kita menetapkan list dengan nama kolom yang dicadangkan. Dengan kata lain, nama-nama dalam list *reservedNames* tidak boleh ada dalam tabel input. Setiap pelanggaran yang terdeteksi akan disimpan dalam list *nameViolations*. Ini memungkinkan kita untuk memberikan umpan balik kepada pengguna:

Mari kita dalami inti solusi ini dan buat komponen penting dalam membangun tabel pelaporan kesalahan, dengan memanfaatkan beberapa pernyataan *if-then-else*:

a. Pertama, kita periksa apakah ada pelanggaran nama kolom. Jika tidak, kita lanjutkan dengan ekspresi let bersarang untuk menangani dua skenario berbeda berdasarkan keberadaan kunci yang valid.
b. Jika ada kunci yang valid, kita simpan semua kesalahan yang berisi baris dari tabel input yang dimodifikasi.
c. Jika tidak ada kunci yang valid, kita akan menambahkan kolom indeks baris berbasis nol, yang disebut *rowIndex*, ke tabel input yang dimodifikasi sebelum memanggil fungsi *Table.SelectRowsWithErrors*.

Ini terlihat seperti berikut:

```
runLogic = if List.IsEmpty( nameViolations )
  then let
  t = if hasKeys = true
    then Table.SelectRowsWithErrors( newTable )
     else Table.SelectRowsWithErrors( Table.AddIndexColumn(
     newTable, "rowIndex", 0, 1, Int64.Type)),
```

Variabel t sekarang menyimpan tabel yang berisi baris dengan satu atau beberapa kesalahan di selnya. Namun, fokus kita hanya pada kesalahan itu sendiri. Kita akan menambahkan kolom baru yang disebut *newCol*, menggunakan *Record.ToTable(_)*, untuk menghasilkan tabel dua kolom di setiap baris. Kolom pertama, yang disebut Name, terdiri dari nama-nama bidang, sedangkan kolom kedua, yang disebut *Value*, berisi nilai-nilai bidang yang sesuai. Sekarang kita dapat memanggil fungsi *Table.SelectRowsWithErrors* lagi, kali ini menargetkan kolom Value untuk hanya mengambil kesalahan.

Tujuan akhir kita adalah laporan kesalahan, yang menunjukkan alasan kesalahan, pesan, dan detailnya. Untuk mencapainya, kita membuat kolom tambahan di tabel bersarang, memanfaatkan konstruksi *try-catch* untuk mengekstrak informasi yang relevan dari nilai kesalahan, *try [Value] catch (e) => e* dan menyimpan informasi itu ke kolom baru bernama *Errors*.

Sekarang setelah kita berhasil mengekstrak data yang diperlukan, tidak perlu lagi menyimpan nilai kesalahan aktual yang tersimpan dalam kolom Nilai pada tabel bersarang. Anda dapat menggunakan proyeksi yang diperlukan untuk membuat tabel dengan kolom yang lebih sedikit: *[[Name], [Errors]]*.

Terakhir, kita juga dapat mengoptimalkan tabel luar dengan menghilangkan semua kolom kecuali *newCol*, dan kolom kunci atau kolom *rowIndex*:

```
getErrors = Table.SelectColumns( Table.AddColumn( t, "newCol", each
Table.AddColumn( Table.SelectRowsWithErrors( Record.ToTable(_),
{"Value"}), "Errors",
each try [Value] catch (e)=> e )[[Name], [Errors]]),
if hasKeys then validKeys & {"newCol"} else {"rowIndex",
"newCol"} ),
```

Setelah mengoptimalkan kolom-kolom di tabel kita, sekarang kita lanjutkan untuk memperluas tabel bersarang di *newCol*, yang berisi kolom *Name* dan *Error*. Operasi ekstraksi ini akan menghasilkan pelebaran dan perluasan tabel, yang memungkinkan kita untuk mengekspos nilai *Record* di kolom *Error* yang baru ditambahkan.

Mengekstrak kolom Alasan, Pesan, dan *Detail* dari nilai *Record* di kolom *Error* akan memperluas tabel lebih jauh, memberikan pengguna wawasan terperinci tentang setiap kesalahan yang ditemukan:

```
20      errorTable = Table.ExpandRecordColumn(
21        Table.ExpandTableColumn( getErrors, "newCol", {"Name",
22        "Errors"} ), "Errors", {"Reason", "Message", "Detail"} )
23    in errorTable
```

Kita perlu melengkapi cabang terakhir dari pernyataan *if-then-else* pertama, artinya jika kita menemukan adanya pelanggaran nama kolom, kita akan memunculkan kesalahan khusus. Pesan kesalahan meminta pengguna untuk mengganti nama kolom yang tercantum dan mencoba lagi:

```
24    else error
25    "Please rename these columns in your table first: " &
      Text.Combine(

nameViolations, ", " )
```

Terakhir, fungsi ini mengeluarkan nilai *Record* yang berisi empat bidang: angka untuk *Error count*, tabel pelaporan kesalahan Data, list dengan *Invalid key columns*, dan list dengan *Invalid scan columns*:

```
28    Error count = Table.RowCount( runLogic ),
29    Data = runLogic,
30    Invalid key columns = List.Difference( keyCols ?? {},
      validKeys),
31    Invalid scan columns = List.Difference( scanewCols ??
```

```
      {}, selCols)
32   ]
```

Berikut skrip kode M lengkap termasuk dokumentasinya:

```
let
  fxErrorReport = (myTable as table, optional keyCols as list, optional
scanCols as list ) as record =>
  let
    colNames = Table.ColumnNames( myTable ),
    validKeys = List.Intersect( { colNames, keyCols ?? {} } ),
    hasKeys = List.Count( validKeys ) >0,
    selCols = List.Intersect( { colNames, scanCols ?? {} } ),
    nTable = Table.SelectColumns( myTable, if List.Count( selCols ) >0 then
validKeys & selCols else colNames ),
    reservedNames = { "Name", "Errors", "Reason", "Message", "Detail",
"rowIndex", "nCol" },
    nameViolations = List.Intersect( { colNames, reservedNames } ),
    runLogic = if List.IsEmpty( nameViolations )
      then let
        t = if hasKeys = true
          then Table.SelectRowsWithErrors( nTable )
          else Table.SelectRowsWithErrors( Table.AddIndexColumn( nTable,
"rowIndex", 0, 1, Int64.Type)),
        getErrors = Table.SelectColumns( Table.AddColumn( t, "nCol", each
          Table.AddColumn( Table.SelectRowsWithErrors( Record.ToTable(_),
{"Value"}), "Errors",
            each try [Value] catch (e)=> e )[[Name], [Errors]] ),
            if hasKeys then validKeys & {"nCol"} else {"rowIndex", "nCol"} ),
        errorTable = Table.ExpandRecordColumn(
          Table.ExpandTableColumn( getErrors, "nCol", {"Name", "Errors"} ),
          "Errors", {"Reason", "Message", "Detail"} )
      in errorTable
      else error
        "Please rename these columns in your table first: " & Text.Combine(
nameViolations, ", " )
    in
      [
        Error count = Table.RowCount( runLogic ),
        Data = runLogic,
        Invalid key columns = List.Difference( keyCols ?? {}, validKeys ),
        Invalid scan columns = List.Difference( scanCols ?? {}, selCols )
      ],
      fxDocumentation = [
        Documentation.Name = " Create an Error Report ",
```

```
        Documentation.Description = " Returns a record containing 4 fields.
        'Error Count' shows the number of errors found in the input table.
        'Data' contains the error reporting table.
        'Invalid key columns' returns a list containing specified invalid key
columns if any.
        'Invalid scan columns' returns a list containing specified invalid scan
columns if any. ",
        Documentation.Author = " Melissa de Korte ",
        Documentation.Version = " 1.0 "
  ]
in
  Value.ReplaceType( fxErrorReport, Value.ReplaceMetadata(
      Value.Type( fxErrorReport ),
      fxDocumentation
    )
  )
```

Penanganan kesalahan merupakan aspek multifaset dari desain kueri yang tangguh yang memerlukan pemahaman menyeluruh tentang berbagai jenis kesalahan dan metode penahanan, deteksi, pencegahan, dan penyelesaiannya. Dengan menerapkan langkah-langkah pencegahan, Anda dapat mengembangkan serangkaian keterampilan penanganan kesalahan yang tidak hanya mengatasi kesalahan saat terjadi tetapi juga meminimalkan kemunculannya. Membuat pilihan yang bijaksana atau bahkan menerapkan penanganan kesalahan yang dinamis akan memungkinkan Anda untuk membangun kueri yang lebih andal. Saat Anda terus menyempurnakan strategi penanganan kesalahan, Anda akan menemukan bahwa ada peluang yang tak terbatas untuk pendekatan yang kreatif dan pemecahan masalah dalam bahasa *Power Query* M.

K. Ringkasan

Bab ini dimulai dengan membuat fondasi: pemahaman bersama tentang apa itu kesalahan, dan bagaimana kesalahan dapat dimunculkan, diatasi, dan dideteksi. Bab ini mengeksplorasi berbagai aspek penanganan kesalahan, yang menyoroti pentingnya mengadopsi pendekatan holistik. Selain bereaksi terhadap kesalahan saja, penting untuk secara proaktif memasukkan tindakan pencegahan ke dalam desain kueri Anda. Dengan menggunakan kombinasi teknik ini secara cermat, Anda dapat membuat kueri yang

lebih tangguh, sehingga mengurangi risiko kesalahan dan meningkatkan kualitas data secara keseluruhan.

Di bab berikutnya, kita akan mempelajari tentang fungsi dan operator utama yang memungkinkan iterasi dan rekursi di *Power Query* M.

L. Daftar Pustaka

Neco, R. P., and Forcada, M. L. (1997), Asynchronous translations with recurrent neural nets. Neural Networks, 1997., International Conference on (Vol. 4, pp. 2535–2540). IEEE: https://ieeexplore.ieee.org/document/614693.

Kalchbrenner, N., and Blunsom, P. (2013), Recurrent Continuous Translation Models. EMNLP (Vol. 3, No. 39, p. 413): https://aclanthology.org/D13-1176/.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. (2014), Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, Doha, Qatar. Association for Computational Linguistics: https://aclanthology.org/D14-1179/.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. (2014), Sequence to sequence learning with neural networks. Proceedings of the 27th International Conference on Neural Information Processing Systems – Volume 2: https://dl.acm.org/doi/10.5555/2969033.2969173.

Rumelhart, D., Hinton, G., and Williams, R (1986). Learning representations by back-propagating errors. Nature 323, 533–536: https://doi.org/10.1038/323533a0.

Schuster, M., and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), 2673–2681: https://doi.org/10.1109/78.650093.

Sepp Hochreiter (1991) Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich: https://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf.

Y. Bengio, P. Frasconi, and P. Simard (1993), The problem of learning long-term dependencies in recurrent networks. IEEE

International Conference on Neural Networks, pp. 1183-1188 vol.3: 10.1109/ICNN.1993.298725.
Y. Bengio, P. Simard, and P. Frasconi (1994) Learning long-term dependencies with gradient descent is difficult in IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, March 1994: 10.1109/72.279181.
Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735–1780: https://doi.org/10.1162/neco.1997.9.8.1735.
Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. EMNLP: https://www.aclweb.org/anthology/D14-1179.pdf.
Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybernetics 36, 193–202 (1980): https://doi.org/10.1007/BF00344251.
Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. 1990. Handwritten digit recognition with a back-propagation network. Advances in neural information processing systems 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 396–404: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

M. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan apa yang dimaksud dengan kesalahan (error) dalam konteks *Power Query* M dan bagaimana kesalahan tersebut dapat terjadi dalam suatu kueri. Berikan contoh kasus sederhana.	10
2.	Uraikan konsep "penahanan kesalahan" (error trapping) dalam *Power Query*. Fungsi atau ekspresi apa yang biasa digunakan untuk menahan kesalahan? Berikan ilustrasi penggunaannya.	10

3.	Bagaimana cara mendeteksi kesalahan secara manual di Editor *Power Query*? Jelaskan peran panel 'Applied Steps' dalam proses ini.	10
4.	Diskusikan bagaimana Anda dapat secara sengaja menimbulkan kesalahan dalam *Power Query* M. Kapan skenario ini mungkin diperlukan?	10
5.	Jelaskan fungsi try ... otherwise dalam *Power Query* M. Bagaimana fungsi ini membantu dalam menangani error saat transformasi data berlangsung?	10
6.	Apa saja strategi *debugging* yang efektif saat menghadapi error dalam *Power Query*? Berikan contoh nyata penerapan salah satu strategi tersebut.	10
7.	Sebutkan dan jelaskan minimal tiga jenis kesalahan umum yang sering terjadi dalam *Power Query*. Bagaimana pendekatan Anda untuk menghindarinya?	10
8.	Dalam konteks *Power Query*, mengapa penting untuk memahami informasi detail dari kesalahan (error *Record*)? Jelaskan cara mengakses dan menggunakan informasi ini.	10
9.	Apa dampak dari mengabaikan kesalahan kecil dalam dataset saat membangun model *Power* BI? Berikan analisis risiko dan solusinya.	10
10.	Analisis bagaimana kombinasi teknik penahanan kesalahan, deteksi dini, dan strategi *debugging* dapat meningkatkan ketahanan dan kualitas kueri *Power Query* Anda.	10

GLOSARIUM

AAS	**Azure Analysis Services:** Layanan Platform-as-a-Service (PaaS) yang disediakan oleh Microsoft Azure untuk membuat solusi Business Intelligence (BI) tingkat perusahaan berbasis model data analitik
ACE	**Access Database Engine:** Komponen inti Microsoft Access yang bertanggung jawab untuk menyimpan dan mengambil data dalam format basis data Access (.accdb dan .mdb).
AD	**Active Directory:** Layanan direktori yang dikembangkan oleh Microsoft untuk mengelola dan mengatur sumber daya dalam jaringan komputer berbasis Windows Server.
ADO.NET	**ActiveX Data Objects .NET:** Seperangkat kelas dalam .NET Framework yang menyediakan akses terpadu ke berbagai sumber data.
AI	**Artificial Intelligence:** Bidang ilmu komputer yang berfokus pada pengembangan sistem komputer yang dapat melakukan tugas-tugas yang biasanya membutuhkan kecerdasan manusia.
API	**Application Programming Interface:** Sekumpulan aturan, protokol, dan alat yang memungkinkan berbagai aplikasi perangkat lunak untuk saling berkomunikasi dan bertukar data.
BI	**Business Intelligence:** Proses pengumpulan, pengorganisasian, analisis, interpretasi, dan penyajian data bisnis untuk membantu para pengambil keputusan membuat keputusan yang lebih baik dan lebih terinformasi.
COM	**Component Object Model:** Kerangka kerja biner yang dikembangkan oleh Microsoft yang memungkinkan aplikasi untuk berinteraksi dengan objek perangkat lunak tanpa perlu mengetahui detail implementasi internal objek tersebut.
CRM	**Customer Relationship Management:** Strategi bisnis yang berfokus pada membangun dan memelihara hubungan yang kuat dan positif dengan pelanggan.
CSS	**Cascading Style Sheets:** Bahasa stylesheet yang digunakan untuk menggambarkan tampilan (presentasi) dokumen yang ditulis dengan bahasa markup seperti HTML atau XML.
CSV	**Comma-Separated Value:** Format file teks sederhana yang digunakan untuk menyimpan data tabular (seperti spreadsheet atau database) di mana setiap baris mewakili satu catatan dan setiap kolom dalam catatan tersebut dipisahkan oleh tanda koma.
DBMS	**Database Management System:** Perangkat lunak yang memungkinkan pengguna untuk membuat, memelihara, dan berinteraksi dengan database secara efisien dan aman.
DOM	**Document Object Model:** Representasi terstruktur (berbentuk pohon) dari dokumen HTML atau XML yang memungkinkan program (seperti JavaScript) untuk mengakses dan memanipulasi

	konten, struktur, dan gaya dokumen tersebut.
DSN	**Data Source Name:** *String* koneksi yang berisi informasi yang dibutuhkan oleh aplikasi untuk terhubung ke sumber data tertentu.
ECMA	**European Computer Manufacturers Association:** Organisasi standar internasional untuk sistem informasi dan komunikasi.
ELT	**Extract-Load-Transform:** Proses integrasi data yang terdiri dari tiga tahap utama: Ekstraksi (Extract), Pemuatan (Load), dan Transformasi (Transform).
ERP	**Enterprise Resource Management:** Sebuah sistem yang mengintegrasikan proses bisnis inti perusahaan
EST	**Eastern Standard Time:** Zona waktu yang digunakan di sebagian Amerika Utara dan beberapa wilayah Karibia.
ETL	**Extract, Transform, And Load:** Proses integrasi data yang umum digunakan untuk memindahkan data dari berbagai sumber ke sistem target, seperti data warehouse, untuk tujuan analisis dan pelaporan.
GUI	**Graphical User Interface:** Jenis antarmuka pengguna yang memungkinkan pengguna berinteraksi dengan perangkat elektronik melalui elemen visual seperti ikon, menu, jendela, tombol, dan pointer (biasanya dikendalikan oleh mouse, trackpad, atau layar sentuh).
HDFS	**Hadoop Distributed File System:** Sistem file terdistribusi yang dirancang untuk menyimpan dan mengelola dataset yang sangat besar yang berjalan di atas perangkat keras komoditas (perangkat keras standar yang tidak terlalu mahal).
HL7	**Health Level 7:** Seperangkat standar internasional yang digunakan untuk pertukaran, integrasi, berbagi, dan pengambilan informasi kesehatan elektronik antara berbagai sistem perangkat lunak yang digunakan oleh penyedia layanan kesehatan.
HTML	**HyperText Markup Language:** Bahasa markup standar untuk membuat halaman web.
IT	**Information Technology:** Penerapan komputer dan sistem telekomunikasi untuk menyimpan, mengambil, mengirim, dan memanipulasi data atau informasi.
JDN	**Julian Day Number:** Jumlah hari yang telah berlalu sejak epoch Julian tengah siang di Greenwich pada tanggal 1 Januari 4713 SM (kalender Julian proleptik).
JSON	**JavaScript Object Notation:** Format data ringan yang digunakan untuk mentransmisikan data antara server dan aplikasi web, serta untuk menyimpan data terstruktur.
ODBC	**Open Database Connectivity:** Antarmuka pemrograman aplikasi (API) standar yang memungkinkan aplikasi untuk mengakses berbagai sistem manajemen basis data (DBMS) secara independen dari DBMS yang mendasarinya.
OLAP	**Online Analytics Processing:** Pendekatan untuk menganalisis data multidimensi dalam jumlah besar dengan cepat dari berbagai perspektif.

OLE DB	**Object Linking and Embedding Database:** Sekumpulan antarmuka yang dikembangkan oleh Microsoft yang menyediakan akses terpadu ke berbagai sumber data.
OLTP	**Online Transactional Processing:** Jenis pemrosesan data yang berfokus pada eksekusi dan pencatatan sejumlah besar transaksi kecil dan bersamaan secara real-time.
OPC	**Open Packaging Conventions:** Sebuah teknologi format file kontainer yang awalnya dibuat oleh Microsoft untuk menyimpan kombinasi file XML dan non-XML yang bersama-sama membentuk satu entitas logis.
PBRS	***Power* BI Report Server:** Solusi pelaporan on-premises (di tempat) yang memungkinkan Anda untuk membuat, menerbitkan, dan mengelola laporan serta KPI (Key Performance Indicators) di dalam infrastruktur organisasi Anda sendiri.
PDF	**Portable Document Format:** Format file yang dikembangkan oleh Adobe untuk menyajikan dokumen, termasuk teks, gambar, hyperlink, font yang disematkan, dan lainnya, dengan cara yang independen dari perangkat lunak aplikasi, perangkat keras, dan sistem operasi.
SSAS	**SQL Server Analysis Services:** Komponen dalam Microsoft SQL Server yang menyediakan kemampuan pemrosesan analitik daring (OLAP) dan data mining.
SSIS	**SQL Server Integration Services:** Platform untuk membangun solusi integrasi data dan transformasi data berkinerja tinggi.
UI	**User Interface:** Bagian dari sistem (perangkat lunak atau perangkat keras) yang memungkinkan manusia berinteraksi dengannya.
URI	**Uniform Resource Identifier:** *String* karakter yang mengidentifikasi sumber daya fisik atau abstrak.
UTC	**Coordinated Universal Time:** Standar waktu utama dunia yang digunakan sebagai dasar untuk semua zona waktu sipil di seluruh dunia.
VoIP	**Voice over Internet Protocol:** Sekelompok teknologi yang memungkinkan Anda melakukan panggilan suara menggunakan koneksi internet broadband, bukan saluran telepon analog biasa.
WMS	**Warehouse Management System:** Sistem perangkat lunak yang dirancang untuk mengelola dan mengoptimalkan operasi gudang.
XML	**eXtensible Markup Language:** Bahasa markup yang dirancang untuk membawa data dan mendeskripsikan struktur data.

INDEKS

L

M

O

P

R

T

V

W

TENTANG PENULIS

Dr. Phil. Dony Novaliendry, S.Kom., M.Kom., Lahir dan besar di Padang tanggal 4 November 1975. Merupakan anak dari Prof. Dr. H. Aljufri B. Syarif, M.Sc (Alm) dengan Endang Ratna Sulistri. Anak ke-4 dari 4 orang bersaudara. Menamatkan S1 di Universitas Gunadarma Jurusan Sistem Informasi dan di lanjutkan S2 di Universitas Gadjah Mada Jurusan Ilmu Komputer dan melanjutkan S3 di National Kaohsiung University of Science and Technology (NKUST) di Taiwan dengan bidang Bio-Informatics. Saat ini tertarik untuk mengembangkan diri di bidang bio-informatics, bio-medics, Artificial Intelligence, Decision Support System, Multimedia, Big Data dan Data Mining. Ini adalah buku yang ke enam yang diterbitkan. Semoga ditahun mendatang masih bisa terus berkarya menciptakan beberapa buku lagi.

Quote: ***Life Must Go On.***

Saran, kritik dan kerjasama: dony.novaliendry@ft.unp.ac.id

RINGKASAN ISI BUKU

Buku ini merupakan kelanjutan dari seri Power Query yang mendalami kemampuan lanjutan dalam mengelola dan mentransformasi data menggunakan bahasa M. Disusun secara sistematis dan praktis, buku ini membahas cara membuat parameter dan custom function, serta penerapannya dalam pembuatan solusi dinamis.

Anda juga akan mempelajari teknik lengkap dalam menangani tanggal, waktu, durasi, serta manipulasi data menggunakan fungsi pembanding, pengganti, penggabung, dan pemisah—semuanya disertai contoh kasus yang aplikatif.Bagian penting lainnya adalah pembahasan mendalam tentang penanganan kesalahan (error handling) dan debugging, yang akan membantu Anda membangun transformasi data yang kokoh dan bebas error.

Ditujukan bagi pengguna Excel dan Power BI tingkat menengah hingga mahir, buku ini memperkuat pemahaman teknis dengan tes formatif, daftar fungsi, dan glosarium lengkap—menjadikannya referensi ideal untuk praktisi data yang ingin menguasai Power Query secara profesional.