PANDUAN DEFINITIF UNTUK POWER QUERY (M)

Menguasai Transformasi Data Kompleks dengan Power Query

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

Buku "Panduan Definitif untuk Power Query (M) – Jilid 4" melanjutkan eksplorasi mendalam tentang kemampuan bahasa M dalam mengelola dan mengoptimalkan transformasi data. Jilid ini membahas topik-topik lanjutan seperti iterasi dan rekursi, penerapan pola data yang bermasalah, strategi optimasi kinerja termasuk query folding dan firewall, hingga teknik pengembangan ekstensi Power Query untuk membangun konektor khusus.

Disusun secara sistematis dengan contoh kasus, tes formatif, glosarium, dan lampiran, buku ini dirancang untuk membantu pembaca memahami konsep lanjutan sekaligus praktik implementasi di dunia nyata. Kehadiran jilid keempat ini menjadikan seri buku Power Query (M) semakin lengkap sebagai rujukan bagi mahasiswa, dosen, peneliti, dan praktisi data yang ingin menguasai transformasi data secara komprehensif dan profesional.

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

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

PANDUAN DEFINITIF UNTUK POWER QUERY (M)
Menguasai Transformasi Data Kompleks dengan Power Query

JILID 4

Penerbitan & Percetakan
UNP PRESS

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

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 4

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 4

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

DUMMY
Penerbitan & Percetakan
UNP PRESS

UNP PRESS

2025

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 4

editor, Tim editor UNP Press
Penerbit UNP Press, Padang, 2025
1 (satu) jilid; 17.6 x 25 cm (B5)
Jumlah Halaman xiii + 280 Halaman Buku

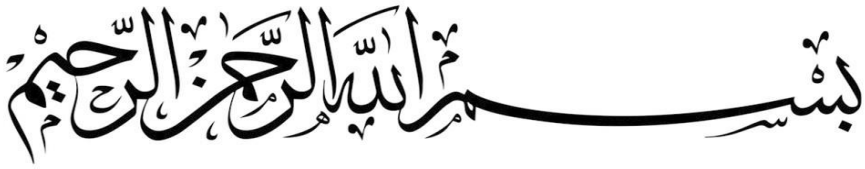
ISBN:

DUMMY
Penerbitan & Percetakan
UNP PRESS

PANDUAN DEFINITIF UNTUK POWER QUERY (M) JILID 4

Penyusun: Dr. Phil. Dony Novaliendry, S. Kom., M. Kom
Editor Substansi: Fadhillah Majid Saragih, S.Pd., M.Pd.T
Editor Bahasa: Prof. Dr. Harris Effendi Thahar, M.Pd.
Desain Sampul & Layout: Mukhlis Zaki Insani

KATA PENGANTAR



Puji syukur kehadirat Allah SWT atas berkat limpahan rahmat dan karunia-Nya sehingga Buku ajar Panduan Definitif untuk Power Query (M) dapat di buat. Buku ajar 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, Juni 2025

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DUMMY
Penerbitan & Percetakan
UNP PRESS

DAFTAR TABEL

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

Iterasi dan Rekursi

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Iterasi:
  - *List.Transform*
  - *List.Accumulate*
  - *List.Generate*
- Rekursi:
  - Operator cakupan @

A. Pendahuluan

Iterasi dan rekursi adalah konsep dasar dalam pemrograman yang memungkinkan eksekusi kode yang berulang. Dalam konteks Power Query M, teknik ini sangat meningkatkan kemampuan transformasi dan manipulasi data Anda. Apakah Anda ingin menerapkan fungsi di seluruh list nilai, mengumpulkan hasil, atau bahkan merujuk ke langkah sebelumnya dalam urutan, memahami iterasi dan rekursi membuat perbedaan besar dalam situasi ini.

Bab ini membahas fungsi dan operator utama yang memungkinkan iterasi dan rekursi dalam bahasa Power Query M. Anda akan mempelajari cara melakukan loop melalui list dengan List.Transform, menjalankan fungsi pada nilai dengan List.Accumulate, membuat list secara kondisional dengan List.Generate, dan menerapkan rekursi sejati menggunakan operator cakupan @.

Tujuannya adalah untuk memberi Anda keterampilan yang dibutuhkan untuk menggabungkan teknik ini ke dalam alur kerja data Anda sendiri. Topik utama yang dibahas adalah:

- Iterasi:
  - *List.Transform*
  - *List.Accumulate*
  - *List.Generate*
- Rekursi:
  - Operator cakupan @

Untuk mendapatkan manfaat maksimal dari bab ini, akan sangat membantu jika Anda memahami dasar-dasar Power Query M, seperti yang dibahas di bab sebelumnya. Topik seperti memahami nilai terstruktur dan mengakses nilai, yang keduanya dibahas di Bab 2 Jilid 2, akan sangat berguna.

1. **Kasus Pemantik Berfikir Kritis: Menghitung Total Diskon Bertingkat pada Transaksi Pelanggan**

Sebuah toko online memiliki kebijakan diskon bertingkat sebagai berikut:

- Transaksi pertama mendapatkan diskon 5%
- Transaksi kedua mendapatkan diskon 10%
- Transaksi ketiga dan seterusnya mendapatkan diskon 15%

Data transaksi pelanggan tersimpan dalam bentuk list jumlah pembelian (dalam satuan mata uang). Tim analis diminta menggunakan Power Query untuk menghitung total pembayaran akhir yang sudah dipotong diskon sesuai aturan tersebut menggunakan pendekatan iterasi atau rekursi.

Pertanyaan Pemantik:

1) Jenis fungsi iterasi apa yang cocok digunakan untuk menyelesaikan kasus ini? Mengapa?
2) Jika Anda menggunakan List.Accumulate, bagaimana Anda akan menyimpan dan menghitung diskon untuk setiap transaksi?
3) Apa risiko jika pendekatan iterasi tidak disusun dengan benar dalam kasus ini?
4) Bagaimana pendekatan rekursif dapat menyelesaikan kasus ini, dan apa kelemahannya dibanding iterasi dalam konteks kinerja?
5) Jika transaksi dalam jumlah besar (ribuan), pendekatan apa yang lebih efisien secara memori dan waktu proses? Jelaskan alasannya.
6) Bagaimana Anda memastikan bahwa proses iterasi yang dibuat tidak menghasilkan nilai yang salah saat list transaksi kosong atau hanya berisi satu item?
7) Jika diskon bersifat dinamis (berubah tergantung kategori pelanggan), bagaimana Anda akan menyesuaikan fungsi List.Generate untuk menangani kondisi tersebut?

B. Pengantar iterasi

Iterasi merupakan konsep inti dalam pemrograman, yang penting untuk melakukan tindakan berulang pada data. Dalam Power Query, iterasi sering kali terjadi pada baris dalam tabel atau item dalam list.

Misalkan Anda ingin mengkuadratkan list yang berisi 1.000 angka. Melakukan hal ini tanpa iterasi akan memerlukan penulisan 1.000 baris kode terpisah, yang memakan waktu dan rawan kesalahan. Dengan iterasi, tugas yang sama dapat diselesaikan hanya dengan beberapa baris kode.

Bahasa tradisional seperti Python atau Java menggunakan for dan while loop untuk tugas tersebut. Namun, bahasa Power Query M mengambil pendekatan yang berbeda terhadap iterasi. Bahasa ini melakukan transformasi pada struktur data yang mendasarinya berdasarkan desain.

Misalnya, saat Anda menggunakan *Table.AddColumn* untuk membuat kolom baru, Power Query secara otomatis menerapkan operasi yang ditentukan ke setiap baris dalam tabel, mirip dengan cara for loop akan mengulangi setiap elemen dalam list. Bahasa M secara otomatis melakukan iterasi untuk Anda menggunakan fungsi pustaka standarnya.

Namun, untuk mempelajari lebih dalam kemampuan iterasi dalam Power Query M, ada beberapa fungsi yang secara khusus dirancang untuk tujuan ini: *List.Transform, List.Accumulate,* dan *List.Generate*. Fungsi-fungsi ini selaras erat dengan konsep iterasi sebagaimana dipahami dalam konteks pemrograman yang lebih luas.

List.Transform dan *List.Accumulate* memungkinkan Anda untuk melakukan operasi pada setiap item dalam list (seperti for loop), sedangkan List.Generate mencerminkan fungsionalitas while loop karena ia beriterasi berdasarkan kondisi penghentian. List.Generate memungkinkan pembuatan list dengan menerapkan fungsi berulang kali, berlanjut hingga kondisi yang ditentukan tidak lagi terpenuhi.

Di bagian selanjutnya, kita akan menjelajahi tiga fungsi ini—*List.Transform*, *List.Accumulate*, dan *List.Generate*—yang memungkinkan iterasi di Power Query M. Masing-masing menawarkan tingkat kompleksitas dan fleksibilitas yang berbeda, yang sejajar dengan kemampuan iteratif loop for dan while dalam pemrograman tradisional. Dengan menguasai fungsi-fungsi ini, Anda dapat memilih pendekatan yang paling sesuai untuk kebutuhan transformasi data Anda, dengan cara yang sama seperti Anda memilih antara loop for atau while dalam bahasa pemrograman lain.

1. List.Transform

Ketika memikirkan iterasi, *List.Transform* adalah salah satu fungsi yang tepat. Fungsi *List.Transform* menerapkan operasi pada setiap item dalam list. Dengan kata lain, fungsi ini mengambil list dan fungsi sebagai input, lalu membuat list baru yang setiap item outputnya merupakan hasil penerapan fungsi tersebut pada item terkait dalam list asli.

Mari kita bahas beberapa situasi praktis di mana *List.Transform* terbukti berguna. Bayangkan Anda memiliki list angka, misalnya 2, 4, dan 6, dan Anda perlu menggandakan setiap angka. Skenario ini adalah contoh umum iterasi, yang mana operasi diterapkan berulang kali pada setiap item dalam urutan.

Dalam bahasa pemrograman umum, Anda dapat melakukan pendekatan ini dengan for loop, seperti ini:

```
for x in [2, 4, 6]
  {
    array[x] = array[x] * 2;
    x++;
  }
```

Kode ini mengulang setiap elemen dalam array, sehingga nilainya menjadi dua kali lipat. Namun, di Power Query, Anda memperoleh hasil yang sama dengan pendekatan yang lebih ringkas menggunakan fungsi *List.Transform*. Berikut ilustrasinya:

```
List.Transform( { 2, 4 6 }, each _ * 2
```

Dalam contoh ini, *List.Transform* mengambil list {2, 4, 6} dan menerapkan fungsi ke setiap elemen. Fungsi tersebut, yang didefinisikan sebagai each _ * 2, mengalikan setiap item dengan 2, menghasilkan list baru, {4, 8, 12}. Garis bawah (_) mewakili setiap elemen dalam list selama iterasi.

Perbandingan ini menunjukkan perbedaan pendekatan antara for loop tradisional dan gaya fungsional Power Query. Sementara for loop secara eksplisit mengiterasi indeks array, List.Transform mengabstraksi proses iterasi, dengan fokus pada operasi yang diterapkan ke setiap elemen list.

Untuk penjelasan lebih rinci tentang konstruksi *each_* di Power Query, silakan lihat Bab 9.

Mengekstrak Item Dari List Berdasarkan Posisi

Beralih dari contoh iterasi sederhana, mari kita bahas skenario yang lebih kompleks. Bayangkan Anda sedang menyelami pengaturan aplikasi untuk menganalisis preferensi pengguna. Preferensi ini tersimpan dalam list, di mana setiap pengaturan aktif (*true*) atau tidak aktif (*false*). List kita memiliki 8 nilai Boolean, masing-masing terkait dengan pengaturan yang berbeda, seperti **Notifikasi, Mode Gelap, Berbagi Lokas**i, dan **Pembaruan Otomatis**, untuk menyebutkan beberapa di antaranya.

Misalnya list kita disimpan dalam langkah bernama *myList*, yang terlihat seperti ini:

```
{ true, false, true, false, true, true, false, true }
```

Tujuannya adalah untuk mengekstrak nilai pada posisi 2, 3, 5, 6, dan 8 dari list ini. Namun, ada tantangannya: tidak ada fungsi langsung untuk menarik beberapa item dari list berdasarkan posisi indeksnya.

Anda mungkin menganggap fungsi seperti *List.Select* untuk mengatasi hal ini. Fungsi ini mengambil list sebagai input, dan memungkinkan Anda untuk memilih item berdasarkan suatu kondisi. Misalnya:

```
List.Select(  { true, false, true, false, true, true, false, true },
each _ = true )
```

Ekspresi ini mengembalikan list dengan hanya nilai true. Namun, karena nilai duplikat dalam list dan persyaratan untuk memilih nilai yang berbeda, fungsi ini tidak akan berhasil. Alasannya adalah karena ia menentukan item hanya berdasarkan nilainya dan bukan berdasarkan posisinya dalam list. Mari kita jelajahi pendekatan alternatif untuk menyelesaikan masalah ini menggunakan *List.Transform*.

Mengekstrak satu item dari list berdasarkan posisi indeksnya mudah. Misalnya, untuk mendapatkan item kedua dari list, Anda akan menggunakan:

```
myList{1}
```

Cuplikan ini menggunakan indeks berbasis nol dalam tanda kurung kurawal untuk menentukan posisi item. Proses ini dikenal sebagai **field selection**. Namun, bagaimana jika Anda ingin mengumpulkan beberapa item, misalnya, nilai pada posisi 2, 3, 5, 6, dan 8?

Berikut ini adalah pendekatan manual yang berhasil:

```
{ myList{1}, myList{2}, myList{4}, myList{5}, myList{7} }
```

Hasil dari kode di atas adalah:

```
{ false, true, true, true, false }
```

Meskipun benar, kodenya terasa agak bertele-tele dan kurang fleksibel. Ini adalah kasus penggunaan umum di mana fungsi *List.Transform* sangat berguna. Kita dapat melakukan operasi yang identik dengan kode yang lebih sedikit. Melakukan tugas ini menggunakan *List.Transform* memerlukan dua argumen:

- **List**: Mulailah dengan list posisi indeks item yang Anda cari.
- **Function**: Buat fungsi untuk mengambil posisi dari list.

Dengan metode sebelumnya, kita dapat meniru fungsi untuk melakukan operasi serupa tetapi dengan cara yang lebih ringkas. Berikut cara Anda dapat menulis ulang operasi menggunakan *List.Transform*:

```
let
  myList =
    { true, false, true, false, true, true, false, true },
  RetrieveItems =
    List.Transform( {1, 2, 4, 5, 7}, each myList{_} )
in
  RetrieveItems
```

Dalam metode yang direvisi ini, *List.Transform* mengulangi posisi indeks yang diberikan. Untuk setiap nilai, metode ini menerapkan fungsi dari argumen kedua, mengambil item yang sesuai dari *myList*. Pendekatan ini tidak hanya lebih pendek tetapi juga dapat disesuaikan untuk penyesuaian selanjutnya. Jika Anda perlu mengubah ke posisi yang berbeda nanti, cukup ubah posisi indeks dalam argumen pertama *List.Transform* dan Anda siap melakukannya.

Mengalokasikan Anggaran Tahunan ke Bulan

Untuk skenario lain, bayangkan Anda bekerja dengan kumpulan data yang berisi anggaran penjualan untuk tahun 2024. Kumpulan data tersebut disimpan dalam langkah yang disebut Sumber dan terlihat seperti berikut:

	Product	Budget	Year
1	iPhone 14	1000000	2024
2	MacBook Air	500000	2024
3	Amazon Echo Dot	250000	2024
4	Tesla Model 3	100000	2024
5	Nike Air Jordans	50000	2024

Gambar 1. 1 Tabel Nilai Anggaran Tahun 2024

Untuk mengikuti contoh ini, Anda dapat menemukan kumpulan data di atas dalam kueri *Start of month dates* dari berkas latihan yang menyertai buku.

Perhatikan bahwa, untuk tugas ini, data di atas diberikan per tahun. Sasaran Anda adalah mendistribusikan angka anggaran di setiap bulan dalam setahun.

Untuk mencapainya, berikut ini yang dapat Anda lakukan:

1) Buat list tanggal awal untuk setiap bulan di tahun 2024.
2) Perluas list ini menjadi beberapa baris.
3) Bagi nilai anggaran tahunan dengan 12 untuk mendapatkan angka bulanan.

Sebelum menambahkan kolom apa pun ke tabel kita, mari kita pahami elemen yang diperlukan untuk pendekatan di atas.

Pertama, kita ingin membuat list tanggal untuk setiap bulan di tahun 2024. Karena jumlah nilai yang akan dihasilkan untuk skenario ini sudah tetap, ini adalah skenario yang bagus untuk *List.Transform*. Berikut ini salah satu cara untuk melakukannya. Untuk memulai, Anda membuat list dari 1 hingga 12:

```
{ 1 .. 12 }
```

Sekarang kita perlu cara untuk mengubahnya menjadi tanggal. Kumpulan data sudah mencakup tahun 2024, yang dapat kita gunakan, dan kita memiliki list 12 nilai untuk nomor bulan.

Untuk membuat list tanggal bulanan, kita ingin memulai dengan 1 Januari 2024. Untuk setiap bulan berikutnya, kita dapat menggunakan fungsi *Date.AddMonths* di samping list nomor untuk menambah tanggal sebanyak satu bulan di setiap langkah.

Anda tetap tidak perlu menambahkan apa pun ke tabel. Mari kita mulai dengan memahami contoh mudah dan fokus hanya pada angka 6 dalam list yang kita buat. Kita dapat mengubah 6 menjadi 1 Juni 2024, dengan menulis:

```
Date.AddMonths( #date( 2024, 1, 1 ), 6 - 1 )
```

Di sini, angka 6 dikurangi 1 lalu digunakan untuk menambah tanggal awal sebanyak bulan tersebut. Jika Anda memahami konsep di atas, akan lebih mudah untuk memanipulasi kumpulan data kita.

Tujuan kita adalah menambahkan kolom dengan list ke setiap baris. List ini akan menampilkan semua tanggal awal bulan untuk tahun baris tersebut. Kita melakukannya dengan memproses urutan 12 nilai dengan fungsi *List.Transform*.

Dengan pengetahuan tentang cara menambahkan bulan ke suatu tanggal, berikut cara kita menerapkannya. Mari perkenalkan kolom baru di tabel utama kita. Kolom baru tersebut mengambil kolom tahun yang ada dan memasukkan urutan 12 nilai. Kode untuk ini terlihat seperti ini:

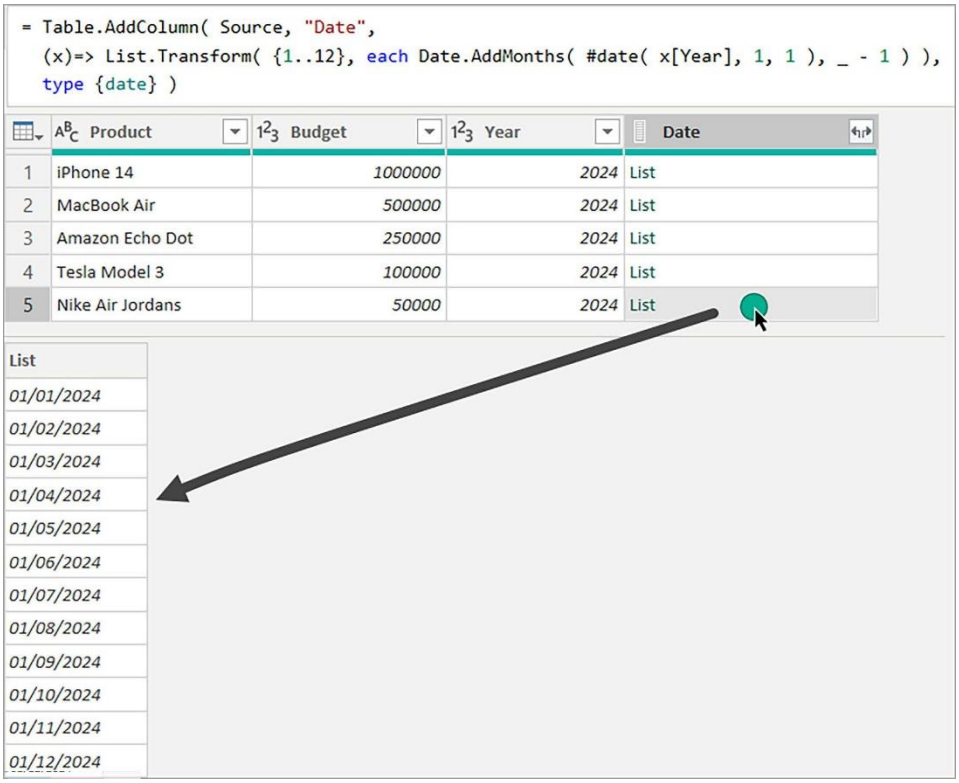
```
Table.AddColumn(
  Source,
  "Date",
  (x)=> List.Transform( {1..12},
         each Date.AddMonths( #date(x[Year], 1, 1), _ - 1 )),
  type {date} )
```

Skrip ini membuat kolom Tanggal baru ke tabel yang disebut *Source*, yang merupakan tabel utama kita di panel **Applied Steps**. Skrip ini menggunakan informasi Tahun dari setiap baris tabel ini. Tantangan di sini terletak pada cakupan yang digunakan dalam ekspresi tersebut. Biasanya, fungsi *Table.AddColumn* akan menggunakan ekspresi each dalam argumen ketiganya untuk mendefinisikan suatu fungsi. Ini adalah cara singkat untuk membuat suatu fungsi dengan simbol garis bawah sebagai nama variabel. *List.Transform* juga secara default menggunakan ekspresi each untuk mendefinisikan ekspresi fungsi dalam argumen keduanya. Namun, menggunakan dua fungsi dengan nama variabel yang sama (garis bawah) menciptakan ambiguitas.

Jika fungsi *List.Transform* dan fungsi *Table.AddColumn* menggunakan kata kunci each, referensi ke garis bawah (nama parameter) hanya dapat mengakses variabel dalam konteksnya sendiri. Namun, untuk contoh kita, kita memerlukan fungsi *List.Transform* (cakupan dalam) untuk mengakses nilai Tahun dalam tabel (cakupan luar). Untuk mengatasi hal ini, kode kita mendefinisikan fungsi kustom untuk cakupan luar (tabel *Source*) menggunakan variabel x. Hal ini memungkinkan kita untuk merujuk kolom

Tahun dari tabel Sumber saat kita berada di dalam konteks fungsi List.Transform.

Sebagai langkah terakhir, kode ini menetapkan tipe data dalam argumen keempat *Table.AddColumn*. Itu didefinisikan sebagai tipe list yang berisi tanggal. Hasil penambahan langkah ini ditunjukkan dalam gambar berikut:



Gambar 1. 2 Baris yang berisi list dengan semua tanggal awal bulan dalam setahun

Setelah membuat kolom, klik tanda panah di sebelah kanan tajuk kolom Tanggal, lalu pilih **Expand to New Rows**. Tindakan ini akan menghasilkan tabel dengan tanggal mulai untuk setiap bulan pada tahun 2024:

	Product	Budget	Year	Date
1	iPhone 14	1000000	2024	01/01/2024
2	iPhone 14	1000000	2024	01/02/2024
3	iPhone 14	1000000	2024	01/03/2024
4	iPhone 14	1000000	2024	01/04/2024
5	iPhone 14	1000000	2024	01/05/2024
6	iPhone 14	1000000	2024	01/06/2024
7	iPhone 14	1000000	2024	01/07/2024
8	iPhone 14	1000000	2024	01/08/2024
9	iPhone 14	1000000	2024	01/09/2024
10	iPhone 14	1000000	2024	01/10/2024
11	iPhone 14	1000000	2024	01/11/2024
12	iPhone 14	1000000	2024	01/12/2024

Gambar 1. 3 Tabel dengan anggaran yang diulang setiap bulan dalam setahun

Untuk menyelesaikannya, bagi kolom anggaran dengan 12. Cara termudah untuk melakukannya adalah dengan memilih kolom **Budget**, navigasikan ke **Transform**, pilih **Standard**, lalu **Divide**, dan masukkan nilai 12. Setelah mengonfirmasi, Anda akan memiliki kumpulan data yang menampilkan anggaran yang dialokasikan per bulan.

	Product	1.2 Budget	Year	Date
1	iPhone 14	83333.33333	2024	01/01/2024
2	iPhone 14	83333.33333	2024	01/02/2024
3	iPhone 14	83333.33333	2024	01/03/2024
4	iPhone 14	83333.33333	2024	01/04/2024
5	iPhone 14	83333.33333	2024	01/05/2024
6	iPhone 14	83333.33333	2024	01/06/2024
7	iPhone 14	83333.33333	2024	01/07/2024
8	iPhone 14	83333.33333	2024	01/08/2024
9	iPhone 14	83333.33333	2024	01/09/2024
10	iPhone 14	83333.33333	2024	01/10/2024
11	iPhone 14	83333.33333	2024	01/11/2024
12	iPhone 14	83333.33333	2024	01/12/2024

Gambar 1. 4 Anggaran tahunan dialokasikan per bulan

Sejauh ini, kita telah melihat bagaimana *List.Transform* merupakan fungsi yang berguna untuk mengulang list dan melakukan tindakan pada setiap elemen. Namun, bagaimana jika Anda memerlukan kontrol lebih, terutama ketika hasil dari satu pengulangan perlu memengaruhi pengulangan

berikutnya? Di sinilah *List.Accumulate* berperan, menyediakan serangkaian fitur yang lebih canggih.

2. List.Accumulate

List.Accumulate adalah iterator hebat lainnya. Sama seperti *List.Transform*, fungsi ini menelusuri item dalam list satu demi satu, melakukan tindakan tertentu pada setiap item. Namun, tidak seperti kebanyakan iterator, *List.Accumulate* memiliki memori.

Saat menghasilkan nilai baru, fungsi ini dapat mengakses item baru dalam list dan hasil iterasi sebelumnya. Sifat ini memungkinkan fungsi ini untuk melakukan operasi dan menggunakan hasil operasi ini sebagai input untuk langkah berikutnya.

Sekilas, Anda mungkin melihat beberapa kesamaan dengan rekursi, yang akan kita bahas nanti di bab ini. Namun, esensi *List.Accumulate* tidak sepenuhnya rekursi. Fungsi ini lebih seperti operasi kumulatif, di mana setiap langkah dibangun di atas hasil langkah sebelumnya, oleh karena itu dinamakan acquire. Perbedaan ini memungkinkan *List.Accumulate* untuk melakukan tugas-tugas di mana hasil dari langkah-langkah sebelumnya memengaruhi operasi berikutnya. Jadi, bagaimana cara kerjanya?

Anatomi Fungsi

Sintaks fungsinya adalah sebagai berikut:

```
List.Accumulate(
   list as list,
   seed as any,
   accumulator as function )
```

Istilah-istilah ini mungkin sedikit abstrak, jadi mari kita bahas lebih dalam masing-masing istilah:

- *list*: Ini adalah list awal yang akan digunakan. Fungsi ini mengulangi setiap nilai dalam list ini saat menerapkan logika.

- *seed*: Seed adalah titik awal Anda. Ini adalah nilai awal yang Anda gunakan sebelum menelusuri item dalam list Anda.
- *accumulator*: Berisi serangkaian instruksi tentang apa yang harus dilakukan *List.Accumulate* dengan setiap item dalam list saat menelusurinya satu per satu. *Accumulator* adalah fungsi dengan dua variabel: nilai akumulasi (yang dimulai sebagai seed) dan item saat ini dari list. Accumulator melakukan tindakan tertentu, membuat nilai akumulasi baru, lalu beralih ke item berikutnya dalam list, sambil membawa serta nilai akumulasi tersebut.

Mari kita lihat cara kerjanya dengan sebuah contoh. Pertimbangkan skenario di mana Anda memiliki list yang berisi angka 1 hingga 5, dan Anda ingin mengalikan nilai-nilai ini bersama-sama. Hasil yang diinginkan adalah hasil perkalian 1 * 2 * 3 * 4 * 5, yang sama dengan 120.

Dalam lingkungan pemrograman tradisional, Anda dapat menggunakan struktur loop untuk mengakumulasikan hasil perkalian ini, seperti ini:

```
y = 1
for x = 1 to 5
  {

    y = y * array[x];
  }
```

Perulangan ini secara berulang mengalikan variabel akumulator y dengan setiap elemen dalam array. Demikian pula, dalam Power Query, fungsi *List.Accumulate* mencapai hasil yang sama dengan gaya yang lebih fungsional:

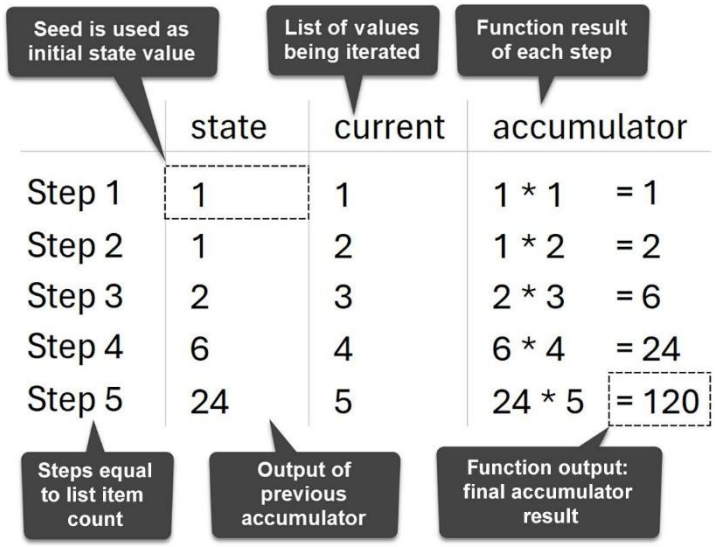
```
List.Accumulate(
  { 1, 2, 3, 4, 5 },
  1,
  ( state, current ) => state * current )
```

Output dari fungsi ini adalah 120. Berikut cara kerjanya:

- *list*: Fungsi ini mengiterasi nilai 1 hingga 5.
- *seed*: Operasi dimulai dengan nilai 1, yang disebut sebagai nilai status pada langkah pertama.

- *accumulator*: Fungsi ini mengambil nilai status dan mengalikannya dengan item saat ini dalam list yang sedang diiterasi.

Prosesnya dapat divisualisasikan seperti yang diilustrasikan pada Gambar 13.5. Setiap langkah fungsi mencerminkan iterasi loop, mengalikan status yang terakumulasi dengan elemen saat ini.



Gambar 1. 5 Langkah-langkah iterasi untuk List.Accumulate

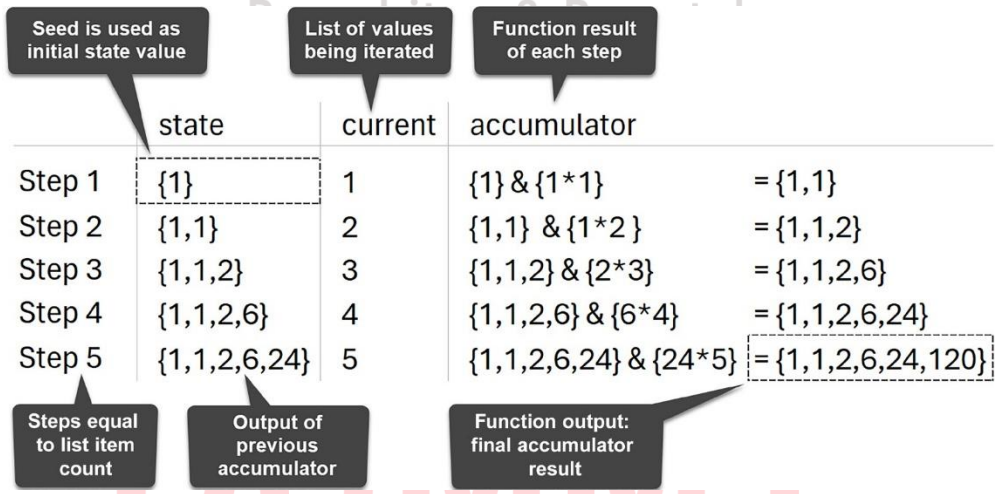
Mari kita lihat lebih dekat cara kerja fungsi akumulator. Fungsi akumulator menggunakan dua parameter input. Meskipun Anda bebas memberi nama parameter ini sesuai keinginan, untuk kejelasan, di sini mereka diberi nama state dan current:

- state mewakili nilai yang terakumulasi. Pada iterasi pertama, nilainya sama dengan nilai awal. Saat fungsi beriterasi dan menerapkan fungsi akumulator, nilai state berevolusi pada setiap langkah, yang mencerminkan output dari hasil akumulator terakhir.
- current mewakili nilai dari list yang sedang diproses. Pada iterasi pertama, nilainya sama dengan nilai pertama dalam list; pada langkah berikutnya, nilainya berpindah ke nilai kedua, terus berlanjut dengan cara ini hingga nilainya berpindah melalui semua nilai dalam list.

Untuk memvisualisasikannya dengan lebih baik, kita dapat menyesuaikan pernyataan *List.Accumulate* untuk mempertahankan nilai total yang berjalan:

```
List.Accumulate(
  { 1, 2, 3, 4, 5 },
  {1},
  ( state, current ) =>
    state & { List.Last( state ) * current }
)
```

Output dari fungsi yang direvisi adalah {*1,1,2,6,24,120*}. Logika langkah-langkah ini dijabarkan dalam gambar berikut:



Gambar 1. 6 *List.Accumulate* logika untuk menyimpan hasil dalam list

Alih-alih hanya mengembalikan hasil perkalian, logika di atas menyimpan hasil akumulator dari setiap langkah dalam sebuah list. Pada setiap iterasi, akumulator mengambil item terakhir dari list dan mengalikannya dengan nilai saat ini. Kemudian akumulator menambahkan hasilnya ke hasil akumulator terakhir. Akumulator melakukannya dengan menggabungkan dua nilai list. Ekspresi seperti {1} & {1} menggabungkan dua list menjadi satu, menghasilkan {1,1}.

Contoh-contoh di atas menggambarkan alur fungsi *List.Accumulate*. Namun, Anda biasanya akan menemukan skenario yang berbeda di mana

fungsi tersebut terbukti berguna. Mari kita lihat contoh yang lebih berguna di mana *List.Accumulate* sangat berguna.

Mengganti Beberapa Nilai

Jadi, untuk skenario mana Anda akan menggunakan *List.Accumulate* alih-alih iterator lainnya? Fungsi tersebut adalah sebuah iterator, artinya fungsi tersebut melakukan operasi berulang kali. Yang lebih penting, *List.Accumulate* dapat mengakses hasil dari iterasi sebelumnya dan membawanya ke iterasi berikutnya. Dalam skenario-skenario seperti itulah *List.Accumulate* memiliki kemampuan yang tidak dimiliki sebagian besar fungsi lainnya. Untuk mengikuti contoh berikut, Anda dapat menemukan data dalam berkas latihan yang menyertai bab ini dalam kueri *ListAccumulate Manual Replacement*.

Bayangkan sebuah skenario di mana Anda perlu membersihkan data berikut:

	Names
1	Rick_de_Groot!
2	Greg-Deckler!
3	Melissa.de_Korte
4	Brian.Julius!
5	John.Doe!
6	Jane_Smith!
7	Sally.Jones!
8	Tommy_Shelby!

Gambar 1. 7 Kumpulan data yang akan dibersihkan

Anda ingin mengembalikan kolom yang berisi nama-nama, dengan spasi yang tepat di antaranya dan tanpa karakter khusus. Menghapus karakter khusus biasanya memerlukan empat operasi penggantian dengan logika yang sangat mirip. Satu-satunya perbedaan adalah serangkaian nilai yang berbeda yang diganti. Karakter _, -, dan . harus diganti dengan spasi, dan tanda seru harus dihapus.

Operasi-operasi ini sangat mirip, dan kita harus dapat mereplikasinya dalam fungsi *List.Accumulate*. Untuk memulai, mari kita lihat seperti apa kode untuk operasi penggantian nilai biasa.

Jika Anda mengklik kanan kolom **Names**, pilih **Replace Values**, dan mengganti setiap garis bawah dengan spasi, Anda akan mendapatkan kode berikut:

```
Table.ReplaceValue(
  Source,
  "_",
  " ",
  Replacer.ReplaceText,
  {"Names"}
)
```

Melakukan operasi ini empat kali akan menambahkan empat langkah ke kueri Anda. Mari kita lihat bagaimana kita dapat menggunakan *List.Accumulate* untuk menyimpan logika ini dalam satu langkah.

Saat menyiapkan pernyataan *List.Accumulate*, pertama-tama kita perlu mengidentifikasi nilai yang akan diulang dan nilai awal:

- **Nilai yang akan diulang (*Values to iterate*)**: Tujuan fungsi ini adalah mengganti nilai lama dengan nilai baru. Dalam setiap pengulangan, kita memerlukan akses ke nilai lama dan baru. Itu berarti kita perlu menyimpan item dalam list atau record.
- **Nilai awal (*Seed value*)**: Awal menentukan nilai awal operasi *List.Accumulate*. Karena kita akan mengganti nilai dalam tabel, kita menyediakan tabel asli di sini.

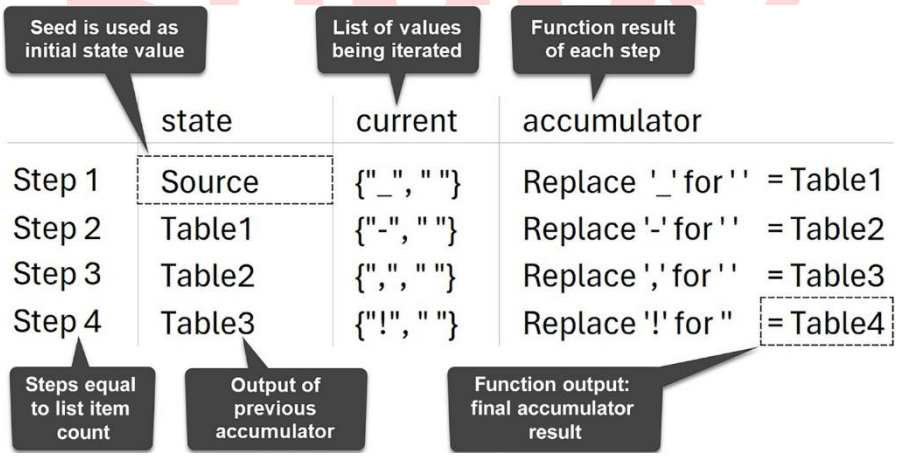
Berikut ini adalah tampilan ekspresi *List.Accumulate*:

```
List.Accumulate(
    { {"_"," "}, {"-", " "}, {".", " "}, {"!",""} }, // Sets of replacements
    Source, // Initial table value (seed)
    ( state, current ) =>
      Table.ReplaceValue(
        state,      // Table being updated with each replacement
        current{0}, // Character to be replaced - first item from the current pair
        current{1}, // Replacement character - second item from the current pair
        Replacer.ReplaceText,{"Names"} // Column name where replacements are made
      )
)
```

Gambar 1. 8 Menggunakan ekspresi *List.Accumulate*

Dalam kode ini, baris 2 berisi nilai pengganti. Ini adalah list yang setiap list bagian dalamnya berisi nilai lama dan baru. Baris 3 merujuk ke tabel asli yang disebut Sumber tempat kita mengganti nilai.

Saat menjalankan fungsi, dengan setiap iterasi (dan karenanya setiap penggantian), nilai status berubah ke tabel baru dengan nilai yang diganti. Untuk mengilustrasikannya, lihat gambar berikut.



Gambar 1. 9 Logika penggantian langkah demi langkah untuk *List.Accumulate*

Gambar 1.9 menunjukkan bagaimana kita memulai dengan tabel yang disebut *Source*. Kemudian, dengan setiap penggantian, isi tabel berubah, diilustrasikan oleh versi **Table1** hingga **Table3**. Hasil **Table1** hingga **Table3** bukanlah tabel terpisah yang dapat diakses tetapi dihitung dalam memori untuk membangun hasil, **Table4**.

Setelah menyelesaikan langkah-langkah ini, Anda akan mendapatkan kumpulan data yang telah dibersihkan, seperti yang ditunjukkan pada Gambar 1.10.

	Names
1	Rick de Groot
2	Greg Deckler
3	Melissa de Korte
4	Brian Julius
5	John Doe
6	Jane Smith
7	Sally Jones
8	Tommy Shelby

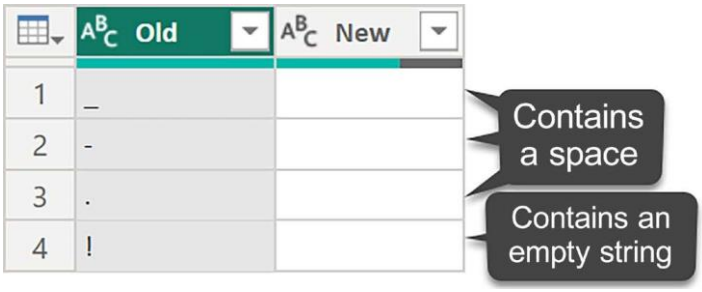
Gambar 1. 10 Membersihkan dataset menggunakan fungsi List.Transform

Anda mungkin telah memperhatikan bahwa kode kita menentukan penggantian yang tepat, yang mungkin tidak memberi Anda fleksibilitas yang Anda inginkan. Pendekatan yang lebih mudah dikelola adalah dengan menyimpan nilai lama dan nilai baru yang sesuai dalam tabel dan menggunakannya sebagai input. Berikut cara melakukannya. List penggantian asli kita terlihat seperti ini:

```
{ {"_"," "}, {"-", " "}, {".", " "}, {"!",""} }
```

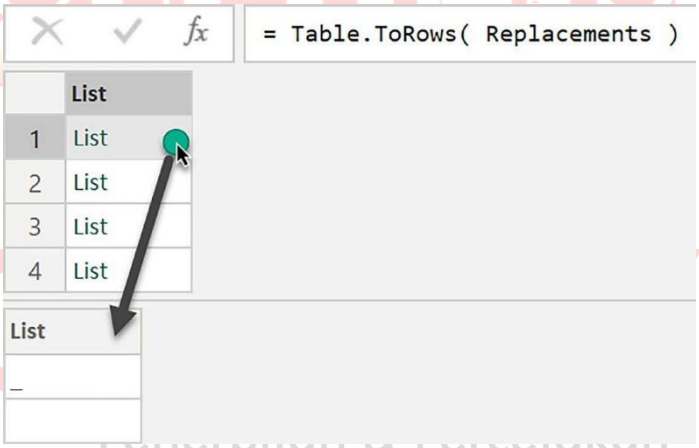
Ini adalah list yang berisi list bertingkat dengan nilai yang akan diganti dan nilai yang akan diganti. Anggaplah Anda menyimpan nilai-nilai ini dalam tabel yang disebut *Replacements*.

Kode tersebut memerlukan list yang lebih kecil sebagai input. Setiap list bagian dalam memiliki dua elemen: nilai yang akan diganti dan nilai yang akan diganti. Bagaimana jika Anda menyimpan pasangan ini dalam tabel bernama Penggantian? Lihat Gambar 1.11 untuk melihat seperti apa tabel Penggantian ini.



Gambar 1. 11 Meja dengan penggantinya

Tujuannya adalah mengonversi tabel ini ke dalam format list yang mirip dengan yang digunakan *List.Accumulate*. Fungsi *Table.ToRows* dapat membantu kita di sini. Fungsi ini mengubah setiap baris tabel menjadi list terpisah. Lihat Gambar 1.12, di mana *Table.ToRows* mengubah tabel di atas menjadi list berisi list, satu untuk setiap baris:



Gambar 1. 12 *Table.ToRows* mengubah tabel menjadi list yang berisi list untuk setiap baris dalam tabel

Itulah format yang kita perlukan sebagai input untuk fungsi kita. Saat kita mengganti list pengganti asli dengan ekspresi *Table.ToRows*, kita akan mendapatkan rumus akhir:

```
List.Accumulate(
    Table.ToRows( Replacements ), // Dynamic replacement list
    Source,
    ( state, current ) =>
      Table.ReplaceValue(
        state,
        current{0},
        current{1},
        Replacer.ReplaceText,{"Names"}
      )
)
```

Rumus tersebut sekarang merujuk ke tabel Penggantian di baris kedua. Pilihan desain ini berguna saat Anda perlu membuat perubahan. Jika Anda ingin menambahkan atau menyesuaikan penggantian, cukup perbarui tabel. Fungsi tersebut secara otomatis mengambil perubahan ini. Anda dapat menemukan solusi dan data dasar dalam berkas latihan yang disertakan dalam buku ini.

Seperti yang telah kita lihat, *List.Accumulate* menawarkan tingkat kontrol yang tidak dapat ditandingi oleh fungsi yang lebih sederhana. Untuk iterasi sederhana, *List.Transform* mungkin adalah semua yang Anda butuhkan. Namun, jika pekerjaan Anda mengharuskan membawa hasil dari satu iterasi ke iterasi berikutnya, *List.Accumulate* menjadi sangat diperlukan.

Selanjutnya, kita akan menyelami *List.Generate*. Fungsi ini memiliki tujuan yang sama tetapi dengan sedikit perubahan: alih-alih mengiterasi melalui sejumlah item yang ditetapkan, fungsi ini terus berjalan selama kondisi tertentu berlaku.

3. List.Generate

Fungsi *List.Generate* adalah salah satu fungsi paling canggih dalam bahasa M. Fungsi ini memungkinkan Anda untuk melakukan iterasi berulang kali, berdasarkan kondisi dan transformasi yang ditentukan. Namun, fungsi ini juga merupakan salah satu fungsi paling rumit yang tersedia.

List.Generate berbeda dari metode statis seperti *List.Numbers* dan *List.Accumulate* (yang telah kita bahas di Bab 2 Jilid 2, Nilai Terstruktur).

Sementara metode statis bergantung pada input yang telah ditentukan sebelumnya, *List.Generate* menghadirkan elemen fleksibilitas. Fungsi ini memungkinkan Anda untuk memasukkan fungsi sebagai argumen. Fungsi ini menginstruksikan bagaimana list terbentuk, dari awal hingga akhir, berdasarkan logika unik Anda. Konsep ini sangat berguna ketika Anda mencapai batas kemampuan metode statis. Fungsi *List.Generate* memiliki aspek iteratif dan rekursif:

- **Aspek iterasi**: *List.Generate* mengiterasi melalui suatu proses, membuat list berdasarkan kondisi yang ditentukan. Proses ini memerlukan nilai awal, memiliki fungsi langkah yang menentukan cara untuk mencapai nilai berikutnya, dan berlanjut selama kondisi yang ditentukan *true*. Proses iteratif ini mirip dengan *while loop*, yang mengeksekusi suatu langkah, memeriksa suatu kondisi, dan melanjutkan atau mengakhiri berdasarkan kondisi tersebut. Struktur List.Generate mewujudkan esensi iterasi.
- **Aspek rekursif**: Meskipun tidak rekursif dalam pengertian konvensional, cara *List.Generate* dapat membuat urutan nilai berdasarkan perhitungan sebelumnya memiliki kemiripan dengan rekursi. Fungsi *step* dapat merujuk ke nilai yang dihitung sebelumnya, seperti yang dilakukan *List.Accumulate*. Proses ini mirip dengan cara rekursi menggunakan kembali keluaran suatu operasi, meskipun tanpa fungsi yang memanggil dirinya sendiri, yang akan menjadi rekursi *true*.

Jadi, mengapa Anda mungkin lebih memilih *List.Generate* daripada metode lain?

Keuntungan dari *List.Generate*

Fungsi *List.Generate* memiliki manfaat dibandingkan metode lain, terutama rekursi tradisional, yang dibahas di bagian berikutnya. Keuntungan terpentingnya adalah:

- **Efisiensi kinerja**: *List.Generate* biasanya berkinerja lebih baik daripada rekursi tradisional. Rekursi membuat beberapa panggilan fungsi terpisah

dan menambahkan hasil perantara ke tumpukan. Proses ini terkenal lambat. Sebaliknya, *List.Generate* bekerja dalam satu konteks. Ini berarti cenderung menggunakan lebih sedikit memori dan berjalan lebih cepat.

- **Iterasi yang dapat dilacak**: Saat menggunakan *List.Generate*, mudah untuk melacak hasil setiap iterasi karena hasilnya disimpan sebagai nilai dalam list yang dihasilkan. Ini memberikan pandangan yang jelas tentang bagaimana setiap iterasi berkontribusi pada list akhir. Di sisi lain, dalam pengaturan rekursif, seperti yang dibahas di bab berikutnya, Anda hanya melihat hasil dari semua iterasi. Itu dapat mempersulit untuk mengikuti alur logis dan men-debug setiap langkah.
- **Mudah dipahami**: *List.Generate* memiliki struktur yang jelas yang membuat kode lebih mudah dibaca dan dipahami. Setiap bagian dari fungsi didefinisikan dengan baik, membantu Anda memahami logikanya dengan cepat. Sebaliknya, rekursi sering kali melibatkan panggilan yang rumit dan saling terkait yang mungkin memerlukan lebih banyak waktu untuk dipahami.

Jadi, bagaimana cara kerja fungsi tersebut?

Anatomi Fungsi

Mengklarifikasi cara kerja *List.Generate* tidak hanya melibatkan pemahaman sintaksisnya tetapi juga pemahaman alur berurutan dari empat argumen utamanya: *Initial*, *Condition*, *Next*, dan *Selector*. Argumen-argumen ini membentuk instruksi untuk list dinamis yang akan dihasilkan oleh fungsi tersebut. Fungsi *List.Generate* dalam bahasa M menggunakan sintaksis berikut:

```
List.Generate(
  Initial as function,
  Condition as function,
  Next as function,
  optional Selector as nullable function
) as list
```

Setiap argumen dalam *List.Generate* mengambil fungsi sebagai nilainya. Ini membedakannya dari banyak fungsi bahasa M lainnya. Berikut adalah penjelasan lebih rinci tentang setiap argumen: Kita dapat mendeskripsikan argumen ini sebagai:

- *Initial*: Ini adalah titik awal Anda. Ini bisa berupa nilai sederhana seperti angka atau struktur yang lebih kompleks seperti record. List mulai dibangun dari sini.
- *Condition*: Argumen Condition bertindak sebagai pengujian yang harus dilalui suatu nilai sebelum List.Generate mempertimbangkan untuk menambahkannya ke list. Awalnya, argumen ini mengevaluasi nilai awal. Jika nilai awal ini tidak memenuhi kondisi, fungsi akan segera mengembalikan list kosong. Jika nilai awal lulus pengujian, fungsi akan melanjutkan untuk menghasilkan item berikutnya dalam list menggunakan argumen Next. Namun, nilai baru ini hanya ditambahkan ke list jika juga memenuhi kondisi. Fungsi akan terus menghasilkan dan mengevaluasi item baru dengan cara ini hingga menemukan nilai yang gagal memenuhi kondisi, dan pada titik ini, pembuatan list akan berhenti.
- *Next*: Di sinilah setiap nilai baru dalam list dibentuk. Fungsi yang Anda tempatkan di sini menentukan logika untuk menghasilkan nilai-nilai berikutnya dalam list.
- Selector: Argumen opsional ini memungkinkan Anda mengubah list akhir. Misalnya, jika Anda memulai dengan list record, Anda dapat menggunakan ini untuk memilih hanya bidang-bidang tertentu. Atau, Anda dapat menggunakannya untuk mengubah setiap nilai, seperti menambahkan awalan atau mengubahnya ke jenis nilai yang berbeda.

Mari kita periksa cara kerja fungsi tersebut menggunakan contoh. Semua contoh yang digunakan di seluruh bagian ini tersedia dalam berkas latihan yang disertakan dalam buku ini. Kode berikut membuat list angka mulai dari 1 hingga 9:

```
List.Generate(
   () => 1,      // starting value: 1
   each _ < 10,  // as long as the value is smaller than 10
   each _ + 1    // increment each value by 1 )
```

Jadi apa yang terjadi di sini?

- *() => 1*: Fungsi dimulai dengan nilai awal 1, yang diberikan sebagai fungsi parameter nol.
- *each _ < 10*: Fungsi mengevaluasi kondisi ini untuk setiap nilai yang dihasilkannya, termasuk nilai awal. Setelah kondisi gagal, fungsi berhenti menghasilkan nilai list baru. Dalam contoh kita, kondisi memastikan bahwa list akan berhenti bertambah setelah nilai mencapai 9.
- *each _ + 1*: Ini adalah aturan untuk membuat setiap nilai berikutnya. Kita menginstruksikan fungsi untuk menambah setiap nilai sebesar 1.

Operasi *List.Generate* dapat dibandingkan dengan *while loop* tradisional yang ditemukan dalam banyak bahasa pemrograman. Kesamaannya terletak pada bagaimana kedua struktur mengelola iterasi dan pemeriksaan kondisi. Berikut adalah tampilan *while loop*, yang mencerminkan logika *List.Generate*:

```
y = 1
while (y < 10)
  {
    y = y + 1;
  }
```

Dalam while loop ini:

- *Loop* dimulai dengan y yang ditetapkan ke 1, mirip dengan nilai awal di *List.Generate*.
- *Loop* berlanjut selama y kurang dari 10, sejajar dengan pemeriksaan kondisi di *List.Generate*.
- Nilai y meningkat sebesar 1 di setiap iterasi, seperti langkah penambahan di *List.Generate*.

List.Generate dan *while loop* dimulai dengan nilai awal, menghitung nilai berikutnya dalam urutan, memeriksa nilai baru ini terhadap suatu kondisi, dan memutuskan apakah akan melanjutkan proses atau berhenti. Perbandingan ini

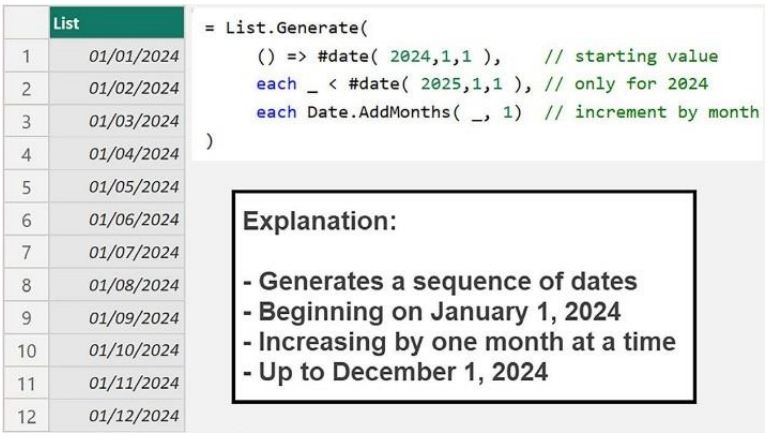
menunjukkan bagaimana *List.Generate* beroperasi secara rekursif, melacak nilai yang dihasilkan terakhir, menggunakannya sebagai input untuk perhitungan berikutnya, dan menentukan apakah akan memasukkannya dalam list atau berhenti berdasarkan kondisi yang ditentukan.

Contoh sebelumnya menggunakan tiga argumen wajib. Mereka menyediakan logika input untuk membuat list utama. Akan tetapi, ada argumen keempat yang opsional di *List.Generate*: pemilih, yang memungkinkan Anda mengubah list yang mendasarinya. Mari kita lihat betapa mudahnya cara kerjanya dengan contoh lain.

Misalkan Anda ingin membuat list dengan 12 kalimat, satu untuk setiap bulan dalam setahun. Di dalamnya, Anda ingin mengembalikan kalimat yang mengatakan: 1 Januari 2024 jatuh pada hari Senin. Untuk mencapainya, kita harus melakukan langkah-langkah berikut:

- Gunakan *List.Generate* untuk membuat list semua tanggal dalam setahun yang berada di awal bulan.
- Gunakan pemilih untuk mengubah tanggal-tanggal ini ke dalam format yang tepat.

Untuk membuat list yang berisi semua nilai awal bulan pada tahun 2024, Anda dapat menggunakan pernyataan pada Gambar 1.13.



Gambar 1. 13 Buat urutan tanggal menggunakan *List.Generate*

Logika ini memerintahkan fungsi untuk mulai pada tanggal 1 Januari 2024. Kemudian, selama tanggal tersebut sebelum tanggal 1 Januari 2025, maka setiap nilai akan bertambah 1 bulan.

Untuk mengubah list ke dalam format yang diinginkan, Anda dapat menggunakan fungsi seperti *List.Transform*. Akan tetapi, akan lebih mudah menggunakan argumen pemilih *List.Generate* untuk mengubah list Anda. Argumen pemilih menerapkan fungsi pada list yang dihasilkan oleh tiga argumen pertama *List.Generate*. Hal itu memungkinkan Anda untuk tetap menggunakan logika dalam fungsi yang sama. Lihat ini:

```
List.Generate(
    () => #date( 2024,1,1 ),     // starting value
    each _ < #date( 2025,1,1 ), // only for 2024
    each Date.AddMonths( _, 1), // increment by month
    each Date.ToText( _, "MMMM d, yyyy") & " is on a "
           & Date.ToText( _, "dddd") )
```

Dalam contoh ini, argumen selector melakukan tiga hal:

- Menggunakan *Date.ToText* untuk memformat tanggal yang mendasarinya dalam format *MMMM d yyyy*, yang, untuk nilai pertama, sama dengan 1 Januari 2024.
- Menggabungkan string "*is on a*" ke nilai pertama.
- Menggunakan fungsi *Date.ToText* untuk menambahkan nilai hari dalam seminggu ke string.

Kode ini memberi kita list dengan nilai teks yang diinginkan:

	List
1	January 1, 2024 is on a Monday
2	February 1, 2024 is on a Thursday
3	March 1, 2024 is on a Friday
4	April 1, 2024 is on a Monday
5	May 1, 2024 is on a Wednesday
6	June 1, 2024 is on a Saturday
7	July 1, 2024 is on a Monday
8	August 1, 2024 is on a Thursday
9	September 1, 2024 is on a Sunday
10	October 1, 2024 is on a Tuesday
11	November 1, 2024 is on a Friday
12	December 1, 2024 is on a Sunday

Gambar 1. 14 Output aker menerapkan fungsi pemilih

Untuk mencapai hasil ini, kita dapat bekerja dengan pernyataan *List.Generate* yang relatif sederhana. Namun, ada situasi di mana Anda perlu memasukkan beberapa nilai untuk mencapai hasil yang diinginkan. Untuk melakukannya, kita memerlukan variabel, yang akan kita bahas selanjutnya.

Menangani Variabel Menggunakan *Record*

Dalam skenario yang lebih kompleks, Anda mungkin perlu melacak beberapa variabel saat membuat list. Dalam fungsi *List.Generate* Power Query, Anda dapat menggunakan record untuk mengelola kompleksitas ini. Seperti yang mungkin Anda ingat dari Bab 6, record adalah kumpulan pasangan nama-nilai, yang memudahkan pelacakan beberapa variabel secara bersamaan.

Pertimbangkan skenario saat Anda ingin membuat list angka dari 1 hingga 10 dan, selain angka itu sendiri, Anda ingin menunjukkan apakah setiap angka genap. Untuk mencapainya, Anda harus mempertahankan dua variabel kunci selama proses pembuatan list:

- Angka saat ini dalam urutan.
- Apakah angka itu genap atau tidak.

Karena variabel-variabel ini akan berubah pada setiap iterasi, kita dapat menggunakan record untuk menyimpan perubahan nilai. Mari kita lihat sebuah contoh.

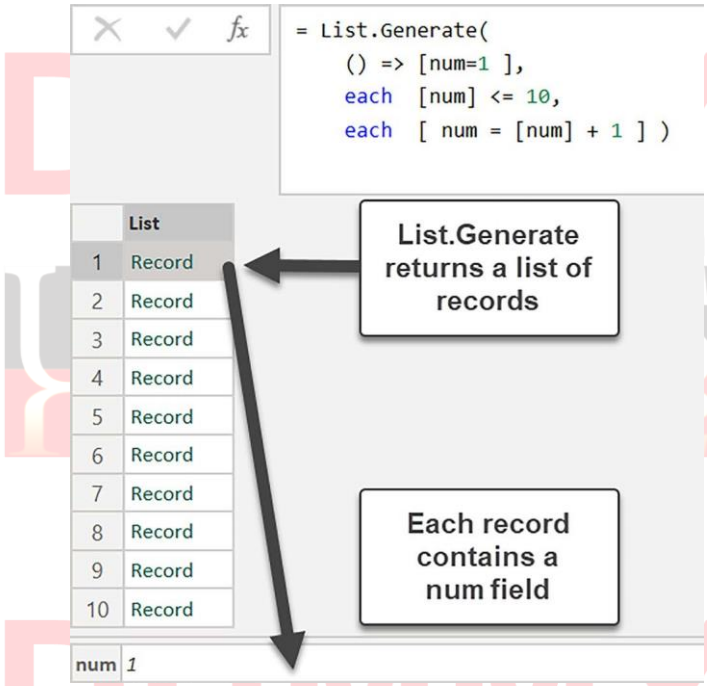
Sebelumnya, kita melihat contoh dasar yang menghasilkan list angka dari 1 hingga 10 tanpa kerumitan tambahan apa pun, yang tampak seperti berikut:

```
List.Generate(
     () => 1,       // starting value: 1
     each _ < 10,   // as long as the value is smaller than 10
     each _ + 1     // increments each value by 1
)
```

- **Nilai Awal**: Kita mulai dengan record yang berisi satu kolom bernama *num*, yang ditetapkan ke 1.

- **Kondisi**: Kondisi menerapkan pemilihan kolom ke *[num]* untuk mengambil nilainya. Kemudian, kondisi tersebut memverifikasi apakah nilai ini kurang dari atau sama dengan 10.
- ***Next***: Dalam setiap iterasi, kolom num dalam record bertambah 1.

Operasi ini mengembalikan list *record*:



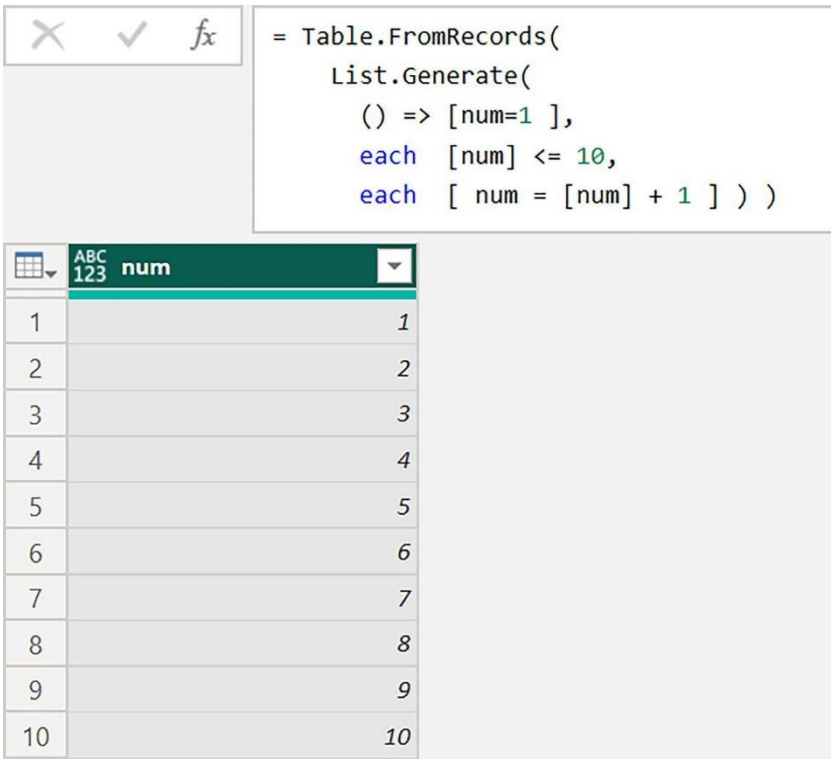
Gambar 1. 15 *List.Generate* mengembalikan list record

Seperti yang ditunjukkan gambar di atas, memeriksa detail setiap record tidak selalu mudah. Ada dua cara umum untuk melakukannya:

- Klik ruang kosong di samping setiap record untuk menampilkan apa yang ada di dalamnya.
- Gunakan argumen selector untuk mengubah record menjadi nilai yang lebih mudah diperiksa.

Namun, kita menyarankan pendekatan yang berbeda dan lebih efektif. Untuk mempermudah pemeriksaan konten record saat Anda mengerjakan kode, Anda dapat mengubah list record menjadi tabel. Anda dapat melakukannya dengan mudah menggunakan fungsi *Table.FromRecords*.

Simpan saja fungsi *List.Generate* di dalam *Table.FromRecords*, dan Anda sudah siap:



```
= Table.FromRecords(
    List.Generate(
      () => [num=1 ],
      each  [num] <= 10,
      each  [ num = [num] + 1 ] ) )
```

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

Gambar 1. 16 Menggunakan *Table.FromRecord* untuk memvisualisasikan konten record

Untuk mempelajari lebih dalam penggunaan *List.Generate*, mari sertakan variabel lain dalam record untuk mengidentifikasi apakah nilai num genap. Ini melibatkan dua langkah utama:

- Tambahkan kolom baru ke *record* yang ditentukan dalam argumen awal.
- Tentukan bagaimana kolom ini akan berubah di setiap iterasi dengan memodifikasi fungsi berikutnya.

Seperti inilah tampilannya:

```
= Table.FromRecords(
    List.Generate(
      () => [num=1,            IsEven = Number.IsEven( 1 ) ],
      each  [num] <= 10,
      each  [ num = [num] + 1, IsEven = Number.IsEven( [num] + 1 ) ] ) )
```

	num	IsEven
1	1	FALSE
2	2	TRUE
3	3	FALSE
4	4	TRUE
5	5	FALSE
6	6	TRUE
7	7	FALSE
8	8	TRUE
9	9	FALSE
10	10	TRUE

Gambar 1. 17 Menggunakan catatan untuk menentukan beberapa variabel di *List.Generate*

Dalam contoh yang diperbarui, record sekarang memiliki kolom *IsEven* tambahan. Perhatikan bahwa kondisi kita tetap sama, masih fokus pada evaluasi apakah kolom num kurang dari atau sama dengan 10. Kolom tambahan dalam record kita tidak memengaruhi hal ini, karena pemilihan kolom masih dapat mengisolasi nilai kolom num.

Keuntungan menggunakan *Table.FromRecords* adalah transparansinya; Anda dapat dengan mudah memverifikasi output. Setelah memastikan hasilnya memenuhi harapan Anda, Anda dapat menghapus fungsi *Table.FromRecords* dari kode Anda.

Setelah memverifikasi kode Anda, Anda akan sering mendapati diri Anda hanya membutuhkan satu kolom dari record output. Misalkan Anda hanya tertarik untuk mengekstrak nilai dari kolom *IsEven*. Anda kemudian dapat menggunakan pemilih sebagai berikut:

```
List.Generate(
    () => [num=1, IsEven = Number.IsEven( 1 ) ],
    each [num] <= 10,
    each [ num = [num] + 1,
           IsEven = Number.IsEven( [num] + 1 ) ],
    each [IsEven]
)
```

List.Generate Alternatives

Meskipun contoh ini valid, perlu dicatat bahwa skenario ini dapat diatasi dengan lebih mudah menggunakan *List.Transform*:

```
List.Transform(
  {1..10},
   each Number.IsEven( _ )
)
```

Fungsi *List.Transform* mengambil list sebagai input dan mengulangi setiap item dalam list dengan menerapkan fungsi tersebut pada argumen kedua. Pendekatan ini tidak hanya menyederhanakan kode tetapi juga kemungkinan besar akan berjalan lebih cepat.

Hal yang perlu diingat di sini adalah mempertimbangkan opsi Anda dengan saksama sebelum menggunakan *List.Generate*, yang kodenya dapat rumit dan relatif kompleks. Opsi seperti *List.Transform* lebih mudah dipahami dan dikelola.

Saat memutuskan fungsi mana yang akan digunakan dan Anda mengetahui jumlah iterasi sebelumnya, pertimbangkan opsi seperti:

- *List.Transform*: Lebih cocok saat setiap elemen dalam list Anda beroperasi secara independen, tanpa memerlukan logika rekursif apa pun.
- *List.Accumulate*: Berguna untuk tugas yang menyerupai operasi rekursif, karena ia membawa hasil dari satu iterasi ke iterasi berikutnya hingga semua iterasi selesai.

Apa saja skenario *List.Generate* yang Berguna?

Ada banyak kasus penggunaan yang dapat dipecahkan oleh *List.Generate*, tetapi Anda akan sering menemukan bahwa Anda dapat memperoleh hasil yang sama menggunakan *List.Accumulate* atau *List.Generate*. Beberapa skenario yang menggunakan *List.Generate* adalah sebagai berikut:

- **Pagination**: Saat menangani API yang mengembalikan hasil yang dibagi menjadi beberapa halaman, *List.Generate* dapat mengulangi setiap halaman untuk mengkompilasi kumpulan data yang lengkap. Hanya saat tidak ada lagi halaman, List.Generate berhenti mengulangi.
- **Rentang Tanggal**: Saat Anda ingin membuat list tanggal yang berada di antara dua tanggal tertentu.
- **Urutan Custom**: Saat Anda tidak terbatas pada angka atau tanggal, *List.Generate* dapat menghasilkan list yang mengikuti logika kustom yang kompleks, seperti total berjalan, urutan Fibonacci, atau bilangan prima.
- *Manipulasi Teks*: Mirip dengan masalah spasi ganda di antara kata yang akan kita pecahkan dengan rekursi di bab ini, Anda dapat menggunakan *List.Generate* untuk membuat list setiap string teks yang dimodifikasi hingga kondisi Anda terpenuhi.

Mari kita lihat beberapa contoh praktis yang menggunakan *List.Generate* untuk menghasilkan hasil yang tepat.

Melakukan Pengulangan Melalui Data API Menggunakan *List.Generate*

Fungsi *List.Generate* membedakan dirinya dari *List.Accumulate* dengan menggunakan kondisi untuk terus menghasilkan nilai. Hal ini memungkinkan fungsi untuk terus menghasilkan nilai baru hingga kondisi tidak lagi terpenuhi. Kemampuan ini sangat berharga saat Anda bekerja dengan API, yang akan kita bahas sekarang.

Jadi, apa itu API? API adalah singkatan dari **Application Programming Interface**. API pada dasarnya adalah serangkaian aturan dan protokol untuk

mengambil data dari aplikasi. Misalnya, katakanlah kita menggunakan API dari basis data perpustakaan untuk mengumpulkan semua buku yang ditulis oleh seorang penulis. Salah satu tantangannya adalah kita mungkin tidak tahu berapa banyak data yang akan dikembalikan. Di sinilah argumen kondisi *List.Generate* berguna. Anda dapat mengaturnya agar terus berjalan hingga tidak ada lagi data yang harus diambil.

Memahami API Perpustakaan Terbuka

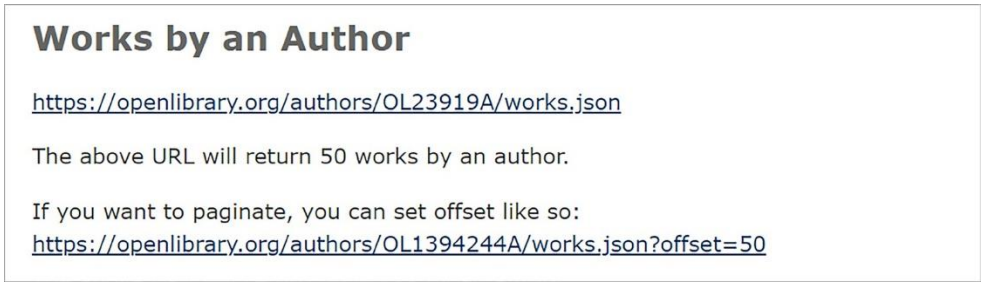
Kita akan menggunakan Open Library API sebagai contoh. Open Library berfungsi sebagai katalog hampir setiap buku yang pernah diterbitkan. Situs web ini tidak hanya memiliki halaman web yang dapat Anda telusuri, tetapi juga menawarkan API untuk menarik data secara langsung.

Sebelum memulai, penting untuk memahami cara kerja API ini. Kunjungi dokumentasi API Open Library di tautan berikut untuk mempelajari lebih lanjut: https://openlibrary.org/developers/api.

Manual daring ini menyediakan petunjuk tentang cara berinteraksi dengan API. Untuk latihan kita, kita akan fokus pada pengumpulan list lengkap karya penulis Stephen King. Untuk melakukannya, kita perlu mengetahui cara bekerja dengan **Authors API**.

Anda dapat membaca lebih lanjut tentangnya dengan membuka tautan ini: https://openlibrary.org/dev/docs/api/authors.

Halaman ini menyediakan format yang tepat yang perlu Anda gunakan saat mengirim permintaan ke basis data Open Library. Jika Anda menggulir ke bawah ke bagian Works by an Authors pada halaman tersebut, Anda akan menemukan permintaan khusus yang dapat Anda buat ke API:



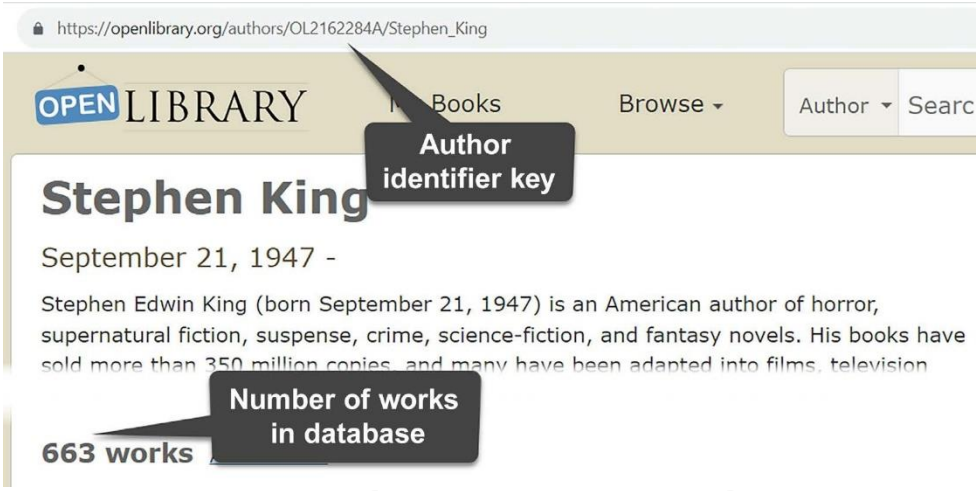
Gambar 1. 18 Instruksi untuk panggilan API

Berikut ini beberapa hal yang perlu Anda ketahui tentang panggilan API:

- **Format URL untuk Panggilan API**: Panggilan API untuk mengambil karya penulis akan terlihat seperti ini: https://openlibrary.org/authors/OL23919A/works.json.
- ***Author Identifier Key***: Untuk mengambil data untuk penulis tertentu, Anda perlu menentukan kunci pengenal mereka. Misalnya, pengenal untuk penulis di URL di atas adalah *OL23919A*.
- ***Data per API Call***: Setiap panggilan API mengambil hingga 50 karya oleh penulis.
- ***Offsetting Records***: Anda dapat menentukan pengimbang di URL untuk memilih kumpulan 50 record yang ingin Anda ambil.

Mengambil Kunci Pengenal Penulis

Untuk menemukan URL yang benar untuk karya Stephen King, kita dapat membuka https://openlibrary.org/ dan memasukkan namanya di kotak pencarian. Di hasil pencarian, klik namanya untuk membuka halaman penulis. Anda akan diarahkan ke:



Gambar 1. 19 Halaman penulis untuk Stephen King di openlibrary.org

Pada saat tulisan ini dibuat, Open Library mencantumkan 663 karya Stephen King. URL halaman pengarangnya memuat pengenalnya: OL2162284A. Berbekal informasi ini, sekarang kita dapat mulai menulis panggilan API kita.

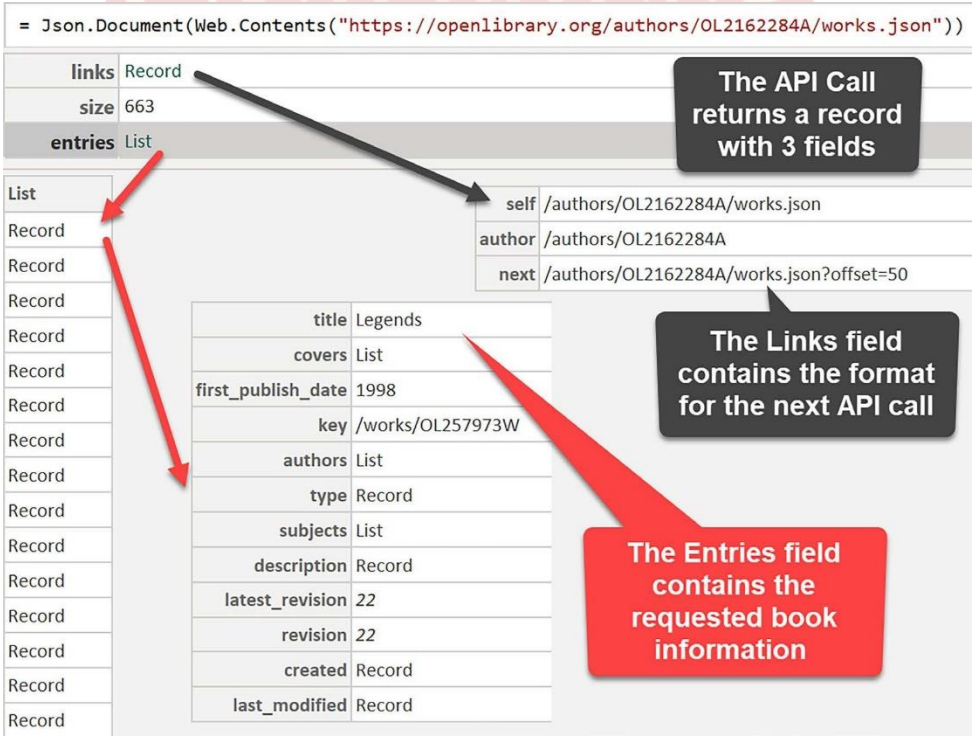
Memvalidasi Panggilan API

Sebelum menyelami logika yang lebih kompleks, pertama-tama kita harus memastikan bahwa panggilan API valid dan mengembalikan data. Jika Anda mengambil contoh URL yang disediakan dalam dokumentasi dan menyertakan pengenal kunci Stepben King, Anda akan mendapatkan URL ini: https://openlibrary.org/authors/OL2162284A/works.json. Meskipun Anda dapat mengujinya dengan menempelkannya ke browser web, kita akan memvalidasinya dalam Power Query. Untuk melakukannya:

1) Buka **Power Query**.
2) Klik **New Source**, lalu pilih **Web**.
3) Di kotak pop-up yang muncul, tempel URL dan klik **OK**:

Gambar 1. 20 Melakukan panggilan API menggunakan fungsionalitas web

Tindakan ini mengimpor data yang dikembalikan oleh API. Power Query juga dapat secara otomatis membuat langkah tambahan, seperti memperluas record atau membuat list kolom. Jika itu terjadi, hapus semua langkah yang dibuat secara otomatis ini dan simpan hanya yang disebut *Source*. Ini mengembalikan record yang berisi nilai bertingkat, seperti berikut:



Gambar 1. 21 Hasil panggilan API untuk API Penulis

Memeriksa Hasil Panggilan API

Saat Anda memilih langkah Sumber, Anda akan melihat bahwa panggilan API mengembalikan record dengan tiga bidang:

- ***links***: Ini berisi record dengan detail tautan, termasuk pengenal penulis dan URL untuk panggilan API saat ini dan berikutnya.
- ***size***: Ini memberi tahu Anda jumlah record yang tersedia untuk penulis.
- ***entries***: List ini berisi informasi aktual yang ingin Anda ambil, seperti judul buku.

Memahami bidang-bidang ini membantu Anda memutuskan cara melakukan pagination.

Menangani Keterbatasan Data

Penting untuk diperhatikan bagaimana API mengomunikasikan akhir dari data yang tersedia. Jika tidak ada lagi record yang dapat diambil, **next** URL akan hilang dari kolom *links*. Selain itu, jika Anda secara manual menjalankan panggilan API yang melompat ke offset tanpa data, kolom entri akan kembali kosong.

Kita dapat mengujinya dengan mencari panggilan API terbaru dengan data atau memberikan angka acak sebagai offset. Misalnya, jika Anda mencoba panggilan API yang bertujuan untuk melewati 999 record dan mengambil 50 record berikutnya, jika tidak ada record tersebut, list entri akan kembali kosong:

Gambar 1. 22 Panggilan API yang mengembalikan list kosong dan tidak menyertakan URL “next”

Strategi Untuk Mengambil Data

Berdasarkan kolom-kolom ini, ada beberapa strategi berbeda untuk mengambil semua record dari API. Berikut ini beberapa strategi yang mungkin berdasarkan kolom yang dikembalikan:

- Gunakan *List.Generate* untuk membuat fungsi yang menggunakan **next** URL dalam record **links** untuk setiap panggilan API berikutnya. Hentikan saat API tidak lagi mengembalikan **next** URL.
- Gunakan *List.Generate* untuk memeriksa entri dalam list **entries**. Terus lakukan panggilan API selama list ini memiliki data.
- Hitung jumlah total halaman yang diperlukan untuk semua record, dengan mempertimbangkan batas 50 item per panggilan API dan jumlah record dalam basis data (**size**). Kemudian, buat fungsi yang mengulangi halaman-halaman ini menggunakan offset yang dihitung.

Untuk tujuan tutorial ini, kita akan fokus pada dua strategi pertama yang melibatkan *List.Generate*.

Komponen Untuk Pernyataan *List.Generate*

Untuk menggunakan fungsi *List.Generate* untuk pagination, Anda perlu:

1) Membuat fungsi untuk berinteraksi dengan API.
2) Mengatur komponen URL.
3) Menentukan logika untuk iterasi melalui respons API.

Buat Fungsi Untuk Berinteraksi Dengan API

Untuk kode yang lebih bersih, sebaiknya tentukan dulu fungsi yang akan memanggil API. Saat Anda awalnya memilih **New Source**, memilih **Web**, lalu memasukkan URL API di Power Query, kode berikut akan dihasilkan:

```
Json.Document(
  Web.Contents(
    "https://openlibrary.org/authors/OL2162284A/works.json"
  )
)
```

Untuk membuat fungsi *List.Generate* lebih mudah dikelola, kita dapat mengubah kode ini menjadi fungsi mandiri. Ini memungkinkan kita untuk memasukkan URL API ke dalam fungsi tersebut saat dibutuhkan. Untuk melakukannya, buat fungsi bernama *fxGetData* menggunakan kode berikut:

```
( myURL as text ) => Json.Document(Web.Contents( myURL) )
```

Atur Komponen URL

Untuk menjalankan panggilan API secara efisien, khususnya untuk pagination, ada baiknya untuk mengkategorikan dan menyimpan berbagai elemen URL API dengan rapi. Melakukan hal ini akan mempersiapkan kueri Anda untuk panggilan API di masa mendatang.

Anda akan melihat bahwa panggilan API pertama menggunakan URL lengkap:

```
"https://openlibrary.org/authors/OL2162284A/works.json"
```

Bandingkan ini dengan parameter berikutnya yang Anda dapatkan untuk pagination:

```
"/authors/OL2162284A/works.json"
```

Parameter berikutnya tidak lengkap. Bagian depan (https://openlibrary.org) tidak ada, yang penting untuk membentuk URL yang fungsional.

Atur Komponen URL

Untuk mempermudah tugas pagination, kita akan membagi URL API lengkap menjadi dua segmen:

- *BaseURL*: Awal yang konstan, https://openlibrary.org, yang tetap konstan di antara panggilan.

- *OffsetSuffix*: Ekor variabel URL yang disesuaikan menurut data spesifik yang diambil, seperti /authors/OL2162284A/works.json untuk panggilan pertama.

Manfaat Pendekatan URL Modular

Membagi URL menjadi beberapa bagian ini memberikan fleksibilitas. Gunakan BaseURL untuk panggilan API awal, dan pasangkan dengan *OffsetSuffix* yang khusus untuk panggilan tersebut. Untuk panggilan API berikutnya, gabungkan *BaseURL* dengan fragmen berikutnya yang diterima untuk pagination.

Dengan pendekatan ini, berikut cara Anda dapat menyiapkan kueri dengan berbagai komponen URL:

```
let
 fxGetData =
  ( myURL as text ) => Json.Document(Web.Contents( myURL ) ),
 BaseURL = "https://openlibrary.org/",
 OffsetSuffix = "authors/OL2162284A/works.json"
in
 OffsetSuffix
```

Pendekatan ini menawarkan cara yang bersih dan terorganisasi untuk mengelola panggilan API, sehingga memudahkan pembuatan URL untuk panggilan awal dan selanjutnya.

Setelah memecah dan mengatur komponen URL, kini kita siap untuk melanjutkan ke tahap persiapan pernyataan pagination terstruktur.

List.Generate* Pernyataan Menggunakan URL *Offset

Sekarang setelah kita mengorganisasikan komponen URL kita ke dalam *BaseURL* dan *OffsetSuffix*, kita dapat menggunakannya untuk membangun pernyataan List.Generate yang kuat. Pernyataan ini akan menangani panggilan API kita dalam satu putaran, membuat beberapa permintaan hingga tidak ada lagi data yang tersedia untuk penulis tertentu. Fungsi *List.Generate* yang kita butuhkan terlihat seperti ini:

```
1  List.Generate(
2    () =>
3      [
4       Request = fxGetData( BaseURL & OffsetSuffix ),          Initial
5       HasNext = true
6      ],
7    each [HasNext],                                           Condition
8    each
9     [
10     Request = fxGetData( BaseURL & [Request][links][next] ), Next
11     HasNext = Record.HasFields( [Request][links], "next")
12    ],
13   each [Request]                                            Selector
14 )
```

Gambar 1. 23 Pernyataan untuk melakukan perulangan melalui API menggunakan URL offset

Fungsi ini terdiri dari empat argumen utama: *Initial value*, *Condition*, *Next argument*, dan *Selector*.

Nilai Awal (*Initial Value*)

Nilai awal adalah record yang terdiri dari dua bidang:

- Bidang Permintaan: Bidang Permintaan melakukan panggilan API pertama. Bidang ini menggabungkan *BaseURL* dan *AuthorSuffix* untuk membentuk URL API awal. Misalnya, URL mungkin terlihat seperti ini: https://openlibrary.org/authors/OL2162284A/works.json

Gambar 1. 24 URL APInya

- Bidang *HasNext*: Bidang *HasNext* menetapkan kondisi awal untuk penomoran halaman menjadi benar. Hal ini karena URL API yang valid akan selalu mengembalikan beberapa record. Jika tidak ada record yang dikembalikan, biasanya berarti penulis tidak ada dalam basis data.

Kondisi (*Condition*)

Di sini, kita cukup memeriksa kolom *HasNext* untuk memutuskan apakah kita perlu membuat panggilan API lainnya. Jika *HasNext* bernilai *true*, fungsi akan membuat permintaan lainnya.

Argumen Berikutnya

Argumen berikutnya menyiapkan fungsi untuk panggilan API berikutnya. Argumen ini terdiri dari dua kolom:

- Memperbarui *Request*: Kolom ini membentuk URL untuk panggilan API berikutnya. Kolom ini mengambil BaseURL dan menambahkan tautan berikutnya dari panggilan API terakhir.
- Memperbarui *HasNext*: Kolom ini memeriksa apakah panggilan API terakhir memiliki tautan berikutnya. Hal ini dicatat dalam

```
[ Request = fxGetData( BaseURL & [Request][links][next] ) ]
```

Jika memiliki *url* berikutnya, maka akan mengembalikan *true*. Jika tidak, maka akan mengembalikan *false*, yang mengakhiri loop. Kode sebelumnya menelusuri beberapa catatan dengan menggunakan operasi pemilihan bidang sebanyak tiga kali. Untuk meringkas cara kerjanya, lihat Bab 6, Nilai Terstruktur.

Pemilih (*Selector*)

Bagian terakhir dari fungsi ini adalah *Selector*. Ini menentukan apa yang harus dikembalikan oleh fungsi tersebut. Dalam kasus ini, fungsi hanya akan mengembalikan kolom *Request* karena di situlah data API yang relevan disimpan. Jika kita menghilangkan argumen ini, kita akan mendapatkan list record sebagai hasilnya.

Pengujian Pernyataan *List.Generate* Untuk API Kosong

Pendekatan yang kita ambil di atas memanfaatkan komponen *next* untuk URL yang menentukan offset. Namun, itu bukan satu-satunya pendekatan

yang dapat kita ambil. Atau, kita dapat menguji apakah panggilan API berisi data. Kita dapat menggabungkan logika itu dengan cara berikut:

```
1  List.Generate(
2    () =>
3      [
4        Counter = 0,                                          Initial
5        Request = fxGetData( 0 )
6      ],
7    each not List.IsEmpty( [Request][entries] ),            Condition
8    each
9      [
10       Counter = [Counter] + 50,                           Next
11       Request = fxGetData( [Counter] + 50 ) ],
12   each [Request]                                          Selector
13 )
```

Gambar 1. 25 Pernyataan untuk melakukan perulangan melalui API menggunakan offset manual

Initial Value

Fungsi ini dimulai dengan record awal yang memiliki dua kolom:

- *Counter*: Kolom ini dimulai dari nol dan akan bertambah 50 untuk setiap panggilan API berikutnya. Ini berfungsi sebagai offset untuk URL permintaan API.
- *Request*: Kolom ini berisi data yang dikembalikan oleh panggilan API. Fungsi ini menggunakan fungsi *fxGetData* untuk mengambil data yang dimulai dari offset nol.

Condition

Argumen *Condition* memutuskan kapan harus menghentikan panggilan API selanjutnya. Argumen ini berfokus pada kolom Permintaan, yang berisi list nilai API yang diambil. Fungsi akan berlanjut jika list ini tidak kosong, yang kita periksa menggunakan:

```
each not List.IsEmpty( [Request][entries] )
```

Jika listnya kosong, perulangan berhenti, yang menunjukkan bahwa semua data telah dikumpulkan.

Next Argument

Bagian ini memperbarui record untuk iterasi berikutnya:

- *Counter*: Menambah counter sebesar 50, yang akan berfungsi sebagai offset untuk panggilan API berikutnya.
- *Request*: Mengambil set data berikutnya berdasarkan offset baru.

Selector

Pemilih *each [Request]* menentukan apa yang akan dikembalikan oleh fungsi tersebut. Kita hanya tertarik pada kolom Request karena kolom tersebut berisi data API yang kita butuhkan.

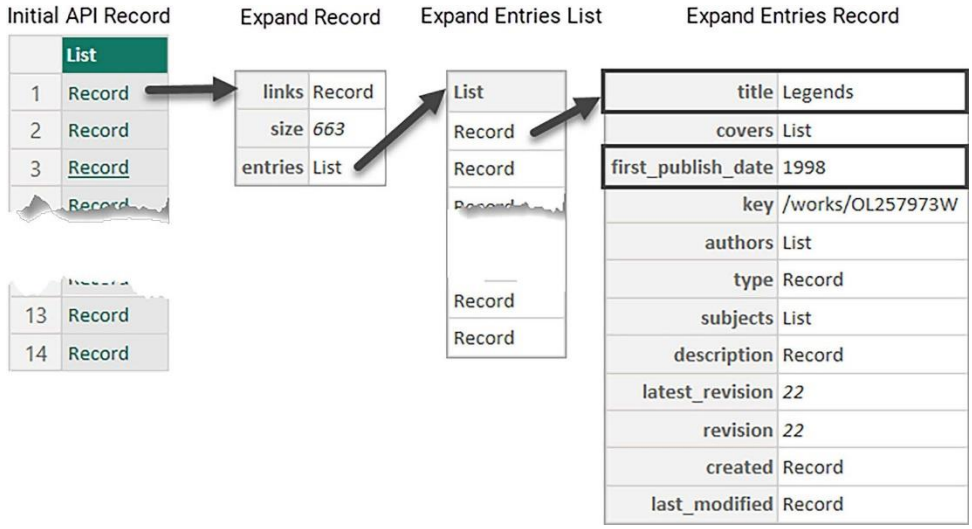
Dibandingkan dengan pendekatan yang menggunakan komponen URL berikutnya, pendekatan offset menyederhanakan banyak hal. Tidak perlu melacak apakah tautan berikutnya ada atau tidak. Yang perlu kita khawatirkan hanyalah apakah ada lebih banyak entri yang akan diambil, yang dapat kita periksa dengan mudah menggunakan *List.IsEmpty*.

Mengubah Catatan Menjadi Keluaran yang Diinginkan

Output dari kedua fungsi *List.Generate* adalah list record:

Gambar 1. 26 List record sebagai output dari *List.Generate*

Misalkan kita ingin mengembalikan list judul buku dan tanggal rilisnya. Kita harus menyelidiki lebih dalam nilai record untuk mengambil informasi tersebut. Saat Anda menelusuri berbagai list dan record dari panggilan API, Anda dapat menemukan informasi berikut:

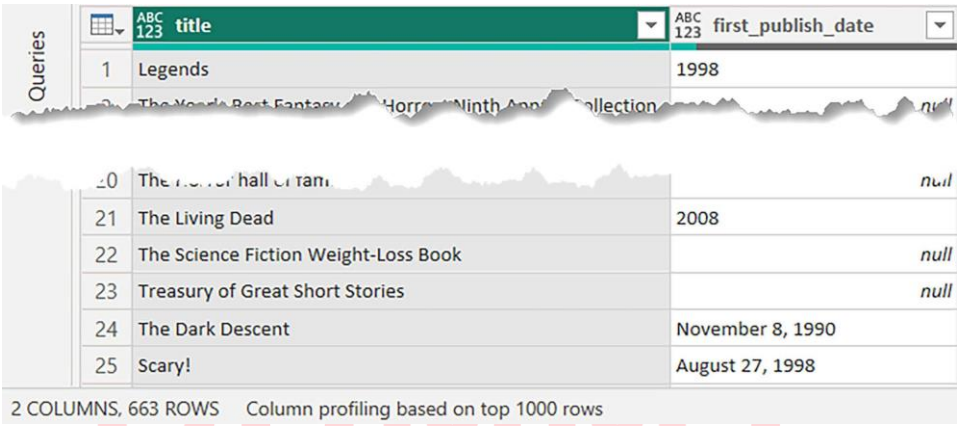


Gambar 1. 27 Objek yang dikembalikan oleh panggilan API

Memiliki list record bukanlah format keluaran yang kita inginkan. Sebaliknya, akan lebih masuk akal untuk melihat informasi sebenarnya dalam record. Itu berarti kita harus mengubah list record. Dari pengaturan saat ini, cara termudah untuk mengembalikan informasi yang diinginkan dalam format yang dapat dibaca adalah:

1) Buka tab **Transform** dan pilih **To Table**.
2) Klik dua panah berlawanan di tajuk kolom Column1 dan perluas bidang **entries**.
3) Klik dua panah berlawanan di tajuk kolom entri dan pilih Expand to New Rows
4) Ulangi tindakan sebelumnya dan, kali ini, pilih bidang **title** dan **first_publish_date**.

Anda akan mendapatkan tabel berikut:



Queries

	title	first_publish_date
1	Legends	1998
20	The [illegible] hall of fam[illegible]	null
21	The Living Dead	2008
22	The Science Fiction Weight-Loss Book	null
23	Treasury of Great Short Stories	null
24	The Dark Descent	November 8, 1990
25	Scary!	August 27, 1998

2 COLUMNS, 663 ROWS Column profiling based on top 1000 rows

Gambar 1. 28 Tabel dari API dengan semua karya Stephen King

Output dari operasi ini adalah tabel dengan kolom *title* dan *first_publish_date*. Tabel ini mudah dibaca, dan Anda dapat menggabungkannya ke dalam kumpulan data Anda dengan cara yang mudah. Itulah ilustrasi pertama *List.Generate*. Seperti yang telah Anda lihat, menyiapkan koneksi API bisa terasa rumit. Anda harus memahami cara mengakses berbagai objek di M dan mempelajari cara melakukan pengulangan melalui objek-objek tersebut. Namun, setelah Anda merasa nyaman dengan ini, ini adalah teknik yang sangat ampuh untuk perangkat Anda.

Selanjutnya, kita memiliki kasus penggunaan umum lainnya yang memanfaatkan *List.Generate*; kita akan melihat pembuatan total berjalan.

Menciptakan Total Berjalan yang Efisien

Skenario umum bagi analis data adalah membuat total berjalan. Meskipun lebih mudah dan sering kali lebih masuk akal untuk membuatnya di DAX, ada skenario di mana total berjalan di Power Query masuk akal.

Total berjalan adalah jumlah kumulatif dari serangkaian nilai. Contoh di mana Anda melihat total berjalan digunakan adalah dalam pelaporan keuangan. Penjualan aktual sering kali dibandingkan dengan penjualan anggaran menggunakan perhitungan tahun-ke-tahun.

Power Query tidak memiliki fungsi khusus untuk menjalankan total. Sebaliknya, orang sering menggunakan fungsi seperti *List.Range* atau *List.FirstN*. Namun, metode ini bisa jadi tidak efisien. Metode ini menghitung total dengan menjumlahkan rentang angka yang terus bertambah, artinya angka yang sama dimasukkan dalam operasi penjumlahan berulang kali. Hal ini dapat memperlambat perhitungan.

Pendekatan yang lebih efisien adalah menggunakan fungsi dengan kapabilitas seperti rekursif, seperti *List.Generate* atau *List.Accumulate*. Fungsi ini melakukan perhitungan, menyimpan hasilnya dalam list, lalu melanjutkan ke angka berikutnya. Setiap angka baru ditambahkan ke total yang disimpan sebelumnya. Dengan cara ini, fungsi tidak perlu menghitung ulang seluruh rentang setiap kali, dengan peningkatan kinerja yang signifikan sebagai hasilnya.

Misalkan Anda memiliki tabel dengan penjualan sepanjang tahun, seperti yang ditunjukkan pada gambar berikut:

	Period	Sales
1	Jan 2024	900
2	Feb 2024	850
3	Mar 2024	925
4	Apr 2024	875
5	May 2024	910
6	Jun 2024	725
7	Jul 2024	750
8	Aug 2024	740
9	Sep 2024	900
10	Oct 2024	925

Gambar 1. 29 Data penjualan

Untuk mengikuti, mulailah dengan membuka berkas latihan yang menyertai bab ini.

Jika Anda ingin menambahkan kolom yang menampilkan penjualan tahun berjalan, menggunakan *List.Generate* dapat menjadi solusinya. Namun, membuat list nilai total berjalan hanyalah sebagian dari pekerjaan. Anda juga perlu mengintegrasikan nilai-nilai ini ke dalam tabel yang sudah ada. Berikut cara memecah tugas:

1) Dapatkan list nilai yang akan digunakan untuk menghitung total berjalan.
2) Hitung nilai total berjalan menggunakan *List.Generate*.
3) Pisahkan tabel yang sudah ada menjadi beberapa list terpisah, yang masing-masing mewakili satu kolom.
4) Pulihkan tabel dengan menggabungkan semua list.

Mari kita lihat cara kerjanya dalam praktik.

Dapatkan List Nilai

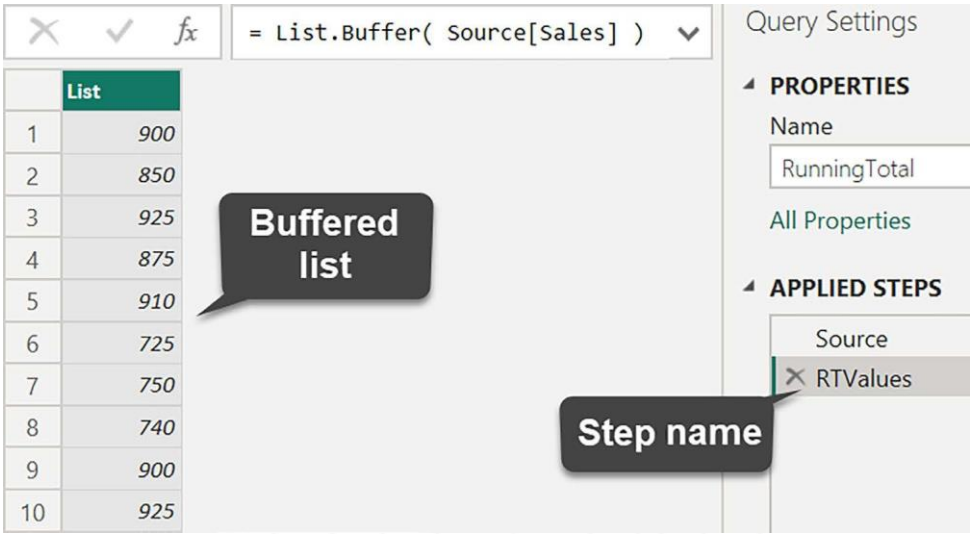
Untuk memulai, kita memiliki data dari gambar di atas yang disimpan dalam langkah bernama *Source*. Kita akan menyimpan nilai yang ingin kita gunakan untuk rumus *List.Generate* dalam langkah terpisah. Untuk melakukannya, buat langkah baru dengan mengeklik simbol **fx** di samping bilah rumus. Ini membuat langkah baru yang merujuk ke Source. Ubah nama langkah ini menjadi *RTValues*.

Selanjutnya, kita merujuk ke kolom Penjualan menggunakan pemilihan bidang. Rumus untuk melakukannya adalah:

```
Source[Sales]
```

Untuk total berjalan, kita akan merujuk list ini berulang kali. Untuk memastikan kinerja yang baik, sebaiknya Anda menyimpan list nilai ke dalam memori menggunakan *List.Buffer*. Hasilnya adalah:

```
List.Buffer( Source[Sales] )
```



Gambar 1. 30 List nilai untuk total berjalan

Itu memberi kita list nilai yang menjadi dasar untuk total berjalan kita.

Hitung Nilai Total Berjalan dengan *List.Generate*

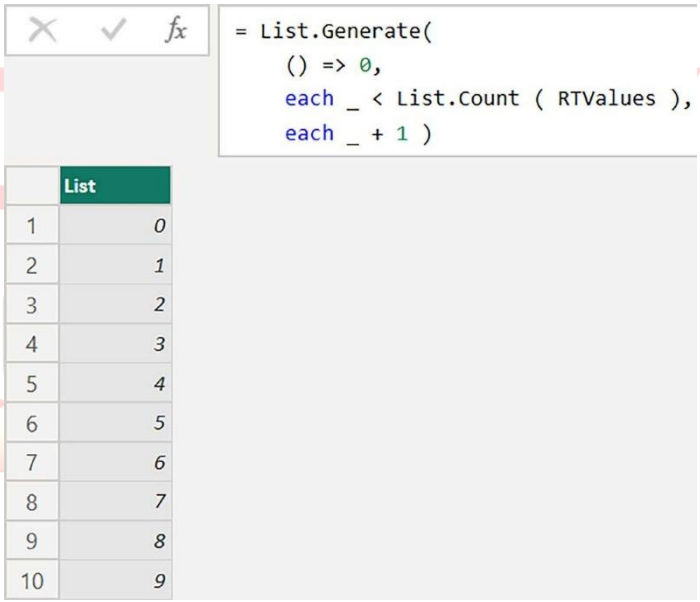
Sekarang saatnya membuat ekspresi *List.Generate* yang mengembalikan total berjalan dari nilai-nilai ini. Sebelumnya dalam bab ini, Anda mempelajari bahwa *List.Generate* menggunakan kondisi untuk mengevaluasi kapan harus berhenti menghitung nilai. Setelah kita berakhir dalam situasi yang diilustrasikan dalam gambar di atas, berikut ini yang dapat Anda lakukan:

- Buat langkah baru di panel **Applied Steps** yang merujuk ke langkah RTValues.
- Buat rumus *List.Generate* yang menggunakan *RTValues* untuk total berjalan.

Sekarang, mari buat fungsi *List.Generate*. Fungsi ini menggunakan list *RTValues* untuk menghitung item dalam total berjalan. Kita akan memasukkan penghitung dalam *List.Generate*. Penghitung ini bertambah pada setiap langkah. Penghitungan nilai total berjalan berhenti ketika penghitung sama dengan jumlah total nilai dalam list. Untuk menyiapkan penghitung ini, Anda dapat menulis:

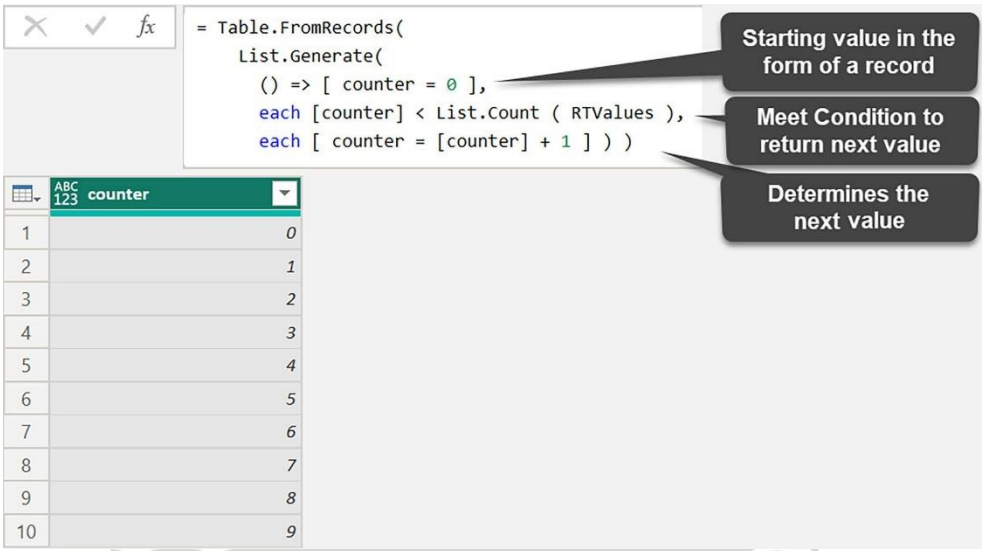
```
List.Generate(
    () => 0,
    each _ < List.Count ( RTValues ),
    each _ + 1
)
```

Hasilnya berupa list dengan angka 0 hingga 9:



Gambar 1. 31 List yang bertambah yang berfungsi sebagai penghitung untuk *List.Generate*

Karena kita memerlukan penghitung dan nilai total yang berjalan, kita akan menggunakan struktur record yang memungkinkan kita menyimpan beberapa variabel. Untuk memeriksa data, kita juga akan melampirkan rumus dengan *Tabel.FromRecords*. Kode ekuivalen yang menggunakan record terlihat seperti berikut:



Gambar 1. 32 *List.Generate* fungsi menggunakan record untuk variabel

Perhatikan bahwa ekspresi baru ini tidak akan berfungsi hanya dengan merujuk nilai dengan memberikan skor rendah. Ini hanya mungkin dalam versi awal karena ada satu nilai untuk setiap iterasi. Karena sekarang kita ingin memilih nilai dari record, kita juga harus menentukan bidang mana yang kita rujuk.

Untuk menghitung total berjalan, Anda perlu memperkenalkan variabel tambahan. Variabel ini akan ada baik dalam record awal maupun dalam fungsi yang bertanggung jawab untuk memperbarui nilai. Berikut kode yang dimodifikasi:

```
    Table.FromRecords(
      List.Generate(
        () => [ counter = 0,

              RT     = RTValues{0} ],
     each [counter] < List.Count ( RTValues ),
     each [ counter = [counter] + 1,
              RT      = [RT] + RTValues{[counter] + 1} ]

   )
 )
```

Dalam kode ini:

- Kolom baru bernama RT telah ditambahkan ke record awal. Nilai *RTValues{0}* menetapkan total berjalan awal menggunakan elemen pertama dari list *RTValues*.
- Pada argumen ketiga, kolom tambahan untuk total berjalan (*RT*) juga disertakan. Ini menghitung total berjalan dengan menambahkan nilai berikutnya ke jumlah yang ada.

Untuk menguraikan logikanya:

- *counter* adalah variabel yang bertambah satu setiap kali fungsi tersebut beriterasi.
- *RT* mewakili total berjalan.

Dengan menggunakan *counter*, Anda dapat mengambil elemen yang sesuai dari list *RTValues* untuk memperbarui total berjalan. Misalnya, saat counter bernilai 0, kode tersebut mengambil *RTValues{0}*; saat bernilai 1, kode tersebut mengambil *RTValues{1}*, dan seterusnya. Dengan cara ini, Anda dapat menghitung total berjalan secara efisien. Ini adalah hasil dari kode sebelumnya:

	counter	RT
1	0	900
2	1	1750
3	2	2675
4	3	3550
5	4	4460
6	5	5185
7	6	5935
8	7	6675
9	8	7575
10	9	8500

Gambar 1. 33 Buat total berjalan menggunakan *List.Generate*

Output dari Gambar 1.33 adalah persis apa yang kita cari. Ada dua perubahan yang tersisa untuk dilakukan:

1) Kode yang kita gunakan masih menggunakan *Table.FromRecords* untuk memvisualisasikan konten setiap record. Kita sekarang dapat menghapusnya karena output kita sudah benar.
2) Setelah melakukan langkah 1, kita akan mendapatkan list record. Kita hanya memerlukan kolom total berjalan dari record tersebut, bukan mengembalikan record itu sendiri. Kita dapat menggunakan pemilih untuk memilih kolom tersebut.

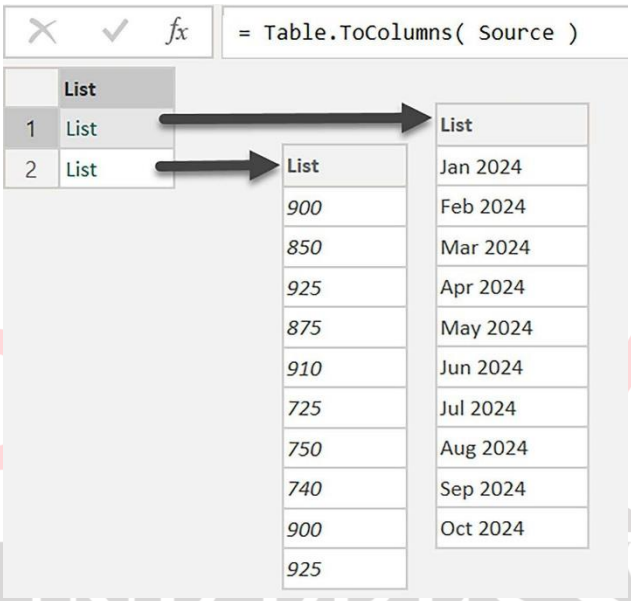
Kode berikut memberi kita list yang hanya berisi total berjalan:

```
List.Generate(
      () => [ counter = 0, RT = RTValues{0} ],
      each [counter] < List.Count ( RTValues ),
      each [ counter = [counter] + 1,
             RT      =  [RT] + RTValues{[counter] + 1} ],
      each [RT]
)
```

Selanjutnya, kita perlu menemukan cara untuk melampirkan nilai total berjalan ke list kita. Cara yang efektif untuk melakukannya adalah dengan membagi tabel asli menjadi beberapa list, di mana setiap list mewakili satu kolom. Kita kemudian menggabungkan list ini dengan list total berjalan dan memulihkan tabel.

Memisahkan Tabel yang ada Menjadi *List* Terpisah

Langkah pertama adalah mengubah tabel yang ditunjukkan pada Gambar 13.34 menjadi list individual, dengan setiap list mewakili satu kolom. Untuk melakukannya, buat langkah baru yang disebut *TableColumns* dan terapkan fungsi *Table.ToColumns* pada langkah *Source*:



Gambar 1. 34 Tabel penjualan diubah menjadi list dengan dua list nilai kolom

Pulihkan Tabel dengan Menggabungkan Semua List

Langkah berikutnya adalah menyusun kembali tabel dengan menggabungkan semua list yang terpisah. Kandidat yang tepat untuk operasi ini adalah fungsi Table.FromColumns.

Fungsi Table.FromColumns menerima dua argumen untuk membuat tabel:

- *Lists*: List yang menyertakan list lain, yang masing-masing berisi nilai untuk kolom tertentu.
- *Columns*: List yang menyertakan nama kolom.

Untuk menggabungkan nilai kolom menjadi satu list bersarang, Anda dapat menggunakan ekspresi berikut:

```
TableColumns & { RunningTotal }
```

Ini menggabungkan list nilai kolom yang terpisah menjadi satu list terpadu:

	List
1	List
2	List
3	List

Gambar 1. 35 List nilai kolom

Untuk membuat tabel menggunakan list gabungan ini, Anda dapat menggunakan fungsi *Table.FromColumns* seperti ini:

```
Table.FromColumns( TableColumns & { RunningTotal } )
```

	Column1	Column2	Column3
1	Jan 2024	900	900
2	Feb 2024	850	1750
3	Mar 2024	925	2675
4	Apr 2024	875	3550
5	May 2024	910	4460
6	Jun 2024	725	5185
7	Jul 2024	750	5935
8	Aug 2024	740	6675
9	Sep 2024	900	7575
10	Oct 2024	925	8500

Gambar 1. 36 Tabel dengan nama kolom dan tipe yang hilang

Ekspresi ini akan mengembalikan tabel yang mencakup semua kolom asli, beserta kolom *RunningTotal* baru. Tabel tersebut hampir lengkap, tetapi tidak memiliki dua elemen penting:

- Tipe data untuk kolom RunningTotal.
- Nama kolom yang tepat.

Anda memiliki dua opsi untuk menetapkan tipe data untuk kolom *RunningTotal*. Yang pertama adalah menambahkan langkah terpisah yang menentukan tipe kolom. Opsi kedua adalah langsung menetapkan tipe data ke kolom *RunningTotal* menggunakan fungsi *Value.ReplaceType*. Berikut cara melakukannya:

```
Value.ReplaceType( RunningTotal, type {Int64.Type} )
```

Tipe data di atas hanya berlaku untuk bilangan bulat. Jika nilai Anda memiliki desimal, Anda dapat mengganti *{Int64.Type}* dengan *{number}*.

Untuk menentukan nama kolom saat menggunakan fungsi *Table.FromColumns*, Anda dapat menyertakannya sebagai list dalam argumen kedua. Untuk mengambil nama kolom dari tabel asli, Anda dapat menggunakan *Table.ColumnNames*. Selain itu, Anda perlu menambahkan *RunningTotal* sebagai nama kolom baru. Ekspresi di bawah ini membuat list nama kolom:

```
Table.ColumnNames( Source ) & {"Running Total"}
```

Anda dapat menggabungkan langkah-langkah ini menjadi satu ekspresi dan hasilnya adalah:

```
Table.FromColumns(
  TableColumns &

    {Value.ReplaceType( RunningTotal, type { Int64.Type } )},
  Table.ColumnNames( Source ) & {"Running Total"} )
```

Pernyataan tunggal ini akan membuat tabel dengan kolom *Running Total*, diketik dan diberi nama dengan benar, beserta kolom asli:

	Period	Sales	Running Total
1	Jan 2024	900	900
2	Feb 2024	850	1750
3	Mar 2024	925	2675
4	Apr 2024	875	3550
5	May 2024	910	4460
6	Jun 2024	725	5185
7	Jul 2024	750	5935
8	Aug 2024	740	6675
9	Sep 2024	900	7575
10	Oct 2024	925	8500

Gambar 1. 37 Tabel keluaran termasuk kolom Total Berjalan

Contoh total berjalan adalah ilustrasi yang bagus tentang bagaimana Anda dapat menerapkan logika secara berulang hingga suatu kondisi gagal. Namun,

perlu diingat bahwa kita dapat memecahkan skenario yang sama ini menggunakan *List.Accumulate*. Kedua pendekatan tersebut bekerja secara efektif. Terserah Anda untuk memutuskan fungsi mana yang Anda sukai.

Sejauh ini, kita telah melihat iterator seperti *List.Transform* dan *List.Accumulate*. Keduanya adalah iterator hebat yang bekerja untuk sejumlah iterasi yang tetap. Bab ini juga memperkenalkan *List.Generate*, yang memungkinkan Anda melakukan operasi hingga kondisi tertentu tidak lagi terpenuhi. Topik yang tersisa untuk bab ini adalah rekursi. Anda akan mempelajari apa itu rekursi dan cara terbaik untuk menggunakannya.

C. Rekursi

Rekursi adalah istilah pemrograman yang kedengarannya lebih rumit daripada yang sebenarnya. Sederhananya, rekursi terjadi ketika suatu fungsi memanggil dirinya sendiri dalam definisinya sendiri. Anggap saja sebagai suatu perulangan, tetapi alih-alih menggunakan perulangan for atau while yang umum, fungsi tersebut menggunakan dirinya sendiri untuk melakukan suatu operasi beberapa kali.

Rekursi berguna untuk mengatasi masalah ketika Anda perlu mengulang suatu tugas tetapi jumlah pengulangannya tidak diketahui sebelumnya. Operasi ini memerlukan kondisi akhir sehingga iterasi berhenti ketika mencapai kondisi tertentu, mencegah pengulangan tak terbatas.

Konsep rekursi juga telah dibandingkan dengan konsep boneka Rusia. Bayangkan membuka boneka Rusia dan menemukan boneka lain di dalamnya, dan boneka lain di dalamnya, dan seterusnya. Proses ini berlanjut hingga Anda mencapai boneka terkecil yang tidak dapat dibuka lebih jauh. Setiap langkah seperti fungsi yang memanggil dirinya sendiri, yang pada akhirnya mencapai titik di mana ia tidak dapat lagi melakukannya.

1. Mengapa Rekursi Penting?

Meskipun bahasa M sudah menawarkan serangkaian fungsi bawaan yang lengkap, beberapa operasi memerlukan fleksibilitas ekstra dan elemen rekursif. Misalnya, Anda mungkin memiliki data hierarkis—seperti bagan organisasi—yang perlu dibongkar. Atau, Anda mungkin memiliki objek bertingkat yang ingin diratakan.

Harus diakui, untuk sebagian besar situasi yang memerlukan elemen rekursif, kita sarankan untuk menggunakan *List.Generate*. Ia berkinerja lebih baik daripada rekursi tradisional dan lebih mudah dipecahkan masalahnya. Akan tetapi, buku ini tidak akan lengkap tanpa mempelajari rekursi tradisional.

Jadi, kapan kita menganggap rekursif dapat diterima? Anda mungkin menghadapi situasi di mana fungsi rekursif lebih mudah ditulis dan jumlah iterasinya relatif rendah. Karena kinerja tidak akan menjadi masalah di sini, penggunaan rekursi sepenuhnya baik-baik saja. Dalam semua situasi lain, kita tetap pada rekomendasi kita, menggunakan *List.Generate* sebagai gantinya.

2. Rekursi Versus Iterasi: Perbandingan Singkat

Sebelum membahasnya lebih lanjut, ada baiknya untuk membedakan rekursi dari padanannya yang sangat erat kaitannya: iterasi. Baik rekursi maupun iterasi melakukan tugas berulang, tetapi keduanya melakukannya dengan cara yang sangat berbeda.

Dalam iterasi, serangkaian instruksi (loop) dieksekusi berulang kali hingga mencapai akhir input. Misalkan Anda memiliki list nilai untuk diubah. Fungsi *List.Transform* dapat menerapkan operasi pada setiap item dalam list, yang dianggap sebagai iterasi. Panjang list diketahui sebelumnya, dan jumlah iterasi dapat diprediksi.

Di sisi lain, rekursi menangkap tugas berulang dalam fungsi itu sendiri. Ia berulang kali menjalankan ekspresi yang menyertakan suatu kondisi hingga kondisi itu tidak lagi terpenuhi. Hal ini sering kali menghasilkan kode yang lebih bersih dan lebih elegan, tetapi dengan mengorbankan penggunaan

memori yang lebih besar dan biasanya kinerja yang lebih lambat. Jadi, bagaimana rekursi bekerja dalam praktiknya?

3. Fungsi Rekursif

Saat menulis fungsi rekursif, konsep cakupan penting. Konsep cakupan mengacu pada lingkungan tempat variabel atau fungsi dapat diakses. Secara umum, cakupan membantu Anda merujuk variabel yang didefinisikan dalam langkah lain atau kueri lain. Namun, untuk fungsi rekursif, kita ingin membuat referensi ke fungsi itu sendiri.

Dengan mengingat hal itu, mari kita perkenalkan simbol @, yang dikenal sebagai operator cakupan atau referensi pengenal inklusif.

Apa Itu Operator @ Scoping?

Dalam bahasa M, simbol @ dirancang untuk mengatasi tantangan saat mengakses variabel. @ memungkinkan suatu fungsi merujuk ke dirinya sendiri atau variabel lain dalam definisinya sendiri.

Variabel dalam M biasanya bukan bagian dari lingkungan ekspresi tempat variabel tersebut didefinisikan. Itu berarti referensi diri umumnya tidak memungkinkan. Di sinilah simbol @ berperan. Simbol ini memungkinkan variabel menjadi bagian dari lingkungannya, sehingga mendukung pemanggilan fungsi rekursif.

Referensi Pengenal Inklusif

@ secara formal dikenal sebagai referensi-pengidentifikasi-inklusif, istilah yang mengacu pada konsep cakupan dalam pemrograman. Istilah ini terdiri dari tiga kata:

- **Inclusive**: Simbol @ menyertakan pengidentifikasi (misalnya, variabel atau fungsi) dalam cakupannya sendiri.
- **Identifier**: Ini adalah variabel atau fungsi yang ingin Anda akses.
- **Reference**: Anda menunjuk ke pengidentifikasi tertentu dalam kode.

@ adalah kebalikan dari apa yang dikenal sebagai referensi pengidentifikasi eksklusif. Pengidentifikasi eksklusif tidak dapat menjadi bagian dari lingkungannya sendiri, yang menyebabkan kesalahan jika Anda mencoba merujuk diri sendiri.

Menggunakan Fungsi Rekursif

Beberapa tantangan mengharuskan Anda untuk memeriksa kondisi dan melakukan operasi secara berulang. Dalam situasi ini, fungsi rekursif menunjukkan nilainya.

Subjek umum yang cocok untuk rekursi adalah perhitungan bilangan faktorial. Bilangan faktorial adalah hasil perkalian semua bilangan bulat positif yang kurang dari atau sama dengan bilangan bulat positif tertentu.

Untuk menghitung faktorial 5, yang direpresentasikan sebagai 5!, Anda mengalikan 5 x 4 x 3 x 2 x 1, sehingga hasilnya adalah 120. Untuk menuliskannya di Power Query, Anda dapat menulis:

```
let
  Factorial =
    ( n ) =>
      if n <= 1
       then 1
      else n * @Factorial ( n - 1 ) // '@' is used here
in
  Factorial ( 5 )
```

Dalam kode ini, simbol @ memungkinkan fungsi Factorial memanggil dirinya sendiri, sehingga fungsi tersebut bersifat rekursif. Pemanggilan diri ini terlihat jelas pada bagian yang bertuliskan *@Factorial*. Ketika pernyataan if bernilai *false*, kode tersebut mengalikan angka n saat ini dengan angka n dikurangi 1. Kemudian, fungsi tersebut memulai kembali, kali ini menggunakan n-1 sebagai nilai awalnya.

Proses berlanjut hingga pernyataan if bernilai benar. Pada titik tersebut, fungsi tersebut berhenti berjalan karena tidak ada referensi diri. Jika Anda menghilangkan simbol @ dan hanya menggunakan Factorial, Power Query

akan menghasilkan kesalahan. Hal ini terjadi karena fungsi tersebut tidak dapat mengenali dirinya sendiri tanpa @.

Sekarang kita akan membahas lebih dalam tentang cara memasukkan operator @ ke dalam kode Anda.

4. Cara Menggunakan Operator @

Setelah melihat operator @ yang digunakan dalam sebuah contoh, Anda mungkin bertanya-tanya bagaimana cara menggabungkannya ke dalam kode Power Query M Anda. Berikut adalah empat langkah untuk menggabungkan operator scope dalam fungsi kustom Anda:

Langkah 1: Tulis Kode Awal Anda

Mulailah dengan menulis fungsi Anda dengan cara biasa, tanpa simbol @. Misalnya, jika Anda membuat fungsi *Factorial*, kode awal Anda mungkin terlihat seperti ini:

```
Factorial =
  (n) => if n <= 1 then 1 else n * (n - 1)
```

Langkah pertama ini membuat fungsi dengan mengambil nilai input n dan melakukan operasi satu kali. Fungsi tersebut memeriksa apakah n bernilai satu atau kurang. Jika lebih besar, fungsi tersebut mengalikan n dengan *n - 1*.

Sebelum berpikir untuk menyertakan rekursi, penting untuk menyertakan kondisi if. Kondisi ini berfungsi sebagai tindakan pengamanan untuk mencegah fungsi memasuki loop tak terbatas. Pastikan juga untuk memberi nama pada fungsi Anda, karena Anda akan membutuhkannya nanti untuk panggilan rekursif.

Langkah 2: Identifikasi Panggilan Rekursif

Selanjutnya, identifikasi di mana fungsi Anda harus memanggil dirinya sendiri. Dalam contoh Faktorial kita, panggilan rekursif melibatkan n * (n - 1).

Dalam setiap iterasi, fungsi mengalikan nilai saat ini dengan dirinya sendiri - 1.

Langkah 3: Tambahkan Operator @

Sekarang saatnya untuk memasukkan simbol @ ke dalam kode Anda. Ingat kembali bahwa kita telah memberi nama fungsi kita sebelumnya. Anda akan menggunakan nama itu di sini.

Ubah fungsi Anda untuk menyertakan simbol @ dan nama fungsi di bagian tempat fungsi tersebut memanggil dirinya sendiri. Fungsi Faktorial yang telah Anda revisi akan terlihat seperti ini:

```
Factorial =
  (n) => if n <= 1 then 1 else n * @Factorial(n - 1)
```

Fungsi akan terus memanggil dirinya sendiri selama kondisi *if* bernilai *false*. Ketika ini terjadi, kode akan mencapai bagian @Factorial dan kemudian mulai dari awal menggunakan *n-1* sebagai input.

Langkah 4: Uji Fungsi Anda

Jalankan fungsi Anda untuk memastikannya berfungsi sebagaimana mestinya. Jalankan melalui berbagai skenario untuk memastikan operator @ menjalankan rekursi sebagaimana diharapkan.

Dengan mengikuti keempat langkah ini, Anda siap menggunakan operator @ dalam fungsi kustom Anda. Anda tidak hanya akan mengetahui cara mengimplementasikan rekursi, tetapi juga melindunginya dari pengulangan yang tak berujung. Siap untuk contoh lainnya?

5. Menghapus Spasi Berturut-Turut

Mari kita telusuri contoh dunia nyata untuk lebih memahami rekursi. Misalkan Anda memiliki kumpulan data dengan kolom yang diisi dengan kalimat, tetapi spasi antarkata tidak konsisten. Tujuan Anda adalah merapikannya sehingga tidak ada lebih dari satu spasi antarkata.

Misalnya, Anda mungkin memiliki kalimat seperti *Hard "work beats luck"*, di mana spasi antarkata bervariasi. Anda dapat menyelesaikannya dengan menggunakan rumus seperti *Text.Replace*:

```
Text.Replace( MyString, "  ", " " )
```

Ini mengganti spasi ganda dengan spasi tunggal. Melakukan ini sekali akan menghasilkan *"Hard work beats luck"*. Meskipun sudah ditingkatkan, tiga kata pertama masih memiliki spasi tambahan—tiga dan dua, tepatnya. Anda perlu menjalankan operasi ini dua kali lagi untuk membersihkan spasi sepenuhnya.

Namun, karena setiap kalimat dalam kumpulan data Anda mungkin memiliki jumlah spasi yang berbeda-beda di antara kata-kata, Anda perlu mengulangi tindakan ini sesuai kebutuhan. Untuk menghindari perhitungan yang berlebihan, penting untuk menyertakan kondisi pengujian yang memeriksa spasi berurutan yang tersisa dalam string. Berikut cara melakukannya:

```
let
  MyString = "Hard    work   beats  luck",
  Replace =  if Text.Contains( MyString, "  " )
               then Text.Replace( MyString, "  ", " " )
               else MyString
in
  Replace
```

Dengan cara ini, Anda hanya menerapkan fungsi *Text.Replace* jika kondisinya terpenuhi, sehingga menghemat sumber daya komputasi.

Sekarang kita memiliki operasi penggantian dan kondisi untuk mengidentifikasi spasi berurutan. Langkah berikutnya adalah membuat cara untuk mengulang operasi ini, dan di situlah rekursi berguna. Untuk mengimplementasikan rekursi, kita perlu menyimpan logika kita dalam suatu fungsi. Ini penting karena simbol @ beroperasi dengan memanggil suatu fungsi berulang kali. Berikut cara mengubah kode menjadi suatu fungsi:

```
( MyString as text ) as text =>
  let
    Replace =
      if Text.Contains ( MyString, "  " )
         then Text.Replace ( MyString, "  ", " " )
         else MyString
  in
    Replace
```

Dalam fungsi ini, kita mendefinisikan parameter bernama *MyString* yang mengambil string teks sebagai input. Kita kemudian menggunakan parameter ini dalam logika fungsi kita.

Versi fungsi kita saat ini hanya dapat mengganti satu spasi ganda sekaligus. Untuk mengaktifkan pemeriksaan dan penggantian yang sedang berlangsung, kita dapat menggabungkan rekursi menggunakan operator @. Berikut kode fungsi yang ditingkatkan:

```
let
  fxDeSpace = ( MyString as text ) as text =>
    let
      Replace =
        if Text.Contains ( MyString, "  " ) then
          @fxDeSpace ( Text.Replace ( MyString, "  ", " " ) )
        else
          MyString
    in
      Replace
in
  fxDeSpace
```

Untuk mendapatkan kode ini, kita melakukan langkah-langkah berikut:

- Menetapkan nama *fxDeSpace* ke fungsi kita.
- Di dalam fungsi, kita menambahkan referensi ke namanya sendiri, diawali dengan operator @ *(@ fxDeSpace*).

Detail penting di sini adalah membungkus *Text.Replace* dengan *@fxDeSpace*. Yang terjadi adalah, jika fungsi *Text.Contains* mengonfirmasi keberadaan spasi berurutan, fungsi tersebut akan mengganti spasi ganda dengan spasi tunggal lalu memanggil dirinya sendiri lagi dari awal. Siklus ini

berlanjut hingga tidak ada lagi spasi ganda yang ditemukan. Pada titik ini, fungsi tersebut berhenti memanggil dirinya sendiri.

6. Pertimbangan Kinerja Menggunakan Rekursi

Rekursi dapat menjadi cara yang elegan untuk memecahkan masalah yang memiliki struktur rekursif, seperti menghasilkan faktorial atau mengakses data hierarkis. Namun, rekursi bukannya tanpa kekurangan, terutama dalam hal kinerja.

Saat Anda menggunakan rekursi, setiap panggilan rekursif membuat lapisan baru dalam tumpukan panggilan—penyimpanan memori sementara tempat program Anda melacak tugasnya. Jika fungsi Anda memanggil dirinya sendiri terlalu sering, tumpukan dapat meluap, menyebabkan kueri Anda gagal.

Saat terjadi luapan tumpukan, Anda akan menemukan pesan kesalahan yang menunjukkan ukuran tumpukan telah melampaui batasnya. Ini terjadi saat fungsi rekursif Anda tidak memiliki kondisi keluar yang tepat, atau kedalaman rekursi terlalu tinggi.

> Untuk mencegah hal ini, pastikan untuk menyertakan logika dalam fungsi rekursif Anda untuk memastikannya akan berakhir. Selain itu, perhatikan kedalaman rekursi. Jika Anda mendapatkan kesalahan batas tumpukan, itu pertanda Anda harus mempertimbangkan pendekatan alternatif.

Rekursi sering kali bukan pilihan pertama kita untuk melakukan perhitungan. Ada metode yang lebih efisien yang sering kali berkaitan dengan *List.Generate* atau *List.Accumulate*, yang telah kita bahas sebelumnya dalam bab ini.

D. Ringkasan

Dalam bab ini, kita mengeksplorasi dua konsep penting: iterasi dan rekursi. Dimulai dengan iterasi, kita meneliti peran fungsi *List.Transform* dan *List.Accumulate*. Keduanya menjalankan tugasnya pada list dengan jumlah item

yang tetap, seperti *for loo*p. Sementara *List.Transform* berfungsi untuk operasi langsung, *List.Accumulate* juga berguna untuk skenario yang memerlukan akses ke hasil iterasi sebelumnya.

Kita kemudian mempelajari fungsi *List.Generate*, yang memungkinkan Anda membuat list secara dinamis berdasarkan kondisi dan logika khusus, mirip dengan while loop. Fungsi ini memiliki kinerja yang baik, menyediakan cara mudah untuk men-debug hasilnya, dan merupakan pilihan yang lebih disukai daripada rekursi tradisional dalam sebagian besar kasus penggunaan.

Terakhir, kita membahas konsep rekursi sejati, yang difasilitasi dalam M melalui operator @ scoping. Meskipun rekursi memerlukan biaya dalam hal memori dan kecepatan, Anda dapat mempertimbangkan untuk menggunakannya untuk tugas yang melibatkan sejumlah iterasi terbatas dan di mana penulisan fungsi rekursif menyederhanakan kode.

Sepanjang bab ini, kita menekankan pentingnya memilih metode yang tepat—iterasi atau rekursi—berdasarkan persyaratan khusus tugas transformasi data Anda. Buat ekspresi Anda sesederhana mungkin, tetapi bila diperlukan, jangan takut untuk menggunakan teknik rumit yang dijelaskan dalam bab ini. Teknik-teknik tersebut dapat meningkatkan kemampuan manipulasi data Anda secara signifikan di Power Query M.

E. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan perbedaan mendasar antara fungsi List.Transform dan List.Accumulate dalam Power Query. Kapan sebaiknya masing-masing digunakan?	10
2.	Apa itu List.Generate dalam Power Query M, dan bagaimana perbedaannya dengan fungsi iterasi lainnya? Berikan satu contoh kasus penggunaannya.	10
3.	Jelaskan konsep "rekursi" dalam Power Query dan peran operator cakupan @ dalam implementasinya.	10

4.	Deskripsikan sintaks dasar dari fungsi List.Accumulate. Jelaskan fungsi dari masing-masing parameter dalam sintaks tersebut.	10
5.	Bagaimana cara kerja fungsi List.Accumulate untuk mengganti beberapa nilai dalam sebuah tabel? Jelaskan secara bertahap.	10
6.	Apa kelebihan menggunakan pendekatan List.Generate untuk membuat total berjalan dibandingkan metode konvensional lainnya di Power Query?	10
7.	Mengapa penting untuk menyimpan nilai lama dan nilai baru dalam bentuk tabel untuk digunakan dalam fungsi iterasi? Jelaskan keuntungannya dari sisi fleksibilitas.	10
8.	Apa perbedaan antara hasil iterasi List.Transform dan List.Accumulate dalam konteks penyimpanan hasil setiap langkah? Berikan ilustrasi sederhana.	10
9.	Bagaimana fungsi Table.ToRows mendukung penggunaan List.Accumulate saat melakukan penggantian nilai dinamis dalam tabel?	10
10.	Analisis situasi di mana lebih baik menggunakan rekursi daripada iterasi dalam Power Query M. Apa yang perlu dipertimbangkan saat membuat fungsi rekursif?	10

BAB 2

Pola Data yang Bermasalah

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Mengekstraksi pola tetap
- Mengekstrak dan menggabungkan data

A. Pendahuluan

Membentuk ulang dan menyiapkan data untuk analisis sering kali merupakan kombinasi seni dan sains, memadukan teori dan keterampilan dengan pemecahan masalah yang kreatif. Ada berbagai macam pola data yang merepotkan, mulai dari menangani data bertumpuk hingga tajuk multibaris dan seterusnya. Tantangan tidak terbatas pada struktur data saja, tetapi sering kali juga mencakup ketidakkonsistenan dan kerumitan lain yang dapat muncul dari sifat unik data itu sendiri atau aturan dan persyaratan bisnis tertentu.

Ketika kita berpikir tentang pemecahan masalah, penting untuk diingat bahwa ada banyak pendekatan dan teknik yang berbeda. Dalam buku ini, kita tidak mengklaim memiliki semua jawaban atau bahwa metode ini adalah yang terbaik. Sebaliknya, kita bertujuan untuk menawarkan kumpulan ide dan memicu imajinasi Anda untuk membantu Anda memulai perjalanan pemecahan masalah, membuat Anda percaya diri dalam menghadapi pola data merepotkan apa pun yang melintasi jalan Anda.

Contoh-contoh yang diberikan di seluruh bab ini merupakan titik awal. Saat Anda membaca dan memahaminya, harapan kita bukan hanya untuk menggabungkan teori dengan pengetahuan praktis, tetapi juga untuk menginspirasi Anda agar memandang pemecahan masalah sebagai proses kreatif karena, pada akhirnya, solusi terbaik adalah solusi yang Anda kembangkan untuk tantangan data unik Anda sendiri.

Ada banyak sekali skenario yang cocok untuk dibahas dalam bab ini; namun, tidak ada cukup halaman untuk membahas semuanya. Oleh karena itu, kita memilih untuk membahas dua contoh secara mendalam, dan topik-topik tersebut adalah:

- Mengekstrak pola tetap
- Mengekstrak dan menggabungkan data

Mengingat adanya diskusi berulang di forum daring, jelas bahwa diskusi tersebut mencerminkan tantangan umum yang dihadapi pengguna di dunia nyata.

Kita akan mengalihkan perhatian kita ke pencocokan pola terlebih dahulu, memulainya dengan mengeksplorasi aspek-aspek yang dapat berperan dalam mengidentifikasi dan mengekstraksi pola yang bermakna dari data.

1. **Kasus Pemantik Berfikir Kritis: Penanganan Kesalahan dalam Power Query**

Sebuah perusahaan retail menggunakan Power Query untuk menganalisis data penjualan mereka. Tim analisis data menemukan beberapa masalah saat memuat data dari berbagai sumber, termasuk file Excel dan database.

Masalah yang Ditemui:

1) Kesalahan Tipe Data: Beberapa nilai di kolom "Jumlah Terjual" yang seharusnya angka, ternyata terdapat teks yang tidak valid.
2) Kesalahan Nilai Kosong: Kolom "Total Pendapatan" memiliki beberapa entri yang kosong dan menyebabkan kesalahan saat melakukan perhitungan.
3) Kesalahan Koneksi: Terkadang, kueri gagal terhubung ke database karena masalah jaringan, yang mengakibatkan kesalahan saat memuat data.

Tugas:

1) Identifikasi: Apa saja jenis kesalahan yang mungkin muncul dari masalah di atas? Bagaimana kesalahan tersebut dapat memengaruhi analisis data?
2) Strategi Penanganan: Diskusikan bagaimana Anda akan menggunakan fungsi try dan otherwise untuk menangani kesalahan yang mungkin terjadi. Berikan contoh sederhana bagaimana Anda akan menerapkan ini dalam kueri.
3) Pencegahan: Apa langkah-langkah yang dapat diambil untuk mencegah kesalahan serupa di masa depan? Pertimbangkan penggunaan validasi data dan pengaturan tipe data yang tepat.

B. Pencocokan Pola

Pencocokan pola merupakan keterampilan penting yang sering dikaitkan dengan ekspresi reguler (atau regex), yang penting untuk mengidentifikasi dan memanipulasi pola tertentu dalam data tekstual. Meskipun bahasa M Power Query tidak memiliki dukungan langsung untuk regex, bahasa ini menyediakan banyak fungsi yang dapat membantu mengatasi jenis persyaratan ini. Di bagian ini, kita mengeksplorasi pencocokan pola dalam batasan Power Query, menyoroti beberapa teknik dan strategi utama untuk mengatasi tantangan ini.

1. Dasar-Dasar Pencocokan Pola

Pencocokan pola sangat berguna untuk menemukan dan mengekstrak urutan karakter tertentu dari data. Proses tersebut sangat bergantung pada penggunaan fungsi teks. Bahasa M dilengkapi dengan berbagai fungsi yang beroperasi pada nilai teks. Fungsi-fungsi ini dapat memfasilitasi aspek-aspek tertentu yang relevan dengan pencocokan pola. Fungsi-fungsi ini menawarkan berbagai kemampuan, dari operasi string dasar hingga yang lebih canggih.

Di sini, kita akan membahas berbagai pertimbangan dan fungsi yang relevan dengan pencocokan pola. Meskipun ini bukanlah list yang lengkap atau lengkap, list ini akan memberi Anda gambaran tentang potensi dan fleksibilitas fungsi M dalam menangani skenario pencocokan pola.

Sensitivitas Huruf Besar/Kecil

Dalam bahasa M Power Query, kepekaan huruf besar/kecil mengacu pada kemampuan untuk membedakan antara huruf besar dan huruf kecil sebagai karakter yang terpisah. Mengenali aspek M ini sangat penting saat menangani data tekstual, karena fungsi yang melibatkan perbandingan atau penggantian teks semuanya dipengaruhi oleh kepekaan tersebut saat mencocokkan karakter. Namun, ada fungsi teks yang dapat melakukan operasi yang tidak peka huruf besar/kecil; artinya saat Anda membandingkan dua string, perbandingan tidak akan mempertimbangkan huruf besar/kecil. Misalnya,

Teks akan sama dengan teks dalam perbandingan peka huruf besar/kecil. Fungsi-fungsi ini dilengkapi dengan pembanding opsional.

Dalam situasi di mana suatu fungsi tidak memiliki pembanding, memodifikasi huruf besar/kecil teks untuk mencapai keseragaman dapat menjadi strategi yang baik. Menggunakan *Text.Lower*, *Text.Upper*, atau *Text.Proper* untuk menstandardisasi huruf besar/kecil dapat menghindarinya. Sebagai ilustrasi:

Text.Lower("Power Query") will return power query.

Text.Upper("Power Query") will return POWER QUERY.

Text.Proper("power query") will return Power Query.

Ini adalah metode sederhana untuk menstandardisasi teks sebelum diproses lebih lanjut.

Berisi Versus Pencocokan Tepat

Saat mencari Power Query dalam string Power Query is *awesome!*, kita dapat menentukan string kedua berisi string pertama tetapi tidak ada kecocokan persis. Persyaratan untuk pencocokan pola dapat bervariasi – terkadang Anda mungkin perlu mengonfirmasi keberadaan substring, skenario tipe berisi, sementara di waktu lain kecocokan persis sangat penting. Fungsi seperti *Text.Start*, *Text.End*, *Text.Middle*, dan *Text.Range* dapat mengembalikan substring untuk dicocokkan. Fungsi-fungsi tersebut juga membantu dalam mengidentifikasi pola tertentu.

Sementara *Text.Contains* menawarkan metode yang mudah untuk memeriksa apakah substring ada dalam string, *Text.PositionOf* dapat digunakan untuk menemukan posisi substring dalam string. Jika mengembalikan angka yang lebih besar dari atau sama dengan 0, substring tersebut ada. Jika mengembalikan -1, substring tersebut tidak ada.

Karakter yang Diperbolehkan

Menentukan karakter mana yang relevan dengan pola Anda sangat penting, seperti huruf besar atau kecil, angka, pilihan karakter khusus, atau kombinasi dari semuanya. Bahasa M mencakup fungsi yang dapat secara selektif menyimpan atau menghapus karakter, seperti *Text.Select* atau *Text.Remove*, misalnya. Sebagai ilustrasi:

Text.Select("Power Query", {"o", "e"}) will return oee.

Text.Remove("Power Query", {"o", "e"}) will return Pwr Qury.

Menangani Satu atau Lebih Elemen

Apakah Anda menangani satu hal, seperti satu kata, atau beberapa hal, seperti string atau kalimat? Jika beberapa, apakah itu perlu dipecah menjadi unit yang lebih kecil? Saat menangani lebih dari satu item, Anda akan menggunakan operasi list, seperti *List.Select*, *List.Transform*, dan lainnya. Mengembalikan satu nilai sering kali melibatkan *Text.Combine*. Fungsi itu mengambil list nilai teks dan menggabungkan atau menggabungkannya menjadi satu string; secara opsional, menyertakan pemisah di antara setiap item list.

Karakter Pengganti (Wildcards)

Karakter pengganti adalah simbol yang digunakan untuk mewakili satu atau beberapa karakter dalam string teks. Karakter pengganti sering digunakan dalam operasi pencarian dan pencocokan pola. Meskipun Power Query tidak mendukung karakter pengganti secara asli, bukan berarti karakter pengganti tidak dapat ditiru dalam M. Fungsi pustaka standar seperti *Text.StartsWith* atau *Text.EndsWith*, misalnya, memudahkan pencarian string yang dimulai atau diakhiri dengan teks tertentu. Namun, perilaku karakter pengganti lainnya sering kali lebih rumit dan memerlukan logika khusus untuk direplikasi.

Sekarang setelah kita memperkenalkan aspek-aspek pencocokan pola, mari selami pemeriksaan beberapa contoh spesifik. Kita akan menunjukkan

bagaimana pencocokan pola dapat digunakan dalam berbagai konteks. Melalui contoh-contoh ini, termasuk aplikasi praktis dan nuansa pencocokan pola, kita berharap dapat meningkatkan kemampuan Anda untuk mengenali dan menerapkan pendekatan yang solid terhadap pencocokan pola dalam alur kerja Anda sendiri.

2. Mengekstraksi Pola Tetap

Mengekstrak pola tetap dari string teks merupakan tugas yang dianggap menantang oleh banyak pengguna. Tugas ini sering kali melibatkan ekstraksi berbagai jenis kode ID – seperti ID sistem, SKU, nomor dokumen, nomor pelacakan, atau kode penagihan – dari string teks yang lebih besar. Tantangannya terletak pada kenyataan bahwa tidak ada dua skenario yang persis sama. Untuk membantu Anda memahami topik ini, kita akan membahas beberapa contoh untuk mengatasi berbagai jenis persyaratan ini, memastikan Anda memiliki pengetahuan untuk menangani berbagai pola tetap dengan sukses, seperti kode awalan atau kode yang mengikuti pola, seperti empat huruf diikuti oleh lima angka dan seterusnya. Setiap contoh dilengkapi dengan kumpulan datanya sendiri yang dapat Anda gunakan untuk mengikutinya.

Contoh 1, *prefixed*

Mari kita lihat contoh praktisnya. Tugas kita adalah mengekstrak kode-kode tertentu. Kode-kode yang kita minati memiliki pola yang berbeda: masing-masing dimulai dengan huruf *DGPQ*, diikuti oleh serangkaian lima angka. Ini berarti kode-kode kita akan selalu dimulai dengan *DGPQ* dan diikuti oleh serangkaian angka, seperti *DGPQ12345*. Penting untuk dicatat bahwa semua kode secara konsisten mengikuti pola ini dan hanya akan ada satu kode yang terkandung dalam suatu string.

Pendekatan kita melibatkan identifikasi dan ekstraksi kode-kode ini dari dalam string teks yang lebih besar. Kita perlu memindai setiap teks, mengenali awalan *DGPQ*, dan mengekstrak kode tersebut:

```
let
    Source = Table.FromColumns(
        {{"The code DGPQ33446 is important for this task.",
        "Unique document identifier: DGPQ13295.",
        "Please use the code DGPQ36006 to access the system.",
        "For verification, enter DGPQ30881 on the website.",
        "The transaction ID DGPQ78388 must be noted.",
        "DGPQ10273 is the code for your appointment.",
        "Reference code DGPQ36144 is included in the report.",
        "To complete registration, use DGPQ90158 as your code.",
        "Package with tracking number DGPQ52287 has been shipped."}},
        type table [String=text]
    )
in
    Source
```

Mari kita uraikan kode untuk kumpulan data contoh ini, dengan fokus pada setiap bagiannya:

1) *let*

Klausul let adalah bagian pertama dari ekspresi *let*, sedangkan klausa in adalah bagian terakhir. Ekspresi *let* memungkinkan pembuatan satu atau beberapa variabel yang masing-masing memiliki ekspresi yang ditetapkan padanya dan dipisahkan dengan koma.

2) *Source*

Ini adalah langkah atau nama variabel dalam kueri Anda. Diikuti oleh tanda sama dengan dan ditetapkan ekspresi untuk menyimpan nilai.

3) Table.FromColumns

Fungsi ini digunakan untuk membuat tabel dari list.

a) {{"String1", "String2", "String3"}}

Argumen pertama adalah list yang berisi list nilai kolom untuk setiap kolom yang akan dibuat. Contoh ini menunjukkan satu list dalam list tersebut.

b) type table [String=text]

Argumen kedua opsional, yang telah ditetapkan tipe tabel yang menentukan nama kolom dan menetapkan tipe kolom untuk setiap kolom dalam tabel.

4) *in*

Bagian akhir dari ekspresi *let* adalah klausa in. Apa pun yang ditentukan setelahnya akan menghasilkan hasil untuk ekspresi let tersebut – dalam hal ini, nilai yang disimpan dalam variabel Source.

Ikuti langkah-langkah berikut untuk menambahkan kolom **Code** ke tabel contoh. Proses ini memerlukan beberapa fungsi. Merupakan praktik umum untuk menumpuk fungsi-fungsi ini, bekerja dari dalam ke luar. Namun, menumpuk ekspresi record memungkinkan Anda untuk meninjau hasil dari semua langkah perantara dengan mudah, yang sangat bagus untuk mengoptimalkan dan memecahkan masalah kode:

1) Navigasi ke **Add Column** dan pilih **Custom Column**.
2) Masukkan Kode sebagai nama kolom baru.
3) [ ]

Inisialisasi ekspresi record dengan memasukkan [ ], serangkaian tanda kurung siku. Di dalam record, list nama bidang dan nilai bidang yang dibatasi koma dapat dibuat.

4) *n = Text.PositionOf([String], "DGPQ", Occurrence.First)*

Untuk menentukan posisi awalan DGPQ, fungsi *Text.PositionOf* dapat digunakan. Jika mengembalikan nomor indeks posisi lebih besar dari atau sama dengan 0, substring ada; jika tidak, jika substring tidak ditemukan, mengembalikan -1.

5) *result = Text.Range([String], n, 9)*

Masukkan koma untuk membuat bidang baru di dalam record. Dengan posisi tersebut, kode lengkap dapat diperoleh dengan mengambil rentang karakter yang dimulai dari indeks posisi tersebut, n, dan mengekstrak 4+5 =9 karakter dari teks.

6) [*result*]

Terakhir, untuk mengembalikan nilai hasil, terapkan akses bidang dengan menempatkan kursor Anda setelah tanda kurung siku penutup dari inisialisasi record, dan dalam serangkaian tanda kurung siku baru, masukkan **field name** tempat memperoleh nilai dan klik **OK**.

Meskipun diformat di sini untuk kejelasan, dalam bilah rumus, kode M akan terlihat seperti ini:

```
Table.AddColumn(Source, "Code", each
    [
        n = Text.PositionOf([String], "DGPQ", Occurrence.First),
        result = Text.Range( [String], n, 9 )
    ][result]
)
```

Atau, bila Anda lebih suka ekspresi bersarang, seperti ini:

```
Table.AddColumn(Source, "Code", each
    Text.Range( [String],
        Text.PositionOf([String], "DGPQ", Occurrence.First
        ), 9
    )
)
```

Gambar 2.1 menunjukkan hasil pendekatan ini.

	String	Code
1	The code DGPQ33446 is important for this task.	DGPQ33446
2	Unique document identifier: DGPQ13295.	DGPQ13295
3	Please use the code DGPQ36006 to access the system.	DGPQ36006
4	For verification, enter DGPQ30881 on the website.	DGPQ30881
5	The transaction ID DGPQ78388 must be noted.	DGPQ78388
6	DGPQ10273 is the code for your appointment.	DGPQ10273
7	Reference code DGPQ36144 is included in the report.	DGPQ36144
8	To complete registration, use DGPQ90158 as your code.	DGPQ90158
9	Package with tracking number DGPQ52287 has been shipped.	DGPQ52287

Gambar 2. 1 Hasil contoh 1

Sempurna. Sekarang, sebaiknya Anda meninjau pendekatan Anda dan mengidentifikasi batasan serta area yang perlu ditingkatkan. Untuk latihan ini, mari kita lanjutkan dengan asumsi bahwa kemungkinan perubahan pada pola

atau persyaratan kita tidak ada. Dengan mengingat hal itu, perhatian utama yang muncul adalah kurangnya penanganan kesalahan. Penanganan kesalahan sangat penting untuk skenario di mana substring DGPQ, tidak ditemukan dalam teks. Dalam contoh seperti itu, fungsi *Text.PositionOf* mengembalikan -1. Jika tidak dikelola, ini akan menimbulkan kesalahan Argumen *'offset' argument is out of range*. dalam fungsi *Text.Range*.

Faktanya, menggabungkan pernyataan kondisional akan menjadi metode yang sangat efektif untuk mengelola jenis skenario ini. Kode M yang ditingkatkan dapat terlihat seperti ini:

```
Table.AddColumn(Source, "Code", each
    let n = Text.PositionOf([String], "DGPQ", Occurrence.First) in
    if n > -1 then Text.Range( [String], n, 9 ) else ""
)
```

1) *let n = Text.PositionOf([String], "DGPQ", Occurrence.First) in*

   Menyusun ekspresi *let* memungkinkan kita untuk menetapkan hasil dari suatu ekspresi ke suatu variabel. Dalam contoh ini, kita membuat variabel bernama n, yang menampung hasil dari ekspresi *Text.PositionOf*. Sekarang kita dapat merujuk ke hasil dari ekspresi tersebut beberapa kali dengan merujuk ke n, tanpa perlu mengulang rumus ini dalam kode.

2) *if n > -1 then Text.Range([String], n, 9) else ""*

   Ekspresi *let* selalu diakhiri dengan klausa in. Setelah klausa in, Anda dapat mengembalikan variabel atau menulis ekspresi lain. Di sini, kita telah memilih pernyataan kondisional yang melakukan uji logika. Ketika ini menghasilkan *true*, ekspresi hasil dieksekusi, tetapi ketika menghasilkan *false*, klausa *else* berlaku, dan string teks kosong dikembalikan. Ini bertindak sebagai pengaman, memverifikasi keberadaan substring sebelum mengeksekusi fungsi *Text.Range*. Ini penting untuk mencegah kesalahan dan memastikan kode berfungsi sebagaimana mestinya dalam skenario di mana substring mungkin hilang dalam string input.

Untuk mencapai hasil yang sama menggunakan ekspresi record, Anda akan menetapkan pernyataan kondisional ke kolom baru dalam record, seperti ini, sebelum mengembalikan nilai kolomnya:

```
Table.AddColumn(Source, "Code", each
    [
        n = Text.PositionOf([String], "DGPQ", Occurrence.First),
        r = if n > -1 then Text.Range( [String], n, 9 ) else ""
    ][r]
)
```

Dalam contoh ini saja, kita telah mengilustrasikan fungsi bersarang, menggunakan ekspresi record dan let. Kita membahas pro, kontra, dan pertimbangan metode ini di Bab 8, Bekerja dengan Struktur Bersarang, tetapi pada akhirnya setiap orang mengembangkan gaya pengodean mereka sendiri. Dalam hal mendorong adopsi, pemecahan masalah, dan kejelasan, kita mempromosikan penggunaan ekspresi record.

Contoh 2, *Pattern*

Tidak seperti skenario sebelumnya, di mana semua kode memiliki awalan tetap, kode dalam contoh ini dimulai dengan empat huruf kapital, diikuti oleh serangkaian lima digit. Ini berarti kode target kita dapat dimulai dengan kombinasi huruf kapital empat huruf, seperti *ABCD12345* atau *EFGH67890*. Perhatikan bahwa pola empat huruf diikuti oleh lima angka ini konsisten di seluruh kumpulan data.

Strategi kita dalam kasus ini adalah membagi string menjadi serangkaian karakter, alias kata-kata, dan memindainya, mengidentifikasi dan mengekstrak setiap kode yang memenuhi kriteria ini. Berikut adalah kumpulan data contoh; berisi 6 kode yang valid:

```
let
    Source = Table.FromColumns(
        {{"The code CHLO33446 is important for this task.",
        "Unique document identifier: JXKE13295.",
        "Please use the code SBT36006 to access the system.",
        "For verification, enter ERKZ30881 on the website.",
        "The transaction ID YNZ78388 must be noted.",
        "IHCY10273 is the code for your appointment.",
        "Reference code SSK36144 is included in the report.",
        "To complete registration, use PRLL90158 as your code.",
        "Package with tracking number MGAK52287 has been shipped."}},
        type table [String=text]
    )
in
    Source
```

Ikuti langkah-langkah berikut untuk menambahkan kolom **Code** ke tabel contoh ini. Proses tersebut memerlukan beberapa fungsi. Kita akan memanfaatkan ekspresi record:

1) Navigasi ke **Add Column** dan pilih **Custom Column**.
2) Masukkan *Code* sebagai nama kolom baru.
3) [ ]

   Inisialisasi ekspresi record dengan memasukkan [ ], serangkaian tanda kurung siku.

4) *w = Text.Select([String], {"A".."Z", "0".."9", " "})*

   Karakter yang valid adalah huruf kapital dan angka. Mari sertakan spasi. Spasi tersebut dapat digunakan untuk membagi setiap string menjadi unit atau kata yang lebih kecil. Dengan menggunakan fungsi *Text.Select*, kita hanya dapat menyimpan karakter tersebut dari string input. Tambahkan koma di akhir untuk membuat kolom baru dalam record ini.

5) *s = Text.Split(w, " ")*

   Pisahkan setiap string pada karakter spasi, untuk mengembalikan list dengan nilai teks, dan tambahkan koma di akhir untuk membuat kolom baru dalam record ini.

6) *r = List.Select(s, each Text.Length(_) =9)*

Untuk percobaan awal, kita akan memilih elemen list apa pun dengan panjang total sembilan karakter, karena itu tampaknya merupakan titik awal yang baik. Tambahkan koma di akhir untuk membuat bidang baru dalam record ini.

7) *a = List.First(r)*

Setelah memeriksa hasil antara, ini memberikan hasil yang diharapkan. Kita dapat mengekstrak jawaban dengan fungsi seperti *List.First*.

8) *[a]*

Akses bidang a untuk mengembalikan nilainya, setelah tanda kurung tutup ekspresi record.

Meskipun diformat di sini untuk kejelasan, dalam bilah rumus kode M akan terlihat seperti ini:

```
Table.AddColumn(Source,
    "Code", each [
        w = Text.Select([String], {"A".."Z", "0".."9", " "}),
        s = Text.Split( w, " "),
        r = List.Select( s, each Text.Length(_) =9),
        a = List.First( r )
    ][a]
)
```

Gambar 2.2 Menunjukkan keluaran pendekatan ini.

	String	Code
1	The code CHLO33446 is important for this task.	CHLO33446
2	Unique document identifier: JXKE13295.	JXKE13295
3	Please use the code SBT36006 to access the system.	null
4	For verification, enter ERKZ30881 on the website.	ERKZ30881
5	The transaction ID YNZ78388 must be noted.	null
6	IHCY10273 is the code for your appointment.	IHCY10273
7	Reference code SSK36144 is included in the report.	null
8	To complete registration, use PRLL90158 as your code.	PRLL90158
9	Package with tracking number MGAK52287 has been shipped.	MGAK52287

Gambar 2. 2 Hasil contoh 2

Tugas telah berhasil diselesaikan. Mari luangkan waktu sejenak untuk meninjau pendekatan ini guna mengetahui potensi kelalaian dan area yang perlu ditingkatkan. Pikirkan tentang bagaimana proses dapat disempurnakan dan pertimbangkan kemungkinan pola atau persyaratan dapat berubah seiring waktu. Berikut adalah beberapa hal yang perlu dipertimbangkan untuk perbaikan:

- **Pemformatan huruf**: Metode saat ini hanya mempertimbangkan huruf kapital. Ini secara otomatis mengecualikan kata sembilan huruf lainnya yang mengandung huruf kecil, termasuk kode yang salah kapitalisasi – yang tidak akan diidentifikasi.
- **Validasi pola**: Pola itu sendiri, empat huruf diikuti oleh lima digit, belum divalidasi. Kelalaian ini dapat menyebabkan hasil yang tidak valid.
- **Risiko kehilangan data**: Ada risiko kehilangan data dalam kasus di mana lebih dari satu kode terdapat dalam string. Proses saat ini tidak akan menangkap semua contoh; proses ini hanya akan menangkap nilai pertama yang bertahan dari kriteria *List.Select*.

Untuk mengembangkan solusi yang lebih tangguh yang mengatasi semua masalah potensial ini, serta meningkatkan akurasi dan efektivitas, kita akan memanfaatkan kumpulan data sampel yang dimodifikasi ini dalam contoh berikut:

```
let
    Source = Table.FromColumns(
        {{"The code CHLO33446 is IMPORTANT.",
        "Unique document identifier: JXKE13295.",
        "Please use the code SBT36006 to access the system.",
        "For verification, enter ERKZ30881 and MdKZ85426.",
        "The transaction ID YNZ78388 must be noted.",
        "IHCY10273 is the code for your appointment.",
        "Reference code SSK36144 is included in the report.",
        "To complete registration, use PRLL90158 as your code.",
        "Package with tracking number MGA5K2287 has been shipped."}},
        type table [String=text]
```

```
    )
in
    Source
```

Sekali lagi, Anda dapat mengikuti langkah-langkah untuk menambahkan kolom Kode ke tabel:

1) Navigasi ke **Add Column** dan pilih **Custom Column**.
2) Masukkan Kode sebagai nama kolom baru.
3) [ ]

   Inisialisasi ekspresi record dengan memasukkan [ ], serangkaian tanda kurung siku. Di dalam record, list nama bidang dan nilai bidang yang dibatasi koma dapat dibuat.

4) *w = Text.SplitAny([String], "., :")*

   Pisahkan string input. Sertakan spasi dan karakter tanda baca, untuk membagi setiap string menjadi bagian yang lebih kecil. Pemisahan terjadi pada karakter apa pun yang ditentukan dalam argumen kedua *Text.SplitAny*, yang mengembalikan list nilai teks. Masukkan koma di akhir untuk membuat bidang baru dalam record.

5) *r = List.Select(w, each Text.Length(_) =9 dan Text.Start(_, 4)=Text.Select(_, {"A".."Z", "a".."z"}) dan Text.Range(_, 4, 5)=Text.Select(_,{"0".."9"}))*

   Pilih elemen mana pun dari list tersebut, *r = List.Select(w, each*, yang sesuai dengan pola yang diinginkan. Ini berarti panjang total sembilan karakter, *Text.Length(_) =9* dan, di mana empat karakter pertama adalah huruf, *Text.Start(_, 4)=Text.Select(_,{"A".."Z", "a".."z"})*, dan lima karakter terakhir adalah angka, *Text.Range(_, 4, 5)=Text.Select(_,{"0".."9"}))*. Sertakan tanda kurung tutup dalam fungsi List.Select dan koma untuk menambahkan kolom baru.

6) *a = Text.Combine(r, ", ")*

   Ekstrak dan gabungkan semua hasil menjadi satu string.

7) [a]

   Terapkan akses bidang untuk mendapatkan nilai a.

Meskipun diformat di sini agar lebih jelas, di bilah rumus, kode M akan terlihat seperti ini:

```
Table.AddColumn(Source, "Code", each
    [
        w = Text.SplitAny( [String], "., :"), r
        = List.Select( w, each
                Text.Length(_) =9 and
                Text.Start(_, 4)=Text.Select(_,{"A".."Z", "a".."z"}) and
                Text.Range(_, 4, 5)=Text.Select(_,{"0".."9"})
            ),
        a = Text.Combine( r, ", ")
    ][a]
)
```

Gambar 2.3 menunjukkan keluaran metode yang direvisi.

	String	Code
1	The code CHLO33446 is IMPORTANT.	CHLO33446
2	Unique document identifier: JXKE13295.	JXKE13295
3	Please use the code SBT36006 to access the system.	
4	For verification, enter ERKZ30881 and MdKZ85426.	ERKZ30881, MdKZ85426
5	The transaction ID YNZ78388 must be noted.	
6	IHCY10273 is the code for your appointment.	IHCY10273
7	Reference code SSK36144 is included in the report.	
8	To complete registration, use PRLL90158 as your code.	PRLL90158
9	Package with tracking number MGA5K2287 has been shipped.	

Gambar 2. 3 Hasil revisi contoh 2

Contoh 3, *Splitters*

Fungsi pemisah dalam bahasa M sangat berharga untuk skenario ekstraksi pola tertentu. Ada sekumpulan fungsi ini, yang semuanya telah dibahas secara menyeluruh dan dicontohkan dalam Bab 3 Jilid 3, Gomparers, Replacers, Gombiners, dan Splitters. Namun, karena format data dengan lebar tetap sangat umum di banyak sistem lama, sudah sepantasnya kita membahas skenario khusus itu di sini juga. Tidak seperti format yang dibatasi, di mana bidang dipisahkan oleh karakter seperti koma atau tab, format dengan lebar tetap hanya bergantung pada posisi data dalam string. Perbedaan ini sering kali menimbulkan tantangan: bidang disematkan dalam blok teks yang besar tanpa

pemisah yang jelas. Memahami cara mengurai data tersebut secara efektif sangatlah penting. Berikut adalah contoh kumpulan data:

```
let
    Source =
     Table.FromRows({
    {"Hardware ZEQNNZE    Nails - 2 inch    4SPYBBU8    WRPBOSGEHTFD300
      2024-01-10"},
     {"Software CGUL2L    Antivirus SW      ZH1R987S2  SADBT0     150
      2024-01-11"},
      {"Hardware OY6IL4VFH21 Hammer - 5kg   O08AUF8JG  80LC6OMO    75
       2024-01-12"},
    {"Software DLTQ80V7X  Operating System  NZ8DD797     AB9MTEI09L
      2024-01-13"},
    {"FurnitureWCPKVSZJX                    FTHJK      QQMR1QZMR71 50
      2024-01-14"},      Office Chair
    {"FurnitureZRE4CCR1                     OYOWTO7IOQMFSNGRQ5    40
      2024-01-15"},      Office Desk
    {"Hardware WJAA7DHJ                     EO65VPCJ   IGC2FHI8G  120
      2024-01-16"},      Screwdriver Set
    {"Software JX6URB9HI                    143OD1ADFC4 3YD7QU07T8OF300
      2024-01-17"},      Database Software

      {"Hardware 1ADXHN   Electric Drill    AK3M3MJMY6M XUBQZP7R    60
        2024-01-18"},
      {"FurnitureD20R63MNM6  Bookshelf          4G6GD3G    373NWC1I9JT55
        2024-01-19"}
  }, type table [Column1=text])
in
  Source
```

Pemisah yang paling efektif untuk menangani data dengan lebar tetap adalah *Splitter.SplitTextByPositions*. Pemisah ini mengambil list nomor indeks posisi. Setiap indeks yang tercantum harus lebih besar dari atau sama dengan 0, dan setiap nilai berikutnya sama dengan atau lebih besar dari yang sebelumnya. Indeks ini menentukan lokasi berbasis nol dalam teks tempat pemisahan terjadi, yang selaras sempurna dengan struktur inheren format data dengan lebar tetap. Tabel berikut menunjukkan cara menentukan posisi ini berdasarkan panjang bidang yang diketahui dari sistem lama dan, yang terpenting, juga menyertakan urutan bidang:

Tabel 2. 1 Menentukan posisi dan urutan lapangan

Field	Length	Position	Method	Order
Item type	10	0	Zero-based starting position = 0	0
Item code	12	9	Previous field length -1	1
Description	20	21	Prev field len + Prev position	2
Supplier code	12	41	Prev field len + Prev position	3
Manufacturer code	12	53	Prev field len + Prev position	4
Quantity	6	65	Prev field len + Prev position	5
Date	10	71	Prev field len + Prev position	6

Ada tiga kode dalam setiap baris data yang perlu kita ekstrak: **Item Code, Sup- plier Code,** dan **Manufacturer Code**. Kita akan mendemonstrasikan dua metode, yang pertama memanfaatkan *Splitter.SplitTextByPositions*. Pengetahuan tentang cara menentukan setiap posisi, ditambah dengan urutan bidang yang diketahui – digambarkan dalam tabel – memastikan kita dapat menyelesaikannya dengan mudah. Mari tambahkan kolom **Code** kustom. Berikut ini:

1) Navigasi ke **Add Column** dan pilih **Custom Column**.
2) Masukkan *Code* sebagai nama kolom baru.
3) [ ]

   Inisialisasi ekspresi record dengan memasukkan [ ], serangkaian tanda kurung siku.

4) *s = Splitter.SplitTextByPositions({0, 9, 21, 41, 53, 65, 71})([Column1])*

   Splitter.SplitTextByPositions mengambil satu argumen, list dengan posisi, dan mengembalikan fungsi. Untuk memanggil fungsi ini, cukup tambahkan satu set tanda kurung lagi, yang merupakan operator pemanggilan fungsi, dan berikan nama kolom. Masukkan koma di akhir untuk membuat kolom baru.

5) *r = List.Transform({1, 3, 4}, (x)=> Text.Trim(s{x}))*

   Dalam tabel, kita dapat mencari urutan bidang, indeks berbasis nol dari setiap bidang yang akan diekstrak. **The Item Code (1), Supplier Code (3),**

dan **Manufacturer Code (4)** sesuai dengan item list di s. Kita dapat membuat list dengan indeks ini untuk diulang, mengekstrak elemen-elemen tersebut dan menghapus spasi berlebih dalam satu operasi. Tambahkan koma di akhir untuk membuat bidang lain.

Ingat: *(x)=>* sama dengan *(_)=>* dan sama dengan ekspresi each, yang juga mengambil garis bawah sebagai satu-satunya variabelnya, seperti yang dijelaskan dalam Bab 1 Jilid 3, Parameter dan Fungsi Gustom.

6) a = Text.Combine(r, ", ")

Gabungkan semua hasil menjadi satu string.

7) [a]

Kembalikan nilai untuk bidang a dengan menerapkan akses bidang ke ekspresi record.

Saat diformat di sini, di bilah rumus, kode M akan terlihat seperti ini:

```
Table.AddColumn( Source, "Code",
    each [
        s = Splitter.SplitTextByPositions({0, 9, 21, 41, 53, 65, 71})
([Column1]),
        r = List.Transform( {1, 3, 4}, (x)=> Text.Trim( s{x} )),
        a = Text.Combine( r, ", ")
    ][a]
)
```

Gambar 2.4 menampilkan output metode ini saat opsi **Monospaced**, yang terdapat di tab **View**, diaktifkan. Penggunaan tampilan monospaced untuk sementara memungkinkan pengguna memverifikasi secara visual dan mudah bahwa mereka menangani data dengan lebar tetap.

	Column1	Code
1	Hardware ZEQNNZE Nails - 2 i…	ZEQNNZE, 4SPYBBU8, WRPBOSGEHTFD
2	Software CGUL2L Antivirus S…	CGUL2L, ZH1R987S2, SADBT0
3	Hardware OY6IL4VFH21 Hammer - 5k…	OY6IL4VFH21, O08AUF8JG, 80LC6OMO
4	Software DLTQ80V7X Operating S…	DLTQ80V7X, NZ8DD797, AB9MTEI09L
5	FurnitureWCPKVSZJZ Office Chai…	WCPKVSZJX, FTHJK, QQMR1QZNR71
6	FurnitureZRE4CCR1 Office Desk…	ZRE4CCR1, OYOWTO7IOQMF, SNGRQ5
7	Hardware WJAA7DHJ Screwdriver…	WJAA7DHJ, EO65VPCJ, IGC2FHI8G
8	Software JX6URB9HI Database So…	JX6URB9HI, 143OD1ADFC4, 3YD7QU07T8OF
9	Hardware 1ADXHN Electric Dr…	1ADXHN, AK3M3MJMY6M, XUBQZP7R
10	FurnitureD20R63MNM6 Bookshelf …	D20R63MNM6, 4G6GD3G, 373NWC1I9JT5

Gambar 2. 4 Hasil contoh 3

Kecuali jika modifikasi dilakukan pada laporan dasar sistem lama, metode seperti ini hampir tidak memerlukan pemeliharaan dan tidak perlu menyertakan penanganan kesalahan. Oleh karena itu, untuk mengeksplorasi jenis skenario ini lebih lanjut, kita akan mengeksplorasi alternatif, yang tidak bergantung pada fungsi pemisah tetapi dapat memanfaatkan posisi tetap ini secara setara. Kita akan menggunakan kumpulan data sampel yang sama persis.

Penting untuk dipahami bahwa mesin M menghasilkan kode M saat Anda berinteraksi dengan User Interface (UI). Proses ini mencakup pembaruan referensi ke variabel, yang juga dikenal sebagai nama langkah, dalam ekspresi. Namun, jika Anda memasukkan kode kustom yang bergantung pada variabel tertentu, interaksi UI lebih lanjut dapat mengganggu langkah ini atau langkah terkait lainnya.

Untuk mencegah masalah tersebut, ingatlah bahwa mesin M tidak peduli dengan urutan langkah karena mengikuti rantai ketergantungan ekspresi let. Dengan menempatkan variabel kustom Anda di atas langkah Sumber, Anda meminimalkan risiko interaksi UI yang memengaruhi langkah yang ditulis secara manual secara tidak sengaja.

Bagian penting dari metode berikut ini melibatkan penulisan variabel kustom secara manual, yang menyimpan informasi penting untuk operasi kita. Dalam Bab 4 Jilid 2, Bekerja dengan Struktur Bersarang, kita menunjukkan bagaimana fungsi *List.Zip* membentuk list baru dari list-list dengan memasangkan elemen-elemen dari semua list masukannya dengan posisi indeks yang sesuai, di mana setiap list bersarang berisi elemen dari List 1 (posisi) diikuti oleh elemen dari *List 2* (panjang bidang). Untuk meningkatkan efisiensi, terutama karena kita akan merujuk list ini berulang kali, kita akan menyimpannya dalam memori. Meskipun kode khusus ini tidak secara langsung merujuk variabel dari ekspresi let, sebagai praktik terbaik umum, kita sarankan untuk meletakkannya di atas langkah *Source*:

```
s = List.Buffer( List.Zip(
    {
        {0, 9, 21, 41, 53, 65, 71},
        {10, 12, 20, 12, 12, 6, 10}
    }
)),
```

Sekarang, Anda seharusnya sudah familier dengan langkah awal proses tersebut; oleh karena itu, kita terutama akan fokus pada logika untuk rumus kolom kustom itu sendiri:

1) Inisialisasi ekspresi *record* dan beri nama kolom pertamanya r.
2) Kita mengetahui urutan kolom dari sistem lama, indeks berbasis nol dari setiap kolom, dan bertujuan untuk mengekstrak **Item Code (1), Supplier Code (3),** dan **Manufacturer Code (4)** dari string teks. Ini dapat dilakukan dengan menempatkan posisi mereka dalam list untuk iterasi: *List.Transform({1, 3, 4}, (x)=> ...)*.
3) Kita akan mengekstrak panjang penuh setiap kolom, termasuk spasi di belakangnya, untuk menghapusnya: *Text.Trim(...)*.
4) Di dalam *Text.Trim*, gunakan fungsi *Text.Range* untuk mengekstrak tiga kolom, dengan meneruskan string dari *Column1* sebagai nilai argumen pertama: *Text.Range([Kolom1], ...)*.
5) Untuk argumen kedua, kita akan menyelami lebih dalam. Variabel s adalah list list, yang sesuai dengan setiap bidang dalam string. Kita dapat mengambil list terkait yang relevan dengan *s{x}*, di mana x adalah nilai saat ini dari list yang kita iterasi di dalam *List.Transform*.
   Karena hasil dari *s{x}* adalah list dengan dua elemen, yang pertama adalah posisi dan yang kedua adalah panjang bidang, kita dapat menerapkan akses item sekali lagi untuk mengekstrak nilai posisi dengan menambahkan {0}. Argumen untuk parameter kedua *Text.Range* sekarang akan terlihat seperti ini: *s{x}{0}*.
6) Sebagai parameter ketiga, ia mengharapkan hitungan yang sesuai dengan panjang bidang, yang merupakan item kedua di setiap list bersarang, yang memerlukan sedikit modifikasi: *s{x}{1}*.
7) Periksa apakah Anda memiliki tanda kurung tutup untuk fungsi *Text.Range*. Jika tidak, sertakan dan tambahkan koma di akhir sehingga kita dapat membuat bidang baru dalam record kita.

8) Tentukan kolom bernama a dan tetapkan ekspresi ini untuk menggabungkan semua hasil: *Text.Combine(r, ", ")*.
9) Terakhir, terapkan akses kolom ke ekspresi record untuk mengambil nilai untuk a:

```
[
    r = List.Transform( {1, 3, 4}, (x)=>
            Text.Trim(
                Text.Range( [Column1], s{x}{0}, s{x}{1} )
            )
        ),
    a = Text.Combine( r, ", ")
][a]
```

Berikut ini adalah tampilan ekspresi lengkap di dalam bilah rumus:

```
Table.AddColumn( Source, "Code", each
    [
        r = List.Transform( {1, 3, 4}, (x)=>
                Text.Trim(
                    Text.Range( [Column1], s{x}{0}, s{x}{1} )
                )
            ),
        a = Text.Combine( r, ", ")
    ][a]
)
```

Contoh 4, *Substitution*

Pendekatan lain untuk mengekstraksi pola tetap adalah substitusi. Teknik ini, yang mungkin paling rumit dari semuanya, memerlukan transformasi setiap karakter yang valid menjadi simbol yang mewakili suatu tipe. Misalnya, setiap huruf diubah menjadi tanda sisipan (^), setiap angka menjadi tanda pagar (#), dan setiap pemisah menjadi garis bawah (_). Proses ini menghasilkan satu pola yang seragam, sehingga menyederhanakan identifikasi dan ekstraksi.

Meskipun mendefinisikan pendekatan yang rumit seperti itu bisa efektif, pendekatan itu sering kali tidak diperlukan mengingat banyaknya alternatif yang lebih mudah. Namun, dalam kasus khusus, substitusi mungkin dapat dibenarkan. Berikut adalah kumpulan data contoh kita untuk contoh ini:

```
let
    Source = Table.FromColumns(
        {{"The code CHLO-33-446 is IMPORTANT.",
        "Unique document identifier: JXKE-13-295.",
        "Please use the code SBT3-6006 to access the system.",
        "For verification, enter ERKZ-30-881 and MdKZ-85.426.",
        "The transaction ID YNZ-78-388 must be noted.",
        "IHC-Y10-273 is the code for your appointment.",
        "Reference code SSK-36-144 is included in the report.",
        "To complete registration, use PRLL-90-158 as your code.",
        "Package with tracking number MGA5-K22.87 has been shipped."}},
        type table [String=text]
    )
in
    Source
```

Dalam skenario ini, kode yang valid didefinisikan sebagai berikut: Kode dimulai dengan empat huruf – huruf besar, huruf kecil, atau kombinasinya – diikuti oleh pemisah, yang berupa tanda hubung (-) atau titik (.), dua digit, pemisah lain, dan terakhir, serangkaian tiga digit. Panjang keseluruhan pola ini secara konsisten adalah 11 karakter. Dengan demikian, kode target dapat muncul sebagai *ABCD-12-345*, *efgh-67.890*, atau *IjkL.01.321*, misalnya.

Selain penggantian nilai yang mendasar, langkah-langkah yang tersisa memiliki tingkat kemiripan yang tinggi dengan Contoh 3, pendekatan alternatif Pemisah:

1) Pertama, susun list penggantian. List ini menyertakan nilai substitusi untuk setiap karakter yang valid dan dimuat ke dalam memori. Meskipun ini tidak secara langsung merujuk ke variabel dari ekspresi let, sebagai praktik terbaik umum, kita akan meletakkannya di atas langkah Sumber. Kodenya adalah sebagai berikut:

```
replacements = List.Buffer(
    List.Zip({{"A".."Z"}, List.Repeat({"^"}, 26)}) &
    List.Zip({{"0".."9"}, List.Repeat({"#"}, 10)}) &
    List.Zip({{"-", "."}, List.Repeat({"_"}, 2)})
),
```

2) Tambahkan kolom *Code* khusus ke tabel dan inisialisasi ekspresi record [ ], untuk mengekstrak semua kode yang valid.

3) Manfaatkan *List.ReplaceMatchingItems* untuk mengganti semua karakter di setiap string. Ini memerlukan list penggantian dalam format *{oldValue, newValue}*, dan fungsi ini juga dilengkapi dengan pembanding opsional. Setelah menyelesaikan semua penggantian, gabungkan semua item list untuk membentuk string teks tunggal:

```
replVal = Text.Combine(
    List.ReplaceMatchingItems(
        Text.ToList([String]),
        replacements,
        Comparer.OrdinalIgnoreCase
    )
)
```

4) Dalam string *replVal* kita sekarang dapat mengidentifikasi setiap contoh pola generik yang terlihat seperti ini: ^^^^_##_### dan mengumpulkan posisinya dalam sebuah list:

```
lookUp = Text.PositionOf( replVal, "^^^^_##_###", Occurrence.All)
```

5) Gunakan list pencarian untuk mengulang dan mengekstrak hasil dari string input, seperti ini:

```
getMatches = Text.Combine(
    List.Transform(
        lookUp,
        (x)=> Text.Range([String], x, 11 )
    ), ", "
)
```

6) Karena kita menggunakan ekspresi record untuk menghasilkan semua nilai, yang tersisa adalah mengembalikan hasil akhir yang disimpan di bidang *getMatches* (dengan menerapkan akses bidang):

```
[
    replVal = Text.Combine(
        List.ReplaceMatchingItems(
            Text.ToList([String]),
            replacements,
            Comparer.OrdinalIgnoreCase
        )
    ),
    lookUp = Text.PositionOf( replVal,
        "^^^^_##_###",
        Occurrence.All
    ),
    getMatches = Text.Combine(
        List.Transform(
            lookUp,
            (x)=> Text.Range([String], x, 11 )
        ), ", "
    )
][getMatches]
```

Berikut tampilannya di dalam bilah rumus:

```
Table.AddColumn( Source,
  "Code", each [
      replacements = List.Buffer(
          List.Zip({{"A".."Z"}, List.Repeat({"^"}, 26)}) &
          List.Zip({{"0".."9"}, List.Repeat({"#"}, 10)}) &
          List.Zip({{"-", "."}, List.Repeat({"_"}, 2)})
      ),
      replVal = Text.Combine(
          List.ReplaceMatchingItems(
              Text.ToList([String]),
              replacements,
              Comparer.OrdinalIgnoreCase
          )
      ),
      lookUp = Text.PositionOf( replVal, "^^^^_##_###", Occurrence.All),
      getMatches = Text.Combine(
          List.Transform(
              lookUp,
              (x)=> Text.Range([String], x, 11 )
          ), ", "
      )
  ][getMatches]
)
```

Gambar 2.5 menampilkan hasil yang diharapkan untuk metode ini.

	String	Code
1	The code CHLO-33-446 is IMPORTANT.	CHLO-33-446
2	Unique document identifier: JXKE-13-295.	JXKE-13-295
3	Please use the code SBT3-6006 to access the system.	
4	For verification, enter ERKZ-30-881 and MdKZ-85.426.	ERKZ-30-881, MdKZ-85.426
5	The transaction ID YNZ-78-388 must be noted.	
6	IHC-Y10-273 is the code for your appointment.	
7	Reference code SSK-36-144 is included in the report.	
8	To complete registration, use PRLL-90-158 as your code.	PRLL-90-158
9	Package with tracking number MGA5-K22.87 has been shipped.	

Gambar 2. 5 Hasil contoh 3

Kita telah mencatat bahwa mungkin ada alternatif yang lebih mudah, bahkan dalam kasus ketika Anda mengandalkan pemeriksaan setiap karakter. Untuk mendemonstrasikannya, mari kita gunakan kumpulan data contoh yang sama seperti contoh sebelumnya.

1) Tambahkan kolom *Code* kustom dan inisialisasi ekspresi record: [ ].
2) Pisahkan setiap string menjadi kata-kata. Tambahkan koma untuk membuat kolom baru.

```
w = Text.Split( [String], " ")
```

3) Pilih item list yang sesuai dengan pola. Abaikan kata-kata yang tidak memenuhi persyaratan karakter minimum (kurang dari 11 karakter). Dalam semua kasus lainnya, uji nilai dengan menghapus semua karakter yang valid dari setiap segmen dan periksa apakah panjang teks yang tersisa sama dengan nol. Tambahkan koma di akhir.

```
r = List.Select(w, (x)=> if Text.Length(x) < 11 then false
    else Text.Length(
        Text.Remove(Text.Start(x, 4), {"A".."Z", "a".."z"}) &
        Text.Remove(Text.Range(x, 4, 1) & Text.Range(x, 7, 1), {"-",
"."}) &
        Text.Remove(Text.Range(x, 5, 2) & Text.Range(x, 8, 3),
{"0".."9"})
    ) = 0
)
```

4) a = Text.Combine(List.Transform(r, (x)=> Text.Start(x, 11)), ", ")

Dari item list yang tersisa, ekstrak 11 karakter pertama.

5) [a]

Ekstrak nilai dari bidang a dengan menerapkan akses bidang.

Di dalam bilah rumus, kode M akan terlihat seperti ini:

```
Table.AddColumn( Source,
  "Code", each [
    w = Text.Split( [String], " "),
    r = List.Select( w, (x)=> if Text.Length(x) <11 then false
    else Text.Length(
    Text.Remove(Text.Start( x, 4),{"A".."Z", "a".."z"}) &
    Text.Remove(Text.Range( x, 4, 1) & Text.Range( x, 7, 1),{"-", "."})
    &
    Text.Remove(Text.Range( x, 5, 2) & Text.Range( x, 8, 3),{"0".."9"})
    )=0 ),
    a = Text.Combine( List.Transform(r, (x)=> Text.Start(x, 11)), ", ")
  ][a]
)
```

Contoh 5, *Regex*

Power Query tidak mendukung regex secara asli, tetapi dapat diimplementasikan dengan menggunakan fungsi *Web.Page* untuk mengeksekusi kode **JavaScript (JS)**. Secara umum, kinerja metode ini menjadi perhatian utama, terutama saat diterapkan pada kumpulan data yang lebih besar. Oleh karena itu, sangat penting untuk menyeimbangkan keinginan dengan kebutuhan regex guna memastikan manfaatnya lebih besar daripada kekurangannya dalam hal kinerja dan pengalaman pengguna.

Tugas kita adalah mengekstrak kode pos dari kumpulan data contoh ini. Kode pos ini masing-masing memiliki panjang 5 digit dan dapat ditemukan di tempat yang berbeda dalam rangkaian alamat:

```
let
  Source = Table.FromColumns({
    {
      "Boulevard des Écoles 73, 31000 Lyon",
      "6 Boulevard du Château, 69001 La Ville Rose Toulouse",
      "Rue Saint-Martin 65, 31000 Lyon",
      "Chemin Victor Hugo 143, 69001 Bordeaux",
      "Avenue des Vignes 7, 67000 Toulouse",
      "74 Quai de la République 69001 Latin Quarter Paris",
      "55 Boulevard de la Liberté, 67000 La Petite France Strasbourg",
      "82 Chemin des Jardins, 59000 Paris"
     }}, type table [Address = text]
  )
in
  Source
```

Metode yang umum adalah membuat fungsi kustom yang menggunakan fungsi *Web.Page* untuk menjalankan *JavaScript*. Proses ini memerlukan nilai teks dengan tag <*script*> untuk memasukkan kode JavaScript ke dalam dokumen HTML. Apa pun antara <*script*> dan </*script*> dieksekusi sebagai JavaScript. Tugas kita adalah membuat kode JS yang menjalankan operasi regex. Kode tersebut perlu mengidentifikasi dan mengekstrak urutan lima digit dari string teks input yang diteruskan ke fungsi kustom. Setelah itu, fungsi *Web.Page* akan menampilkan tabel yang menunjukkan konten dokumen HTML, yang disusun dalam elemen dasar. Tabel ini kemudian dapat diakses untuk mengambil hasilnya. Meskipun kita bukan ahli dalam kode JS, berikut tampilannya. Dengan mengikuti praktik terbaik, kita akan menerapkan fungsi kustom ini di atas langkah Source:

```
fxRegex = (input as text) as text =>
    Web.Page(
        "<script>
            var a ='" & input & "';
            var b = a.match(/\d{5}/g);
            document.write(b);
        </script>"
    ){0}[Data]{0}[Children]{1}[Children]{0}[Text],
```

Gambar 14.6 menunjukkan bagaimana tampilannya dari dalam **Advanced Editor**:

```
let
    fxRegex = (input as text) as text =>
        Web.Page(
            "<script>
                var a = '" & input & "';
                var b = a.match(/\d{5}/g);
                document.write(b);
            </script>"
        ){0}[Data]{0}[Children]{1}[Children]{0}[Text],
    Source = Table.FromColumns({
        {
```

Gambar 2. 6 Menggabungkan fungsi fxRegex di atas langkah Sumber

Berikut adalah rincian komponen dalam kode ini dan fungsinya:

1) *fxRegex = (input as text) as text =>*

   Ini mendeklarasikan fungsi bernama fxRegex yang mengambil satu parameter, input, bertipe teks. Fungsi tersebut akan mengembalikan nilai bertipe teks.

2) *Web.Page(...)*

   Fungsi ini memungkinkan kita membuat halaman web dalam memori, solusi untuk menjalankan JavaScript, yang mendukung ekspresi reguler, di Power Query. Apa pun di antara tag *<script>* dan *</script>* diperlakukan dan dieksekusi sebagai kode JavaScript.

3) *var a ='" & input & "';*

   Baris ini mendeklarasikan variabel JavaScript bernama a. Tanda kutip tunggal (') setelah tanda sama dengan (=) memulai string dalam JavaScript. Selanjutnya, tanda kutip ganda (") menandakan akhir string teks dalam M, yang untuk sementara menghindari notasi JS. Ini memungkinkan penggabungan nilai parameter input (& input) dalam kode M. Setelah menyuntikkan nilai input, M menggabungkan string baru (& ") dan melanjutkan dalam JavaScript, menutup string dengan tanda kutip tunggal (').

4) *var b = a.match(/\d{5}/g);*

Baris ini menginisialisasi variabel JavaScript, b, dan menggunakan metode match JavaScript dengan pola regex, di mana \d cocok dengan digit mana pun, {5} menentukan tepat lima digit dalam satu baris, dan g adalah tanda pencarian global untuk menemukan semua kecocokan. Ekspresi reguler bisa sangat spesifik. Anda mungkin perlu menyesuaikan pola ini saat menerapkannya pada skenario yang berbeda.

5) *document.write(b);*

Baris ini menuliskan hasil pencocokan regex ke dokumen HTML.

6) *){0}[Data]{0}[Children]{1}[Children]{0}[Text]*

Bagian ini mengakses output yang dihasilkan oleh *Web.Page* dan menavigasi melalui **Document Object Model (DOM)** untuk mendapatkan teks yang ditulis oleh *document.write(b)*. Secara berurutan menerapkan akses item dan akses bidang, beberapa kali, untuk mengambil konten.

Dengan fungsi *fxRegex* kustom, kita dapat menerapkan logikanya ke setiap baris dalam tabel, menyimpan hasilnya di kolom *Postal Code*. Karena fungsi kustom sama seperti fungsi lainnya di M, kita akan memilih *Custom Column* dan memasukkan: *fxRegex([Address])*.

Pada bilah rumus, kode M akan terlihat seperti ini:

```
Table.AddColumn(Source, "Postal Code", each fxRegex( [Address] ))
```

Gambar 2.7 menunjukkan keluaran metode ini.

	Address	Postal Code
1	Boulevard des Écoles 73, 31000 Lyon	31000
2	6 Boulevard du Château, 69001 La Ville Rose Toulouse	69001
3	Rue Saint-Martin 65, 31000 Lyon	31000
4	Chemin Victor Hugo 143, 69001 Bordeaux	69001
5	Avenue des Vignes 7, 67000 Toulouse	67000
6	74 Quai de la République 69001 Latin Quarter Paris	69001
7	55 Boulevard de la Liberté, 67000 La Petite France Strasbourg	67000
8	82 Chemin des Jardins, 59000 Paris	59000

Gambar 2. 7 Hasil aker memanggil fungsi fxRegex pada sampel kita

Catatan penting: Meskipun metode JavaScript ini didukung di Power BI Desktop dan Excel, misalnya, metode ini tidak tersedia di layanan Power BI. Di layanan Power BI, ekspresi reguler dapat diimplementasikan melalui skrip dalam Python atau R. Namun, mengaktifkan pembaruan terjadwal juga akan memerlukan gateway pribadi yang diinstal di komputer tempat buku kerja dan instalasi R atau Python berada.

Tugas tersebut berhasil diselesaikan. Mari kita evaluasi pendekatan ini. Kita telah menyebutkan bahwa kinerja teknik khusus ini dapat dengan cepat menjadi masalah karena teknik ini membuat halaman web di memori untuk setiap baris dalam tabel guna menjalankan kode JavaScript dan mengeksekusi *regex*. Akan tetapi, masalah utamanya, dari sudut pandang kita adalah kita sama sekali tidak memikirkan atau mencoba menggunakan fungsi M pustaka standar. Hal ini memunculkan pertanyaan penting: Apakah mungkin untuk menyelesaikan tantangan ini hanya dengan fungsi M standar? Mari kita gunakan kembali kumpulan data contoh kita dan cari tahu.

Anda dapat mengikuti langkah-langkah berikut untuk menambahkan kolom Kode Pos ke tabel:

1) Navigasi ke *Add Column* dan pilih *Custom Column*.
2) Masukkan *Postal Code* sebagai nama kolom baru.
3) [ ]

   Inisialisasi ekspresi record dengan memasukkan [ ], serangkaian tanda kurung siku.

4) *w = Text.SplitAny([Address], Text.Remove([Address], {"0".."9"}))*

   Pisahkan string input. Sebagai pemisah, kita akan menggunakan fungsi *Text.Remove* untuk menyertakan semua karakter dari string input, kecuali angka apa pun. Tambahkan koma di akhir untuk membuat kolom baru dalam record.

5) *r = List.Select(w, each Text.Length(_) =5)*

   Pilih semua elemen dari list yang memiliki total panjang 5 karakter. Masukkan koma di akhir baris untuk menambahkan kolom lain.

6) *a = Text.Combine(..., ", ")*

Gabungkan semua hasil menjadi satu string.

7) [a]

Ekstrak nilai kolom dengan menerapkan akses kolom ke record.

Meskipun diformat di sini agar lebih jelas, kode M di bilah rumus akan terlihat seperti ini:

```
Table.AddColumn(Source, "Postal Code", each
    [
        w = Text.SplitAny([Address],
            Text.Remove([Address], {"0".."9"})),
        r = List.Select( w, each Text.Length(_)=5),
        a = Text.Combine(r, ", ")
    ][a]
)
```

Kode yang relatif ringkas ini mencapai hasil yang diharapkan dan dijamin berkinerja lebih baik daripada operasi regex awal. Namun, apakah itu berarti tidak ada harapan jika Anda berada dalam posisi sulit dan satu-satunya solusi tampaknya adalah menggunakan regex? Belum tentu. Ada satu pilihan yang tersisa: mengoptimalkan pendekatan *regex* awal.

Pengoptimalan sering kali terasa seperti misteri. Ini melibatkan pengujian ekstensif dan merupakan perpaduan antara seni dan sains. Namun, terkadang hanya memikirkan ulang suatu pendekatan dapat membuat semua perbedaan. Kita tidak yakin siapa yang pertama kali memperkenalkan konsep ini, tetapi prinsipnya sangat sederhana dan efektif. Buat halaman web untuk menjalankan kode JavaScript hanya sekali.

Kita akan memanfaatkan kumpulan data sampel yang sama. Sekarang, dengan maksud untuk mengeksekusi kode JS hanya sekali, semua nilai harus diteruskan ke panggilan fungsi tunggal itu. Dalam istilah JavaScript, ia perlu menerima array nilai, melakukan operasi regex pada array itu, dan menyusun output sedemikian rupa sehingga Power Query dapat memahaminya. Berikut bagian terpenting dari teka-teki tersebut, yaitu membuat array:

1) Dapatkan nilai kolom Alamat sebagai list: *Source[Address]*.

2) Bungkus *Json.FromValue(...)* di sekitarnya untuk mengubah list tersebut menjadi format JSON, yang mempersiapkannya untuk diproses dalam lingkungan JavaScript.
3) Bungkus *Text.FromBinary(...)* di sekitarnya karena fungsi *Json.FromValue* mengembalikan tipe data biner. Namun, fungsi *Web.Page* memerlukan input string teks. Fungsi *Text.FromBinary* mengubah data biner (list alamat berformat JSON) menjadi string teks:

```
Text.FromBinary( Json.FromValue( Source[Address] ))
```

Hal ini memungkinkan kita untuk meneruskan seluruh kolom Alamat sebagai array nilai, yang dapat dimasukkan ke dalam kode JavaScript. Akibatnya, kode JavaScript yang menjalankan operasi regex harus dimodifikasi untuk mengulang array ini. Sekali lagi, kita sama sekali bukan ahli dalam kode JS, tetapi pertimbangkan kode berikut:

```
OutputList = Web.Page(
    Text.Combine({
        "<script>",
        "var arr = ",
        Text.FromBinary( Json.FromValue( Source[Address] )),
        ";",
        "var output = []; ",
        "for (var i in arr){",
        "  var matches = arr[i].match(/\d{5}/g);",
        "  output[i] = matches ? '\"' + matches.join('|') + '\"' :
'\"\"\"';",
        "}; ",
        "document.write('{' + output.join(',') + '}');",
        "</script>"
    })
){0}[Data]{0}[Children]{1}[Children]{0}[Text]
```

Berikut adalah rincian terperinci elemen dalam kode yang diberikan beserta fungsinya:

1) Web.Page(...)

Fungsi ini mengambil nilai teks dan membuat halaman web dalam memori yang mengeksekusi kode JavaScript yang ada di antara tag <*script*> dan </*script*>.

2) *Text.Combine({...})*

Fungsi ini mengambil list nilai teks yang dipisahkan koma dan menggabungkannya menjadi satu teks. List tersebut berisi string teks berbeda yang dipisahkan koma. Semuanya, kecuali satu ekspresi M, adalah segmen kode JS.

3) *var arr =*

Menginisialisasi array JavaScript dengan nama arr.

4) *Text.FromBinary(Json.FromValue(Source[Address]))*

Mengonversi kolom Address dari tabel Source menjadi string JSON, lalu menjadi nilai teks, yang akan menjadi konten array JavaScript bernama arr.

5) ; (*semicolon*)

Ini menandai akhir pernyataan dalam JavaScript, instruksi yang akan dieksekusi.

6) *var output = [];*

Mendeklarasikan array JavaScript kosong bernama output.

7) *for (var i in arr){*

Perulangan for dalam JavaScript mengiterasi setiap alamat dalam array *arr*.

8) *var matches = arr[i].match(/\d{5}/g);*

Menggunakan ekspresi reguler untuk menemukan urutan lima digit di setiap alamat dan menyimpannya dalam variabel bernama matches. Jika ada beberapa kecocokan dalam satu alamat, semuanya disertakan dalam array matches ini.

9) *output[i] = matches ? '\"' + matches.join('|') + '\"' : '\"\"';*

Operator ternary merupakan singkatan dari pernyataan *if-else*. Operator ini digunakan di sini sebagai *condition? IfTrue : IfFalse* dan memeriksa apakah variabel matches bukan null dan bukan kosong. Jika *true*, maka semua elemen akan digabungkan menjadi satu string, dipisahkan oleh karakter |. Jika *false*, ia menetapkan string teks kosong sebagai placeholder.

Ini memastikan array output memiliki panjang yang sama jika dibandingkan dengan array input arr, yang berarti satu nilai per baris tabel.

10) *document.write('{' + output.join(',') + '}');*

Menulis array output sebagai string ke dokumen HTML dan menggabungkan { (kurung kurawal) di awal dan satu } di akhir string yang dibatasi koma, yang menciptakan representasi tekstual dari sintaksis list dalam kode M.

11) *){0}[Data]{0}[Children]{1}[Children]{0}[Text]*

Bagian ini menelusuri dokumen halaman HTML yang dihasilkan untuk mengakses teks yang ditulis oleh kode JavaScript.

Gambar 14.8 menunjukkan nilai pengembalian untuk ekspresi *OutputList*, sebuah string.

Gambar 2. 8 Nilai pengembalian untuk OutputList, nilai teks

Langkah terakhir – mengonversi nilai teks ini, *OutputList*, ke dalam list aktual dengan nilai teks – ternyata mudah: *Expression.Evaluate(OutputList)*. List ini berisi satu nilai untuk setiap baris, yang akan kita tambahkan sebagai kolom *Postal Code* baru ke tabel Sumber. Anda dapat mengikuti petunjuk berikut:

1) Untuk menyisipkan langkah manual, klik tombol **fx**.
2) Ganti nama variabel yang dikembalikan dengan *Table.ToColumns(Source)*. Ini mengubah tabel Sumber menjadi list kolomnya.
3) Untuk menambahkan list nilai kolom lain yang berisi nilai *OutputList* yang telah diubah, kita dapat memanfaatkan operator kombinasi – ampersand, &.
4) Tambahkan *{Expression.Evaluate(OutputList)}* setelah ampersand. Ekspresi tersebut diapit oleh serangkaian tanda kurung kurawal, {}, yang mengubahnya menjadi list di dalam list.

5) Letakkan kursor Anda di depan ekspresi dan masukkan *Table.FromColumns(* untuk mulai mengubah list gabungan kembali menjadi tabel.
6) Letakkan kursor Anda di akhir ekspresi, tambahkan koma, lalu berikan list nama kolom. Karena kita menambahkan kolom baru, sertakan namanya, *Table.ColumnNames(Sumber)* & {"*Postal Code*"}, dengan cara yang sama.
7) Tambahkan tanda kurung tutup,), untuk menyelesaikan ekspresi.

Meskipun diformat di sini, di dalam bilah rumus kode M akan terlihat seperti ini:

```
Table.FromColumns(
    Table.ToColumns(Source) & {Expression.Evaluate(OutputList)},
    Table.ColumnNames(Source) & {"Postal Code"}
)
```

Ini menyimpulkan bagian mengekstraksi pola tetap. Setiap metode, baik yang langsung atau lebih rumit atau kompleks, menyediakan kemungkinan yang unik dan menunjukkan fleksibilitas Power Query dalam menangani data teks. Tentunya, jika kita berusaha, kita dapat dengan mudah menemukan selusin cara lain untuk mengatasi jenis tantangan ini. Namun, ingatlah bahwa pilihan metode pada akhirnya bergantung pada persyaratan, data, dan kompleksitas pola tertentu yang perlu diekstraksi. Keahlian dan imajinasi Anda yang terus berkembang adalah kunci untuk membuka lebih banyak potensi.

Beralih ke bagian kedua bab ini, kita mengalihkan fokus kita untuk menggabungkan data. Proses penting ini memungkinkan pengguna untuk menambahkan informasi dari berbagai tabel, lembar, atau buku kerja, misalnya.

C. Menggabungkan Data

Meskipun proses mengekstraksi dan menggabungkan data sekilas tampak mudah, proses ini dapat dengan cepat berubah menjadi tugas yang rumit dan

menuntut. Tingkat kerumitan sangat bergantung pada karakteristik data dan organisasinya. Dari perspektif bagian ini, penting untuk dipahami bahwa yang kita maksud dengan menggabungkan data adalah operasi penambahan, yang biasanya terlihat saat menggabungkan file.

Di dunia nyata, data sering kali berasal dari berbagai sumber dan dikelola oleh banyak pengguna, yang mempersulit pemeliharaan konsistensi di semua file. Buku kerja Excel, khususnya, adalah format yang sangat umum, karena Excel digunakan secara luas di banyak industri untuk berbagai tugas terkait data.

Di bagian ini, kita akan mempelajari teknik dan strategi untuk mengatasi tantangan yang umum dihadapi saat bekerja dengan banyak file Excel. Sementara beberapa bersifat khusus untuk sifat Excel, yang lain lebih umum dan berlaku di berbagai skenario.

1. Dasar-Dasar Untuk Menggabungkan Data

Proses **ETL** (**Extract, Transform,** dan **Load**) dimulai dengan menghubungkan ke sumber data. Sebelum memulai, penting untuk memiliki pemahaman yang baik tentang tugas dan mengklarifikasi persyaratan sebanyak mungkin. Meskipun ini bukanlah list yang lengkap, berikut adalah beberapa hal yang perlu diingat, terutama saat menangani beberapa file:

1) **Lokasi data**: Ini merujuk pada alamat unik yang mengidentifikasi tempat penyimpanan data. Ini dapat berupa jalur ke direktori atau file lokal, URL yang mengarah ke layanan penyimpanan cloud, atau jalur ke drive jaringan, misalnya. Penting untuk dipahami bahwa lokasi dan nama file tidak statis; keduanya dapat berubah seiring waktu karena berbagai alasan, seperti migrasi sistem, reorganisasi data, dan sebagainya.

   Di sinilah parameter berperan. Parameter berfungsi seperti variabel dan memungkinkan Anda menyimpan dan mengelola nilai yang dapat digunakan di dalam kueri Anda. Ini tidak hanya efisien tetapi juga menyederhanakan pemeliharaan. Parameter memungkinkan Anda memperbarui dan mengubah nilai dengan mudah tanpa harus mengubah

setiap kueri tempat nilai tersebut digunakan. Saat file, folder, atau bahkan seluruh basis data dipindahkan, Anda dapat dengan cepat memulihkan semua kueri dependen dengan memperbarui parameter yang menyimpan nilai tersebut. Parameter dibahas dalam Bab 9, Parameter dan Fungsi Gustom.

2) **Sumber data**: Agar dapat membaca data, konektor digunakan. Oleh karena itu, Anda perlu mengidentifikasi asal data dan apakah konektor tersedia untuk sumber tersebut. Misalnya, apakah itu file yang disimpan secara lokal, folder di drive jaringan, folder SharePoint Online, atau yang lainnya? Jika tidak ada konektor default yang tersedia untuk sumber tersebut, Anda dapat mengembangkannya sendiri. Proses tersebut dibahas dalam Bab 16, Mengaktifkan Ekstensi, dalam buku ini.

   Sebagian besar fungsi pengaksesan data memanfaatkan konektor dan lokasi untuk mengekstrak data dari sumber. Fungsi ini biasanya mengembalikan tabel **Navigation**, yang terutama digunakan oleh antarmuka pengguna Power Query untuk memberikan pengalaman navigasi atas data yang diekspos. Ini membantu pengguna mengidentifikasi dan memilih data tertentu secara visual untuk diproses lebih lanjut.

3) **Pemilihan data**: Data dapat dibuang ke dalam folder dan disebarkan ke beberapa file. Penting untuk mengetahui file mana yang harus dipertimbangkan, dan jika pemilihan diperlukan, kriteria apa yang harus diterapkan. Misalnya, rentang tanggal, ekstensi file tertentu, atau atribut relevan lainnya.

4) **Organisasi data**: Memahami bagaimana data diatur dalam file sangatlah penting. Apakah Anda berurusan dengan satu atau beberapa lembar dan tabel dalam buku kerja? Dapatkah Anda menemukan file tanpa data? Apakah ketiadaan data menjadi perhatian dan bagaimana Anda harus menangani keadaan tersebut?

5) **Struktur data**: Karena fokus kita adalah pada file Excel, Anda perlu memeriksa bagaimana data dalam setiap file terstruktur. Apakah data disajikan pada lembar atau tabel Excel? Dalam kasus terakhir, apakah semua tabel Excel diberi nama dengan benar untuk memudahkan identifikasi? Dalam kasus pertama, metode harus dikembangkan untuk mengidentifikasi di mana data relevan berada yang dapat diterapkan ke semua file yang terlibat.
6) **Konsistensi data**: Periksa keseragaman di seluruh file yang berbeda, seperti nama kolom yang cocok, tipe nilai yang konsisten di kolom dengan nama yang sama, dan seterusnya. Anda mungkin menemukan konvensi penamaan. Karena tajuk kolom dapat digunakan sebagai label pada laporan, tanyakan pada diri Anda apakah nama-nama tersebut sesuai dan jelaskan datanya. Atau, Anda mungkin menemukan nama kolom dengan awalan, seperti misalnya *1-1-2024 Stock On Hand*. Itu sendiri akan membuat semua kolom *Stock On Hand* menjadi unik. Namun, dengan mengekstrak tanggal, pola seperti itu dapat dengan mudah diperbaiki.
7) **Pengumpulan data**: Apakah semua data dalam file yang dipilih sama? Jika homogen, data tersebut dapat dikonsolidasikan ke dalam satu tabel. Jika tidak, kemungkinan besar terdapat beberapa set data yang berbeda. Identifikasi setiap set data: apa yang diwakilinya, dan apa, jika ada, keluaran yang diperlukan dari set data tersebut? Ingatlah bahwa kueri terpisah harus dikembangkan untuk setiap tabel guna menghasilkan data.

   Bayangkan file dari departemen keuangan, yang masing-masing memiliki neraca, anggaran, laporan laba rugi, pengeluaran, dan sebagainya. Jelas, file-file tersebut tidak dan tidak menggambarkan hal yang "sama". Data apa yang harus dikumpulkan untuk analisis?
8) **Transformasi data**: Terakhir, apakah data harus ditransformasikan sebelum dapat digabungkan? Apakah Anda menemukan pola atau masalah

dalam data yang perlu ditangani? Apakah transformasi yang diterapkan bersifat dinamis? Apakah Anda perlu menyertakan pelaporan kesalahan?

Strategi pengoptimalan model data yang paling sederhana dan efektif adalah hanya menyertakan apa yang Anda butuhkan, tidak ada yang lain. Memilih kolom dan baris lebih awal dapat memberikan manfaat tambahan seperti pelipatan kueri. Pelipatan kueri dibahas dalam Bab 15, Mengoptimalkan Kinerja.

Mudah-mudahan, semakin jelas bahwa apa yang sering dianggap sebagai tugas sederhana sebenarnya dapat memiliki sejumlah lapisan yang menambah kompleksitas. Meskipun demikian, kunci keberhasilan terletak pada desain kueri yang cermat. Perlu dipikirkan tentang bagaimana persiapan dan pengembangan kueri memengaruhi proses ETL, bahkan pada intinya, dengan mempertimbangkan hal-hal mendasar seperti urutan langkah dan menghindari redundansi, misalnya. Intinya, waktu dan upaya yang dihabiskan untuk merancang kueri yang terstruktur dengan baik terbayar dalam hal kemudahan perawatan, keandalan, dan pada akhirnya kualitas wawasan yang dapat Anda peroleh dari analisis berikut.

Setelah memperkenalkan aspek-aspek penggabungan data, mari kita bahas contoh praktis. Tujuan kita adalah untuk mengilustrasikan berbagai elemen utama yang dijelaskan di sini. Meskipun setiap kumpulan data dan persyaratannya memiliki karakter yang berbeda, tujuannya adalah untuk meletakkan kerangka dasar yang dapat membantu mengembangkan strategi yang lebih tangguh untuk jenis tantangan ini, sehingga Anda dapat menggabungkan elemen-elemen tersebut ke dalam alur kerja Anda sendiri.

Dengan mengingat hal ini, mari kita mulai eksplorasi kita dengan membahas contoh praktis.

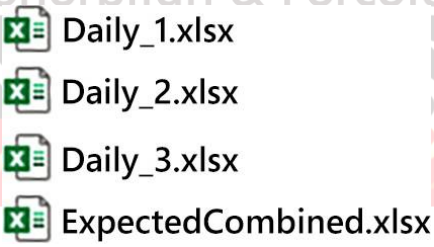
2. Ekstrak, ubah, dan gabungkan

Di bagian ini, kita berfokus pada pengerjaan beberapa file Excel yang disimpan dalam satu folder. Untuk mendapatkan hasil maksimal, kita sarankan untuk mengikuti langkah-langkah yang diberikan secara aktif. Penting untuk dicatat bahwa ada banyak metode untuk mencapai hasil yang diinginkan. Di bagian bab ini, tujuan kita adalah memandu Anda melalui satu solusi khusus, dengan mengilustrasikan berbagai teknik yang telah dibahas di seluruh buku ini. Catatan akhir: tangkapan layar berasal dari Power BI Desktop (versi: 2.124.1805.0); mungkin ada beberapa perbedaan jika Anda menggunakan versi yang berbeda.

Sebelum melanjutkan, penting untuk memahami tugas yang ada dan memeriksa bagaimana data mentah diatur dalam file.

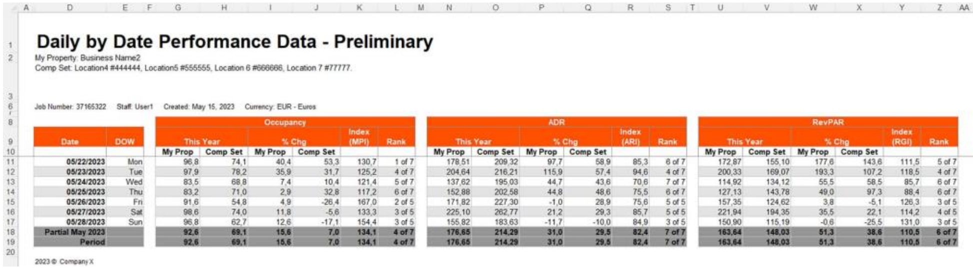
Dapatkan dan Periksa Data

Data yang digunakan dalam contoh ini tersedia untuk diunduh di sini: https://github.com/PacktPublishing/The-Definitive-Guide-to-Power-Query-M-/. Ini adalah folder yang berisi sembilan file. Tiga file pertama, yang diilustrasikan dalam Gambar 2.9, harus digabungkan untuk mencerminkan hasil yang diharapkan yang ditunjukkan dalam file akhir yang ditampilkan di sini:



Gambar 2. 9 File yang akan digabungkan dan file hasil yang diharapkan

Luangkan waktu sejenak untuk meninjau tiga berkas sumber. Cuplikan gambar ditunjukkan pada Gambar 2.10.



Daily by Date Performance Data - Preliminary

My Property: Business Name2

Comp Set: Location4 #444444, Location5 #555555, Location 6 #666666, Location 7 #77777.

Job Number: 37165322 Staff: User1 Created: May 15, 2023 Currency: EUR - Euros

Date	DOW	Occupancy This Year My Prop	Occupancy This Year Comp Set	Occupancy % Chg My Prop	Occupancy % Chg Comp Set	Occupancy Index (MPI)	Occupancy Rank	ADR This Year My Prop	ADR This Year Comp Set	ADR % Chg My Prop	ADR % Chg Comp Set	ADR Index (ARI)	ADR Rank	RevPAR This Year My Prop	RevPAR This Year Comp Set	RevPAR % Chg My Prop	RevPAR % Chg Comp Set	RevPAR Index (RGI)	RevPAR Rank
05/22/2023	Mon	96,8	74,1	40,4	53,3	130,7	1 of 7	178,51	209,32	97,7	58,9	85,3	6 of 7	172,87	155,10	177,6	143,6	111,5	5 of 7
05/23/2023	Tue	97,9	78,2	35,9	31,7	125,2	4 of 7	204,64	216,21	115,9	57,4	94,6	4 of 7	200,33	169,07	193,3	107,2	118,5	4 of 7
05/24/2023	Wed	83,5	68,8	7,4	10,4	121,4	5 of 7	137,62	195,03	44,7	43,6	70,6	7 of 7	114,92	134,12	55,5	58,5	85,7	6 of 7
05/25/2023	Thu	83,2	71,0	2,9	32,8	117,2	6 of 7	152,88	202,58	44,8	48,6	75,5	6 of 7	127,13	143,78	49,0	97,3	88,4	6 of 7
05/26/2023	Fri	91,6	54,8	4,9	-26,4	167,0	2 of 5	171,82	227,30	-1,0	28,9	75,6	5 of 5	157,35	124,62	3,8	-5,1	126,3	3 of 5
05/27/2023	Sat	98,6	74,0	11,8	-5,6	133,3	3 of 5	225,10	262,77	21,2	29,3	85,7	5 of 5	221,94	194,35	35,5	22,1	114,2	4 of 5
05/28/2023	Sun	96,8	62,7	12,6	-17,1	154,4	3 of 5	155,82	183,63	-11,7	-10,0	84,9	3 of 5	150,90	115,19	-0,6	-25,5	131,0	3 of 5
Partial May 2023		92,6	69,1	15,6	7,0	134,1	4 of 7	176,65	214,29	31,0	29,5	82,4	7 of 7	163,64	148,03	51,3	38,6	110,5	6 of 7
Period		92,6	69,1	15,6	7,0	134,1	4 of 7	176,65	214,29	31,0	29,5	82,4	7 of 7	163,64	148,03	51,3	38,6	110,5	6 of 7

2023 © Company X

Gambar 2. 10 Tampilan terbatas dari file sumber

Bandingkan masukan dengan hasil yang diharapkan dari *ExpectedCombined.xlsx*, yang sebagian ditunjukkan pada Gambar 2.11.

Name	Date	DOW	Occupancy | This Year | My Prop	Occupancy | This Year | Comp Set	Occupancy | % Chg | My Prop	Occupancy | % Chg | Comp Set	Occupancy | Index (MPI)	Occupancy | Rank
Daily_1.xlsx	5/22/2023	Mon	89.393939393939391	93.126385809312623	69.540229885057471	73.553719008264466	95.992063492063494	5 of 6
Daily_1.xlsx	5/23/2023	Tue	100	98.59571322985957	87.5	77.158034528552463	101.42428785607196	1 of 6
Daily_1.xlsx	5/24/2023	Wed	70.707070707070713	84.405025868440504	28.205128205128204	48.119325551232166	83.771161704611785	6 of 6
Daily_1.xlsx	5/25/2023	Thu	61.616161616161612	77.605321507760536	9.3189964157706093	12.419700214132762	79.396825396825392	5 of 6
Daily_1.xlsx	[illegible]/2023	Fri	[illegible]	[illegible]	10.59190031152648	[illegible]	[illegible]	5 of 6
Daily_3.xlsx	[illegible]2023	Thu	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Daily_3.xlsx	5/26/2023	Fri	73.684210526315795	77.611940298507463	-19.185059422750424	-18.16082983773941	94.939271255060731	4 of 5
Daily_3.xlsx	5/27/2023	Sat	88.421052631578945	96.865671641791039	-6.052631578947369	3.723418306696606	91.282134457870399	5 of 5
Daily_3.xlsx	5/28/2023	Sun	84.210526315789465	93.283582089552226	-10.526315789473683	-0.22264368763915232	90.273684210526312	5 of 5

Gambar 2. 11 Pandangan terbatas terhadap hasil yang diharapkan

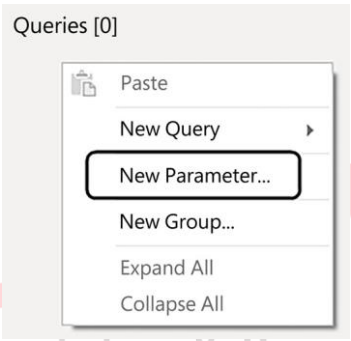
Metode yang akan kita bahas menunjukkan cara membangun solusi yang dapat dikelola, yang mencerminkan apa yang disediakan UI saat **Combine & Transform Data** dipilih. Ini adalah pendekatan yang sangat efektif, dapat disesuaikan, dan modular, yang memungkinkan kolaborasi dan serah terima yang mudah kepada orang lain, bahkan saat berisi kode M tingkat lanjut atau konsep yang canggih.

Mengingat lokasi file cenderung berubah seiring waktu, kita akan memulai pendekatan modular dengan membuat parameter.

Parameter Lokasi

Kita telah menyimpan semua file yang diunduh dalam folder bernama *Sample files*, di direktori akar sistem. Daripada mengodekan jalur tersebut secara permanen dalam kueri akses data apa pun, buat parameter untuk menyimpan nilainya. Nilai parameter tersebut dapat dengan mudah diperbarui tanpa perlu mengubah kode M apa pun dan, dalam kebanyakan kasus, tanpa pernah membuka Editor Power Query.

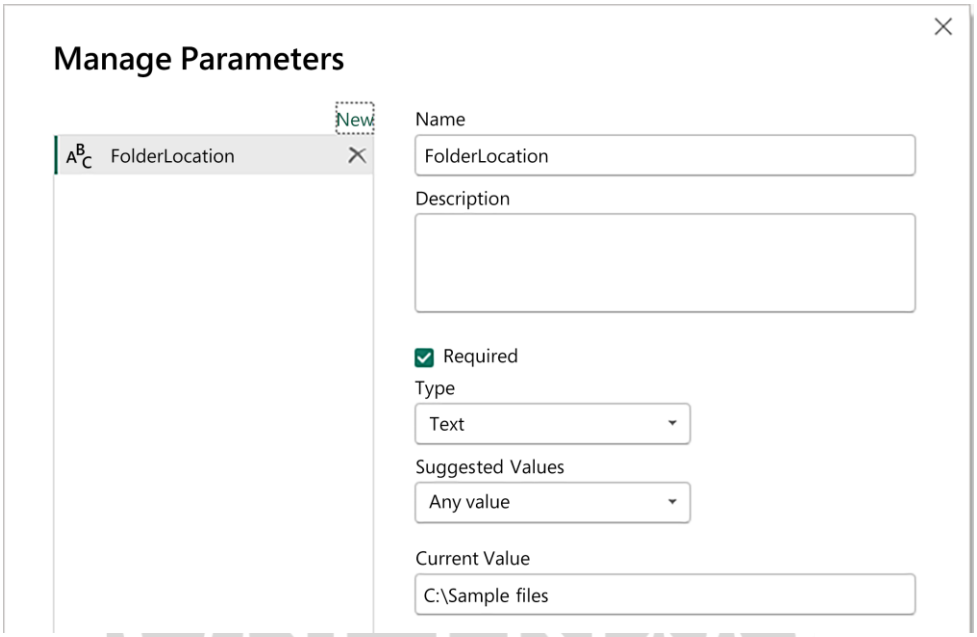
Klik kanan pada area kosong di dalam panel **Queries** dan pilih **New Parameter**, dari menu akses cepat, yang digambarkan pada Gambar 2.12, atau pilih dari menu tombol **Manage Parameters** yang terletak di tab **Home**:



Gambar 2. 12 Buat new parameter dengan cepat dari panel Queries

Tindakan ini membuka kotak dialog **Manage Parameters**, yang ditunjukkan pada Gambar 2.13, yang menampilkan semua parameter dalam berkas saat ini dan menyediakan tempat terpusat untuk membuat, mengelola, dan menghapusnya. Setiap parameter disimpan dan terlihat sebagai kueri terpisah di panel **Queries**.

Dalam M, nama kueri dianggap sebagai pengenal, dan notasi yang dikutip berlaku dalam kondisi tertentu, seperti saat nama kueri menyertakan spasi. Untuk menghindarinya, pilih nama yang bebas spasi seperti *FolderLocation*. Pastikan untuk tetap mencentang opsi **Required**. Untuk *Type*, pilih *Text*, dan tentukan **Current Value**. Dalam kasus kita, yaitu *C:\Sample files* (perhatikan bahwa jalur file Anda mungkin berbeda):



Gambar 2. 13 Dialog Kelola Parameter

Dengan adanya blok penyusun ini, kita sekarang dapat bergerak maju dengan membuat koneksi ke data.

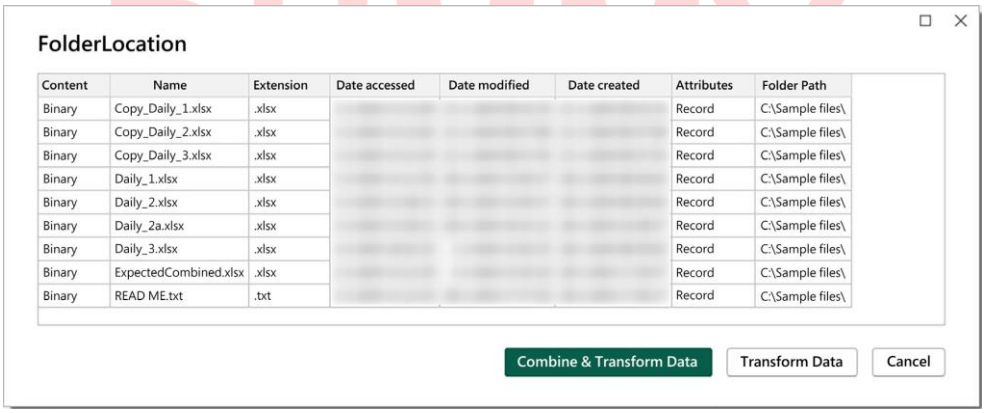
Hubungkan ke Data

Gunakan tombol Sumber Baru yang terletak di tab **Home**. Pilih **Folder connector** dan alihkan **input type** dari **Text** (ditunjukkan oleh ikon **ABC**) ke **Parameter**, atur parameter **FolderLocation** sebagai nilainya. Menyelesaikan tindakan ini akan menampilkan konten folder, yang diilustrasikan dalam Gambar 2.14. Bersamaan dengan itu, Anda akan menemukan tombol perintah, yang masing-masing menawarkan fungsionalitas yang berbeda. Berikut adalah deskripsi singkatnya:

1) **Gabungkan & Ubah Data**: Opsi ini hanya boleh dipilih ketika semua konten folder, termasuk subfolder, akan diproses dengan cara yang persis sama. Ini tidak menawarkan cara untuk memilih file dan penggabungan data akan gagal ketika format, struktur, atau konsistensi file bervariasi. Misalnya, jika "file contoh" adalah file Excel dan tabel bernama myData

dipilih dari pengalaman navigasi, semua file akan diperlakukan sebagai file Excel yang berisi tabel myData.

2) **Transform Data**: Opsi ini harus dipilih untuk mengontrol pemilihan dan transformasi file secara manual di Editor Power Query sebelum menggabungkan data.
3) **Cancel**: Ini akan membatalkan operasi penyambungan ke data.



Gambar 2. 14 Isi folder dan tombol perintah yang tersedia

Tujuannya adalah untuk membuat solusi yang memungkinkan pemilihan file tertentu, oleh karena itu pilih **Transform Data**. Tindakan ini membuat kueri baru, yang disebut Query1 jika nama tersebut belum ada. Kueri ini berisi data yang ditunjukkan pada Gambar 2.14 dan berfungsi sebagai blok penyusun baru.

Setelah menyiapkan koneksi ini, saatnya memilih file yang diperlukan untuk analisis kita.

Filter File

Satu-satunya transformasi yang perlu kita terapkan pada *Query1* adalah filter untuk memilih file yang menarik untuk operasi penggabungan – tiga file teratas yang digambarkan pada Gambar 14.9. Untuk menjalankan proses tersebut, kita akan menggunakan menu drop-down dari tajuk kolom *Extension*,

masuk ke **Text filter**, pilih *Begins with*, dan masukkan *.xls*, yang menghasilkan kode berikut, yang terlihat di bilah rumus:

```
Table.SelectRows(Source, each Text.StartsWith([Extension], ".xls"))
```

Untuk memastikan perbandingan dilakukan tanpa memperhatikan huruf besar/kecil, berikan Comparer.OrdinalIgnoreCase sebagai nilai argumen ketiga ke fungsi Text.StartsWith. Lebih jauh, file hanya boleh dipertimbangkan jika namanya dimulai dengan Daily_. Untuk melakukannya, ikuti langkah-langkah berikut:

1) Dari bilah rumus, salin fungsi *Text.StartsWith* secara lengkap.
2) Letakkan kursor Anda sebelum tanda kurung penutup terakhir.
3) Masukkan operator **and**.
4) Tempel logika yang disalin.
5) Perbarui kode dan masukkan bidang [*Name*] sebagai teks dan *Daily_* sebagai substring.
6) Ubah nama langkah ini menjadi *SelectFiles*.

Meskipun diformat di sini, kodenya sekarang akan terlihat seperti ini:

```
Table.SelectRows(Source,
    each Text.StartsWith([Extension], ".xls") and
    Text.StartsWith([Name], "Daily_", Comparer.OrdinalIgnoreCase)
)
```

Strategi Keseluruhan

Konektor **folder** Power Query menyertakan opsi **Combine & Transform** Data. Opsi ini menggunakan file contoh yang dipilih pengguna untuk mengembangkan logika transformasi yang akan diterapkan secara seragam ke semua file dalam folder. Namun, proses desain tersebut tentu saja tidak memiliki fleksibilitas untuk memilih file tertentu atau mengekstrak kumpulan data tertentu, seperti tabel atau lembar kerja, dari file-file ini. Bagi mereka yang ahli dalam bahasa M, mengubah semua kueri yang relevan dapat dilakukan, sementara banyak yang lain kesulitan.

Oleh karena itu, kita akan mengambil pendekatan yang berbeda: membangun solusi kustom yang menyerupai **Combine & Transform Data** tetapi memperkenalkan fleksibilitas yang diperlukan selama pengembangan langkah demi langkahnya, menyesuaikan proses secara tepat dengan persyaratan kita, sekaligus memperoleh pemahaman yang lebih baik tentang bagaimana solusi ini diatur – modular, mudah dirawat, dan sangat cocok untuk berbagai skenario.

Untuk mencapai hasil yang diinginkan, beberapa transformasi harus dilakukan:

1) Hapus baris dan kolom kosong.
2) Ekstrak bagian nama bisnis dari tajuk dokumen.
3) Ekstrak isi data dari lembar kerja.
4) Menangani beberapa baris header.

Untuk memulai proses pengembangan ini, kita perlu memilih sampel yang sesuai. File mana yang paling sesuai untuk tujuan tersebut? Dan mari kita antisipasi bahwa kita mungkin ingin beralih ke file sampel yang berbeda di masa mendatang. Karena ini adalah input lain, kita dapat menyimpannya dalam parameter.

Pilih File Sampel

Nilai Biner di kolom Konten (Gambar 2.14) berisi data file Excel. Anda harus menganggapnya sebagai nilai yang akan diubah. Dengan kata lain, ini adalah nilai input dan untuk membangun pola transformasi untuk semua file, diperlukan sampel. Setelah memeriksa file, kita menemukan Daily_2.xlsx yang sesuai. Tampaknya ini yang paling menantang dan lengkap. Anda dapat mengikuti langkah-langkah berikut:

1) Klik kanan pada spasi di samping nilai Biner untuk Daily_2.xlsx.
2) Pilih **Add as new Query**.
3) Ubah nama kueri baru itu menjadi *BinarySample*.

	Content	Name	Extension
1	Binary	Daily_1.xlsx	.xlsx
2	Binary	Daily_2.xlsx	.xlsx
3	Binary	Daily_2a.xlsx	.xlsx
4	Binary	Daily_3.xlsx	.xlsx

Gambar 2. 15 Tiga kolom pertama dari kueri Query1

Parameter Berkas

Konten biner adalah nilai input. Saat mengembangkan pendekatan terstruktur dan modular, buat parameter untuk itu. Berikut cara melakukannya:

1) Pilih **Kelola Parameter** dan pilih **New parameter** dari opsi.
2) Tetapkan nama *BinaryFile*, yang mencerminkan tujuannya, sehingga mudah diidentifikasi.
3) Pilih **Binary** untuk **Type** dan **Suggested Values**.
4) Masukkan *BinarySample* untuk **Default Value** dan tetapkan juga untuk **Current Value**.

Pola Transformasi

Kita akan tetap menggunakan pendekatan modular, dengan menyiapkan kueri pengembangan transformasi yang mengambil parameter BinaryFile sebagai nilai inputnya:

1) Klik kanan pada kueri parameter *BinaryFile*.
2) Pilih **Reference** dari opsi yang tersedia. Ini akan menambahkan kueri baru dan menampilkan file di panel **Preview**.
3) Klik dua kali file untuk memanggil fungsi **Excel.Workbook**.
4) Ubah nama kueri ini menjadi *TransformData*.

Sekarang konten file Excel yang dipilih akan terlihat, seperti yang digambarkan pada Gambar 2.16.

	Name	Data	Item	Kind
1	Daily	Table	Daily	Sheet

Gambar 2. 16 Isi dari file contoh

Ini menandai momen penting lainnya dalam proses pengembangan. Ada kemungkinan bahwa konten berkas, tidak seperti Gambar 2.16, menghasilkan tabel yang berisi lebih dari satu baris. Dalam skenario di mana data seragam dan harus dikonsolidasikan ke dalam satu tabel, proses pemilihan baris yang relevan, pemilihan sampel, dan pembuatan parameter seperti yang diilustrasikan sebelumnya diulang.

Di sisi lain, jika konten berkas di sini menunjukkan beberapa set data yang berbeda, kueri baru harus dibuat untuk setiap tabel terpisah untuk dikembangkan. Kemudian, duplikat kueri ini, berikan nama yang sesuai, dan ulangi proses yang akan datang untuk masing-masingnya.

Baiklah, sekarang, dengan mempertimbangkan potensi ketidakkonsistenan di seluruh berkas yang dapat menghambat identifikasi data yang akurat, tindakan proaktif dapat diterapkan. Alih-alih langsung menelusuri kolom Data, perlindungan dapat diterapkan yang mampu mengatasi beberapa tingkat ketidakkonsistenan antara berkas, seperti huruf besar-kecil. Untuk mengilustrasikannya, kita akan menerapkan filter pada baris di mana Jenis sama dengan Lembar dan Nama sama dengan Harian, dan memastikan perbandingan ini dilakukan tanpa memperhatikan huruf besar/kecil:

```
Table.SelectRows( Source,
    each Text.Lower([Name]) = "daily" and [Kind] = "Sheet"
)
```

Pahami bahwa ekspresi filter seperti ini tidak akan pernah mengembalikan lebih dari satu baris karena nama lembar harus unik dalam buku kerja dan penggunaan huruf besar tidak dipertimbangkan. Ini membuatnya aman untuk menelusuri kolom Data dengan menerapkan kombinasi akses item dan bidang opsional:

1) Di dalam bilah rumus, setelah tanda kurung tutup, terapkan akses item opsional, *{0}?*, untuk mendapatkan record pertama dari tabel.
2) Ini harus langsung diikuti oleh akses bidang opsional, *[Data]?*, untuk mengembalikan nilai dari bidang yang diminta.

3) Di panel **Applied Steps**, ganti nama langkah ini menjadi *RAW*.

Sekarang konten lembar yang dipilih terlihat, sebagian, seperti yang digambarkan pada Gambar 2.17.

Column3	Column4	Column5	Column6
Daily by Date Performance Data - Preliminary	null	null	null
My Property: Business Name2	null	null	null
Comp Set: Location4 #444444, Location5 #555555, Location 6 #66666...	null	null	null
null	null	null	null
null	null	null	null
Job Number: 37165322 Staff: User1 Created: May 15, 2023 Curr...	null	null	null
null	null	null	null
null	null	null	Occupancy
Date	DOW	null	This Year
null	null	null	My Prop
05/22/2023	Mon	null	96,84210526
05/23/2023	Tue	null	97,89473684

Gambar 2. 17 Tampilan sebagian data pada lembar harian

Hapus Baris Kosong

Jika melihat datanya, ada cukup banyak kolom dan baris yang benar-benar kosong. Tampaknya strategi yang bagus adalah menghapusnya terlebih dahulu, dan UI menawarkan fitur ini untuk baris:

1) Pada tab **Home**, pilih **Remove Rows** lalu **Remove Blank Rows**.
2) Di panel **Applied Steps**, ganti nama langkah tersebut menjadi *NoEmptyRows*.

Tindakan tersebut menghasilkan kode M berikut, yang terlihat di bilah rumus:

```
Table.SelectRows(RAW,
    each not List.IsEmpty(
        List.RemoveMatchingItems( Record.FieldValues(_), {"", null})
    )
)
```

Hapus Kolom Kosong

Tidak ada fitur yang setara untuk menghapus kolom kosong. Sebaliknya, kita harus mengembangkan logika tersebut. Namun, mari kita ambil inspirasi dari langkah *NoEmptyRows*:

1) Kita dapat menggunakan kembali sebagiannya. Lanjutkan dan salin kode M dari kata kunci *each* hingga tanda kurung penutup kedua hingga terakhir dari bilah rumus.
2) Klik **fx** di depan bilah rumus untuk menyisipkan langkah manual.
3) Di dalam bilah rumus, Anda dapat melihat nama langkah sebelumnya, tempatkan kursor di depannya, dan masukkan fungsi *Table.SelectColumns(*
4) Pindahkan kursor Anda ke akhir dan tempatkan koma. Parameter kedua mengharapkan list dengan nama kolom untuk disimpan. Logika itu akan disediakan oleh ekspresi lain, yang menghasilkan list dengan semua nama kolom berdasarkan suatu kondisi, seperti ini:
   a) Masukkan fungsi *List.Select*.
   b) Berikan nama kolom saat ini: *Table.ColumnNames(NoEmptyRows)*
   c) Ikuti ini dengan koma dan tempel logika yang disalin sebagai kondisi *each not List. IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))*
   d) Operasi ini tidak akan dilakukan baris demi baris tetapi kolom demi kolom; oleh karena itu, *Record.FieldValues(_)* harus diganti dengan *Table.Column(NoEmptyRows, _)*, yang menghasilkan list dengan kolom nilai.
   e) Terakhir, masukkan tanda kurung tutup pada fungsi *Table.SelectColumns*.
5) Ubah nama langkah ini menjadi *NoEmptyCols*.

Dengan mengikuti langkah-langkah yang diberikan dengan saksama, ekspresi tersebut sekarang akan terlihat seperti ini:

```
Table.SelectColumns( NoEmptyRows,
    List.Select( Table.ColumnNames(NoEmptyRows),
        each not List.IsEmpty(
            List.RemoveMatchingItems(
                Table.Column( NoEmptyRows, _),
                {"", null}
            )
        )
    )
)
```

Hasilnya, semua kolom yang hanya berisi kolom kosong dan/atau null telah dihapus.

Mengingat tingginya kemungkinan perlunya menghapus kolom kosong saat bekerja dengan file Excel di masa mendatang, mengubah tugas ini menjadi fungsi kustom tampaknya merupakan langkah yang logis. Pendekatan ini disertai dengan manfaat tambahan – mari kita temukan manfaatnya.

Fungsi *Custom*

Jumlah logika yang signifikan yang terlibat dalam melakukan operasi ini merujuk ke tabel dari langkah *NoEmptyRows* sebanyak tiga kali. Akan sangat bagus untuk memindahkannya ke parameter fungsi. Mari buat fungsi kustom di atas langkah Source di **Advanced Editor**. Ini juga akan meningkatkan keterbacaan dan penggunaan ulang kode di masa mendatang.

1) Salin ekspresi yang ditetapkan ke langkah *NoEmptyCols*.
2) Sisipkan langkah baru di atas *Source* dan beri nama: *fxNoEmptyCols =*
3) Sebelum kita menempelkan logika, kita bermaksud mengubahnya menjadi fungsi kustom:

a) Masukkan inisialisasi fungsi – tanda kurung dengan tanda goes to: () =>
b) Fungsi ini memerlukan satu parameter – nilai input tipe tabel. Sebut saja *tbl* dan tentukan tipenya: *tbl as table*
c) Nilai yang dikembalikan fungsi juga adalah table. Kita dapat menentukannya di antara tanda kurung tutup dan tanda *goes to*: *as table*

d) Setelah tanda goes to, tempel kode yang disalin, dan ganti semua referensi tabel dengan parameter *input tbl*.

Setelah selesai, variabel dan ekspresinya akan terlihat seperti ini:

```
fxNoEmptyCols = (tbl as table) as table => Table.SelectColumns( tbl,
    List.Select( Table.ColumnNames( tbl ),
            each not List.IsEmpty (
                List.RemoveMatchingItems(
                    Table.Column( tbl, _),
                    {null, ""}
                )
            )
        )
    ),
```

Sekarang kita sudah memiliki fungsi kustom tersebut, saatnya memperbarui logika yang ditetapkan pada langkah *NoEmptyCols*. Di dalam bilah rumus, ubah ekspresi menjadi:

```
fxNoEmptyCols( NoEmptyRows )
```

Ekstrak Data *Header*

Sebelum menghapus semua baris tajuk dokumen (lihat Gambar 2.18), informasi penting perlu diekstrak dari tajuk. Kata terakhir pada baris kedua, *Name2*, adalah *Property ID*, yang harus disertakan dalam kolom baru, dalam hasil akhir:

	Column3	Column4
1	Daily by Date Performance Data - Preliminary	*null*
2	My Property: Business Name2	*null*
3	Comp Set: Location4 #444444, Location5 #555555, Location 6 #66666...	*null*
4	Job Number: 37165322 Staff: User1 Created: May 15, 2023 Curr...	*null*

Gambar 2. 18 Menampilkan baris tajuk dokumen

Saat merancang strategi untuk mengekstrak *Property ID* tersebut, penting untuk menghindari referensi bidang yang dikodekan secara keras, karena konsistensi di seluruh file tidak dijamin. Mengandalkan referensi yang

dikodekan secara keras dapat menyebabkan kesalahan atau mengembalikan nilai yang salah.

Ada dua langkah untuk mengambil ID Properti ini: pertama, ekstrak seluruh string dari header. Kedua, ekstrak kata terakhir dari string tersebut. Ekspresi record berguna di sini karena memungkinkan Anda untuk melihat ke dalam dan meninjau hasil antara selama pengembangan atau pada tahap selanjutnya saat memecahkan masalah. Buka Editor Lanjutan, buat variabel baru di bagian atas, dan beri nama GetPropertyID:

1) Inisialisasi ekspresi record: [ ].
2) Panggil bidang pertama *PropertyString* dan tetapkan ekspresi untuk mendapatkan baris kedua dari tabel sebagai record: *NoEmptyCols{1}*.
3) Ubah record menjadi list dengan membungkus *Record.ToList(...)* di sekitarnya.
4) Ekstrak item list pertama dengan kemudian membungkus *List.First(...)* di sekitarnya.
5) Sertakan koma di akhir, untuk membuat bidang baru untuk record *GetPropertyID* ini.
6) Panggil bidang kedua *PropertyID* dan tetapkan ekspresi untuk membagi string: *Text.Split(PropertyString, " ")* Ini akan mengembalikan list.
7) Ekstrak item list terakhir dengan membungkus *List.Last(...)* di sekitarnya.
8) Kembalikan nilai *PropertyID* melalui akses field.

Berikut ini tampilan ekspresi record tersebut. Ini akan mengembalikan nilai teks *Name2*:

```
GetPropertyID = [
    PropertyString = List.First( Record.ToList(NoEmptyCols{1})),
    PropertyID = List.Last( Text.Split( PropertyString, " "))
][PropertyID],
```

Setelah PropertyID berhasil diekstraksi, kita dapat memasukkan nilai ini ke dalam keluaran akhir di tahap selanjutnya dalam proses transformasi data.

Hapus Baris

Informasi header dan footer diposisikan di kolom pertama tabel, dan semua sel yang berdekatan yang memanjang ke paling kanan kosong. Kita dapat memanfaatkan pola yang sudah dikenal untuk menghilangkan baris-baris ini karena prosesnya mirip dengan NoEmptyRows, tetapi kita akan tetap menggunakan langkah itu sebagaimana adanya dan di tempatnya untuk tujuan mengekstrak PropertyID:

1) Salin seluruh nilai argumen kedua dari langkah *NoEmptyRows: each not List.IsEmpty(List.RemoveMatchingItems(Record.FieldValues(_), {"", null}))*.
2) Kemudian, lanjutkan ke langkah terakhir di panel **Applied Steps**, *NoEmptyCols*, dan tempatkan filter pada bidang mana pun. Itu akan menghasilkan langkah baru dan beberapa kode. Ganti argumen kedua dalam langkah ini dengan kode yang disalin sebelumnya dari *NoEmptyRows*.
3) *Record.FieldValues(_)* bagian dalam mengembalikan list nilai bidang dalam urutan yang sama dengan urutan kolom dalam tabel. Untuk menghilangkan nilai dari kolom pertama, bungkus *List.Skip(...)* di sekitar ekspresi ini.
4) Ubah nama langkah *RemoveRows*.
5) Di bagian bawah, tersisa dua baris secara total. Cukup pilih **Remove Rows**, lalu **Remove Bottom Rows** dari tab **Home** untuk mengecualikannya, dan ubah nama langkah ini menjadi *GetDataRange*.

Meskipun diformat di sini, sintaksis gabungan untuk RemoveRows akan terlihat seperti ini:

```
Table.SelectRows(NoEmptyCols,
    each not List.IsEmpty(
        List.RemoveMatchingItems(
            List.Skip( Record.FieldValues(_)), {"", null}
        )
    )
)
```

Proses pengembangan berjalan dengan baik, namun untuk menjamin data dari setiap berkas ditempatkan secara akurat ke dalam kolom yang sesuai selama operasi penambahan, berkas-berkas tersebut harus memiliki nama yang cocok.

Header Baru

Tiga baris pertama dari tabel perantara ini berisi tajuk (lihat Gambar 2.19). Dalam buku kerja Excel, tajuk sering ditempatkan di dalam sel gabungan. Akibatnya, satu sel akan berisi nilai ini dan yang lainnya akan kosong. Ada lima langkah umum untuk mengatasi hal ini: **Transpose** tabel, **Fill Down, Merge Columns, Transpose** kembali, dan **Promote Headers**. Semua langkah ini dapat dilakukan melalui UI, tanpa memerlukan pengodean.

	Column3	Column4	Column6
1	*null*	*null*	Occupancy
2	Date	DOW	This Year
3	*null*	*null*	My Prop
4	05/22/2023	Mon	*96,84210526*
5	05/22/2023	Tue	*97,89473684*

Gambar 2. 19 Tampilan sebagian dari meja perantara

Namun, ada beberapa alternatif. Kita akan menyediakannya di sini. Ini memerlukan pemisahan tabel dan transformasi baris tajuk secara terpisah dari baris data, dikombinasikan dengan trik untuk mengisi kolom di sebelah kanan dengan mengubah setiap baris tabel, sebuah record, menjadi tabel dan menjalankan pengisian ke bawah. Proses itu didemonstrasikan dalam Bab 8, Bekerja dengan Struktur Bersarang.

1) Pada tab **Home**, pilih **Keep Rows**, lalu **Keep Top Rows**, dan masukkan 3. Di dalam bilah rumus, kode tersebut berbunyi *Table.FirstN(GetDataRange,3)*.
2) Letakkan kursor Anda tepat setelah tanda sama dengan di bilah rumus dan masukkan *Table.TransformRows(*.

3) Letakkan kursor Anda di akhir, masukkan koma, ekspresi *each* diikuti oleh *Record.ToTable(_)*, dan tambahkan tanda kurung tutup ke Table.TransformRows untuk meninjau hasil antara.
4) Untuk mengisi nilai dalam kolom Nilai, bungkus *Table.FillDown(..., {"Value"})* di sekelilingnya.
5) Ekstrak kolom *Value* sebagai list dengan menambahkan [*Value*] tepat setelah tanda kurung tutup fungsi *Table.FillDown*. Ini mengembalikan list, dengan list bertingkat untuk setiap baris, seperti yang ditunjukkan pada Gambar 14.20.
6) Ubah nama langkah *GetHeaderRows*:

```
Table.TransformRows(
    Table.FirstN(GetDataRange,3),
    each Table.FillDown( Record.ToTable(_), {"Value"})[Value]
)
```

	List
1	List
2	List
3	List

Gambar 2. 20 *GetHeaderRows* menghasilkan list list.

Dengan pendekatan modular, sekarang adalah waktu yang tepat untuk memecah logika menjadi langkah baru. Sisipkan langkah manual dengan mengeklik **fx** di depan bilah rumus:

1) Nilai perlu digabungkan menurut posisinya: *List.Zip(GetHeaderRows)*.
2) Sekarang semua item list dapat diubah: *List.Transform(..., each Text.Combine(_, " | "))* untuk mengembalikan satu nilai teks untuk setiap kolom, yang digambarkan pada Gambar 2.21.
3) Ubah nama langkah ini menjadi *Headers*:

```
List.Transform(
    List.Zip( GetHeaderRows ),
    each Text.Combine( _, " | ")
)
```

	List
1	Date
2	DOW
3	Occupancy | This Year | My Prop
4	Occupancy | This Year | Comp Set
5	Occupancy | % Chg | My Prop

Gambar 2. 21 List header, menampilkan lima item pertama

Akhirnya, saatnya untuk mendapatkan bagian data dan menetapkan tajuk berikut ke kolom:

1) Klik **fx** untuk membuat langkah manual dan ganti *Headers* dengan *GetDataRange* di dalam bilah rumus. Ini akan mengembalikan nilai yang terkait dengan variabel/langkah tersebut.
2) Lewati tiga baris teratas: *Table.Skip(GetDataRange, 3).*
3) Ganti nama kolom, gunakan *List.Zip* untuk membuat list ganti nama: *Table.RenameColumns(…., List.Zip({Table.ColumnNames(GetDataRange), Headers})).*
4) Ubah nama langkah *GetTable*.

```
Table.RenameColumns(
    Table.Skip( GetDataRange, 3),
    List.Zip({Table.ColumnNames(GetDataRange), Headers} )
)
```

Mengembalikan ke langkah sebelumnya di tengah-tengah kueri sama sekali bukan masalah. Namun, penting untuk berhati-hati karena interaksi kueri setelahnya seperti memasukkan atau mengubah langkah melalui UI dapat mengganggu referensi variabel dalam ekspresi. Solusi praktis untuk meminimalkan risiko tersebut adalah dengan menempatkan langkah-langkah ini di lokasi yang lebih aman, seperti di atas langkah Sumber, tempat interaksi kueri tidak mungkin terjadi, mirip dengan penempatan fungsi kustom dan ekspresi record akses bidang.

Untuk mengubah posisi langkah *GetHeaderRows* dan *Headers*, buka jendela **Advanced Editor**. Dari sana, pilih **Cut**, lalu **Paste** langkah-langkah

ini pada baris baru yang disisipkan di antara klausa let dan ekspresi awal. Penataan ulang ini memastikan langkah-langkah tersebut tidak akan terpengaruh oleh perubahan yang dibuat dari UI di tahap selanjutnya. Setelah mengubah posisi langkah-langkah ini di Editor Lanjutan, urutan langkah akan menyerupai yang ditunjukkan pada Gambar 2.22.

```
let
    GetHeaderRows = Table.TransformRows( Table.FirstN(GetDataRange,3), each
        Table.FillDown( Record.ToTable(_), {"Value"})[Value]
    ),
    Headers = List.Transform( List.Zip( GetHeaderRows ), each Text.Combine( _, " | ")),
    GetPropertyID = [
            PropertyString = List.First( Record.ToList(NoEmptyCols{1})),
            PropertyID = List.Last( Text.Split( PropertyString, " "))
        ][PropertyID],
    fxNoEmptyCols = (tbl as table) as table => Table.SelectColumns( tbl,
        List.Select( Table.ColumnNames(tbl),
            each not List.IsEmpty ( List.RemoveMatchingItems( Table.Column( tbl, _), {null, ""})) )
        ),
    Source = Excel.Workbook( BinaryFile ),
```

Gambar 2. 22 Langkah-langkah yang diposisikan ulang di Editor Lanjutan

Langkah Tipe yang Diubah akan menjadi yang teratas dalam semua jajak pendapat pada langkah yang menimbulkan kesalahan. Mungkin tidak adil, karena mendefinisikan tipe data dimaksudkan untuk membantu menghindari kesalahan dan memastikan hasil yang akurat.

Tipe Data

Sebaiknya tetapkan tipe kolom selambat mungkin, mendekati akhir proses pengembangan kueri. Saat menentukan tipe data, Anda juga dapat menentukan lokal. Ini memungkinkan interpretasi tanggal dan angka yang lebih akurat. Tipe data ditetapkan pada tingkat kolom/bidang. Menetapkan tipe data kolom melalui UI memicu konversi tipe ke tipe yang dapat diubah menjadi null, dan memanggil fungsi *Table.TransformColumnTypes*.

Mari tetapkan tipe untuk kolom Tanggal. Klik **ABC123** di sebelah kiri nama kolom dan tetapkan ke **date**. Itu menghasilkan sintaksis berikut:

```
Table.TransformColumnTypes(ExpandContent,{{"Date", type date}})
```

Budaya dapat ditetapkan sebagai parameter ketiga. Berikan en-US untuk Amerika Serikat:

```
Table.TransformColumnTypes(ExpandContent,{{"Date", type date}}, "en-US")
```

Menurut spesifikasi file hasil yang diharapkan, kita hampir menyelesaikan fase persiapan data awal ini. Semua kolom harus diubah menjadi teks. Meskipun mungkin ada diskusi tentang kesesuaian tipe data tersebut, perlu dicatat bahwa nilai teks dapat diubah menjadi tipe data primitif apa pun – pilihan yang aman.

Periksa ekspresi di dalam bilah rumus dengan saksama. Penting untuk menyadari bahwa kita berurusan dengan struktur bersarang: list di dalam list yang formatnya adalah {namakolom, namatipe}. Dan harus ada satu list untuk setiap kolom yang tipenya akan ditetapkan. Pola ini dapat dibuat dinamis dan lebih ringkas.

1) Di dalam bilah rumus, hapus struktur bersarang *{{"Date", type date}}*.
2) Masukkan fungsi *List.Transform(*.
3) Berikan list Headers sebagai argumen pertama. Ini berisi semua nama kolom.
4) Ubah setiap nama kolom ke format *{ columnName, typeName }*, seperti ini: *each {_, type text}*. Sertakan tanda kurung tutup pada fungsi.

Meskipun diformat di sini, seperti inilah tampilan ekspresinya:

```
Table.TransformColumnTypes( GetTable,
    List.Transform( Headers, each {_, type text}), "en-US"
)
```

Setiap kolom kini telah diberi tipe data teks, yang menampilkan ikon ABC di depan namanya.

Menambahkan *Field*

Langkah terakhir dalam proses transformasi ini adalah memasukkan *Property ID*, yang menetapkan nilai *GetPropertyID* ke setiap baris dalam tabel. Cukup ikuti langkah-langkah berikut:

1) Navigasi ke **Add Column** dan pilih **Custom Column**.
2) Berikan nama kolom: *Property ID*.

3) Masukkan referensi *GetPropertyID* di bagian rumus.
4) Ubah nama langkah menjadi *InsertPropertyID*.

Di dalam bilah rumus, tetapkan tipe ke kolom ini dengan memasukkan koma sebelum tanda kurung tutup, diikuti dengan *type text*:

```
Table.AddColumn(SetColTypes, "Poperty ID", each GetPropertyID, type text)
```

Data dari contoh berkas ini kini telah dibentuk persis seperti berkas *ExpectedCombined*. Sekarang saatnya mempertimbangkan apakah dan bagaimana menangani berkas kosong dan kesalahan.

Kondisi File Kosong

Tentukan apakah tindakan pada file kosong diperlukan untuk tujuan pengoptimalan kueri atau pelaporan. Ini tidak sulit untuk diterapkan. Perlu diingat bahwa jika tidak ada intervensi yang dilakukan, kesalahan akan muncul, tetapi akan dibahas lebih lanjut nanti.

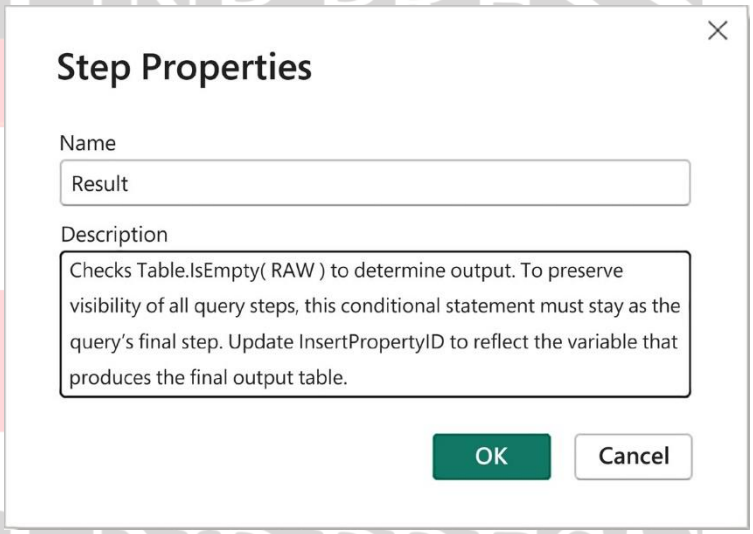
Untuk persyaratan kita, pelaporan aktif pada file kosong tidak diperlukan. Namun, pemantauan dan penilaian apakah ada modifikasi yang perlu dilakukan untuk mengoptimalkan kueri dapat bermanfaat. Jika kriteria identifikasi data dari langkah RAW gagal, tabel kosong dikembalikan, yang memberikan peluang untuk mengubah keluaran kueri. Berikut caranya:

1) Klik **fx** di depan bilah rumus untuk menyisipkan langkah manual.
2) Masukkan ekspresi ini: *= if Table.IsEmpty(RAW) then "File is empty." else InsertPropertyID*
3) Ubah nama langkah tersebut *Result*:

```
if Table.IsEmpty( RAW ) then "File is empty." else InsertPropertyID
```

Pernyataan kondisional ini bertindak seperti sakelar, yang menentukan apakah akan mengeluarkan string teks atau nilai tabel. Dengan menjadikannya sebagai langkah terakhir dalam kueri, seluruh rangkaian operasi transformasi akan tetap terlihat dan dapat diakses dari panel **Applied Steps**.

Jika tidak berurutan, **Applied Steps** akan menampilkan semua operasi yang diringkas menjadi satu langkah tunggal. Tidak ada persyaratan teknis untuk menjadikan variabel Result sebagai langkah kueri terakhir, karena tidak ada dampak pada cara pengoperasiannya atau struktur kode M. Meskipun demikian, penempatan tersebut memengaruhi pengalaman UI secara signifikan, yang memengaruhi pengembangan dan serah terima kueri di masa mendatang. Oleh karena itu, sebaiknya Anda mendokumentasikan pilihan strategis ini dan fungsinya dalam sebuah komentar. Klik kanan nama langkah **Result**, pilih **Properties**, dan masukkan **Description**, seperti yang diilustrasikan pada Gambar 2.23. Ikon informasi akan ditampilkan di bagian **Applied Steps** setelahnya.



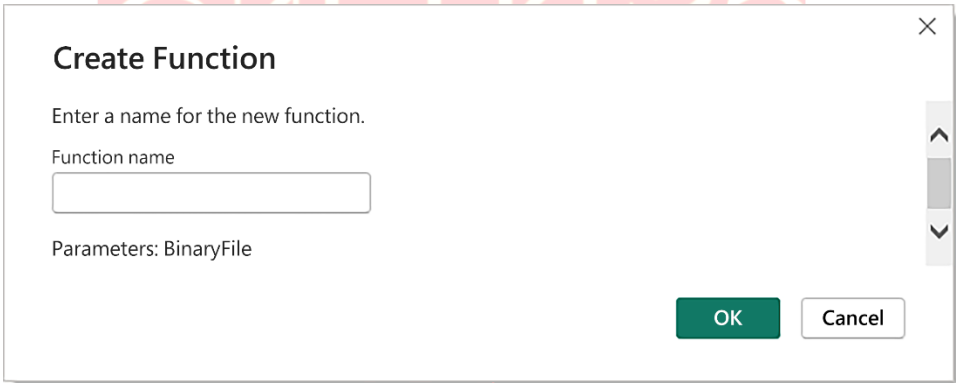
Gambar 2. 23 Sisipkan komentar untuk dokumentasi

Query *TransformData* berisi seluruh proses transformasi data. Query ini telah dikembangkan dan diterapkan pada satu file sampel. Namun, untuk memproses semua file dengan cara yang sama persis, query ini harus diubah menjadi query fungsi yang dapat dipanggil pada semua file yang dipilih dalam folder tersebut.

Query, Buat Fungsi

Anda dapat mengubah kueri menjadi kueri fungsi dengan bantuan UI. Sangat penting untuk memiliki pengaturan modular karena opsi **Create Function** kueri akan meminta untuk memberikan parameter untuk setiap argumen yang diteruskan ke fungsi. Dalam kasus ini, hanya ada satu input, **BinaryFile** yang sudah disiapkan sebagai parameter.

Klik kanan **TransformData** dan pilih **Create Function**. Kotak dialog akan muncul, meminta Anda untuk memasukkan nama untuk fungsi baru. Seperti yang diilustrasikan pada Gambar 2.24, masukkan *fxCollectData*. Tindakan ini menghasilkan kueri fungsi yang disebut *fxCollectData*, dan menempatkan parameter **BinaryFile** dan **TransformData** dalam folder yang disebut *fxCollectData*.



Gambar 2. 24 Dialog Buat Fungsi

Atur kueri Anda dengan memindahkan *FolderLocation* dan *BinarySample* ke folder *fxCollectData*. Ini dapat dilakukan dengan menyeret dan melepaskannya atau dengan mengklik kanan dan memilih **Move to group**.

Kembali ke *Query1*, ubah namanya menjadi **CombinedData**, dan pertahankan kolom *Name* dan *Content*. Ini dapat dilakukan melalui UI dengan memilihnya dan memilih **Remove Columns | Remove Other Columns**, atau, untuk kode yang lebih ringkas dan bersih, dengan menerapkan proyeksi, menambahkan *[[Name], [Content]]* di akhir ekspresi *SelectFiles* di dalam

bilah rumus. Untuk langkah berikutnya, kita akan menganggap proyeksi digunakan.

Ada beberapa metode untuk mengubah nilai terstruktur dalam tabel. Ini dibahas dalam Bab 4 Jilid 2, Bekerja dengan Struktur Bersarang. Di sini, kita akan memanfaatkan fungsi *Table.TransformColumns*:

1) Klik **fx** di depan bilah rumus untuk memasukkan langkah manual.
2) Masukkan fungsi di antara tanda sama dengan dan nama variabel: *Table.TransformColumns(*.
3) Sebagai argumen kedua, ia mengharapkan list operasi transformasi: *{"Content", each fxCollectData(_)}*.
4) Tambahkan tanda kurung penutup ke fungsi *Table.TransformColumns* dan ganti nama langkah ini menjadi *InvokedFunction*:

```
Table.TransformColumns( SelectFiles, {"Content", each fxCollectData(_)})
```

	Name	Content
1	Daily_1.xlsx	Table
2	Daily_2.xlsx	Table
3	Daily_2a.xlsx	File is empty.
4	Daily_3.xlsx	Table

Gambar 2. 25 Isi di kolom Konten aker memanggil *fxCollectData*

Setelah mengubah nilai di dalam kolom **Content** yang ditunjukkan pada Gambar 2.25, file kosong akan muncul. Apa pun itu, alur kerja pemantauan harus diterapkan selama sakelar file kosong diterapkan pada kueri *TransformData*.

Siapkan Pemantauan

Menyiapkan alur kerja pemantauan untuk menerima pemberitahuan tentang file kosong mudah dilakukan di layanan Power BI. Mulailah dengan menduplikasi kueri *CombinedData* dan ganti nama duplikat ini menjadi *EmptyFiles*. Kemudian, pisahkan semua baris yang relevan dengan

menerapkan filter. Lanjutkan dan ganti nama langkah filter ini menjadi *EmptyFiles*:

```
Table.SelectRows(InvokedFunction, each [Content] = "File is empty.")
```

Karena kueri ini dimaksudkan untuk memudahkan pemantauan, sebaiknya sertakan lokasi file dalam output. Pilih langkah Source. Ini menunjukkan kolom yang berisi nilai ini. Namanya Folder Path tetapi sebelumnya tidak disertakan. Sekarang pilih langkah *SelectFiles*. Di sini, kita telah menerapkan proyeksi untuk membatasi kolom. Oleh karena itu, untuk menyertakan kolom ini dan menempatkannya di depan sebagai kolom pertama dalam tabel, masukkan *[Folder Path]*, seperti yang ditunjukkan di sini:

```
[[Folder Path], [Name], [Content]]
```

Kueri *EmptyFiles* ini dapat dimuat ke dalam model data, sebagai tabel tersembunyi yang berdiri sendiri. Untuk mengaktifkan pemberitahuan, buat pengukuran DAX sederhana, seperti ini:

```
EmptyFileCount_ModelName = COUNTROWS( EmptyFiles )
```

Pengukuran ini dapat ditempatkan dalam bentuk visual yang menyediakan kemampuan untuk menyiapkan pemicu ambang batas dalam layanan Power BI, seperti kartu. Setelah semuanya selesai dan nilai ambang batas terlampaui, Anda atau tim Anda dapat diberi tahu secara otomatis untuk melakukan penyelidikan.

Penyempurnaan

Selanjutnya, kembali ke kueri *CombinedData*. Di sini, selama sakelar berkas kosong diterapkan ke kueri *TransformData*, baris yang memenuhi kriteria tersebut harus dikecualikan. Terlepas dari apakah ada baris yang saat ini memenuhi kondisi tersebut, sebut saja *NoEmptyFiles*:

```
Table.SelectRows(InvokedFunction, each [Content] <> "File is empty.")
```

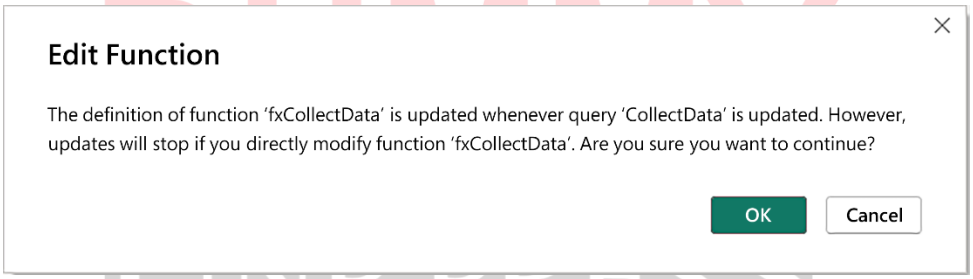
Jika langkah *NoEmptyFiles* ini diabaikan, menemukan file kosong akan menimbulkan kesalahan. Selain itu, memproses file secara massal dapat mengungkap masalah lain dari waktu ke waktu, yang berpotensi menyebabkan munculnya kesalahan. Menggabungkan strategi secara proaktif untuk mengatasi skenario ini dapat dibenarkan. Gbapter 12, Penanganan Kesalahan dan Debugging, menyediakan metode untuk menangkap kesalahan untuk pelaporan, menyediakan sarana untuk menerapkan alur kerja yang serupa tetapi terpisah dengan pengelolaan file kosong.

Jika file yang menyebabkan kegagalan diidentifikasi, logika transformasi tambahan kemungkinan akan diperlukan untuk memperbaiki masalah yang mendasarinya. Berikut ini adalah tampilan proses modifikasi atau pembaruan kode M yang diterapkan melalui fungsi kustom *fxCollectData*:

1) Mulailah dengan menentukan file yang menimbulkan kesalahan selama pemrosesan.
2) Navigasi ke kueri *BinarySample* dan pilih langkah *SelectFiles*.
3) Telusuri kolom Konten untuk file yang diidentifikasi.
4) Hapus langkah *Imported Excel workbook* saat dibuat secara otomatis, untuk memperlihatkan file yang dipilih sebagai file di panel **Preview**.
5) Pindah ke kueri *CollectData* dan identifikasi langkah tempat kesalahan terjadi.
6) Sertakan logika tambahan atau perbarui kode yang sesuai, untuk mengatasinya.
7) Pilih kueri *CombinedData*, dan konfirmasikan tidak ada masalah yang tersisa. Jika kesalahan masih ada, ulangi langkah-langkah ini hingga semua masalah teratasi.

Dalam kondisi apa pun Anda tidak boleh mengubah kueri *fxCollectData* secara langsung; kueri ini ditautkan untuk mencerminkan perubahan apa pun yang dibuat pada *CollectData*. Namun, tautan tersebut akan terputus setelah *fxCollectData* dibuka di Editor Lanjutan. Untungnya, pengguna akan dimintai

pesan peringatan sebelumnya, seperti yang ditunjukkan pada Gambar. Tekan **Cancel** untuk membatalkan. Demikian pula, jika tidak dimintai peringatan ini, yang menunjukkan koneksi telah terputus, modifikasi apa pun pada *CollectData* tidak akan disinkronkan ke *fxCollectData*. Dalam kasus seperti itu, untuk membuat ulang kueri fungsi yang ditautkan, fungsi baru harus dibuat dari kueri *CollectData* dan langkah InvokedFunction perlu diperbarui.



Gambar 2. 26 Pesan peringatan saat mencoba membuka Editor Lanjutan untuk kueri *fxCollectData*

Tanpa menyertakan pelaporan kesalahan, Anda mungkin ingin kesalahan terus berlanjut dan menyebabkan kegagalan untuk menandakan adanya masalah yang perlu ditangani. Untuk saat ini, kita akan menganggap pelaporan kesalahan telah diterapkan, dan aman untuk mengecualikan semua baris dengan kesalahan. Pilih **Remove Rows**, lalu **Remove Errors** pada tab **Home**, dan ganti nama langkah *RemoveErrors*. Konten dari semua file sekarang dapat digabungkan.

Gabungkan File

Klik opsi perluas kolom, di tajuk kolom Konten. Ini memungkinkan Anda untuk memilih dan memperluas kolom dari tabel bersarang. Klik ikon panah samping. Secara default, semua bidang akan dipilih dan, saat diaktifkan, nonaktifkan opsi **Use original column name as prefix** di bagian bawah sebelum mengeklik **OK**.

Ini akan menambahkan langkah baru ke kueri. Di dalam bilah rumus, Anda dapat melihatnya meneruskan dua list berkode keras ke fungsi

Table.ExpandTableColumn. List pertama berisi *columnNames* untuk diperluas dari setiap tabel bersarang dan list kedua, *newColumnNames*, untuk ditetapkan ke tabel induk. Ini membantu menghindari konflik antara nama di kolom yang ada dan yang baru. Namun, risiko file akan tiba-tiba menyertakan bidang yang disebut Nama atau Konten sangat rendah, oleh karena itu list akhir tersebut dapat dihapus. Untuk list pertama, itu dapat diganti dengan fungsi yang menyediakan semua nama kolom dari kueri *CollectData* secara dinamis. Setelah modifikasi ini, ekspresi dalam bilah rumus akan terlihat seperti ini – ganti nama langkah itu *ExpandAllColumns*:

```
Table.ExpandTableColumn(RemoveErrors, "Content",
    Table.ColumnNames(CollectData)
)
```

Setelah berhasil menggabungkan data dari ketiga berkas, masing-masing sesuai dengan data selama seminggu, menghasilkan satu tabel komprehensif yang terdiri dari 21 baris dan 22 kolom. Semua nilai diformat sebagai teks, mencerminkan berkas *ExpectedCombined.xlsx*. Dengan semua data yang diperlukan dikompilasi dan disusun sesuai dengan spesifikasi yang telah ditetapkan, kita berada dalam posisi yang baik untuk melangkah maju dan memulai transformasi untuk fase analitik.

D. Ringkasan

Dalam bab ini, kita telah menyoroti aspek praktis dari manipulasi dan persiapan data dengan berfokus pada dua topik yang sangat berbeda: mengekstraksi pola tetap dan menggabungkan data dari beberapa file Excel. Meskipun keduanya terasa sangat berbeda, keterampilan ini, di antara banyak keterampilan lainnya, sangat penting untuk mengubah data mentah menjadi informasi yang mendalam dan dapat ditindaklanjuti.

Dengan memecah proses menjadi bagian-bagian yang dapat dikelola, menggarisbawahi pentingnya pendekatan metodis serta pemecahan masalah yang kreatif, kita bertujuan untuk menawarkan kumpulan ide, yang memicu

imajinasi Anda untuk mengatasi tantangan apa pun. Bab berikutnya berfokus pada area penting lainnya: mengoptimalkan kinerja. Anda akan mempelajari tentang faktor-faktor yang memengaruhi, dan metode untuk meningkatkan, kinerja kueri.

E. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan pentingnya penanganan kesalahan dalam Power Query. Bagaimana kesalahan dapat memengaruhi hasil yang diharapkan dari kueri?	10
2.	Deskripsikan berbagai jenis kesalahan yang mungkin ditemui saat bekerja dengan M. Berikan contoh untuk masing-masing jenis kesalahan.	10
3.	Bagaimana Anda dapat menggunakan fungsi try dan otherwise untuk menangani kesalahan dalam M? Berikan contoh kode yang relevan.	10
4.	Diskusikan teknik debugging yang efektif dalam Power Query. Apa langkah-langkah yang bisa diambil untuk mengidentifikasi dan memperbaiki kesalahan?	10
5.	Apa yang dimaksud dengan error handling strategies dalam konteks Power Query? Berikan penjelasan tentang minimalisasi kesalahan.	10
6.	Jelaskan bagaimana Anda dapat mengimplementasikan logika kontrol alur dalam kueri M untuk menangani kesalahan tertentu.	10
7.	Analisis perbedaan antara error handling dan exception handling dalam konteks M. Bagaimana keduanya dapat diterapkan dalam kueri?	10
8.	Buatlah skenario di mana penanganan kesalahan penting. Apa langkah-langkah yang harus diambil untuk memastikan kueri tetap berjalan meskipun terjadi kesalahan?	10
9.	Diskusikan bagaimana penggunaan komentar dalam kode M dapat membantu dalam penanganan kesalahan. Apa manfaatnya bagi pengembang?	10
10.	Berikan contoh studi kasus di mana penanganan kesalahan dalam M telah menyelamatkan proses analisis data. Apa hasil yang diperoleh dari penerapan strategi penanganan kesalahan yang baik?	10

BAB 3

Mengoptimalkan Kinerja

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Memahami penggunaan memori saat mengevaluasi kueri
- Melipat kueri
- Rumus firewall
- Mengoptimalkan kinerja kueri
- Tips performa

A. Pendahuluan

Meskipun mengembalikan data yang akurat itu penting, kecepatan kueri Anda sangat memengaruhi pengalaman pengguna. Waktu penyegaran yang lama untuk kumpulan data Anda dapat mengakibatkan batas waktu, dan menunggu hasil transformasi Anda di editor Power Query dapat membuat frustrasi. Dalam bab ini, kita akan membahas strategi untuk mengoptimalkan kinerja kueri Anda. Pertama-tama kita akan melihat penggunaan memori saat mengevaluasi kueri dan bagaimana penggunaan memori yang terlalu banyak memperlambat kueri Anda. Kemudian kita akan melihat berbagai strategi untuk mencegah hal ini terjadi.

Kita akan melihat pentingnya pelipatan kueri dan bagaimana Anda dapat memastikan bahwa sumber data memproses sebanyak mungkin transformasi. Kemudian, kita akan membahas firewall rumus, alat yang dirancang untuk melindungi kueri Anda dari pemaparan informasi sensitif. Kita akan menjelaskan apa yang memicu kesalahan yang terkait dengan ini dan membahas pengaruhnya terhadap kueri Anda. Selain itu, kita akan memeriksa bagaimana menyimpan data dalam memori (buffering) dapat membuat kueri Anda berjalan lebih cepat. Terakhir, kita akan menawarkan kiat untuk memilih sumber data, dengan mempertimbangkan pertimbangan ini. Kita akan membahas topik-topik berikut:

- Memahami penggunaan memori saat mengevaluasi kueri
- Pelipatan kueri
- Firewall rumus
- Mengoptimalkan kinerja kueri
- Kiat-kiat kinerja

1. Kasus Pemantik Berfikir Kritis: Pengoptimalan Kinerja di Power Query

Sebuah perusahaan retail menggunakan Power Query untuk mengolah data penjualan bulanan. Mereka menemukan bahwa proses pemuatan data

semakin lambat seiring bertambahnya jumlah data. Tim analisis data diminta untuk meningkatkan kinerja kueri agar lebih efisien.

Pertanyaan untuk Diskusi:

- Identifikasi Masalah: Apa yang mungkin menyebabkan penurunan kinerja kueri di Power Query? Sebutkan beberapa faktor yang dapat mempengaruhi kecepatan pemrosesan.
- Teknik Pengoptimalan: Apa saja teknik yang dapat diterapkan untuk memperbaiki kinerja kueri ini? Diskusikan minimal tiga metode yang mungkin efektif.
- Folding Query: Bagaimana konsep folding query dapat digunakan untuk meningkatkan efisiensi? Berikan contoh bagaimana ini dapat diterapkan dalam situasi ini.
- Pemilihan Tipe Data: Mengapa pemilihan tipe data yang tepat penting dalam konteks pengoptimalan kinerja? Bagaimana hal ini dapat diterapkan pada data penjualan yang sedang dianalisis?
- Pengujian Kinerja: Jika setelah menerapkan beberapa teknik pengoptimalan, kinerja kueri masih belum memadai, langkah apa yang harus diambil selanjutnya? Apa metode yang dapat digunakan untuk mendiagnosis masalah lebih lanjut?

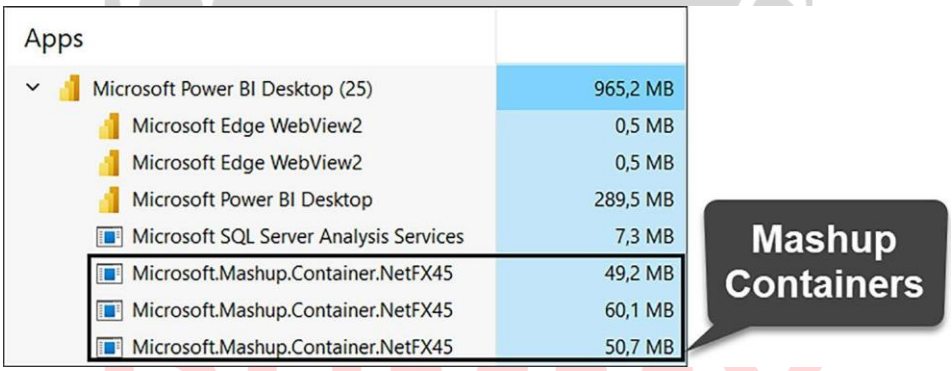
Tugas:

Diskusikan pertanyaan-pertanyaan ini dalam kelompok dan buat rencana tindakan untuk meningkatkan kinerja kueri di Power Query. Presentasikan solusi terbaik yang diusulkan kepada manajemen.

B. Memahami penggunaan memori saat mengevaluasi kueri

Mengambil data dan melakukan transformasi tidaklah gratis. Untuk menjalankan kueri, Power Query memerlukan memori. Beberapa pendekatan memerlukan lebih banyak memori daripada yang lain. Oleh karena itu, memahami cara meningkatkan penggunaan memori membantu saat mengoptimalkan kueri. Jadi, bagaimana cara kerjanya?

Kueri dieksekusi dalam **mashup container**, proses khusus yang bertanggung jawab untuk evaluasi kueri. Anda dapat melihat jumlah wadah mashup yang digunakan dengan membuka **Task Manager** dan membuka proses untuk Power BI Desktop:



Gambar 3. 1 Kontainer mashup muncul di Pengelola Tugas

Memori yang diberikan untuk setiap kontainer bergantung pada tempat Anda menjalankan Power Query. Lingkungan umum meliputi Power BI Desktop, layanan Power BI, dan gateway data lokal. Jadi, apa yang terjadi saat memori kontainer hampir habis?

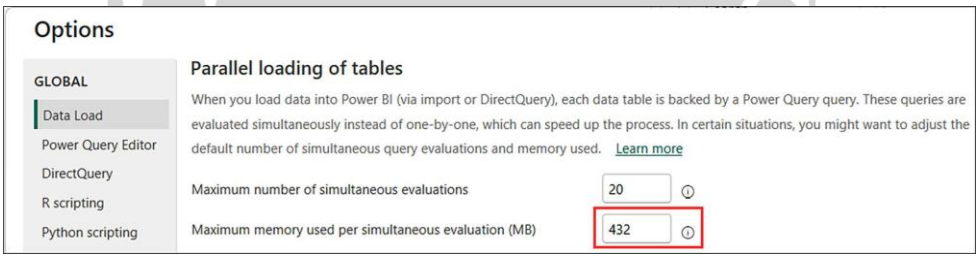
Saat kebutuhan memori kontainer gabungan melebihi batas yang dialokasikan selama evaluasi kueri, Power Query mulai melakukan paging (mentransfer) data kueri ke penyimpanan disk. Proses paging ini menyebabkan penurunan kinerja kueri yang signifikan, karena akses disk jauh lebih lambat daripada akses memori. Oleh karena itu, menyadari keterbatasan memori dan mengelola kueri sesuai kebutuhan penting untuk mengoptimalkan kinerja di Power Query.

1. Variasi dan Penyesuaian Batas Memori

Batas memori untuk setiap kontainer mashup bergantung pada lingkungan eksekusi, dan dalam beberapa kasus, Anda dapat menyesuaikannya. Berikut adalah lingkungan yang biasanya digunakan:

- **Mesin gateway**: Memori per kontainer dihitung secara otomatis untuk mesin yang menjalankan gateway lokal. Meskipun Anda tidak dapat menetapkan batasan apa pun sendiri, meningkatkan memori dan kecepatan mesin dapat meningkatkan waktu penyegaran kueri.
- **Layanan Power BI**: Anda tidak dapat mengubah batas memori di sini. Namun, jika Anda menggunakan Power BI versi premium, Anda mendapatkan lebih banyak sumber daya, yang dapat meningkatkan kinerja.
- **Power BI Desktop**: Anda dapat menyesuaikan berapa banyak memori yang digunakan di sini. Anda akan menemukan opsi ini di bawah **File > Options and Settings > Options**, di tab **Data Load**.

Untuk Power BI Desktop, pada saat penulisan, alokasi memori maksimum standar untuk setiap evaluasi kueri adalah 432 MB. Anda dapat menyesuaikannya di Power BI Desktop dengan membuka **File**, memilih **Options and Settings**, lalu memilih **Options**. Anda dapat mengubah pengaturan memori kueri di tab **Data Load**:



Gambar 3. 2 Anda dapat menetapkan jumlah memori maksimum untuk evaluasi kueri di Power BI Desktop

Ingat, memori maksimum yang dapat ditetapkan ke kueri bergantung pada total memori yang tersedia di komputer Anda.

Untuk mencapai eksekusi kueri yang lebih cepat di Power Query, strategi utama melibatkan pengelolaan penggunaan memori. Ini termasuk:

- Meminimalkan konsumsi memori saat mengimpor.
- Saat menggunakan memori, pastikan volume data sekecil mungkin. Selanjutnya, kita akan membahas beberapa strategi untuk membantu Anda melakukannya.

C. Lipatan Kueri (*Query Folding*)

Pelipatan kueri merupakan salah satu aspek terpenting untuk mengoptimalkan kinerja kueri Anda dan berguna bagi pengembang mana pun yang terhubung ke sumber data eksternal. Ini adalah proses di mana evaluasi kueri dialihkan ke sumber data itu sendiri.

Selama pelipatan kueri, mesin mashup Power Query mengonversi transformasi kueri ke dalam bahasa asli sumber data. Pendekatan ini menggabungkan beberapa langkah transformasi menjadi satu kueri efisien yang dapat dijalankan oleh sumber data.

Dengan demikian, beban komputasi transformasi ini akan ditransfer dari lingkungan lokal, tempat Power Query beroperasi, langsung ke sumber data, seperti database SQL. Transformasi yang tersisa yang tidak dapat dilipat akan dilakukan secara lokal.

Tujuan utama pelipatan kueri adalah untuk mengurangi penggunaan memori di Power Query dengan menggunakan daya komputasi sumber data untuk memproses kueri dan transformasi. Misalnya, anggaplah Anda terhubung ke tabel dalam database SQL dan ingin mengambil data hanya untuk minggu lalu.

Alih-alih mengimpor semua data terlebih dahulu lalu memfilternya, pelipatan kueri malah mengirimkan kueri yang tepat untuk mengambil hanya baris yang relevan. Hal itu memiliki dua manfaat:

1) **Mengoptimalkan penggunaan sumber daya**: Pelipatan kueri menggunakan kekuatan sumber data dengan mentransfer beban kerja

transformasi kueri langsung ke sumber data tersebut. Saat transformasi dijalankan di komputer lokal, diperlukan ekstraksi semua data dari sumber data melalui jaringan, yang kemudian diproses di mesin Power Query. Hal ini dapat menghabiskan banyak sumber daya dan waktu. Sebaliknya, sumber data Anda, biasanya server atau basis data, umumnya dapat menangani tugas-tugas ini dengan lebih cepat dan efisien daripada sistem lokal Anda.

2) **Meminimalkan transfer data jaringan**: Produk sampingan penting dari pendekatan ini adalah pengurangan transfer data melalui jaringan. Alih-alih menarik sejumlah besar data yang belum diproses ke lingkungan lokal Anda untuk transformasi, pelipatan kueri memastikan bahwa hanya data yang diperlukan dan diproses yang diambil. Ini berarti hanya data yang diperlukan (setelah memilih baris/kolom tertentu) yang dikirim melalui jaringan, yang menghemat bandwidth dan waktu yang diperlukan untuk transfer data.

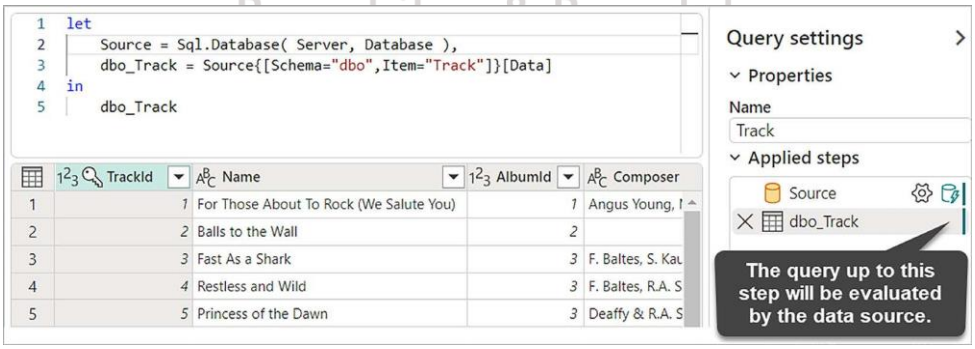
Selain manfaat utama ini, memahami pelipatan kueri sangat penting karena dua alasan lainnya. Pertama, pelipatan kueri sangat penting untuk menggunakan tabel mode Kueri Langsung dan Penyimpanan Ganda di Power BI. Mode ini bergantung pada pelipatan kueri untuk mengirim permintaan data langsung ke database.

Kedua, pelipatan kueri memainkan peran penting dalam skenario penyegaran inkremental. Penyegaran inkremental adalah fitur yang memungkinkan pemuatan data dari titik waktu tertentu. Misalnya, dalam tabel dengan jutaan baris yang mencakup 20 tahun, Anda hanya dapat memperbarui sebagian data terkini menggunakan penyegaran inkremental. Namun, ini hanya dapat dilakukan jika sumber data mendukung pelipatan kueri, yang memungkinkan kueri Anda mengambil hanya sebagian data yang diperlukan. Tanpa pelipatan kueri, seluruh kumpulan data harus diambil dan difilter secara lokal, yang jauh kurang efisien. Jadi, bagaimana cara kerja pelipatan kueri dalam praktiknya?

1. Pelipatan Kueri Dalam Aksi

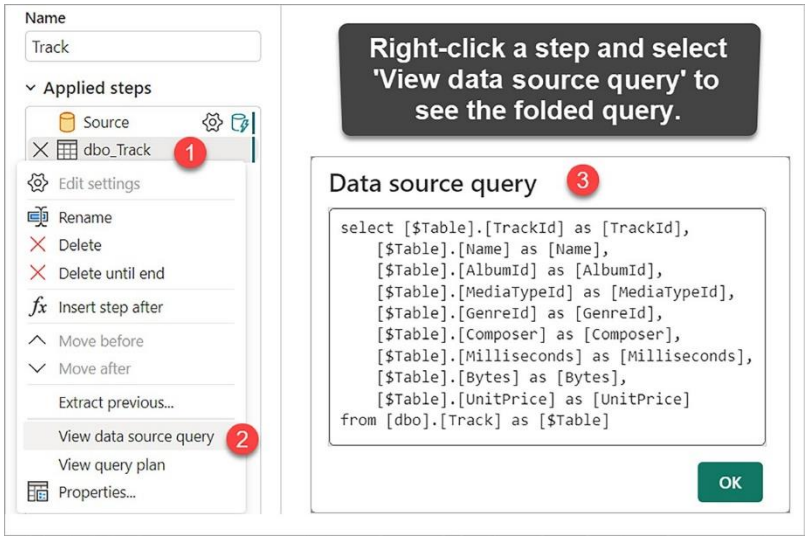
Untuk memahami pelipatan kueri, mari kita bahas skrip M. Skrip M membuat rencana terperinci untuk menangani data. Skrip ini memberi instruksi kepada Power Query tentang langkah apa yang harus diambil, dalam urutan apa, dan cara menggabungkan langkah-langkah ini secara efektif.

Setiap kali kita ingin mengakses sumber data, kita menggunakan fungsi pengaksesan data ke sumber data tersebut. Misalnya, kita terhubung ke database SQL. Gambar di bawah ini menggambarkan kueri yang terhubung ke tabel bernama Track:



Gambar 3. 3 Kueri yang terhubung ke database SQL

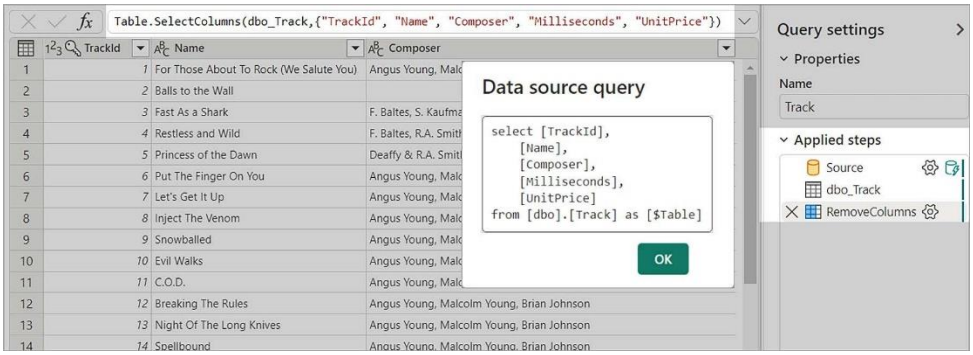
Dalam pengalaman Power Query yang baru, Anda dapat menemukan indikator di samping setiap langkah. Bila langkah tersebut memiliki **database icon with a lightning sign**, ini menunjukkan bahwa langkah tersebut akan dilipat (lebih lanjut tentang ini nanti). Untuk memeriksa apa yang dilakukan Power Query di balik layar, pilih langkah yang dapat dilipat, klik kanan, dan **View data source query**:



Gambar 3. 4 Saat kueri Lihat sumber data diaktifkan untuk suatu langkah, Anda dapat melihat kueri terlipat

Anda akan melihat layar yang menampilkan popup dengan kueri terlipat, kueri SQL asli. Artinya, server basis data menjalankan kueri ini.

Sekarang, mari kita buat beberapa perubahan pada kueri. Alih-alih menampilkan semua kolom, saya hanya memilih lima kolom. Dengan memeriksa kueri sumber data dari langkah **RemoveColumns** yang baru ini, kita sekarang mendapatkan kueri berbeda yang hanya memilih kolom yang relevan, seperti yang ditunjukkan pada tangkapan layar berikut:



Gambar 3. 5 Kueri sumber data ini mewakili pemilihan beberapa kolom

Pikirkan ini sebentar. Tanpa pelipatan kueri, Power Query harus mengimpor semua data (termasuk kolom yang tidak diperlukan). Setelah

mengimpor ini, kolom yang tidak relevan akan dibuang. Dengan kata lain, sumber daya yang tidak diperlukan akan digunakan untuk mengimpor lalu menghapus data.

Namun, berkat pelipatan kueri, Power Query mengirimkan kueri yang hanya mengambil kolom yang diperlukan. Pendekatan ini memiliki dua manfaat mendasar. Pertama, ini meningkatkan kinerja secara signifikan karena hanya data relevan yang ditransfer ke mesin. Kedua, ini memungkinkan pengguna yang mungkin tidak begitu paham dengan bahasa kueri sumber data (seperti SQL) tetap mendapatkan manfaat dari kueri sumber data yang dibuat secara otomatis.

Penting untuk dicatat bahwa tidak ada indikator eksplisit di antarmuka Power Query lama untuk menunjukkan apakah suatu langkah sedang dilipat. Untuk memeriksa apakah suatu langkah dilipat di lingkungan ini, klik kanan padanya dan cari opsi **Native Query**. Jika opsi ini tidak tersedia (berwarna abu-abu), ini menunjukkan bahwa tidak jelas apakah langkah itu dilipat. Ini bisa berarti pelipatan telah berhenti, atau konektor tidak yakin apakah langkah ini dilipat. Kita akan membahasnya lebih lanjut di bab ini.

Sekarang setelah Anda diperkenalkan dengan pelipatan kueri, Anda mungkin bertanya-tanya bagaimana Power Query memutuskan operasi mana yang dapat dilipat. Dengan kata lain, apa proses di balik evaluasi kueri dalam konteks ini?

2. Evaluasi Query

Sebagian besar kueri terdiri dari beberapa langkah yang diterapkan. Untuk mengevaluasi kueri, langkah-langkah berikut dilakukan:

1) **Pengambilan awal**: Mesin Power Query mengumpulkan kode M kueri. Mesin ini juga mengumpulkan kredensial sumber data dan tingkat privasi sumber data.

2) **Membaca metadata sumber data**: Mesin mengkueri sumber data untuk metadata. Langkah ini menentukan kapabilitas sumber data dan apakah operasi tertentu dapat dilipat.
3) **Hasilkan kueri sumber data**: Berdasarkan informasi yang diterima, mesin menentukan (menggunakan mekanisme pelipatan kueri) informasi apa yang dapat diekstrak dari sumber data, dan transformasi mana yang harus terjadi di dalam mesin Power Query. Berdasarkan hal ini, mesin menghasilkan kueri sumber data dan mengirimkan informasi ke mesin transformasi.
4) **Ambil data**: Mesin Power Query kemudian menerima data seperti yang diminta dari data, berdasarkan kueri sebelumnya.
5) **Jalankan yang tersisa**: Setiap transformasi yang tidak dapat dilipat sekarang dilakukan oleh mesin M pada data yang diambil.
6) **Muat data**: Data yang diproses ini dimuat ke tujuan yang ditentukan, seperti Power BI atau Excel.

Seperti yang kita katakan, fungsi utama pelipatan kueri di Power Query adalah untuk mengubah transformasi menjadi kueri sumber data yang dapat dijalankan oleh sumber data. Koneksi ke sumber data awalnya dibuat menggunakan fungsi dari keluarga fungsi akses data. Pelipatan kueri terutama menargetkan langkah-langkah transformasi yang segera mengikuti langkah yang terhubung ke data. Efektivitas pelipatan kueri dalam menerjemahkan transformasi ini menjadi kueri sumber data bergantung pada kemampuan konektor yang digunakan dengan sumber data.

3. Melipat, Tidak Melipat, dan Melipat Sebagian (*Folding, Not Folding, and Partial Folding*)

Sayangnya, konektor Power Query tidak cukup pintar untuk menerjemahkan semua kode M ke kueri sumber data. Beberapa operasi menyebabkan pelipatan kueri rusak. Nanti di bab ini, kita akan membahas strategi untuk mempertahankan pelipatan kueri. Dengan mengingat hal itu, hasil berikut mungkin terjadi:

- **Pelipatan kueri penuh (Full query folding)**: Semua transformasi terjadi di sumber data, dan mesin Power Query menerima output.
- **Pelipatan kueri sebagian (Partial query folding)**: Sebagian transformasi dilipat dan dikirim ke sumber data. Langkah-langkah yang tersisa yang tidak diterjemahkan diproses di mesin Power Query.
- **Tidak ada pelipatan kueri (No query folding)**: Ketika tidak ada langkah yang diterapkan dapat diubah ke bahasa sumber data, semua transformasi dilakukan di mesin Power Query. Mungkin konektor tidak mendukung pelipatan kueri atau transformasi tertentu tidak didukung.

Jika sumber data Anda berfungsi dengan bahasa kueri, kemungkinan besar sumber tersebut mendukung pelipatan kueri. Anda dapat memikirkan sumber seperti basis data, umpan OData, dan Direktori Aktif. Namun, sumber data seperti file Excel, CSV, atau sumber web tidak mendukung pelipatan kueri.

Seperti yang baru saja Anda pelajari, memahami cara kerja pelipatan kueri dalam kueri Power BI Anda penting untuk mengurangi penggunaan memori. Namun, hal itu akan sulit dilakukan jika Anda tidak mengetahui langkah mana yang dapat dilipat dan mana yang tidak. Untuk membantu dalam hal ini, Power Query menyediakan berbagai alat yang dapat memberi tahu Anda apakah transformasi dapat dilipat.

4. Alat Untuk Menentukan Kemampuan Melipat

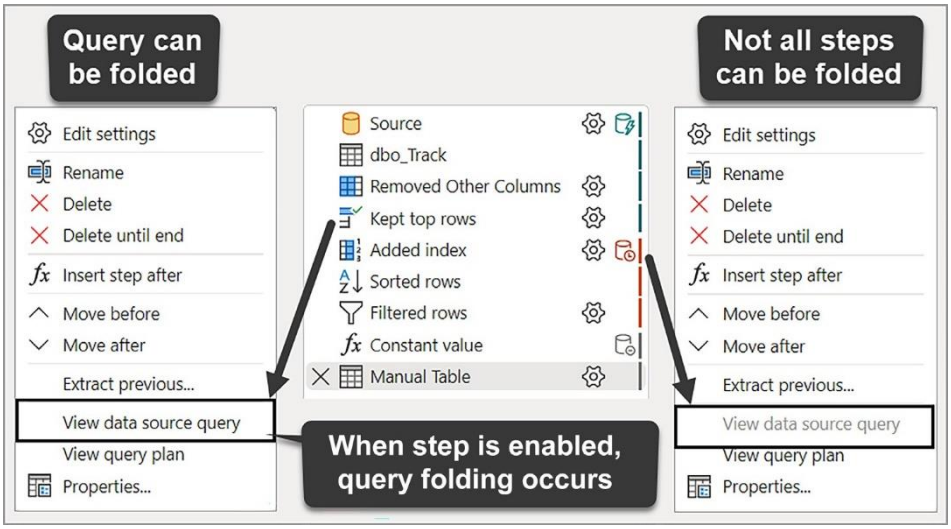
Power Query memiliki tiga alat utama yang memungkinkan Anda menentukan apakah suatu langkah dapat dilipat. Alat-alat ini adalah:

- Melihat kueri sumber data
- Indikator pelipatan langkah
- Rencana kueri

Di bagian berikut, kita akan membahas masing-masing alat dan menunjukkan cara menggunakannya.

Lihat Kueri Sumber Data

Seperti yang kita lihat sebelumnya, opsi **View data source query** adalah alat pertama untuk menentukan apakah suatu langkah dapat dilipat. Saat Anda mengklik kanan suatu langkah, Anda dapat menemukan opsi yang bertuliskan **View data source query**:



Gambar 3. 6 Lihat kueri sumber data

Saat fitur aktif, Anda dapat melihat kueri sumber data yang telah dibuat. Jika fitur tampak berwarna abu-abu, ini menunjukkan bahwa beberapa langkah dalam kueri Anda tidak dapat dilipat. Namun, ini tidak berarti bahwa seluruh kueri Anda tidak dapat dilipat; kueri tersebut dapat dilipat sebagian, atau konektornya mungkin tidak mendukung fitur ini. Opsi **View native query** khusus untuk konektor database tertentu yang membuat kueri SQL. Opsi ini tidak berlaku untuk konektor yang berbasis pada Data.

Keuntungan dari pendekatan ini adalah ia berfungsi baik dalam pengalaman Power Query lama maupun baru. Dalam pengalaman Power Query lama, opsi ini muncul dengan teks **View native query**. Kelemahan dari metode ini adalah Anda harus melalui setiap langkah untuk melihat apakah kueri tersebut dapat dilipat. Untungnya, ada opsi lain yang tersedia, yang akan kita bahas selanjutnya.

Indikator Pelipatan Kueri

Opsi lain untuk menentukan kemampuan melipat adalah dengan menggunakan indikator pelipatan kueri. Indikator ini tersedia dalam pengalaman Power Query baru dan menyediakan isyarat visual dalam Editor Kueri Power BI. Indikator ini menyediakan informasi tentang apakah setiap langkah dalam kueri Anda dilipat atau apakah langkah tertentu merusak lipatan. Hal ini berguna karena memberi sinyal operasi selanjutnya yang akan diproses di luar sumber data.

Setiap langkah dalam kueri Anda akan menampilkan salah satu indikator berikut:

Indicator	Icon	Description
Folding		Indicates that this part of the query will be processed by the data source.
Not Folding		Indicates that this step will be processed outside the data source.
Might Fold		Whether or not a query step will be processed by the data source is uncertain and will be determined during query execution. Likely happens for ODBC or Odata connections.
Uncertain		Indicates an uncertain query plan, often due to providing a manual table or using transformations/connectors unsupported by the query plan tool and indicators.
Unknown		Indicates that there is no query plan available, which could be due to an error or because the query involves data formats other than tables.

** When an applied step in a query displays a specific folding indicator, any subsequent steps that have a vertical line with the same color as this indicator share the same query folding status.*

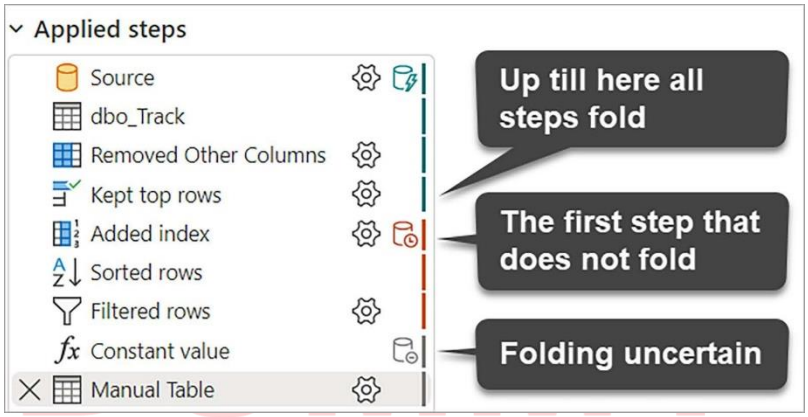
Gambar 3. 7 Tinjauan umum indikator pelipatan kueri yang tersedia

Indikator ini memberikan informasi tentang cara Power Query berinteraksi dengan sumber data Anda dan apakah Power Query memanfaatkan kemampuan pemrosesan sumber data tersebut. Mari kita lihat contohnya.

Misalkan kita menggunakan fungsi *Sql.Database* untuk terhubung ke database SQL. Konektor ini mendukung pelipatan kueri dan mencoba mengubah transformasi apa pun pada data yang mendasarinya menjadi kueri sumber data. Kita dapat menggunakan ekspresi:

```
Sql.Database( ServerAddress, DatabaseName )
```

Gambar berikut adalah hasil dari serangkaian transformasi:



Gambar 3. 8 Indikator pelipatan kueri menandakan status pelipatan setiap langkah yang diterapkan

Empat langkah pertama dalam kueri ini berhasil dilipat, artinya Power Query dapat mengubahnya menjadi kueri yang dipahami sumber data. Langkah pertama menampilkan indikator pelipatan, dan tiga langkah berikutnya ditandai dengan garis vertikal dengan warna yang sama, yang menunjukkan status pelipatannya.

Namun, terjadi pergeseran pada langkah **Added index**, saat kita menemukan indikator **Not Folding** untuk pertama kalinya. Ini adalah titik balik untuk mengoptimalkan kueri Anda, karena ini menunjukkan bahwa langkah selanjutnya yang dibangun di atas langkah ini tidak akan dilipat dan akan ditangani oleh mesin Power Query sebagai gantinya.

Penting untuk dicatat bahwa keberadaan indikator Tidak Melipat tidak berarti seluruh kueri gagal dilipat; ini hanya menyoroti bahwa sebagian kueri, mulai dari titik ini, tidak dilipat, meskipun bagian lain mungkin masih melakukannya.

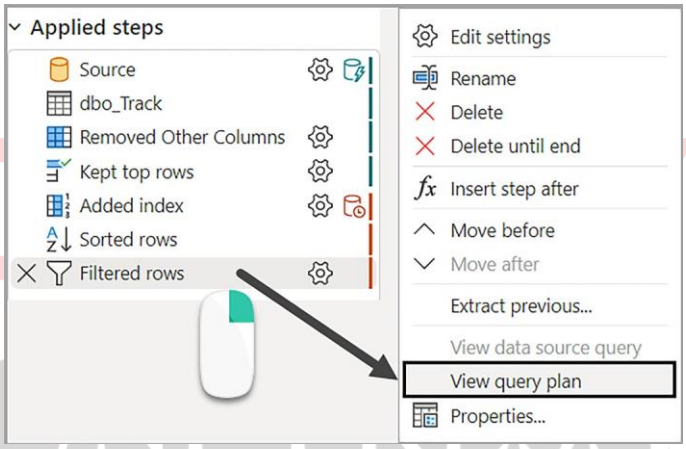
Aspek lain yang perlu diingat adalah bahwa rencana kueri saat ini (diperlukan untuk pelipatan kueri) secara eksklusif mendukung tabel. Oleh karena itu, ketika suatu langkah melibatkan penambahan nilai konstan yang

tidak memiliki rencana kueri yang sesuai, maka langkah tersebut diberi indikator Pelipatan **Uncertain**, yang mencerminkan keterbatasan ini.

Indikator pelipatan kueri menyediakan cara mudah untuk menentukan pelipatan kueri. Namun, dengan pengalaman Power Query yang baru, ada metode lain yang dapat Anda gunakan, yaitu rencana kueri.

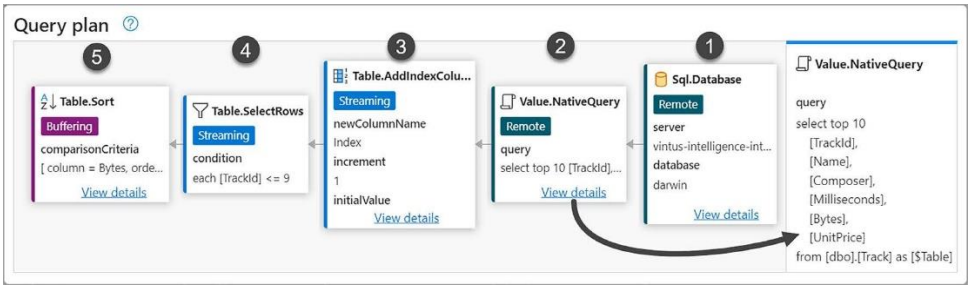
Query Plan

Rencana kueri menawarkan metode ketiga untuk menilai pelipatan kueri. Jadi, apa itu rencana kueri? Rencana kueri berisi instruksi mesin mashup tentang cara menjalankan kueri pada data Anda. Ini adalah alat yang berguna untuk menganalisis urutan mesin menjalankan operasinya. Antarmuka Power Query yang baru memungkinkan Anda menjelajahi rencana kueri yang dihasilkan mesin M. Untuk mengaksesnya, klik kanan pada langkah mana pun dalam kueri Anda dan pilih **View query plan**:



Gambar 3. 9 Melihat rencana kueri

Setelah melakukan ini, sebuah jendela akan muncul, menampilkan berbagai langkah rencana kueri:



Gambar 3. 10 Rencana kueri yang dihasilkan oleh mesin Power Query

Rencana kueri mewakili semua langkah yang mengarah ke langkah yang dipilih. Rencana ini dibaca dari kanan ke kiri, dan Anda mungkin dapat melihat bagaimana rencana ini terkait dengan langkah yang diterapkan pada Gambar 3.10. Berikut ini terjadi:

1) Koneksi ke database SQL, menggunakan **query folding**.
2) Pengambilan 10 baris teratas menggunakan **native query**, juga melalui **query folding**.
3) Penambahan kolom indeks, yang merusak **query folding**. Perubahan ini ditunjukkan dengan pergeseran dari label **Remote** ke **Streaming**.
4) Kueri kemudian memfilter baris yang relevan.
5) Terakhir, baris yang tersisa diurutkan.

Namun, urutan dalam rencana kueri mungkin tidak selalu selaras dengan urutan langkah yang diterapkan. Ini mungkin membingungkan pada awalnya karena Anda mungkin bertanya-tanya mengapa langkah pertama yang Anda terapkan bukanlah transformasi pertama dalam rencana kueri.

Mesin Power Query mengoptimalkan kueri dengan kemungkinan melakukan hal berikut:

- **Menggabungkan langkah**: Mesin dapat memilih untuk menggabungkan langkah, seperti melakukan beberapa langkah pemfilteran sekaligus.
- **Menghapus langkah**: Jika langkah mendatang membuat langkah sebelumnya tidak relevan, rencana kueri dapat menghapusnya.

- **Menyusun ulang langkah**: Jika output tetap sama, mesin dapat menyusun ulang langkah untuk memaksimalkan pelipatan kueri.

Dalam rencana kueri di atas, mesin memfilter baris sebelum mengurutkannya, meskipun urutan langkah yang diterapkan melakukan sebaliknya. Rencana kueri yang direvisi lebih efisien, karena mengurutkan kumpulan data yang diperkecil memerlukan lebih sedikit sumber daya sambil menghasilkan hasil yang sama.

Operasi yang berhasil dilipat dalam rencana kueri ditandai dengan label Jarak Jauh di kotaknya. Misalnya, mengklik teks **Details** di langkah 2 memungkinkan Anda melihat pratinjau kueri sumber data. Anda dapat menggunakan isyarat visual ini untuk melihat apakah langkah penting kueri Anda berhasil dilipat.

Bila Anda melihat langkah-langkah penting tidak dapat dilipat, pertimbangkan untuk merestrukturisasi kueri atau menulis ulang transformasi Anda untuk meningkatkan kemampuan melipat, sehingga berpotensi meningkatkan kinerja. Ini dapat melibatkan pemecahan operasi kompleks menjadi operasi yang lebih sederhana yang dapat dilipat atau mengatur ulang langkah-langkah untuk memungkinkan lebih banyak eksekusi **Remote**.

Anda kini telah mempelajari tiga alat untuk memverifikasi apakah suatu kueri (langkah) terlipat. Kita ingin menunjukkan bahwa Anda juga dapat memverifikasi apakah suatu langkah kueri terlipat dengan menggunakan fitur **Query Diagnostics**. Akan tetapi, kita biasanya tidak menyarankan untuk menggunakannya, karena informasi yang diberikannya sulit ditafsirkan. Selain itu, dibandingkan dengan tiga alat yang diuraikan di atas, diperlukan upaya yang tidak perlu untuk menentukan pelipatan kueri.

Dengan pemahaman tentang cara mengidentifikasi langkah pelipatan, sekarang mari kita bahas jenis operasi yang biasanya terlipat dan yang tidak.

5. Operasi dan dampaknya terhadap pelipatan

Di bagian ini, kita akan membahas operasi mana yang dapat dilipat dan mana yang tidak. Meskipun mekanisme pelipatan kueri tidak mendukung operasi yang sama untuk setiap sumber data, Anda dapat menerapkan beberapa prinsip umum. Sumber seperti basis data relasional (SQL Server, Oracle, dll.), kubus (SSAS, SAP Hana/BW, dan Google Analytics), Azure Data Explorer, OData, dan alur data premium mendukung pelipatan kueri. Karena basis data SQL termasuk sumber data yang paling umum, contoh berikut berkisar pada SQL.

Operasi yang Dapat Dilipat

Satu-satunya operasi yang dapat dilipat adalah transformasi. Biasanya, jika transformasi sesuai dengan pernyataan *SELECT*, kemungkinan besar transformasi tersebut dapat dilipat. Pernyataan tersebut dapat mencakup klausa seperti *WHERE*, *GROUP BY*, dan *JOIN*. Anda juga dapat membuat kolom dengan ekspresi sederhana yang kompatibel dengan basis data SQL.

Jenis operasi berikut biasanya dapat dilipat dengan baik:

- **Memodifikasi kolom**: Melibatkan pemilihan, penghapusan, penggantian nama, atau penataan ulang kolom yang selaras dengan klausa SELECT dalam SQL
- **Perhitungan kustom**: Hanya mendukung logika dasar seperti transformasi teks, matematika, dan bahkan perubahan tipe data tertentu yang sesuai dengan klausa *SELECT*
- **Mengurutkan kolom**: Sesuai dengan klausa *ORDER BY* dalam SQL
- **Memfilter baris**: Sesuai dengan klausa *WHERE* dalam SQL
- **Mengelompokkan data**: Terkait dengan klausa *GROUP BY*
- *Menggabungkan tabel*: Penggabungan non-fuzzy termasuk dalam klausa *JOIN*.

- **Menambahkan kueri**: Gunakan operator *UNION ALL* untuk menambahkan kueri.
- **Memutar dan membatalkan pivot data**: Operasi ini, yang selaras dengan operator *PIVOT/UNPIVOT* dalam SQL, memungkinkan reorganisasi data, mengubah strukturnya untuk analisis yang lebih baik.

Penting untuk menyadari bahwa transformasi tertentu dalam Power Query dapat menghasilkan hasil yang berbeda, tergantung pada apakah transformasi tersebut dijalankan oleh mesin Power Query atau oleh sumber data. Perbedaan ini muncul dari berbagai cara Power Query dan banyak database menangani operasi tertentu. Mari kita pertimbangkan dua contoh untuk mengilustrasikan hal ini:

- **Penanganan nilai null dalam gabungan**: Di Power Query, saat Anda melakukan gabungan luar kiri, ia memperlakukan dua nilai *null* sebagai ekuivalen, dan gabungan dapat mengembalikan nilai. Namun, dalam banyak sistem basis data, nilai *null* tidak dianggap sama. Perbedaan dalam penanganan nilai *null* ini dapat menyebabkan perbedaan dalam hasil operasi gabungan antara Power Query dan basis data.
- **Sensitivitas huruf besar/kecil dalam filter**: Misalkan Anda memfilter kolom untuk nama kota Meksiko di Power Query. Secara default, operasi peka huruf besar/kecil. Namun, saat kueri ini dilipat dan dijalankan di sumber data, ia mungkin mengabaikan kapitalisasi, karena beberapa basis data tidak mempertimbangkan huruf besar/kecil saat memfilter teks. Itu berarti operasi yang sama dapat mengembalikan hasil yang berbeda tergantung di mana ia dijalankan.

Oleh karena itu, saat menerjemahkan kode Power Query ke kode sumber data asli, pertimbangkan perbedaan potensial ini. Perbedaan ini dapat mengakibatkan variasi dalam data yang dikembalikan untuk kumpulan data yang sama berdasarkan di mana dan bagaimana kueri diproses. Alternatif bagi mereka yang menginginkan kontrol lebih besar atas kueri mereka adalah

membuat kueri khusus dan menggunakannya dengan fungsi Value.NativeQuery. Kita akan membahas fungsi ini secara lebih rinci nanti dalam bab ini saat mempelajari strategi untuk mempertahankan pelipatan kueri.

Operasi yang Tidak Dapat Dilipat

Pelipatan kueri merupakan fitur hebat di Power Query, tetapi memiliki keterbatasan karena beberapa operasi yang tidak dapat dilipat. Sekarang kita akan membahas keterbatasannya, yaitu operasi yang tidak dapat dilipat.

Transformasi

Ketika memikirkan operasi yang tidak dapat dilipat, salah satu kategori yang paling penting adalah transformasi. Meskipun banyak konektor yang secara efektif menerjemahkan kode M ke dalam kueri sumber data, banyak transformasi masih dapat berhenti dilipat. Ini biasanya terjadi ketika konektor tidak mendukung transformasi tertentu atau sumber data tidak memiliki padanan. Transformasi umum yang tidak dapat dilipat meliputi:

- Menambahkan kolom *index or ranking*.
- Menggabungkan atau menambahkan sumber data yang berbeda.
- Menggabungkan kueri dengan tingkat privasi yang berbeda.
- Operasi tanpa padanan dalam sumber data. Beberapa contohnya adalah Transpose, Keep Bottom N Rows, dan Kapitalisasi huruf pertama setiap kata dalam string.

Tidak ada jaminan jenis operasi dapat dilipat. Misalnya, mengonversi nilai ke jenis teks mungkin dapat dilipat, tetapi mengubahnya ke jenis waktu mungkin tidak. Konektor juga menerima pembaruan sehingga beberapa operasi dapat didukung di masa mendatang. Sementara sebagian besar pengguna yang ingin mengoptimalkan pelipatan kueri berfokus pada transformasi, area lain juga memengaruhi pelipatan, misalnya, tingkat privasi.

6. Tingkat Privasi Sumber Data

Aspek lain yang memengaruhi pelipatan kueri adalah tingkat privasi sumber data. Bekerja dengan sumber data dengan tingkat privasi yang tidak kompatibel dapat memicu firewall rumus, yang mencegah pelipatan kueri. Rincian tingkat privasi dan pengaruhnya terhadap pelipatan kueri dan firewall rumus akan dibahas nanti dalam bab ini. Mari kita lihat area lain yang dapat mencegah pelipatan kueri, yaitu menyediakan kueri basis data asli.

7. Kueri Basis Data Asli

Penggunaan kueri basis data asli biasanya merusak pelipatan kueri. Hal ini terjadi saat menggunakan fungsi *Value.NativeQuery* atau pernyataan kustom dalam fungsi akses data. Namun, ada beberapa contoh saat kueri kustom masih dapat mengizinkan pelipatan, yang akan kita bahas di bagian selanjutnya tentang mempertahankan pelipatan kueri.

8. Fungsi yang Dirancang Untuk Mencegah Pelipatan Kueri

Beberapa fungsi dirancang untuk mencegah terjadinya pelipatan kueri. Fungsi *Table.StopFolding* adalah contohnya. Fungsi ini memastikan bahwa operasi selanjutnya dijalankan dalam mesin Power Query. Anda dapat menggunakan fungsi ini dalam skenario di mana sumber data, meskipun mendukung pelipatan, sangat lambat.

Serupa dengan itu, fungsi *List.Buffer*, *Table.Buffer*, dan *Binary.Buffer* juga menghentikan pelipatan dengan memuat data ke dalam memori. Fungsi buffer dan perannya dalam pengoptimalan akan dibahas nanti dalam bab ini.

Dengan mempertimbangkan faktor-faktor ini, strategi apa yang dapat Anda gunakan untuk mempertahankan pelipatan kueri selama mungkin?

9. Strategi Untuk Mempertahankan Pelipatan Kueri

Di bagian ini, kita akan menjelajahi berbagai skenario yang dapat merusak pelipatan kueri dan melihat metode untuk mengatasinya. Untuk melakukan contoh pelipatan kueri sendiri, Anda memerlukan akses ke sumber data yang

mendukung pelipatan kueri, seperti basis data. Karena buku ini berfokus pada kode M, contoh berikut, yang menunjukkan cara mempertahankan pelipatan kueri, dimaksudkan untuk dibaca bersama.

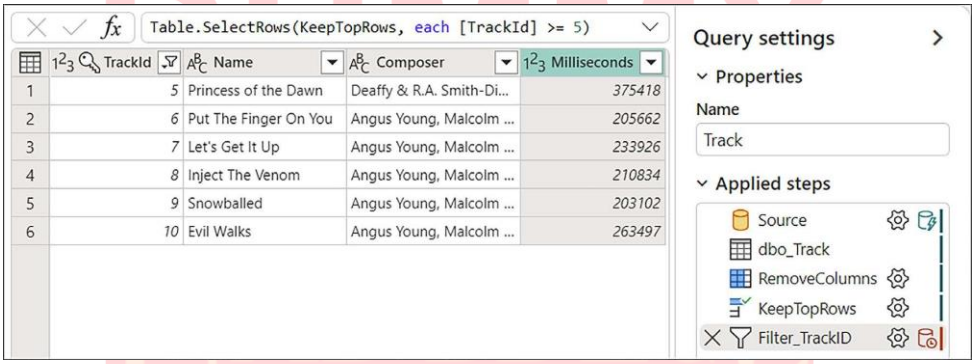
Menata Ulang Langkah-Langkah

Urutan di mana Anda menjalankan operasi dapat memengaruhi apakah kueri akan gagal. Untuk memastikan kueri Anda berjalan secepat mungkin, mengatur ulang langkah-langkah merupakan strategi yang sangat baik sehingga sumber data menjalankan operasi sebanyak mungkin.

Misalnya, anggaplah kita memiliki kumpulan data berisi trek musik, dan kita ingin melakukan transformasi berikut:

- **RemoveColumns**: Pilih kolom *TrackId*, *Name*, *Composer*, dan *Milliseconds* dari tabel asli.
- **KeepTopRows**: Simpan hanya 10 baris pertama dari tabel.
- **Filter_TrackID**: Filter *TrackIds* dengan nilai 5 atau lebih tinggi.

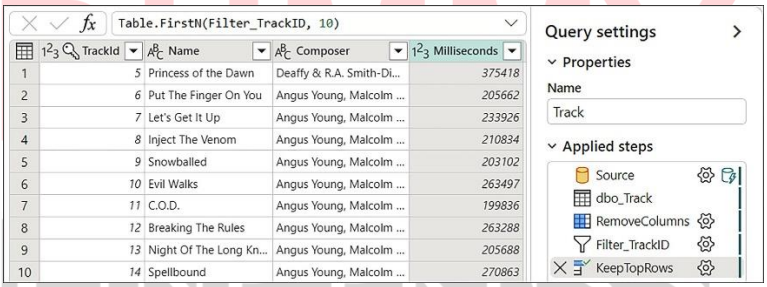
Gambar berikut menunjukkan indikator pelipatan kueri untuk setiap langkah:



Gambar 3. 11 Indikator pelipatan kueri menunjukkan apakah suatu langkah akan dilipat atau tidak

Empat langkah pertama dilipat, artinya sumber data mengeksekusinya. Namun, langkah terakhir, **Filter_TrackID**, tidak dilipat; Power Query mengeksekusinya setelah memuat data sebelumnya. Ini mengejutkan karena pemfilteran nilai biasanya merupakan operasi pelipatan kueri yang didukung.

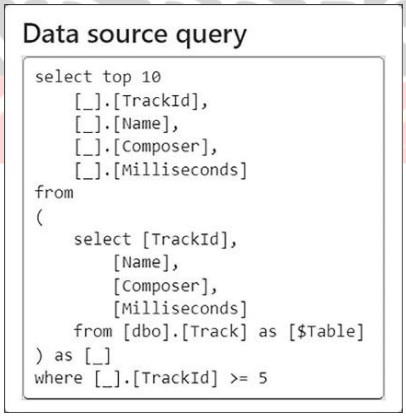
Dalam situasi seperti itu, ada baiknya bereksperimen dengan urutan langkah kueri Anda. Mengubah urutan dapat meningkatkan kemampuan konektor untuk mempertahankan pelipatan kueri melalui lebih banyak langkah kueri. Misalnya, pertimbangkan untuk memindahkan langkah **Filter_TrackID** ke titik sebelumnya dalam urutan tersebut. Berikut tampilan kueri yang direstrukturisasi:



Gambar 3. 12 Semua langkah dalam lipatan kueri

Perhatikan bagaimana pelipatan kueri kini berfungsi untuk semua langkah yang diterapkan? Dalam skenario di atas, memindahkan langkah-langkah juga menghasilkan hasil yang berbeda sehingga mungkin tidak diinginkan. Namun, hal itu menunjukkan bahwa urutan langkah yang Anda lakukan dapat mengubah apakah transformasi akan dilipat.

Anda juga dapat memeriksa kueri yang dihasilkan Power Query untuk sumber data dengan mengklik kanan langkah **KeepTopRows** dan memilih **View data source query**. Kueri keluarannya adalah sebagai berikut:



Gambar 3. 13 Kueri sumber data dihasilkan melalui pelipatan kueri

Kueri ini mungkin tidak seoptimal kueri SQL yang ditulis secara manual, tetapi jauh lebih baik daripada memuat semua data lalu menerapkan transformasi di Power Query. Namun, tidak ada yang menghalangi Anda untuk meneruskan kueri SQL Anda sendiri, yang akan kita bahas selanjutnya.

Bekerja Dengan Kueri Asli

Saat menggunakan Power Query, mereka yang memiliki keahlian SQL dapat meneruskan kueri yang lebih optimal. Namun, penting untuk mengingat beberapa pertimbangan guna memastikan keberhasilan pelipatan kueri dengan kueri SQL kustom.

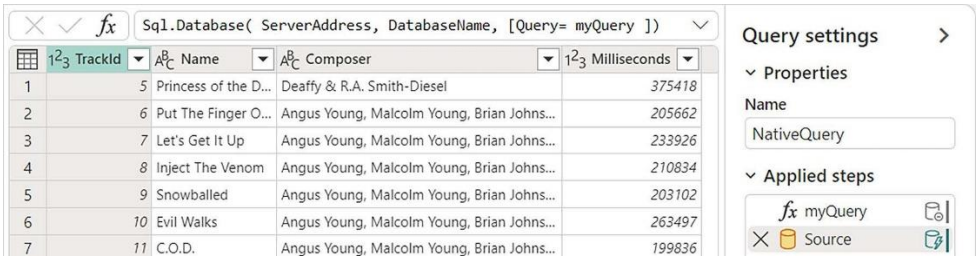
Metode utama untuk menghubungkan ke database SQL di Power Query adalah melalui fungsi *Sql.Database*. Biasanya, ini melibatkan penentuan alamat server dan nama database. Selain itu, Anda dapat menentukan record opsi yang menyertakan kueri SQL yang ingin Anda jalankan. Misalnya, pertimbangkan skenario saat Anda memiliki kueri SQL yang disimpan dalam langkah bernama *myQuery*:

```
myQuery =
  "SELECT TOP (10)
    [TrackId],[Name], [Composer], [Milliseconds]
  FROM [dbo].[Track]
  WHERE [TrackId] >= 5"
```

Anda dapat memasukkan kueri ini ke dalam fungsi **Sql.Database** sebagai berikut:

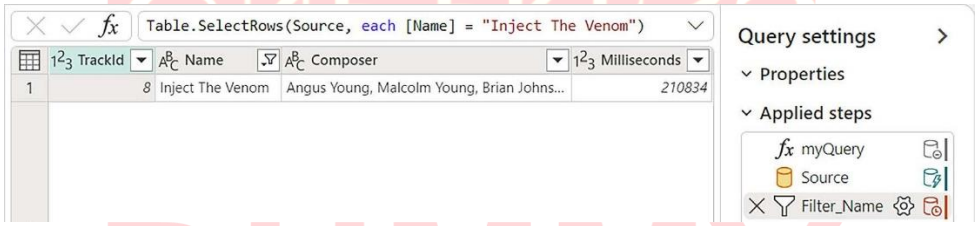
```
  Sql.Database( ServerAddress,
     DatabaseName,
              [ Query= myQuery ] )
```

Menjalankan kueri SQL khusus akan memberi kita keluaran sebagai berikut:



Gambar 3. 14 Pelipatan kueri terjadi untuk kueri SQL kustom

Indikator mengonfirmasi bahwa kueri ini diproses di tingkat sumber data, yang mungkin Anda harapkan. Namun, mari kita periksa apa yang terjadi saat kita menerapkan operasi sederhana, seperti memfilter baris dalam tabel ini – operasi yang biasanya diambil oleh mekanisme pelipatan kueri. Anehnya, indikator **Not Folding** mengungkapkan bahwa pelipatan kueri tidak terjadi untuk langkah **Filter_Name**:



Gambar 3. 15 Pelipatan kueri tidak terjadi untuk langkah-langkah yang mengikuti kueri SQL kustom

Meskipun kita dapat meneruskan kueri SQL kustom ke fungsi *Sql.Database*, langkah selanjutnya tidak akan mendapatkan manfaat dari pelipatan kueri, yang tidak ideal.

Kabar baiknya adalah ada solusi saat bekerja dengan kueri SQL asli. Kita dapat menggunakan fungsi *Value.NativeQuery* untuk menggabungkan pelipatan kueri dengan kueri SQL kustom. Bagaimana cara kerjanya? Daripada memasukkan pernyataan SQL langsung ke fungsi *Sql.Database*, kita harus menyediakannya ke fungsi *Value.NativeQuery*:

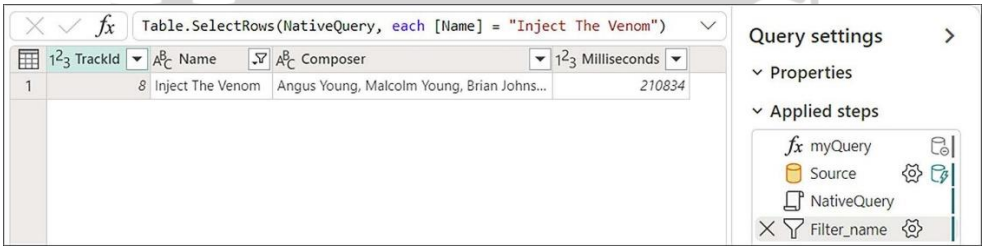
```
let
    myQuery =
    "SELECT TOP (10)
       [TrackId], [Name], [Composer] ,[Milliseconds]
     FROM [dbo].[Track]
     WHERE [TrackId] >= 5",
    Source = Sql.Database( ServerAddress, DatabaseName ),
    NativeQuery = Value.NativeQuery( Source, myQuery, null, [EnableFolding = true] )
in
    NativeQuery
```

Gambar 3. 16 Kueri SQL kustom yang diteruskan ke fungsi *Value.NativeQuery*

Berikut ini adalah proses yang terjadi:

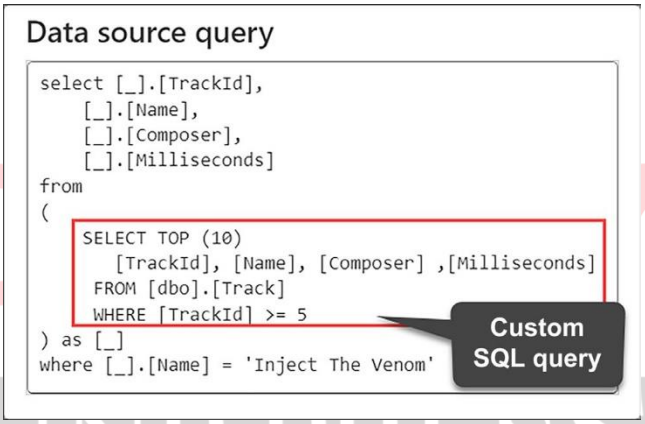
1) Pertama, buat koneksi ke database menggunakan fungsi *Sql.Database*.
2) Selanjutnya, atur fungsi *Value.NativeQuery*:
   a) Berikan koneksi database sebagai argumen pertama ke fungsi *Value.NativeQuery*.
   b) Sertakan query SQL kustom sebagai string dalam argumen kedua.
   c) Tentukan null untuk menunjukkan bahwa tidak ada parameter opsional yang diperlukan.
   d) Untuk memastikan pelipatan query terjadi, sertakan record opsi dengan *EnableFolding* yang disetel ke *true*.

Dengan pengaturan ini, jika kita mencoba memfilter kolom *Name* untuk nilai *Inject the Venom*, kita sekarang mengamati bahwa pelipatan query diterapkan secara efektif ke semua langkah dalam query:



Gambar 3. 17 *Value.NativeQuery* memungkinkan pelipatan kueri sambil menyediakan kueri SQL khusus

Dengan melihat kueri sumber data, kita sekarang dapat melihat kueri SQL khusus yang kita sediakan dan kode terlipat tambahan menggunakan mekanisme pelipatan kueri:



Gambar 3. 18 Kueri sumber data yang dihasilkan oleh mekanisme pelipatan kueri

Ingatlah untuk menyetel opsi *EnableFolding* ke *TRUE* agar ini berfungsi.

Menulis Ulang Kode

Terkadang, Anda mungkin menghadapi skenario saat operasi tertentu tidak mendukung pelipatan kueri, meskipun Anda mengharapkannya. Seperti yang dibahas sebelumnya dalam bab ini, Anda dapat menentukan pelipatan langkah dengan melihat indikator pelipatan kueri, memeriksa rencana kueri, atau menentukan apakah langkah tersebut mengaktifkan **Data source query** saat Anda mengklik kanan langkah tersebut.

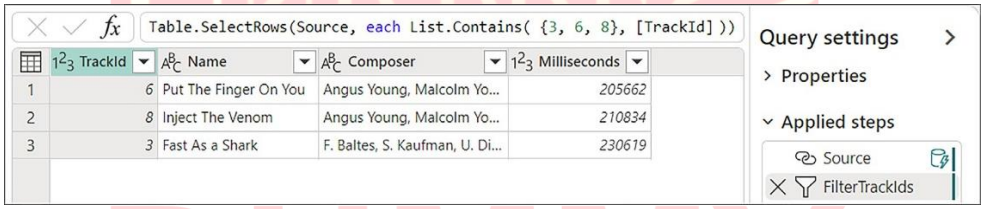
Jika suatu langkah tidak dapat dilipat, penulisan ulang ekspresi dapat menghasilkan versi yang mendukung pelipatan kueri. Mari kita ambil contoh. Dalam tabel berikut, kolom *TrackID* memiliki nilai *null* di baris 5:

	TrackId	Name	Composer	Milliseconds
1	1	For Those About ...	Angus Young, Malc...	343719
2	2	Balls to the Wall	null	342562
3	3	Fast As a Shark	F. Baltes, S. Kaufma...	230619
4	4	Restless and Wild	F. Baltes, R.A. Smit...	252051
5	null	Princess of the D...	Deaffy & R.A. Smit...	375418
6	6	Put The Finger O...	Angus Young, Malc...	205662
7	7	Let's Get It Up	Angus Young, Malc...	233926
8	8	Inject The Venom	Angus Young, Malc...	210834

Gambar 3. 19 Cuplikan data yang digunakan untuk pelipatan kueri

Misalkan Anda ingin memilih *ID 3*, *6*, dan *8*. Anda dapat melakukannya menggunakan fungsi *List.Contains*, yang diterjemahkan menjadi operator *IN* dalam SQL dan mendukung pelipatan kueri. Hasilnya seperti berikut:

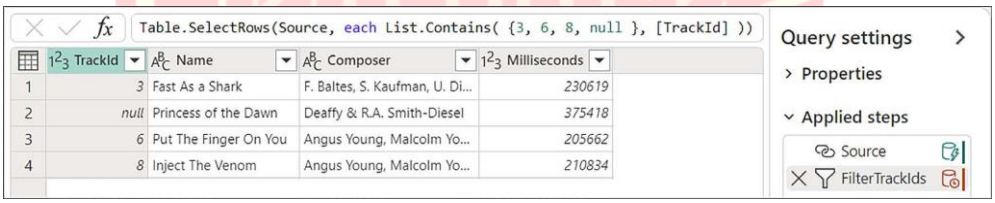
```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8 }, [TrackId] ) )
```



Gambar 3. 20 *List.Contains* adalah transformasi yang dapat dilipat

Namun, tantangan muncul saat Anda perlu menyertakan nilai *null* dalam pilihan Anda. Dalam SQL, nilai null tidak diperbolehkan dalam klausa IN, yang mengarah ke situasi di mana ekspresi asli tidak dapat dilipat:

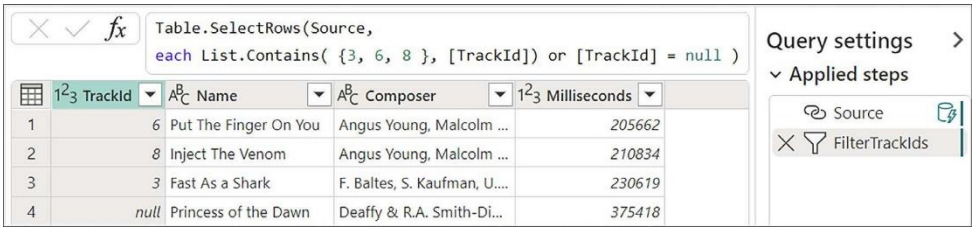
```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8, null}, [TrackId] ) )
```



Gambar 3. 21 Pelipatan tidak terjadi saat memasukkan nilai null ke dalam *List.Contains*

Untuk mengatasi hal ini, Anda dapat bereksperimen dengan menggabungkan kondisi *null* secara terpisah dari fungsi *List.Contains*:

```
Table.SelectRows( Source,
  each List.Contains( {3, 6, 8 }, [TrackId] )
       or [TrackId] = null )
```



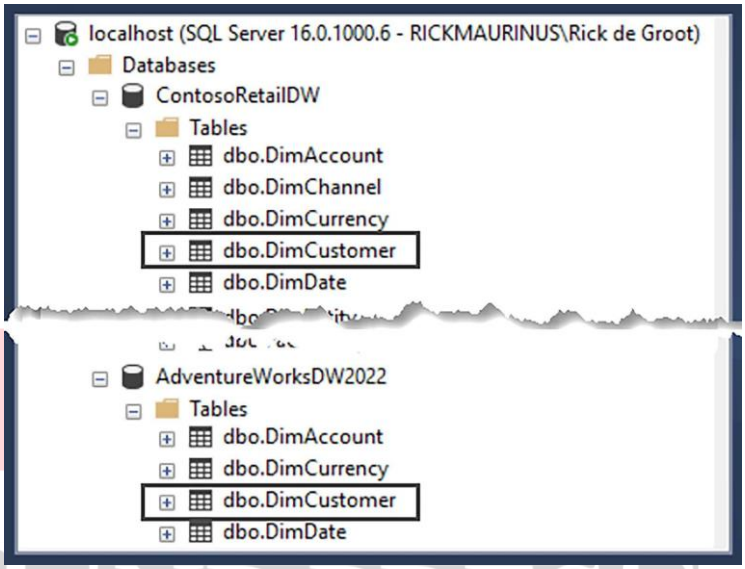
Gambar 3. 22 Pelipatan kueri berfungsi saat menjalankan pengujian untuk nilai null secara terpisah

Seperti yang Anda lihat, pelipatan kueri kini berfungsi untuk keseluruhan kueri dan memberikan keluaran yang sama.

Menggunakan Pelipatan Lintas Basis Data

Tantangan umum dengan pelipatan kueri muncul saat bekerja dengan data di berbagai basis data. Power Query tidak dirancang untuk melipat kueri yang mencakup beberapa basis data secara default. Contoh di mana Anda mungkin mengalami keterbatasan ini adalah saat melakukan operasi seperti penggabungan antar basis data. Namun, saat basis data ini dihosting di server yang sama, ada solusinya.

Mari kita ilustrasikan ini dengan sebuah contoh. Pertimbangkan skenario di mana kita ingin membandingkan alamat email pelanggan antara dua basis data untuk mengidentifikasi kesamaan. Untuk contoh ini, kita akan menyebut basis data ini sebagai *AW (AdventureWorks)* dan *CT (Contoso)*, yang masing-masing berisi tabel dengan struktur yang serupa:



Gambar 3. 23 Dua nama tabel identik dalam database berbeda di server yang sama

Kita menangani tugas tersebut dengan dua kueri berbeda dan menggunakan data dari basis data *AW (AdventureWorks)* dan *CT (Contoso)*. Setiap kueri terhubung ke basis datanya masing-masing dan mengambil informasi dari tabel pelanggan. Secara khusus, kita berfokus pada kolom *BirthDate* dan *EmailAddress* dari kedua basis data:

AW

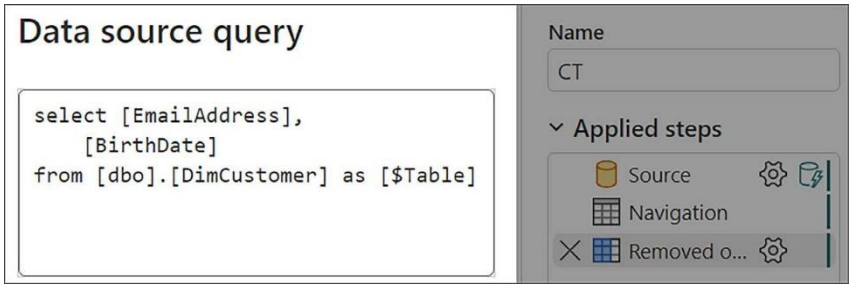
```
let
  Source = Sql.Database("localhost", "ContosoRetailDW"),
  Navigation = Source{[Schema = "dbo", Item = "DimCustomer"]}[Data],
  #"Removed other columns" = Table.SelectColumns(Navigation, {"EmailAddress", "BirthDate"})
in
  #"Removed other columns"
```

CT

```
let
  Source = Sql.Database("localhost", "AdventureWorksDW2022"),
  Navigation = Source{[Item = "DimCustomer", Schema = "dbo"]}[Data],
  #"Removed other columns" = Table.SelectColumns(Navigation, {"EmailAddress", "BirthDate"})
in
  #"Removed other columns"
```

Gambar 3. 24 Kueri yang mengambil kolom dari tabel pelanggan di dua database berbeda

Indikator pelipatan kueri untuk kedua kueri menunjukkan bahwa semua langkah dalam kueri individual ini telah berhasil dilipat. Berikut tampilan kueri sumber data untuk basis data *CT*:

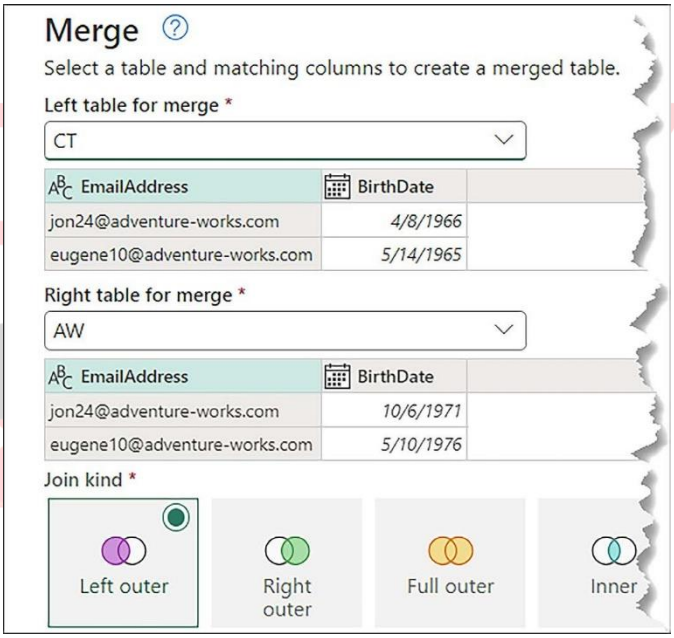


Gambar 3. 25 Kueri sumber data yang dihasilkan oleh mekanisme pelipatan kueri

Langkah selanjutnya melibatkan penggabungan data dari kedua kueri ini. Untuk melakukannya:

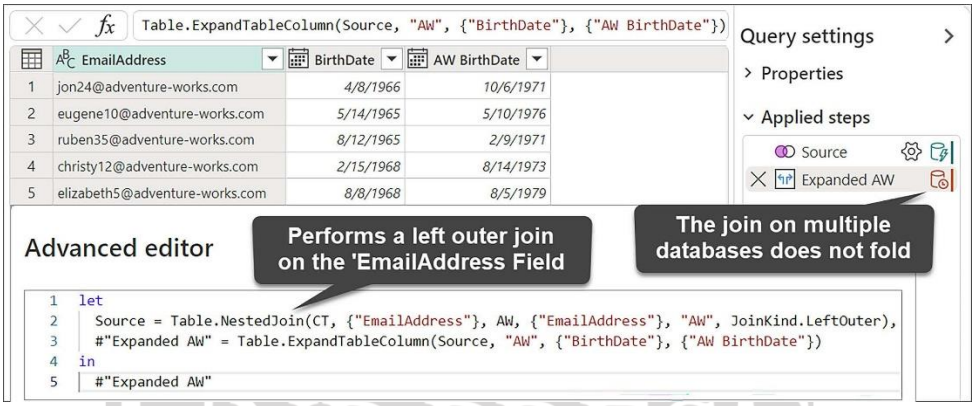
1) Navigasi ke tab **Home** dan pilih **Merge Queries**.
2) Kemudian pilih **Merge queries as New**.
3) Di kotak dialog yang muncul, kita pilih **EmailAddress** sebagai bidang untuk digabungkan dan pilih join **Left Outer**.

Itu memberi kita pengaturan berikut:



Gambar 3. 26 Melakukan lek outer join menggunakan *EmailAddress* sebagai kolom join

Setelah penggabungan, kita perluas kolom *BirthDate* untuk memungkinkan perbandingan langsung dalam kueri. Hasilnya adalah sebagai berikut:



Gambar 3. 27 Menggabungkan kueri dan memperluas bidang tidak gagal saat menggunakan beberapa database

Meskipun upaya awal kita dalam menggabungkan data dari basis data *AW (AdventureWorks)* dan *CT (Contoso)* berhasil menggabungkan alamat email dengan tanggal lahir yang relevan, operasi penggabungan ini merusak pelipatan kueri. Sebaliknya, pemrosesan berlangsung di dalam mesin Power Query. Meskipun kueri yang terpisah dapat dilipat, penggabungan berikutnya tidak. Jadi, apakah ada yang dapat kita lakukan untuk memperbaikinya?

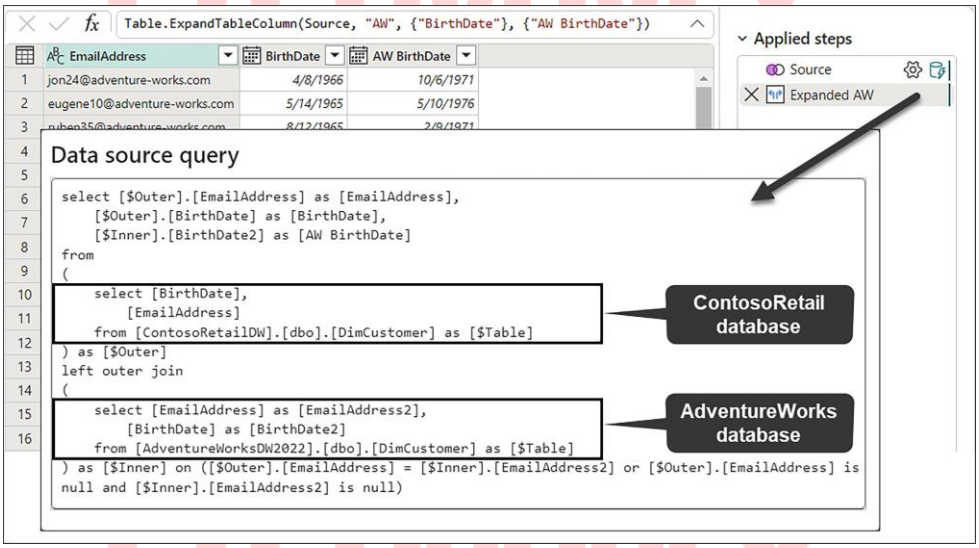
Di sinilah opsi *EnableCrossDatabaseFolding* dalam fungsi *Sql.Database* terbukti berguna. Fungsi *Sql.Database* biasanya memerlukan input seperti nama server dan basis data. Namun, fungsi ini juga memungkinkan parameter tambahan, salah satunya adalah opsi *EnableCrossDatabaseFolding*. Pengaturan ini dirancang untuk memfasilitasi pelipatan kueri di beberapa basis data, selama semuanya berada di server yang sama. Awalnya, kita terhubung ke basis data CT menggunakan ekspresi:

```
Sql.Database("localhost", "ContosoRetailDW")
```

Untuk memanfaatkan pelipatan kueri lintas basis data, ekspresi ini dapat dimodifikasi menjadi:

```
Sql.Database(
  "localhost",
  "ContosoRetailDW",
  [EnableCrossDatabaseFolding = true])
```

Agar pelipatan kueri dapat berfungsi, kita harus menerapkan modifikasi yang sama pada koneksi basis data AW. Pendekatan yang direvisi menghasilkan skenario di mana operasi penggabungan antara dua basis data diproses secara efisien melalui pelipatan kueri:



Gambar 3. 28 Kueri sumber data yang menggabungkan tabel dari beberapa database di server yang sama

Seperti yang ditunjukkan gambar, pelipatan kueri lintas basis data memungkinkan sumber data itu sendiri untuk menangani beban kerja transformasi, daripada mengandalkan mesin Power Query. Itu bahkan berfungsi saat operasi merujuk ke beberapa basis data di server yang sama.

Sebagai penutup, bagian ini telah memperkenalkan pelipatan kueri sebagai aspek penting dalam mengoptimalkan kueri Anda. Praktik terbaik adalah meminta sumber data Anda melakukan pekerjaan berat untuk transformasi Anda. Dengan strategi yang diberikan, Anda sekarang mengetahui beberapa cara untuk mempertahankan pelipatan kueri selama mungkin. Dengan pengetahuan ini, kita mendorong Anda untuk menerapkan

strategi ini pada pekerjaan harian Anda dan melihat sendiri bagaimana hal itu memengaruhi kinerja.

Berikutnya adalah topik penting lainnya, firewall rumus. Ini adalah fitur yang dapat memengaruhi pelipatan kueri, tetapi juga merupakan sumber frustrasi yang umum saat memblokir kueri Anda agar tidak berjalan.

D. Rumus Firewall

Saat bekerja dengan kueri, menggabungkan berbagai sumber data adalah hal yang umum. Kemungkinan besar setelah beberapa kali mengutak-atik, Anda akan mengalami kesalahan yang mencegah Anda menggabungkan data. Kesalahan ini disebabkan oleh firewall rumus yang dimiliki Power Query, yang juga dikenal sebagai firewall privasi data. Jadi, apa sebenarnya firewall rumus itu, dan bagaimana Anda dapat bekerja dengannya?

1. Apa Rumus Firewall?

Firewall rumus adalah fitur Power Query yang mencegah transfer data yang tidak disengaja antara sumber. Fitur ini sangat penting saat menangani informasi sensitif. Fitur ini dirancang karena pelipatan kueri. Pelipatan kueri memungkinkan Power Query mengonversi transformasi data menjadi kueri sumber data yang dapat dijalankan sumber data secara langsung.

Bayangkan situasi saat Anda bekerja dengan dua kueri di Power Query. *Query1* berisi informasi sensitif, seperti nomor jaminan sosial. Dengan menggunakan gabungan internal, Anda ingin menggunakan data ini untuk mengekstrak detail yang relevan dari *Query2*. Namun, demi efisiensi, Anda ingin menghindari pengimporan seluruh tabel SQL yang terkait dengan *Query2* ke Power Query. Mengimpor seluruh tabel hanya untuk memfilter baris yang tidak diperlukan nanti bukanlah hal yang optimal. Lagi pula, menggunakan pelipatan kueri adalah pendekatan yang lebih efisien, karena hanya mengimpor data yang diperlukan.

Namun, meskipun pelipatan kueri efektif, pelipatan kueri juga menimbulkan risiko potensial. Dalam proses pelipatan kueri, informasi sensitif dari *Query1*, seperti nomor jaminan sosial, dapat secara tidak sengaja disematkan ke kueri sumber data untuk *Query2*. Jika seseorang dengan keahlian basis data—atau siapa pun yang memantau jaringan Anda—mencegat kueri ini, mereka berpotensi mengakses informasi rahasia yang disematkan di dalamnya.

Di sinilah firewall rumus berperan. Tujuannya adalah untuk mencegah situasi seperti itu saat data sensitif mungkin secara tidak sengaja disematkan dalam kueri yang dikirim ke sumber eksternal. Untuk mengenali di mana letak potensi risiko, firewall rumus menggunakan partisi.

2. Memahami Partisi

Saat mengevaluasi kueri dengan firewall rumus aktif, Power Query membagi kueri dan dependensinya ke dalam partisi. Partisi ini pada dasarnya adalah kelompok yang terdiri dari satu atau beberapa langkah. Setiap kali partisi merujuk ke partisi lain, firewall rumus akan melakukan intervensi dengan mengganti referensi tersebut dengan panggilan ke fungsi tertentu, yang diberi nama *Value.Firewall*. Fungsi ini memastikan bahwa partisi tidak saling mengakses secara langsung. Sebaliknya, semua referensi harus dievaluasi terlebih dahulu oleh firewall. Data diizinkan masuk ke partisi saat ini hanya setelah firewall mengizinkan interaksi antarpartisi.

Seperti yang dapat Anda bayangkan, cara partisi dibuat sangatlah penting. Menyertakan lebih sedikit atau lebih banyak langkah dalam satu partisi dapat menentukan apakah firewall akan terlibat. Namun, proses mempartisi kueri itu rumit dan berada di luar cakupan buku ini. Bagi mereka yang tertarik untuk memahami proses ini lebih dalam, bacaan lebih lanjut tersedia di https://learn.microsoft.com/en-us/power-query/data-privacy-firewall. Kita akan merujuk ke halaman ini di seluruh bab ini.

3. Prinsip dasar dari rumus firewall

Firewall formula beroperasi berdasarkan prinsip dasar mengenai partisi. Prinsip ini dibagi menjadi dua poin utama:

- **Mengacu pada partisi lain**: Partisi dapat menggunakan data atau hasil dari partisi lain. Partisi tersebut dapat berupa kueri lain atau partisi dalam kueri yang sama.
- **Mengakses sumber data yang kompatibel**: Partisi dapat memperoleh data dari sumber eksternal, asalkan sumber tersebut memiliki tingkat privasi yang kompatibel.

Salah satu dari keduanya mungkin terjadi, tetapi tidak keduanya pada saat yang bersamaan. Melanggar aturan ini akan mengakibatkan salah satu dari dua pesan firewall rumus:

- **Kesalahan saat merujuk**:

  Formula.Firewall: Query 'X' (step 'Y') references other queries or steps, so it may not directly access a data source. Please rebuild this data combination.

- **Kesalahan saat mengakses sumber data yang tidak kompatibel:**

  Formula.Firewall: Query 'X' (step 'Y') is accessing data sources with incompatible privacy levels. Please rebuild this data combination.

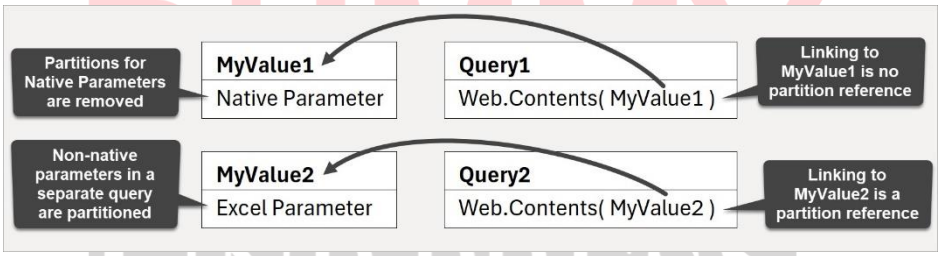
Dari sekadar membaca pernyataan di atas, sulit untuk memahami apa yang terjadi pada kesalahan Firewall Rumus, tetapi tunggu dulu. Kita akan menjelaskannya dengan lebih jelas. Di bagian berikutnya, kita akan membahas alasan di balik kesalahan ini dan membahas strategi untuk mengatasinya.

4. Kesalahan Firewall: Merujuk ke Partisi Lain

Kesalahan firewall pertama yang akan kita bahas adalah kesalahan yang dipicu saat merujuk partisi lain. Pesan kesalahan untuk kesalahan ini adalah:

> *Formula.Firewall: Query 'X' (step 'Y') references other queries or steps, so it may not directly access a data source. Please rebuild this data combination.*

Penting untuk memahami mengapa kesalahan ini muncul karena kesalahan ini memblokir kueri Anda agar tidak menghasilkan hasil apa pun. Ini berarti Anda tidak akan dapat memperbarui data atau melihat hasil transformasi Anda. Sayangnya, memecahkan kesalahan Formula Firewall tanpa mengetahui mekanisme yang mendasarinya bisa jadi sulit. Perhatikan contoh yang ditunjukkan pada gambar berikut:



Gambar 3. 29 Dua kueri serupa yang menyebabkan seseorang mengalami kesalahan Formula Firewall

Di sini, Anda melihat dua kueri serupa, tetapi hanya satu yang mengalami galat Formula Firewall. Di **Query1**, ada referensi ke parameter bernama **MyValue1**, yang dibuat melalui antarmuka pengguna. Proses pemartisian secara otomatis menghapus partisi dari parameter asli tersebut, yang memungkinkan **Query1** berjalan lancar.

Namun, jika Anda secara manual menyiapkan parameter yang menarik nilai dari file Excel, seperti yang terlihat pada **MyValue2**, proses pemartisian akan memperlakukannya secara berbeda. Proses ini tidak menghapus partisi, jadi setiap referensi ke **MyValue2** dianggap merujuk ke partisi.

Akibatnya, saat **Query2** mencoba menggunakan **MyValue2** dengan fungsi **Web.Contents** untuk menjangkau sumber data lain sekaligus merujuk ke partisi lain, firewall rumus akan terpicu.

Mari kita bahas contoh untuk melihat kesalahan ini dalam tindakan. Untuk mengikuti contoh di bawah ini, Anda dapat mengunduh file latihan dari repositori GitHub buku ini.

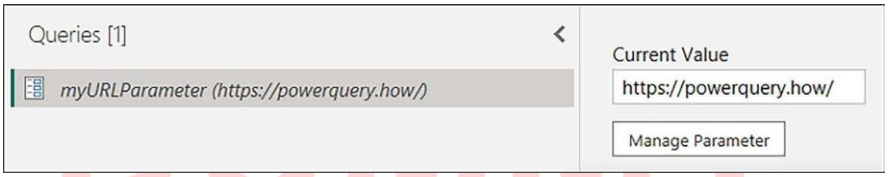
Menghubungkan ke URL Menggunakan Parameter Asli

Operasi umum untuk mengambil data adalah dengan menghubungkan ke URL. Contohnya adalah mengambil skor film dari IMDb, mendapatkan data dari tabel di Wikipedia, atau mengembalikan informasi prakiraan cuaca terkini. Apa pun informasi yang Anda butuhkan, Anda dapat terhubung ke situs web menggunakan konektor web Power Query. Melakukan hal itu akan memudahkan penggunaan untuk mengilustrasikan firewall rumus. Berikut caranya.

Misalkan Anda ingin terhubung ke situs web *PowerQuery.How* dan mengambil kode HTML. Pertama, Anda akan membuat parameter di Power Query:

1) Buka tab **Home** dan pilih **Manage Parameters**.
2) Kemudian pilih **New** dan beri nama parameter ini *myURLParameter*.
3) Atur ke *type text*, dan tetapkan nilai https://powerquery.how/.

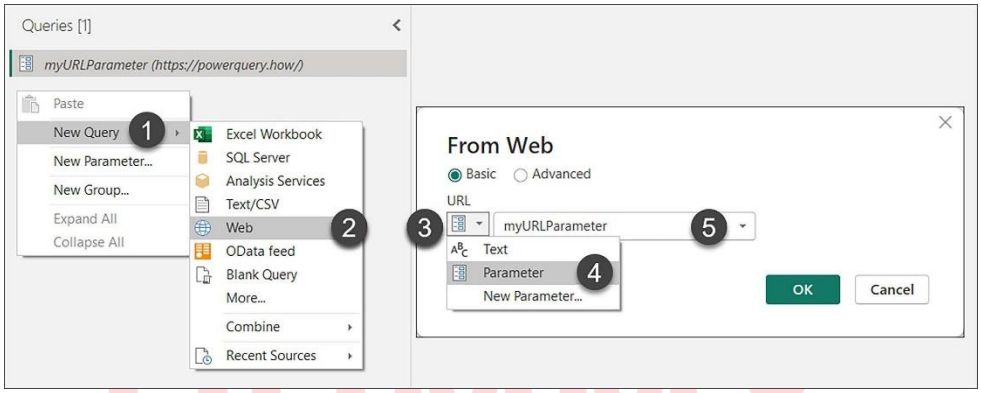
Melakukan langkah-langkah ini akan memberi Anda:



Gambar 3. 30 Parameter yang berisi URL untuk terhubung

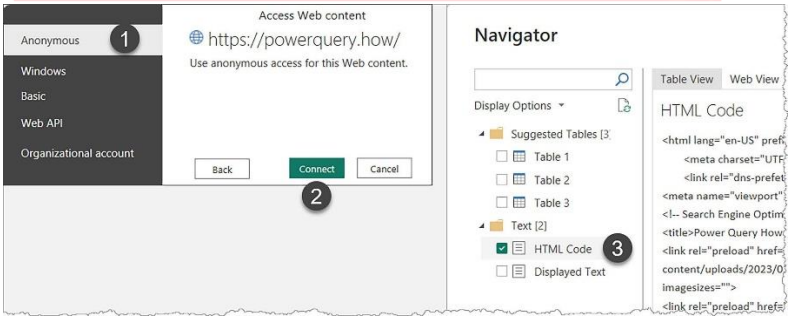
Parameter yang baru dibuat ini, termasuk URL statis, tidak mengakses sumber data secara langsung.

Selanjutnya, mari kita siapkan kueri untuk terhubung ke situs web dan mengambil kode HTML. Di area Kueri, klik kanan, pilih **New Query**, lalu klik **Web**. Di kotak dialog, pilih jenis URL, pilih **Parameter** dari menu tarik-turun, dan rujuk ke **myURLParameter**:



Gambar 3. 31 Membuat kueri web yang menggunakan parameter sebagai URL

Anda akan diminta untuk memilih metode koneksi saat menghubungkan ke sumber web untuk pertama kalinya. Pilih **Anonymous** lalu **Connect**. Di layar berikut, centang kotak di depan **HTML Code** untuk mengembalikan konten HTML. Konfirmasikan dengan mengeklik **OK**:



Gambar 3. 32 Memilih akses konten web dan opsi pengambilan data

Dengan menyelesaikan langkah-langkah ini, Anda mengambil konten HTML dari situs web menggunakan parameter yang dikodekan secara permanen, yang menghasilkan dua kueri. Kueri pertama adalah parameter *myURLParameter*, yang didefinisikan sebagai:

```
"https://powerquery.how/" meta [IsParameterQuery=true, Type="Text",
IsParameterQueryRequired=true]
```

Kueri kedua disebut Kode HTML dan didefinisikan sebagai:

```
Web.BrowserContents(myURLParameter)
```

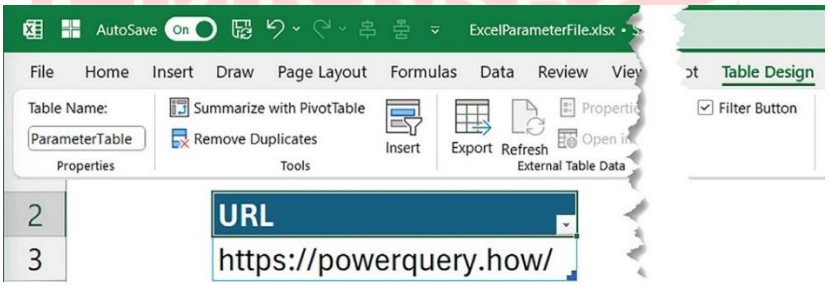
Hingga tahap ini, firewall formula belum menandai masalah apa pun dan telah berhasil mengembalikan kode HTML. Operasi yang lancar ini disebabkan oleh proses partisi, sebagaimana dijelaskan secara terperinci di situs web Microsoft. Proses ini menghapus partisi dari parameter asli yang ditetapkan melalui antarmuka pengguna Power Query.

Dalam kasus kita, kueri Kode HTML merujuk ke *myURLParameter*. Namun, karena parameter ini tidak disertakan dalam partisi dan tidak menautkan ke sumber data apa pun, firewall formula tidak memicu peringatan apa pun. Sekarang mari kita beralih ke situasi di mana sedikit perubahan dalam pengaturan menyebabkan kesalahan Firewall Formula.

Menghubungkan ke URL Menggunakan Parameter Excel

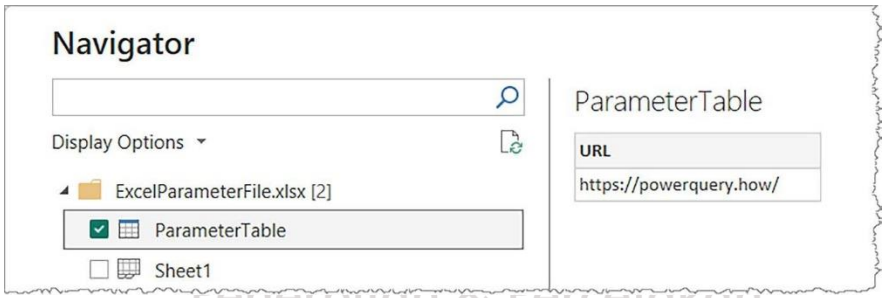
Sekarang, mari kita ubah sedikit pendekatan kita. Alih-alih parameter asli, kita akan mengambil URL untuk terhubung dari nilai tabel dari file Excel. Pengaturan seperti ini memungkinkan pengguna akhir untuk membuat perubahan pada URL dalam file Excel yang mudah diakses, tanpa perlu mengutak-atik Power Query. Ini menyediakan cara yang mudah digunakan untuk memasukkan input, dengan risiko masuk ke firewall rumus. Mari kita cari tahu cara kerjanya.

Untuk mengikuti panduan ini, pastikan Anda telah mengunduh berkas *ExcelParameterFile* dari repositori GitHub buku ini. Berkas ini berisi tabel berjudul *ParameterTable*, yang memiliki satu kolom bernama URL, yang berisi nilai teks https://powerquery.how/. Berikut tampilannya:



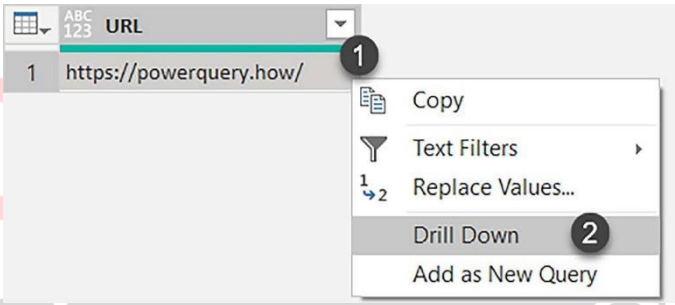
Gambar 3. 33 File Excel yang berisi URL untuk kueri kita

Pertama, simpan berkas ini di lokasi pilihan Anda. Untuk menghubungkannya, buka **New Query**, pilih **Excel Workbook**, dan temukan berkasnya. Saat Anda membukanya, sebuah jendela akan muncul. Di sini, pilih *ParameterTable* dan klik **OK**:



Gambar 3. 34 Memilih data dari Excel Workbook Navigator

Setelah mengimpor data, Anda akan menemukan tabel dengan satu sel yang berisi *URL*. Untuk menggunakan URL ini dalam kueri Anda, Anda harus mengekstrak nilai dari sel ini. Untuk melakukannya, klik kanan pada sel dan pilih opsi **Drill Down**. Langkah ini akan mengembalikan nilai sel untuk Anda:



Gambar 3. 35 Menelusuri nilai tabel untuk mengembalikan nilainya

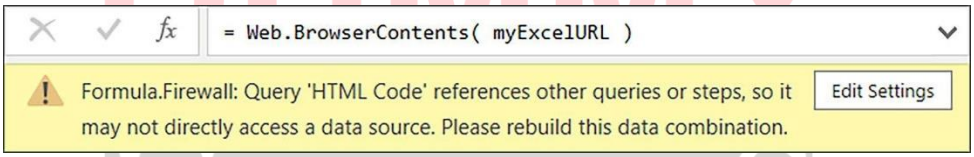
Kueri yang dihasilkan untuk mengekstrak nilai dari berkas Excel terlihat seperti ini:

```
let
  Source = Excel.Workbook(File.Contents(
    "C:\Data\ExcelParameterFile.xlsx"), null, true),
  ParameterTable_Table =
    Source{[Item="ParameterTable",Kind="Table"]}[Data],
  URL = ParameterTable_Table{0}[URL]
in
  URL
```

Ubah nama kueri ini menjadi *myExcelURL*. Apakah Anda ingat situasi sebelumnya saat kita menggunakan *myURLParameter*? Bagaimana jika kita menggantinya dengan *myExcelURL*, parameter dari berkas Excel kita? Coba gunakan ini dalam kueri:

```
Web.BrowserContents( myExcelURL )
```

Setelah melakukan perubahan ini, Anda akan mengalami kesalahan seperti berikut:



Gambar 3. 36 Kesalahan Formula Firewall saat mereferensikan kueri yang terhubung ke sumber data

Mengapa kesalahan ini muncul, terutama saat kita berhasil menggunakan parameter pada contoh sebelumnya? Untuk memahaminya, mari kita fokus pada prinsip dasar rumus firewall: "Sebuah partisi dapat merujuk ke partisi lain atau mengakses data dari sumber yang memiliki tingkat privasi yang kompatibel. Partisi tidak dapat melakukan keduanya pada saat yang bersamaan."

Pada contoh awal kita, ada dua alasan untuk tidak adanya kesalahan. Pertama, myURLParameter adalah parameter asli yang dikodekan secara permanen yang tidak mengakses sumber data eksternal apa pun. Kedua, proses pemartisian mengecualikan parameter asli dari cakupannya, yang berarti referensinya tidak dianggap sebagai referensi ke partisi lain.

Situasi berubah dengan parameter yang berasal dari file Excel. Parameter tersebut mengakses data dari sumber lain (file Excel), dan dengan demikian, parameter tersebut disertakan dalam proses pemartisian. Saat fungsi *Web.BrowserContents* mencoba mengakses situs web *(PowerQuery.How)* dan secara bersamaan merujuk ke partisi (parameter Excel), hal itu melanggar aturan dasar firewall. Pemartisian di Power Query bisa jadi cukup rumit.

Bahkan dengan membaca dokumentasi Microsoft tentangnya, Anda akan memiliki banyak pertanyaan yang tersisa. Namun, poin utama yang perlu diingat adalah ini: jika Anda merujuk satu kueri untuk mendapatkan data dari suatu sumber, seperti file Excel kita, lalu mencoba menggunakan data tersebut di kueri lain untuk mengakses suatu sumber, aturan firewall rumus akan dilanggar. Jadi, apa cara terbaik untuk memperbaiki masalah ini?

Mengatasi Kesalahan Firewall

Ketika kita mencoba merujuk parameter Excel dari partisi yang berbeda ini, kita melanggar aturan firewall utama. Kita tidak dapat mengakses sumber data (dalam hal ini, permintaan web) dan secara bersamaan merujuk partisi lain (parameter Excel). Namun, ada banyak skenario di mana Anda ingin dapat menggunakan titik data yang disimpan di tempat lain, dalam hal ini, URL Excel. Tujuan kita adalah untuk memungkinkan kueri mengambil URL dari sumber eksternal sambil menemukan cara untuk menjaga logika dalam partisi yang sama. Untuk mencapai hal ini, kita akan mengeksplorasi dua metode yang mungkin.

Metode 1: Menggunakan Fungsi

Metode pertama melibatkan penyimpanan logika parameter *myExcelURL* dalam suatu fungsi. Suatu fungsi menyimpan logika ke dalam kode yang dapat digunakan kembali tetapi tidak mengeksekusinya. Ketika kita kemudian memanggil fungsi ini dalam suatu kueri, pengaksesan data terjadi dalam partisi tempat fungsi tersebut dipanggil. Ini berbeda dari merujuk pada suatu kueri karena, dalam situasi tersebut, pengaksesan data terjadi pada kueri lain, yang akan dilihat sebagai partisi terpisah dan berpotensi memicu firewall.

Untuk membuat fungsi ini, Anda dapat menambahkan definisi fungsi sederhana yang merujuk ke langkah sebelumnya di akhir kueri Anda:

```
let
  Source = Excel.Workbook(File.Contents(
      "C:\Data\ExcelParameterFile.xlsx"), null, true),
  ParameterTable_Table =
    Source{[Item="ParameterTable",Kind="Table"]}[Data],
  URL = ParameterTable_Table{0}[URL],
  myFunction = () => URL
in
  myFunction
```

Skrip ini membuat sebuah fungsi. Pastikan untuk menamakannya *fxMyExcelURL*. Gunakan fungsi ini dalam kueri lain untuk membuat permintaan web, seperti ini:

```
Web.BrowserContents( fxMyExcelURL() )
```

Dengan melakukan ini, Anda dapat mengajukan permintaan dengan sukses tanpa harus berhadapan dengan firewall rumus. Anda dapat menemukan solusi ini dalam berkas latihan bab ini.

Metode 2: Mengintegrasikan Logika ke Dalam Satu Kueri

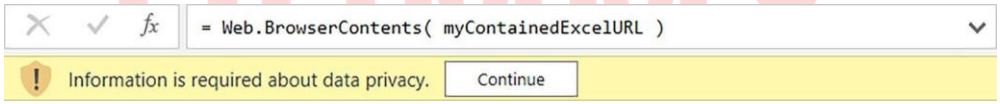
Pendekatan kedua adalah mengintegrasikan semua logika akses ke berkas Excel ke dalam kueri yang sama. Dengan demikian, semua langkah terkait akan berada dalam satu partisi, yang memungkinkan penggunaan parameter untuk mengakses sumber data eksternal di bawah aturan firewall rumus.

Untuk menggabungkan langkah-langkah tersebut, Anda dapat menggunakan logika kueri Buku Kerja Excel dan menambahkan langkah baru di akhir kueri. Pada langkah tersebut, Anda dapat merujuk URL dari buku kerja Excel sebagai berikut:

```
let
  Source = Excel.Workbook(File.Contents(
    "C:\Data\ExcelParameterFile.xlsx"), null, true),
  ParameterTable_Table =
    Source{[Item="ParameterTable",Kind="Table"]}[Data],
  myContainedExcelURL = ParameterTable_Table{0}[URL],
  HTMLCode = Web.BrowserContents( myContainedExcelURL )
in
    HTMLCode
```

Menggabungkan langkah-langkah ini ke dalam satu kueri memungkinkan proses pemartisian untuk menjaga langkah-langkah yang mengakses data dalam partisi yang sama. Pendekatan ini berhasil mengatasi kesalahan Formula Firewall yang pertama. Untuk melihatnya sendiri, Anda dapat menemukan kueri lengkap dalam berkas latihan bab ini.

Dengan menerapkan langkah-langkah di atas, Anda akan berakhir dalam situasi berikut:



Gambar 3. 37 Perintah untuk mengatur tingkat privasi

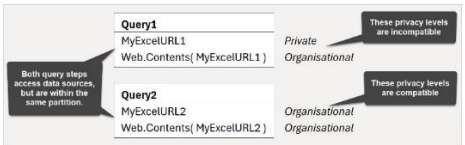
Kesalahan tersebut memberi tahu Anda bahwa diperlukan informasi tambahan tentang privasi data. Jika Anda menekan **Continue** dan memilih Publik, kueri Anda akan menampilkan hasilnya. Namun, dengan memilih **Private** atau **Organizational**, kueri Anda dapat mengarah ke jenis kesalahan Formula Firewall kedua, yang akan kita bahas selanjutnya.

5. Kesalahan Firewall: Mengakses Sumber Data yang Kompatibel

Kesalahan firewall kedua dapat terjadi saat Anda menggabungkan sumber data dengan tingkat privasi yang tidak kompatibel. Ingat bagaimana firewall dimaksudkan untuk mencegah kebocoran data yang tidak diinginkan? Itulah inti dari kesalahan ini. Pesan untuk kesalahan ini adalah:

Formula.Firewall: Query 'X' (step 'Y') is accessing data sources with incompatible privacy levels. Please rebuild this data combination.

Pertimbangkan skenario yang digambarkan dalam gambar berikut:



Gambar 3. 38 Skenario Kesalahan

Gambar tersebut memperlihatkan dua kueri. Dengan berfokus pada **Query1**, kueri tersebut mengakses data dari dua sumber eksternal. Jika langkah **Web.Contents** mencoba merujuk partisi lain, biasanya akan memicu galat "merujuk partisi lain" karena mengakses sumber data dan merujuk partisi lain secara bersamaan. Namun, bukan itu masalahnya di sini karena data berada dalam kueri yang sama. Hal itu memungkinkan proses pemartisian untuk menempatkan kedua langkah tersebut dalam partisi yang sama. Masalah sebenarnya untuk **Query1**adalah ketidakcocokan dalam tingkat privasi: sumber data **MyExcelURL1** ditandai sebagai **'Private**, sementara langkah **Web.Contents** ditetapkan ke tingkat privasi **Organizational**. Karena tingkat ini tidak cocok, Power Query melaporkan galat firewall kedua untuk mengakses sumber data yang tidak kompatibel.

Query2 tidak mengalami galat ini karena kedua langkah dalam kueri melibatkan sumber data dengan tingkat privasi yang cocok. Oleh karena itu, rumus firewall memungkinkan kedua sumber data untuk digabungkan.

Memahami hal ini mungkin tampak rumit, tetapi jangan khawatir. Kita akan menguraikan contoh ini lebih lanjut dengan kasus praktis. Untuk memahami pesan kesalahan secara menyeluruh, penting untuk terlebih dahulu mempelajari tentang tingkat privasi.

Memahami Tingkat Privasi

Di Power Query, setiap sumber data dapat diberi tingkat privasi. Tingkat ini penting untuk menentukan bagaimana data dari berbagai sumber dapat digabungkan.

Berikut ini ikhtisar tingkat privasi dan pengaruhnya terhadap pelipatan kueri:

- **Public**: Data dari sumber **Public** dapat digabungkan secara bebas dengan sumber data lain.
- **Organizational**: Pengaturan ini memungkinkan data untuk digabungkan hanya dengan sumber **Organizational** lainnya.

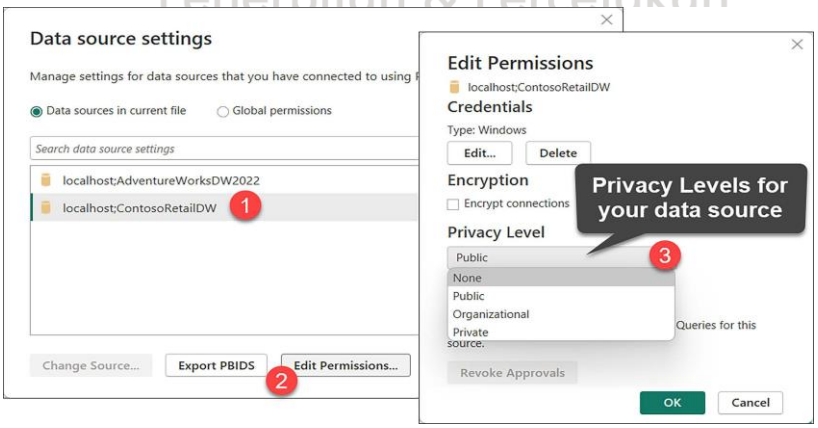
- **Private**: Pengaturan yang paling ketat. Data **Private** tidak digabungkan dengan sumber lain, mempertahankan isolasi yang ketat.

Pikirkan tingkat **Private** seperti ini: jika Anda memberi label sumber data sebagai Pribadi, sumber tersebut tidak akan tercampur dengan data lain dalam kueri. Ini menjaga informasi sensitif tetap aman dan terpisah. Data yang ditandai sebagai **Organizational** hanya dapat dicampur dengan data **Organizational** lainnya. Data **Public** menawarkan fleksibilitas paling tinggi karena dapat dengan mudah digabungkan dengan sumber data lain yang mengizinkannya. Ini biasanya mencakup sumber data dengan tingkat privasi yang ditetapkan ke **Public** atau **None**.

Menetapkan Tingkat Privasi

Saat Anda mencoba menggabungkan sumber data tanpa tingkat privasi yang ditetapkan, Power Query meminta Anda untuk memilih satu. Untuk menetapkan tingkat privasi secara manual:

1) Di Power BI Desktop:
   - Navigasi ke tab **Home** pada pita Power Query dan pilih **Data Source Settings**.
   - Pilih sumber data yang akan diubah dan klik **Edit Permissions** untuk menetapkan tingkat privasinya.



Gambar 3. 39 Mengatur tingkat privasi di pengaturan Sumber data

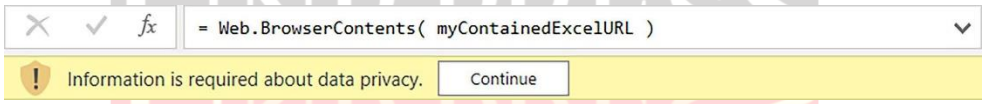
2) Di layanan Power BI:

- Akses pengaturan melalui ikon roda gigi di kanan atas, lalu pilih **Manage Connections and Gate-ways**.
- Temukan sumber data Anda dan atur **Privacy Level** di bagian bawah layar.

Jadi, apa hubungannya ini dengan rumus firewall?

Mengatasi Kesalahan Firewall

Mari kita tinjau kembali contoh sebelumnya saat kita mengambil URL dari file Excel. Tindakan ini memunculkan pesan kesalahan yang meminta informasi tingkat privasi untuk sumber data:



Gambar 3. 40 Pesan kesalahan yang meminta informasi tentang tingkat privasi

Pesan di atas memerlukan informasi tentang tingkat privasi data dari permintaan web. Untuk mengatasi kesalahan ini, kita melakukan salah satu dari dua metode.

Metode 1: Menetapkan Tingkat Privasi yang Kompatibel

Apakah kita menerima galat Formula Firewall atau tidak bergantung pada pengaturan tingkat privasi:

- **Public**: Kueri Anda akan segera dijalankan kecuali file Excel yang kita sambungkan memiliki tingkat privasi yang ditetapkan ke **Private** atau **Organizational**, yang menyebabkan galat firewall. Tingkat privasi Tidak Ada dan **Public** keduanya kompatibel.
- **Organizational**: Menetapkan kueri ke **Organizational** akan meminta pemeriksaan tingkat privasi file Excel. Kueri berjalan lancar jika file yang kita gabungkan disetel ke **Organizational**. Tingkat privasi lainnya akan memicu kesalahan Formula Firewall.

- **Private**: Memilih **Private** pasti akan menyebabkan kesalahan firewall.

Misalnya kita ingin menyetel tingkat privasi ke **Organizational**. Kita mengalami kesalahan firewall saat memilih **Continue** dan menetapkan **Organizational** ke sumber data kita. Ini terjadi karena, saat kita menyetel permintaan web ke **Organizational**, tingkat privasi untuk koneksi file Excel masih belum ditentukan. Untuk mengatasi ini:

1) Navigasi ke **Data Source Settings**.
2) Cari file Excel di bawah **Data sources in current file**.
3) Gunakan **Edit Permissions** untuk menyetel tingkat privasi file Excel ke **Organizational**.
4) Konfirmasikan pilihan Anda, lalu kembali ke kueri Anda.
5) Klik **Refresh Preview** untuk memperbarui kueri Anda.

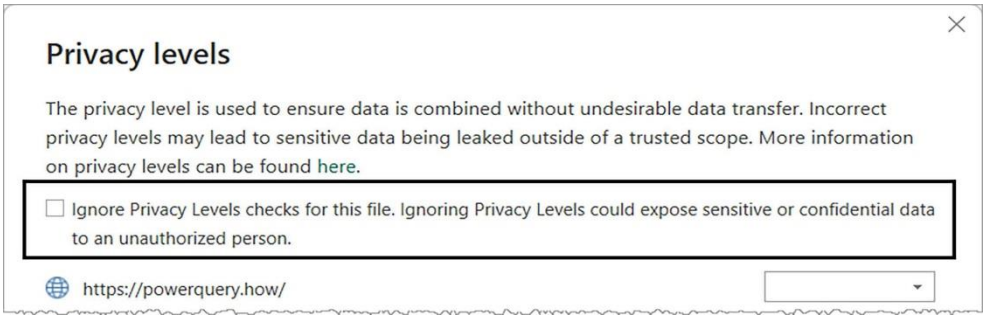
Setelah langkah-langkah ini, kueri Anda akan berfungsi tanpa kesalahan terkait tingkat privasi.

Metode 2: Mengabaikan Tingkat Privasi

Metode lain untuk mencegah kesalahan Formula Firewall adalah dengan mengabaikan pengaturan tingkat privasi. Peran formula firewall sebagai penjaga gerbang antara sumber data Anda dapat memperlambat kinerja kueri. Fitur ini penting untuk mencegah kebocoran data, terutama dalam skenario pelipatan kueri. Namun, Anda memiliki alternatif jika Anda menangani data yang tidak sensitif.

Di Power BI Desktop, Anda dapat memilih untuk mengabaikan tingkat privasi. Ini juga dapat membantu meningkatkan kinerja kueri Anda karena pemeriksaan kompatibilitas apa pun dapat dilewati.

Dalam situasi seperti contoh kita sebelumnya, saat Anda diminta untuk mengatur tingkat privasi, melanjutkan tanpa mengaturnya akan memicu pesan pop-up tertentu:

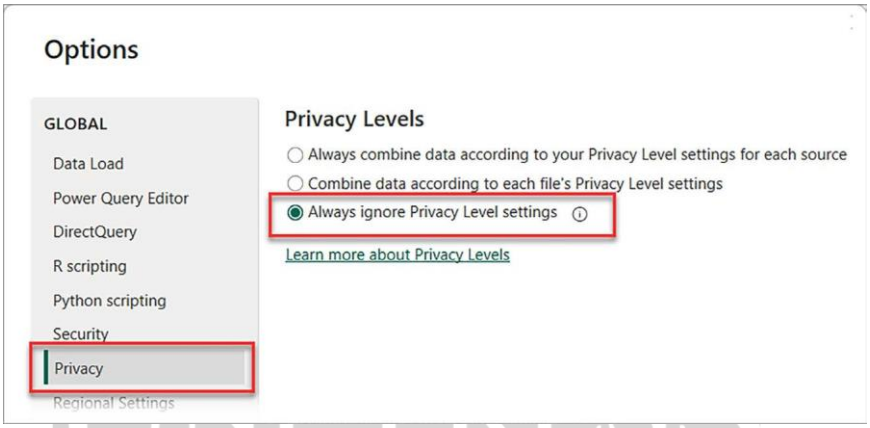


Gambar 3. 41 Pemberitahuan tingkat privasi – opsi untuk mengabaikan tingkat privasi

Anda memiliki pilihan untuk memilih **Ignore Privacy Levels**. Memilih ini memungkinkan kueri Anda dijalankan tanpa mengalami kesalahan Formula Firewall yang terkait dengan privasi. Untuk menemukan pengaturan ini:

- Navigasi ke **File**, lalu **Options and Settings**, dan klik **Options**.
- Pilih **Always ignore Privacy Level settings** di bagian **Privacy**.

Tampilannya seperti berikut:



Gambar 3. 42 Menu Opsi untuk mengonfigurasi Tingkat Privasi

Penting: Penggunaan pengaturan ini dapat mengekspos data sensitif atau rahasia. Selain itu, pengaturan ini hanya berlaku di Power BI Desktop. Layanan Power BI tidak mematuhi pengaturan ini, dan Anda perlu mengonfigurasi tingkat privasi yang sesuai.

Sebagai penutup, bab ini telah menunjukkan cara mengatasi berbagai kesalahan Formula Firewall menggunakan strategi tertentu. Kita memecahkan

kesalahan referensi partisi dengan menggabungkan logika dalam satu kueri dan membuat fungsi kustom. Untuk kesalahan privasi data, kita menyelaraskan tingkat privasi data atau mengabaikan pengaturan privasi sepenuhnya.

Namun, penting untuk dicatat bahwa contoh yang diberikan dalam bab ini tidak mendukung penyegaran dalam layanan Power BI. Batasan ini muncul karena contoh tersebut melibatkan pengambilan URL dalam satu kueri dan menggunakannya di kueri lain. Mesin Power Query mengharuskan URL dasar permintaan web terlihat secara eksplisit oleh mesin M. Referensi dinamis kita terhadap URL mengaburkan hal ini, yang menyebabkan kesalahan kueri dinamis.

Meskipun ada batasan khusus ini untuk permintaan web, teknik yang diuraikan dalam bab ini tetap efektif untuk menyelesaikan situasi Formula Firewall. Misalnya, jika metode ini diterapkan untuk memfilter kueri basis data, kueri akan berhasil disegarkan dalam layanan Power BI.

Sejauh ini, Anda telah mempelajari cara menangani formula firewall dan apa yang dapat Anda lakukan untuk memastikan pelipatan kueri terjadi untuk kueri Anda. Di bagian berikutnya, kita akan melihat strategi lain untuk meningkatkan kinerja kueri kita.

E. Mengoptimalkan Kinerja Kueri

Power Query beroperasi dalam lingkungan yang terbatas. Secara khusus, setiap kontainer mashup memiliki jumlah sumber daya yang terbatas. Keterbatasan ini merupakan faktor penting yang perlu dipertimbangkan saat membuat kueri. Penting untuk mengurangi volume data tepat di awal kueri Anda untuk kueri yang cepat. Dengan melakukannya, Anda tidak hanya mempercepat proses kueri tetapi juga mencegah terlampauinya batas sumber daya yang menyebabkan paging dan dapat menyebabkan kinerja yang lambat atau kegagalan.

Seperti yang dibahas sebelumnya, menggunakan mekanisme pelipatan kueri adalah salah satu strategi yang paling efektif untuk mengoptimalkan kinerja. Namun, ada skenario di mana sumber data Anda tidak mendukung pelipatan kueri, atau transformasi yang diperlukan merusak proses pelipatan. Dalam kasus seperti itu, strategi bergeser ke arah meminimalkan jejak memori kueri Anda. Jadi, apa saja beberapa metode yang efektif untuk mencapai ini?

1. Prioritaskan Pemfilteran Baris dan Penghapusan Kolom

Perlu ditekankan di sini bahwa salah satu langkah pertama yang harus Anda fokuskan adalah mengurangi kumpulan data Anda untuk hanya menyimpan apa yang diperlukan. Anda dapat melakukannya dengan:

- **Memfilter baris**: Sesegera mungkin dalam kueri Anda, prioritaskan pemfilteran baris Anda untuk hanya menyertakan yang diperlukan. Dengan mengurangi jumlah baris, Anda mengurangi jumlah data yang perlu disimpan Power Query dalam memori, yang mengarah ke pemrosesan yang lebih cepat dan penggunaan memori yang lebih sedikit.
- **Menghapus kolom yang tidak diperlukan**: Seperti memfilter baris, menghapus kolom yang tidak diperlukan sama pentingnya. Praktik ini memangkas volume data dan menyederhanakan model data Anda, membuatnya lebih mudah untuk dikerjakan dan dipahami.

Kedua operasi tersebut masuk akal dari perspektif komputasi. Namun selain ini, ada perbedaan lain yang harus dibuat untuk jenis operasi yang harus dilakukan terlebih dahulu.

2. Operasi Buffering Versus Streaming

Melakukan operasi di sumber data sering kali lebih cepat daripada melakukannya secara lokal. Namun, saat melakukan operasi secara lokal, Anda dapat menemukan dua jenis operasi: buffering dan streaming. Jadi, apa saja operasi ini, dan apa hubungannya dengan kinerja?

Operasi Penyanggaan

Operasi buffering adalah operasi yang memerlukan pembacaan seluruh kumpulan data untuk menghasilkan suatu hasil. Operasi ini membutuhkan banyak sumber daya dan menuntut pemindaian data secara menyeluruh hingga langkah sebelumnya. Ini juga berarti bahwa jumlah memori yang digunakan sebanding dengan ukuran input.

Pertimbangkan contoh di mana kita mengurutkan kumpulan data dan kemudian mengambil 1000 baris teratas. Dalam kasus ini, seluruh kumpulan data diperlukan untuk menentukan 1000 baris teratas secara akurat, bahkan jika Anda hanya ingin mengembalikan sebagian kecil data. Di sisi lain, jika Anda terlebih dahulu memilih 1000 baris teratas dan kemudian mengurutkannya, proses pengurutan hanya perlu menyimpan 1000 baris ini dalam memori. Proses ini dapat melakukannya hanya dengan mengambil 1000 baris teratas, tanpa memerlukan kumpulan data lengkap. Dalam skenario ini, pengurutan dianggap sebagai operasi buffering.

Operasi buffering tidak terbatas pada pengurutan data. Operasi ini juga mencakup beberapa jenis penggabungan, pengelompokan data, dan pengubahan struktur tabel, seperti melalui operasi pivot atau transpose. Demikian pula, operasi yang mengembalikan nilai unik atau menambahkan kolom peringkat memerlukan pemindaian seluruh tabel. Ini adalah jenis operasi yang menggunakan banyak sumber daya dan harus dibatasi saat mengoptimalkan kinerja.

Operasi Streaming

Berbeda dengan operasi buffering, operasi streaming tidak memerlukan pemindaian seluruh kumpulan data untuk mengembalikan hasil. Contoh yang bagus dari hal ini adalah operasi penyaringan. Memfilter list atau tabel memproses data dengan cara streaming. Saat data mengalir, Power Query mengevaluasi dan mengembalikan hasil secara progresif. Ini berarti bahwa

untuk layar pratinjau di Editor Power Query, hanya sebagian data yang diperlukan untuk mengisi 1000 baris teratas. Ini juga secara langsung mengarah ke Editor Power Query yang lebih responsif karena dapat menampilkan pratinjau data Anda lebih cepat. Operasi streaming yang umum meliputi penambahan kolom, pemfilteran baris, dan pemilihan kolom. Objek yang mendukung streaming meliputi objek bertipe tabel, list, dan biner.

Memaksimalkan transformasi streaming membantu mengurangi konsumsi memori dan menghindari masalah kinerja saat menjalankan kueri. Mengetahui hal ini, Anda mungkin bertanya-tanya bagaimana cara mengidentifikasi apakah Anda bekerja dengan operasi streaming atau buffering. Jawabannya adalah Anda dapat menentukannya di antarmuka Power Query yang baru dengan menggunakan rencana kueri.

Menggunakan Rencana Kueri

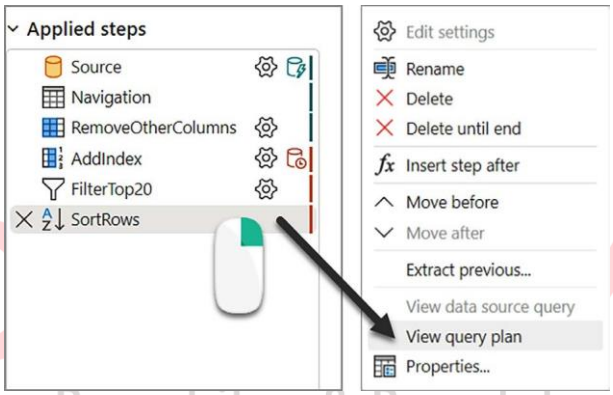
Rencana kueri adalah fitur dalam pengalaman Power Query baru yang dimaksudkan untuk memberikan informasi lebih lanjut tentang evaluasi kueri Anda. Setiap kali Anda ingin mengetahui detail lebih lanjut tentang evaluasi langkah tertentu, klik kanan pada langkah tersebut dan pilih **Query Plan**. Misalnya, mari kita lihat kueri sederhana:

```
let
  Source = Sql.Database("localhost", "ContosoRetailDW"),
  Navigation = Source{[Schema = "dbo", Item = "DimCustomer"]}[Data],
  RemoveOtherColumns = Table.SelectColumns(Navigation, {"EmailAddress", "BirthDate"}),
  AddIndex = Table.AddIndexColumn(RemoveOtherColumns, "Index", 0, 1, Int64.Type),
  FilterTop20 = Table.SelectRows(AddIndex, each [Index] <= 20),
  SortRows = Table.Sort(FilterTop20, {{"BirthDate", Order.Ascending}})
in
  SortRows
```

Gambar 3. 43 Kueri basis data dengan berbagai transformasi

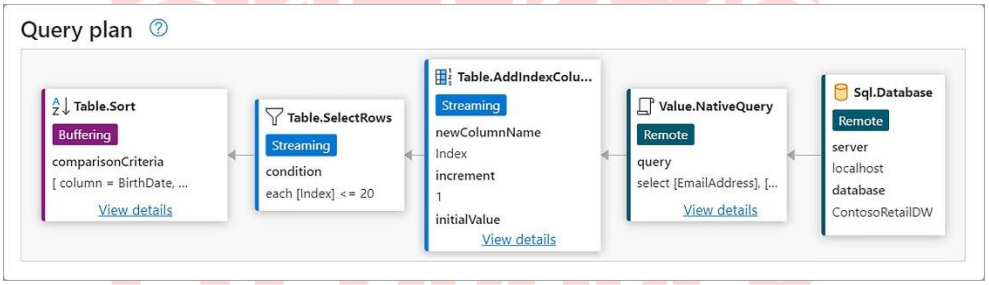
Kueri ini terhubung ke database SQL, memilih beberapa kolom, menambahkan kolom indeks, memfilter 20 baris teratas, dan mengurutkan tabel.

Untuk menganalisis rencana kueri untuk langkah **SortRows**, klik kanan pada langkah tersebut dan pilih **View query plan**:



Gambar 3. 44 Klik kanan pada langkah untuk memilih Lihat rencana kueri

Tindakan ini akan membuka jendela yang menampilkan rencana kueri:



Gambar 3. 45 Rencana kueri menunjukkan bagaimana suatu langkah dievaluasi

Rencana kueri memvisualisasikan jalur evaluasi kueri Anda. Dimulai di sebelah kanan, memperlihatkan koneksi ke basis data SQL, dan berlanjut ke sebelah kiri, mengarah ke langkah terakhir pengurutan tabel.

Di dalam rencana tersebut, Anda dapat menemukan tiga jenis indikator untuk setiap langkah:

- **Remote**: Langkah yang ditandai sebagai **Remote** menunjukkan operasi yang dilakukan di sumber data. Saat langkah dilipat, Anda dapat melihat kueri asli dengan memilih **View details**.

- **Streaming**: Langkah-langkah ini tidak memerlukan pemindaian semua data untuk mengembalikan hasil. Langkah-langkah ini memproses data saat mengalir, hanya mengambil baris yang diperlukan untuk mengisi pratinjau.
- **Buffering**: Langkah-langkah dengan indikator ini memerlukan pemindaian penuh data dari langkah sebelumnya.

Rencana kueri tidak hanya mengungkapkan apakah suatu operasi adalah buffering atau streaming, tetapi juga memperlihatkan langkah mana yang dilipat dan kueri yang mendasarinya di sumber data. Yang perlu diingat tentang memori adalah bahwa saat (bagian dari) kueri Anda tidak dilipat, operasi dari titik tersebut dilakukan dalam memori mesin Power Query.

Kiat Pengoptimalan: Prioritaskan pelipatan kueri jika memungkinkan. Setelah pelipatan tidak lagi memungkinkan, gunakan operasi streaming sebelum melakukan buffering. Pendekatan ini memastikan data Anda dengan cepat mengisi data pratinjau, sehingga mengurangi waktu tunggu selama pengembangan. Gunakan operasi buffering hanya jika diperlukan untuk menyelesaikan kueri Anda. Mengurutkan operasi ini secara strategis dapat secara signifikan mengurangi waktu yang diperlukan untuk memuat pratinjau saat memodifikasi kueri Anda.

Anda baru saja mempelajari tentang jenis operasi yang memerlukan pemindaian sebagian atau penuh pada kumpulan data Anda. Strategi lain untuk meningkatkan kinerja kueri Anda adalah dengan menyimpan bagian kueri Anda dalam memori sebelum menerapkan operasi mahal. Ini dapat dilakukan dengan menggunakan fungsi buffer, yang akan kita bahas selanjutnya.

Menggunakan Fungsi Buffer

Fungsi buffer dalam bahasa M, yang diidentifikasi dengan suffix *.Buffer*, memainkan peran penting dalam meningkatkan kinerja operasi Power Query. Fungsi ini menyimpan data sementara di memori komputer Anda. Proses ini, yang disebut **buffering**, dapat mempercepat kueri Anda tetapi juga memiliki beberapa keterbatasan.

Fungsi buffer paling berguna dalam skenario saat Power Query dapat menjalankan beberapa permintaan untuk mengambil data yang sama dari sumber eksternal. Situasi ini dapat terjadi selama operasi kompleks yang memerlukan akses data berulang kali, seperti dalam transformasi baris demi baris yang terperinci. Anda juga melihat ini saat menggunakan tabel pengganti untuk melakukan beberapa penggantian.

Karena meminta data yang sama berulang kali dapat menjadi lambat, terutama dari sumber eksternal, Anda dapat mempertimbangkan untuk melakukan buffering data ke dalam memori. Dengan cara itu, data hanya diambil sekali dan dapat digunakan kembali dari memori. Skenario lain tempat Anda mendapatkan manfaat dari fungsi buffer adalah saat menghitung total yang berjalan. Kinerja meningkat pesat saat Anda melakukan buffering nilai untuk total yang berjalan.

Di sisi lain, jika Anda melakukan buffer sejumlah besar data dalam memori dan melampaui batas memori kontainer mashup Anda, sistem akan mulai memindahkan data antara RAM dan penyimpanan (paging). Hal ini memperlambat kueri Anda karena kecepatan akses yang lebih lambat dibandingkan dengan RAM. Oleh karena itu, untuk memastikan kueri Anda cepat, penting untuk mempertimbangkan jejak memori kueri Anda. Di bagian berikutnya, kita akan melihat berbagai pendekatan untuk membuat total berjalan dan bagaimana fungsi buffer membantu meningkatkan kinerja kueri.

3. Dampak Buffering dan Total Berjalan

Di bagian ini, kita akan menggunakan contoh untuk mengukur kinerja dan konsumsi memori dari berbagai pendekatan. Sayangnya, Power Query tidak memiliki cara bawaan yang mudah untuk mengukur kinerjanya. Sementara diagnostik kueri tersedia di Power BI Desktop, fitur tersebut menyulitkan untuk menginterpretasikan metriknya.

Untuk pengukuran kecepatan kueri yang lebih akurat, kita telah menggunakan alat yang disebut **SQL-profiler** dan menghubungkannya ke

kumpulan data Power BI kita. Kita telah menghitung waktu penyegaran kita dengan menyegarkan setiap kueri secara terpisah tiga kali dan mengambil waktu penyegaran rata-rata. Mari kita lihat seperti apa pengaturan kita.

Pengaturan

Pertimbangkan kumpulan data yang berisi transaksi parkir, masing-masing dengan jumlah pembayaran terkait:

	Transaction ID	Meter Code	Transaction DateTime	Amount Paid
1	1250162207	12028002	11/7/2023 7:30:00 AM	26
2	1250162278	19232002	11/7/2023 7:31:00 AM	152
3	1250131414	19127010	11/7/2023 4:20:00 AM	967
4	1250134465	5073002	11/7/2023 4:48:00 AM	4
5	1250134860	19161010	11/7/2023 4:52:00 AM	673

Gambar 3. 46 Contoh kumpulan data transaksi parkir

Anda dapat mengikuti contoh ini dengan menggunakan file latihan yang disediakan di halaman GitHub buku ini.

Tujuan kita adalah menghitung total kumulatif pembayaran ini. Untuk menghitung total berjalan, kita menggunakan dua fungsi: *List.Sum* dan *List.FirstN*. *List.FirstN* memilih nilai hingga baris tertentu, dan List.Sum menambahkan nilai ini untuk memberikan total kumulatif. Kita mulai dengan menambahkan kolom indeks yang dimulai dari 1, bertambah 1 untuk setiap baris. Ini menjadi dasar kueri kita:

```
let
  Source = myCSV,
  AddedIndex =
    Table.AddIndexColumn(Source, "Index", 1, 1, Int64.Type)
in
    AddedIndex
```

Berikutnya, kita akan menjelajahi empat metode untuk menghitung total berjalan:

1) Dari tabel biasa
2) Dari tabel buffer
3) Dari kolom buffer

4) Dari kolom buffer dan menggunakan *List.Generate*

Setiap metode menawarkan pendekatan alternatif untuk menghitung total berjalan, yang berpotensi memengaruhi efisiensi dan kinerja proses.

Metode 1: Total Berjalan Dari Tabel Biasa

Untuk metode pertama, kita akan menambahkan kolom baru dengan ekspresi berikut:

```
List.Sum( List.FirstN( AddedIndex[Amount Paid], [Index] ))
```

Rumus ini menghitung total berjalan untuk setiap transaksi:

```
= Table.AddColumn(AddedIndex, "Running Total", each
List.Sum( List.FirstN( AddedIndex[Amount Paid], [Index] ) ) )
```

		Amount Paid	Index	Running Total
1	:00 AM	26	1	26
2	:00 AM	152	2	178
3	:00 AM	967	3	1145
4	:00 AM	4	4	1149

Gambar 3. 47 Menambahkan total berjalan tanpa nilai buffering

Saat mengevaluasi kinerja metode ini di Power BI, kita mengamati:

- **Waktu pemrosesan**: Rata-rata diperlukan waktu 128 detik.
- **Data yang diambil**: Metode ini mengambil data sebanyak 1,88 GB.

Mengingat file data asli hanya berukuran 393 KB, jumlah data yang diproses yang cukup besar menunjukkan bahwa Power Query membaca file tersebut beberapa kali. Ketidakefisienan ini merupakan kelemahan signifikan dari penggunaan tabel biasa untuk menjalankan kalkulasi total.

Metode 2: Menjalankan Total Dari Tabel Buffer

Metode 2 memperkenalkan variasi dalam menghitung total berjalan. Alih-alih menggunakan tabel biasa, metode ini melibatkan penyimpanan tabel dalam memori. Hal ini dicapai dengan fungsi *Table.Buffer*, seperti yang ditunjukkan dalam kueri berikut:

```
let
  Source = myCSV,
  AddedIndex =
    Table.AddIndexColumn(Source, "Index", 1, 1, Int64.Type),
  BufferTable = Table.Buffer( AddedIndex ),
  AddRunningTotal = Table.AddColumn(BufferTable, "Running Total",
    each List.Sum( List.FirstN( BufferTable[Amount Paid], [Index] )))
in
  AddRunningTotal
```

Pada langkah ini, *Table.Buffer* memuat data ke dalam memori. Keuntungannya di sini adalah bahwa alih-alih membuat beberapa permintaan ke berkas sumber, data sekarang disimpan dalam memori. Operasi selanjutnya pada tabel ini dapat menggunakan data dalam memori.

Metrik kinerja untuk metode ini di Power BI Desktop menunjukkan peningkatan yang signifikan:

- **Waktu pemrosesan**: Dikurangi menjadi rata-rata 61 detik.
- **Data yang diambil**: Hanya 394 KB data yang diambil.

Pendekatan ini meningkatkan kecepatan kueri hingga 52,3% dan mengurangi konsumsi datanya. Total data yang diambil kini sesuai dengan ukuran file CSV asli. Ini juga menunjukkan seberapa efektif fungsi *Table.Buffer* dalam mengoptimalkan kinerja.

Metode 3: Menjalankan Total Dari Kolom Buffer

Sebelumnya dalam bab ini, kita menekankan pentingnya meminimalkan penggunaan memori untuk mesin mashup. Mengikuti prinsip ini, metode 3 hanya menyimpan kolom tertentu yang digunakan dalam perhitungan total yang sedang berjalan. Pendekatan ini mengurangi jejak memori dan diharapkan dapat mengurangi waktu penyegaran kueri. Struktur kueri untuk metode ini adalah sebagai berikut:

```
let
  Source = myCSV,
  AddedIndex =
    Table.AddIndexColumn(Source, "Index", 1, 1, Int64.Type),
  BufferColumn = List.Buffer( AddedIndex[Amount Paid] ),
  AddRunningTotal = Table.AddColumn(AddedIndex, "Running Total",
    each List.Sum( List.FirstN( BufferColumn, [Index] )))
in
  AddRunningTotal
```

Dalam kueri ini, langkah BufferColumn secara khusus menargetkan kolom Jumlah yang Dibayar untuk buffering. Hasil kinerja untuk metode ini menunjukkan peningkatan signifikan lainnya:

- **Waktu pemrosesan**: Sangat berkurang hingga rata-rata 1,2 detik.
- **Data yang diambil**: Berjumlah 786 KB.

Metode ini mencapai pengurangan waktu penyegaran yang mencengangkan sebesar 99,1% dibandingkan dengan kueri asli. Pengamatan yang menarik adalah bahwa metode ini mengakses file CSV dua kali—kemungkinan besar sekali untuk tabel asli dan sekali untuk buffer kolom yang digunakan dalam kalkulasi total berjalan. Ini menunjukkan bahwa jumlah data yang diambil tidak selalu menjadi indikator kecepatan kueri. Sebaliknya, skenario ini menunjukkan keseimbangan strategis antara efisiensi memori dan kecepatan pemrosesan.

Apakah ini berarti tidak ada lagi yang perlu ditingkatkan? Yah, tidak juga. Metode *List.FirstN* untuk membuat total berjalan bukanlah yang paling efisien. Untuk meningkatkan hal-hal lebih jauh, kita dapat menggunakan fungsi *List.Generate*.

Metode 4: Menjalankan Total Dari Kolom Buffer Menggunakan *List.Generate*

Metode *List.FirstN*, meskipun fungsional, bukanlah yang paling efisien untuk menjumlahkan nilai dalam kalkulasi total berjalan. Metode ini memerlukan pengambilan dan penambahan rentang nilai untuk setiap baris.

Teknik alternatif dan lebih efisien adalah penggunaan fungsi *List.Generate*. Metode ini menghitung total berjalan dengan menambahkan setiap nilai baru ke total yang dihitung sebelumnya, alih-alih menghitung ulang jumlah tersebut pada rentang nilai yang terus bertambah setiap kali. Untuk pembahasan mendalam tentang metode *List.Generate* ini, lihat Bab 14, tetapi untuk saat ini, mari kita bahas metode ini dalam konteks saat ini. Kueri menggunakan *List.Generate* adalah sebagai berikut:

```
let
  Source = myCSV,
  BufferedColumn = List.Buffer( Source[Amount Paid] ),
  RunningTotal =
    List.Generate (
      () => [ RT = BufferedColumn{0}, RowIndex = 0 ],
      each [RowIndex] < List.Count( BufferedColumn ),
      each [ RT = List.Sum( { [RT], BufferedColumn{[RowIndex] + 1} }),
             RowIndex = [RowIndex] + 1 ],
      each [RT] ),
  CombineTables = Table.FromColumns(
    Table.ToColumns( Source )   & { RunningTotal },
    Table.ColumnNames( Source ) & { "Running Total" }  )
in
  CombineTables
```

Metode ini masih hanya menyimpan satu kolom. Fungsi *List.Generate* kemudian menghitung total berjalan secara efisien. Terakhir, kolom tabel asli digabungkan dengan kolom total berjalan yang baru dibuat.

Kinerja metode yang direvisi ini di Power BI Desktop adalah sebagai berikut:

- **Waktu pemrosesan**: Berkurang drastis hingga rata-rata 201 milidetik.
- **Data yang diambil**: Berjumlah 786 KB.

Metode ini menunjukkan peningkatan luar biasa sebesar 99,8% dalam waktu pemrosesan dibandingkan dengan pendekatan asli. Tabel berikut merangkum kinerja keempat metode berbeda ini:

Tabel 3. 1 Membandingkan kinerja berbagai metode

Metode	Deskripsi	Waktu pengerjaan	Data Diperoleh	Peningkatan
1	`List.FirstN` + Regular Table	128 seconds	1.88 GB	0.0%
2	`List.FirstN` + Buffered Table	61 seconds	394 KB	52.3%
3	`List.FirstN` + Buffered Column	1.2 seconds	786 KB	99.1%
4	`List.Generate` + Buffered Column	201 milliseconds	786 KB	99.8%

Seperti yang ditunjukkan bagian ini, fungsi buffer seperti *List.Buffer* dan *Table.Buffer* dapat memengaruhi memori yang digunakan untuk melakukan kalkulasi dalam M secara signifikan, terutama saat kueri Anda mengakses sumber data beberapa kali. Anda dapat mencoba salah satu fungsi ini setiap kali Anda menyadari kueri Anda menggunakan lebih banyak memori daripada yang dibutuhkan. Meskipun ini merupakan pengoptimalan yang sangat baik, jangan lupa untuk mencari pendekatan alternatif yang menggunakan memori secara lebih efisien, seperti *List.Generate* dalam contoh ini.

Performa kueri Anda tidak hanya bergantung pada bagaimana Anda menyusun kueri Anda. Pertimbangan penting adalah kecepatan sumber data Anda, yang akan kita bahas selanjutnya.

4. Pertimbangan Sumber Data

Saat menangani kueri yang lambat, pertanyaan yang bagus adalah di mana kueri Anda menghabiskan waktunya. Bahkan jika Anda berhasil menjaga jejak memori kueri Anda tetap kecil, jika sumber data Anda lambat, Anda akan menunggu data Anda masuk. Dengan kata lain, memilih sumber data yang tepat dapat sangat memengaruhi kinerja kueri Anda.

Sumber Data dan Kecepatan

Meskipun tidak selalu memungkinkan untuk beralih ke sumber data lain, ada baiknya untuk mengetahui beberapa aturan umum sumber data dan

kinerjanya. Berikut adalah beberapa petunjuk tentang apa yang perlu dipikirkan:

- **Basis data relasional**: Basis data ini cenderung berkinerja lebih baik daripada file. Bonusnya adalah banyak konektor basis data mendukung pelipatan kueri dan memungkinkan Anda memuat data secara bertahap.
- **Files**: Simpan data dalam file CSV atau teks saat bekerja dengan file. Kinerjanya jauh lebih baik daripada file JSON, XML, dan Excel.
- **Lokasi jaringan**: Mengambil file dari lokasi jaringan seperti OneDrive atau SharePoint cenderung lebih lambat daripada file lokal. Saat mengembangkan, sebaiknya pertimbangkan untuk menyimpan file Anda secara lokal.
- **Web/internet**: Koneksi ke layanan web atau API mungkin lambat. Demikian pula, mengimpor data dalam jumlah besar melalui internet bisa lambat.

Saat bekerja dengan sumber data di atas, kemungkinan besar sumber data tersebut berisi data sumber Anda. Anda mungkin menemukan diri Anda dalam situasi di mana sifat sumber-sumber ini memperlambat kueri Anda, terutama saat Anda bekerja dengan sumber data lama atau data dari lokasi jaringan. Jika Anda menemukan diri Anda dalam situasi itu, Anda dapat mempertimbangkan untuk memanfaatkan fitur yang disebut dataflows, yang akan kita bahas selanjutnya.

Menggunakan Aliran Data

Beberapa sumber data mungkin sulit dioptimalkan. Sumber data tersebut mungkin tidak mendukung pelipatan kueri, lambat secara alami, atau tidak memiliki sumber daya untuk menangani permintaan data yang besar. Anda mungkin tidak dapat melakukan banyak hal untuk mempercepatnya dalam kasus ini. Namun, jika Anda bekerja dengan data dari sumber yang lambat, pertimbangkan untuk menggunakan aliran data, yang juga dikenal sebagai aliran data analitis.

Aliran data terhubung ke sumber data menggunakan Power Query. Anda dapat mengaturnya untuk mengumpulkan data dari sumber yang lambat secara teratur. Setelah mengumpulkan data, aliran data menyimpannya di Azure Data Lake Storage milik Power BI. Di sini, data disimpan dalam format yang dikenal sebagai folder **CDM (Common Data Model)**. Folder ini menyertakan file CSV untuk setiap bagian data Anda dan file metadata JSON yang menjelaskan konten dan struktur data.

Sekarang, ingatkah Anda sumber data yang dulunya lambat? Anda dapat menggunakan aliran data untuk menyimpan datanya. Alih-alih terhubung langsung ke sumber yang lambat, kueri Anda dapat terhubung ke aliran data di Power Query, yang dapat mempercepat pemrosesan kueri Anda secara signifikan.

Bagi perusahaan dengan Power Premium, mesin komputasi yang disempurnakan merupakan fitur yang hebat untuk meningkatkan alur data. Mesin ini mempercepat seberapa cepat alur data merespons permintaan Anda. Bagian terbaiknya adalah kemampuannya untuk menggunakan pelipatan kueri. Kueri yang dikirim untuk mengambil data dari alur data dieksekusi terhadap cache di SQL, bukan folder CDM untuk alur data tanpa mengaktifkan fitur ini.

Kita telah membahas bagaimana alur data dapat menjadi platform yang hebat untuk menyimpan data dari sumber data Anda yang lambat. Keuntungan lainnya adalah kemampuannya untuk menyimpan output kueri Anda. Bila Anda memiliki banyak kueri di editor kueri, kueri cenderung menjadi lambat, kemungkinan besar karena beban pemuatan kueri di latar belakang. Untuk mempercepat antarmuka pengguna, ada baiknya mengurangi jumlah kueri. Cara yang baik untuk melakukannya adalah dengan menyimpan sebagian logika kueri Anda dalam alur data. Kemudian, Anda dapat membuat satu kueri yang terhubung ke alur data ini untuk mendapatkan hasil kueri asli Anda. Pendekatan ini tidak hanya mempercepat antarmuka pengguna tetapi juga membuat kueri Anda diperbarui lebih cepat karena aliran data hanya

mengembalikan output kueri Anda alih-alih menghitungnya selama waktu pembaruan.

Untuk mempelajari lebih lanjut tentang aliran data, Anda dapat membaca artikel ini: https://learn.microsoft.com/en-us/power-bi/transform-model/dataflows/dataflows-create.

F. Kiat kinerja

Mengoptimalkan kinerja adalah topik yang rumit, bahkan setelah mempelajari semua perspektif yang berbeda dalam bab ini. Seperti halnya semua topik pengoptimalan, Anda harus menguji berbagai pendekatan untuk situasi Anda. Namun, kita memiliki beberapa kiat kinerja umum:

- Hubungkan kueri Anda ke sumber data yang cepat. Jika sistem sumber Anda lambat, pertimbangkan untuk memindahkan data ke sumber yang lebih cepat atau menyimpan data Anda dalam aliran data.
- Maksimalkan jumlah langkah yang dapat dilipat oleh mekanisme pelipatan kueri Anda. Jika Anda ahli dalam SQL, pertimbangkan untuk membuat kueri SQL yang efisien dan terapkan parameter *EnableFolding*. Jika SQL bukan kekuatan Anda, prioritaskan penempatan langkah yang dapat dilipat di awal kueri Anda. Pendekatan ini membantu memastikan bahwa sumber data Anda menangani sebagian besar transformasi.
- Saat langkah kueri Anda tidak lagi dilipat, fokus berikutnya adalah mengurangi penggunaan memori kueri Anda. Itu berarti memfilter baris dan menghapus kolom sesegera mungkin. Kurangi juga ukuran kueri Anda sebanyak mungkin sebelum melakukan operasi buffering yang mahal seperti group by, pivoting, atau joins. Langkah selanjutnya hanya perlu menerapkan logikanya pada kumpulan data yang lebih kecil.
- Gunakan fungsi buffer untuk langkah yang meminta data berulang kali. Saat melakukannya, pastikan untuk melakukan buffer hanya pada data yang diperlukan. Ini berarti bahwa jika Anda perlu melakukan buffer pada nilai

kolom saja, lakukan buffer pada kolom tersebut saja; jangan melakukan buffer pada seluruh tabel.

Meskipun hal-hal di atas merupakan rekomendasi umum untuk membuat kueri Anda berkinerja baik, kita juga memiliki beberapa rekomendasi yang meningkatkan kecepatan kueri saat mengembangkan di editor kueri:

- Pertimbangkan untuk bekerja dengan sebagian data Anda untuk mempercepat pengembangan. Filter baris sedini mungkin sehingga kueri yang tersisa memiliki kebutuhan pemrosesan yang lebih sedikit.
- Cobalah untuk menggunakan operasi streaming sebanyak mungkin di awal kueri Anda. Ini memastikan layar pratinjau terisi dengan cepat dan memberikan pengalaman pengembangan yang lebih baik. Setelah operasi buffering masuk, kueri Anda perlu mengevaluasi semua baris dalam kumpulan data Anda.
- Setelah jumlah transformasi dan kueri meningkat, Power Query cenderung menjadi lambat. Pertimbangkan untuk menyimpan (sebagian) logika Anda dalam aliran data untuk mengurangi jumlah sumber daya yang digunakan saat mengembangkan.

G. Ringkasan

Dalam bab ini, kita telah mempelajari pendekatan multi-aspek yang diperlukan untuk mengoptimalkan kinerja di Power Query. Kita telah mempelajari area seperti penggunaan pelipatan kueri yang efektif dan strategi untuk menavigasi firewall rumus. Perbedaan antara operasi buffering dan streaming, bersama dengan penggunaan fungsi buffer yang cermat dan pentingnya mengurangi penggunaan memori dalam kueri Anda, merupakan area fokus yang penting.

Kunci dari bab ini adalah pemahaman bahwa manajemen aspek memori yang cermat membantu meningkatkan kinerja kueri. Dengan mengintegrasikan konsep dan pendekatan ini, Anda sekarang dilengkapi dengan pemahaman yang

komprehensif tentang berbagai elemen yang berkontribusi pada pengoptimalan kinerja operasi Power Query Anda.

Dalam bab berikutnya, kita akan fokus pada bekerja dengan ekstensi. Anda akan mempelajari konsep dasar yang diperlukan untuk membuat konektor kustom dan alat apa yang terlibat dalam proses tersebut.

H. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan konsep dasar pengoptimalan kinerja dalam Power Query. Mengapa pengoptimalan ini penting untuk efisiensi kueri?	10
2.	Diskusikan berbagai teknik yang dapat digunakan untuk meningkatkan kinerja kueri di Power Query. Sertakan contoh konkret dari aplikasi teknik tersebut.	10
3.	Apa yang dimaksud dengan 'folding query'? Bagaimana konsep ini berkontribusi pada pengoptimalan kinerja dalam Power Query?	10
4.	Analisis dampak penggunaan fungsi yang tidak efisien pada performa kueri. Berikan contoh fungsi yang umum menyebabkan masalah kinerja.	10
5.	Jelaskan peran tipe data dalam pengoptimalan kinerja. Bagaimana pemilihan tipe data yang tepat dapat mempengaruhi kecepatan pemrosesan?	10
6.	Apa saja perbedaan antara pengolahan data di Power Query dan di sumber data eksternal dalam konteks kinerja?	10
7.	Diskusikan pentingnya penggunaan variabel dalam ekspresi M untuk meningkatkan kinerja kueri. Sertakan contoh penggunaan variabel yang efektif.	10
8.	Bagaimana penggunaan teknik caching dapat mempengaruhi kinerja kueri di Power Query? Berikan penjelasan yang mendalam.	10
9.	Jelaskan bagaimana menerapkan langkah-langkah pengoptimalan kinerja pada kueri yang kompleks. Apa saja tantangan yang mungkin dihadapi?	10
10.	Analisis bagaimana pengujian dan debugging dapat membantu dalam mengidentifikasi masalah kinerja dalam kueri Power Query. Apa alat dan teknik yang dapat digunakan?.	10

BAB 4

Mengaktifkan Ekstensi

Topik Bab

Pada bab ini, topik yang akan dibahas adalah:

- Apa itu ekstensi Power Query?
- Mempersiapkan lingkungan Anda
- Membuat konektor khusus
- Memasang dan menggunakan konektor khusus

A. Pendahuluan

Power Query mendukung beragam konektor sumber data yang berbeda untuk mengakses dan mengambil data dari berbagai sumber data yang berbeda. Faktanya, Power BI Desktop secara native mendukung hampir 200 konektor sumber data yang berbeda, yang sebagian besar telah dibahas dalam Bab 3, Mengakses dan Menggabungkan Data. Selain itu, banyak dari konektor sumber data ini bersifat generik untuk berbagai standar dan protokol format data seperti **Open Data Protocol (OData), Open Database Connectivity (ODBC), JavaScript Object Notation (JSON), Extensible Markup Language (XML)**, dan Parquet. Dukungan untuk standar dan protokol umum ini secara signifikan memperluas potensi sumber data untuk Power Query hingga ribuan, jika tidak puluhan ribu sumber potensial.

Bahkan dengan semua sumber data yang telah didukung oleh Power Query, masih ada kebutuhan untuk mendukung sumber data tambahan. Secara khusus, berbagai layanan web sering kali menyediakan **Application Programming Interface (API)** yang harus digunakan untuk mengakses datanya. Bab ini menunjukkan cara membuat ekstensi khusus untuk Power Query guna menangani skenario semacam ini, termasuk topik berikut:

- Apa itu ekstensi Power Query?
- Mempersiapkan lingkungan Anda
- Membuat konektor khusus
- Menginstal dan menggunakan konektor khusus

1. Kasus Pemantik Berfikir Kritis

Seorang analis data di perusahaan retail sedang mengembangkan laporan penjualan bulanan menggunakan Power Query. Data penjualan disimpan dalam berbagai file Excel dengan nama yang berbeda untuk setiap bulan, seperti *Sales_Jan.xlsx, Sales_Feb.xlsx*, dan seterusnya.

Manajer meminta agar laporan ini bersifat dinamis, sehingga tidak perlu mengubah kode setiap kali bulan baru dimulai. Analis ingin menggunakan

ekstensi dan fungsi Expression.Evaluate untuk menyusun kueri berdasarkan nama file dan tabel yang ditentukan pengguna.

Pertanyaan:

1) Apa langkah-langkah yang perlu diambil oleh analis untuk mengimplementasikan ekstensi yang dapat menangani nama file dinamis?
2) Bagaimana fungsi Expression.Evaluate dapat digunakan untuk menyusun kueri yang fleksibel? Berikan contoh bagaimana ini dapat diterapkan dalam konteks laporan bulanan.
3) Apa tantangan yang mungkin dihadapi analis dalam mengembangkan solusi ini, dan bagaimana cara mengatasinya?
4) Diskusikan pentingnya metadata dalam konteks laporan ini. Informasi apa yang perlu dilacak untuk memastikan kualitas dan akurasi data?
5) Bagaimana analis dapat memastikan bahwa ekstensi yang dikembangkan dapat diintegrasikan dengan baik ke dalam alur kerja Power Query yang ada?
6) Apa peran lingkungan global dalam pengembangan ekstensi ini? Bagaimana variabel yang didefinisikan dalam lingkungan global dapat memengaruhi fungsi yang dijalankan?
7) Jelaskan bagaimana proses pengujian ekstensi dapat dilakukan untuk memastikan bahwa semua fungsionalitas bekerja dengan baik. Apa saja aspek yang perlu diuji?
8) Apa keuntungan menggunakan fungsi Expression.Evaluate dibandingkan dengan metode pengkodean tradisional? Dalam situasi apa penggunaan fungsi ini lebih efektif?
9) Diskusikan bagaimana pengaturan lokal dapat memengaruhi hasil laporan penjualan. Apa langkah yang dapat diambil untuk memastikan bahwa data tetap konsisten di berbagai pengaturan lokal?

B. Persyaran Teknis

Anda akan memerlukan hal berikut untuk menyelesaikan tugas dalam bab ini:

- Visual Studio Code
- Power Query SDK

C. Apa Itu Ekstensi Power Query?

Singkatnya, ekstensi Power Query terutama merupakan bagian dari kode M yang ditambahkan ke lingkungan global. Lihat Bab 7, Konseptualisasi M, untuk memahami lebih lanjut tentang lingkungan global. Ekstensi terutama digunakan untuk membuat konektor data kustom tetapi juga dapat digunakan untuk membuat pustaka fungsi kustom yang dapat digunakan kembali. Ekstensi ini tersedia dalam empat jenis file (ekstensi file) yang berbeda:

- **m**: Ini adalah file teks sederhana yang berisi kode M.
- **Pq**: Ini adalah file teks sederhana yang berisi kode M.
- **mez**: Ini adalah file ZIP standar yang berisi file konektor kustom. Dapat menggunakan file ZIP standar dan mengganti namanya dari *file.zip* menjadi *file.mez*.
- **pqx**: Ini adalah file ZIP **Open Packaging Conventions (OPC)** yang berisi file konektor kustom. Biasanya dikemas menggunakan pustaka *System.IO.Packaging .NET*, format pengemasan ini memungkinkan penandatanganan digital file dan merupakan format untuk konektor bersertifikat Microsoft.

Seperti yang disebutkan, file *.mez* dan *.pqx* adalah kumpulan file yang digunakan untuk membangun konektor sumber data kustom. Kita akan membahas pembuatan konektor kustom nanti di bab ini. Kumpulan file ini selalu menyertakan satu file *.m* atau *.pq*, yang merupakan file utama yang menyimpan kode konektor kustom.

Penting untuk dicatat bahwa ekstensi Power Query diimplementasikan menggunakan kata kunci *section* dan fungsi dalam section diekspos ke lingkungan global menggunakan kata kunci shared. Section diimplementasikan sebagai kode M standar dengan beberapa aturan sintaksis khusus.

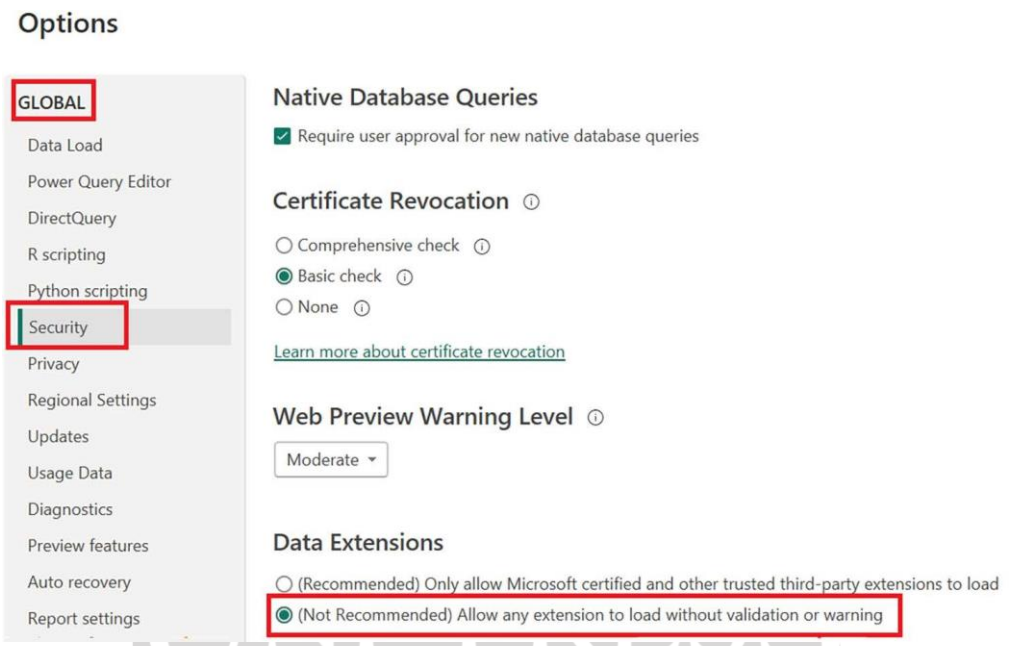
Perhatikan kode berikut. Perhatikan penggunaan khusus titik koma (;) dalam sintaksis section. Sama seperti beberapa ekspresi dalam pernyataan let dipisahkan dengan koma (,), beberapa ekspresi dalam *section* dipisahkan dengan titik koma:

```
section MyAwesomeExtension;

shared MyAwesomeExtension.DoSomething = (firstNumber as number, secondNumber as
number) as number =>
     MyAwesomeExtension.SquareNumber(firstNumber) + MyAwesomeExtension.
SquareNumber(secondNumber);

MyAwesomeExtension.SquareNumber = (paramNumber as number) as number =>
paramNumber * paramNumber;
```

Bagian ini memiliki satu fungsi bersama, *MyAwesomeExtension.DoSomething*, dan satu fungsi internal yang hanya dapat diakses di dalam bagian tersebut, *MyAwesomeExtension.SquareNumber*. Fungsi *MyAwesomeExtension.DoSomething* memanggil *MyAwesomeExtension.SquareNumber* untuk mengkuadratkan kedua angka tersebut dan menjumlahkannya.

Tempel kode ini ke dalam editor teks dan simpan file dengan nama file *MyAwesomeExtension*.pq. Tempatkan file ini ke dalam folder *[Documents]\Microsoft Power BI Desktop\Custom Connectors*. Anda mungkin harus membuat folder ini.

Buka Power BI Desktop dan navigasikan ke **File | Options** and **settings** | **Options**. Di bawah judul **GLOBAL**, pilih **Security**, lalu di bawah **Data Extensions**, pilih tombol radio **untuk (Not Recommended) Allow any extension to load without validation or warning**:



Gambar 4. 1 Aktifkan ekstensi yang tidak bersertifikat

Tutup Power BI Desktop, lalu luncurkan kembali Power BI Desktop. Buka editor Power Query di Power BI Desktop dan buat kueri berikut:

```
let
    Source = MyAwesomeExtension.DoSomething(10, 5)
in
    Source
```

Kueri ini menghasilkan 125, hasil dari mengkuadratkan 10 dan 5 dan menambahkannya bersama-sama. Perhatikan bahwa hanya fungsi *MyAwesomeExtension.DoSomething* yang diekspos dan dapat digunakan dalam kueri. Fungsi *MyAwesomeExtension.SquareNumber* tidak dapat dirujuk karena tidak diawali dengan kata kunci shared.

Pada titik ini, Anda seharusnya dapat memahami struktur dasar dan sifat ekstensi Power Query tetapi mungkin bertanya pada diri sendiri bagaimana Anda dapat menggunakannya.

1. Apa yang Dapat Anda Lakukan Dengan Ekstensi?

Ada dua penggunaan utama untuk ekstensi dalam Power Query:

- Pustaka fungsi kustom

- Konektor sumber data kustom

Seperti yang telah Anda lihat, Anda dapat menggunakan ekstensi untuk menyediakan pustaka fungsi kustom yang dapat digunakan kembali untuk digunakan dalam kueri Anda. Alih-alih menyalin dan menempel kode di antara proyek, Anda dapat menyertakan fungsi kustom Anda di bagian dengan kata kunci shared. Meskipun kemampuan ini ada, itu bukanlah penggunaan utama ekstensi dan tidak digunakan secara luas.

Konektor sumber data kustom adalah penggunaan utama lain untuk ekstensi Power Query. Meskipun Power BI Desktop dilengkapi dengan hampir 200 konektor, masih banyak sumber data lain, terutama sumber data yang menggunakan API kustom, yang tidak disertakan dalam produk. Ekstensi Power Query memungkinkan pengembang membuat konektor kustom yang terhubung ke sumber data tambahan ini. Dalam bab ini, kita menggunakan Discord, platform pesan instan dan **Voice over Internet Protocol (VoIP)** yang populer sebagai contoh.

Konektor kustom dapat digunakan untuk membuat sumber data baru atau menyesuaikan dan memperluas sumber yang sudah ada. Saat dipasang, konektor ini memperluas lingkungan global Power Query, menyediakan fungsi akses data baru yang tidak ada dalam bahasa M secara default.

Sisa bab ini difokuskan pada pembuatan konektor sumber data khusus, yang dimulai dengan pemrosesan persiapan lingkungan Anda untuk pengembangan.

D. Mempersiapkan Lingkungan Anda

Untuk menyiapkan lingkungan Anda, Anda harus mengunduh dan memasang dua komponen perangkat lunak:

- Visual Studio Code
- **software development kit (SDK)** Power Query

Selain itu, contoh konektor kustom yang dibahas dalam bab ini terhubung ke Discord, platform sosial pesan suara dan instan yang populer. Kita akan menggunakan API Discord untuk terhubung ke server Discord dan mengambil informasi. Untuk menggunakan Discord, kita juga perlu memasang dan mengonfigurasi dua komponen tambahan, yaitu:

- Layanan Informasi Internet
- Discord

Mari kita lihat cara memasang dan mengonfigurasi komponen perangkat lunak ini.

1. Mendapatkan Kode Visual Studio

Saat ini ada dua versi Power Query: versi yang dirilis pada tahun 2017 yang berfungsi sebagai ekstensi untuk Visual Studio 2017 dan 2019 dan versi baru (saat ini dalam pratinjau pada tanggal penerbitan buku ini) yang berfungsi dengan Visual Studio Code.

Untuk buku ini, kita akan menggunakan versi baru Power BI SDK untuk **Visual Studio Code**. Untuk memulai, unduh dan instal versi terbaru Visual Studio Code di sini: https://code.visualstudio.com/download.

Bahasa Indonesia: Jika Anda hanya familier dengan Visual Studio 2017, 2019, atau 2022, Anda mungkin akan merasa Visual Studio Code agak asing dan mungkin sedikit menjengkelkan untuk digunakan. Bahkan, Anda tidak akan dikucilkan jika Anda merasa bahwa akronim GUI adalah singkatan dari Gratuitously Unintuitive Interface dalam hal Visual Studio Code. Ada banyak perbedaan dan keanehan aneh dalam produk Visual Studio Code yang dapat menyebabkan kebingungan. Sementara pembahasan lengkap tentang pengalaman dalam Visual Studio Code berada di luar cakupan buku ini, upaya signifikan telah dilakukan untuk menunjukkan di mana dan bagaimana menavigasi Visual Studio Code untuk tujuan membuat konektor kustom. Selain itu, langkah-langkah dalam bagian Greating a custom connector telah

dirancang khusus untuk meminimalkan jumlah kesalahan atau pesan aneh. Oleh karena itu, ikuti langkah-langkah di bagian itu dengan saksama.

Pro tip: Terkadang, hanya dengan menyimpan pekerjaan Anda, menutup folder, lalu membukanya kembali, akan memberikan keajaiban.

Setelah Anda menginstal Visual Studio Code, langkah berikutnya adalah menginstal Power Query SDK.

2. Mendapatkan SDK Power Query

SDK Power Query dapat diunduh dari Visual Studio Marketplace: https://marketplace.visualstudio.com/items?itemName=PowerQuery.vscode-powerquery-sdk. Jika Anda memiliki Visual Studio 2017 atau 2019, maka Anda dapat mengunduh dan menginstal SDK versi lama di sini: https://marketplace.visualstudio.com/items?itemName=Dakahn.PowerQuerySDK. Petunjuk yang ditemukan dalam buku ini mengasumsikan bahwa Anda menggunakan SDK baru untuk Visual Studio Code.

Mengklik tombol **Install** pada halaman web untuk SDK Power Query baru untuk Visual Studio Code akan meluncurkan Visual Studio Code dan menampilkan halaman instalasi untuk SDK Power Query. Cukup klik tombol **Install** di sini untuk mengunduh dan menginstal SDK.

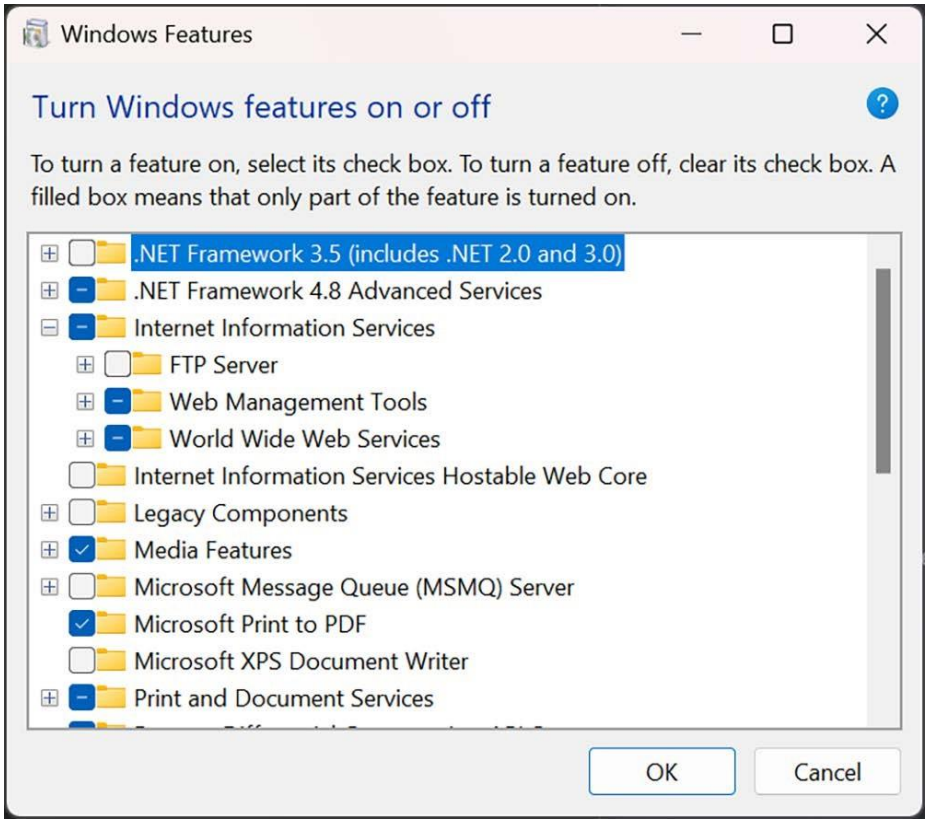
Setelah SDK Power Query diinstal, langkah berikutnya adalah mengonfigurasi server web lokal.

3. Menyiapkan Layanan Informasi Internet

Layanan Informasi Internet adalah server web yang dapat dikonfigurasi pada PC Windows mana pun. Untuk mengonfigurasi Layanan Informasi Internet pada komputer lokal Anda, ikuti langkah-langkah berikut:

1) Ketik *Turn Windows features on and off* pada bilah pencarian Windows dan luncurkan dialog **Windows Features**.

2) Pilih kotak centang di sebelah **Internet Information Services**:



Gambar 4. 2 Aktifkan Layanan Informasi Internet

3) Klik tombol **OK** untuk mengonfigurasi **Internet Information Services**. Secara default, ini akan otomatis memilih **Web Management Tools** dan **World Wide Web Services**, tetapi bukan **FTP Server**. Ini bagus untuk tujuan kita.

Dalam contoh konektor kustom kita, kita akan menggunakan **OAuth** untuk autentikasi. **OAuth** adalah singkatan dari **Open Authentication** dan merupakan standar terbuka yang memungkinkan pengguna untuk memberikan akses terbatas ke situs web atau layanan kepada aplikasi pihak ketiga. Kita akan menggunakan Layanan Informasi Internet lokal sebagai titik akhir untuk pengidentifikasi **Uniform Resource Identifier (URI)** pengalihan yang diperlukan oleh protokol OAuth.

Perlu dicatat bahwa tidak terlalu penting URI pengalihan apa yang disediakan dan URI aktual yang terlibat bahkan tidak harus benar-benar ada dalam beberapa kasus, termasuk dalam kasus contoh ini. Yang diperlukan hanyalah beberapa server web tersedia untuk berfungsi sebagai titik akhir URI pengalihan.

Sebagai alternatif untuk menggunakan server web lokal sebagai titik akhir kita, secara teori, kita dapat menggunakan hampir semua URI pengalihan OAuth, seperti https://oauth.powerbi.com/views/oauthredirect.html. Namun, hal ini dapat berisiko apabila nama domain dan layanan web tidak berada dalam kendali pengembang karena titik akhir tersebut mungkin pada suatu saat tidak lagi valid dan berpotensi menimbulkan kesalahan dalam proses autentikasi.

Perhatikan bahwa jenis kesalahan "**not valid**" ini berbeda dengan halaman yang tidak ada untuk server web. Misalnya, jazzysneakers123.com bukanlah nama domain yang valid dan karenanya akan menghasilkan kesalahan dalam proses autentikasi jika seseorang mencoba menggunakan jazzysneakers.com sebagai URI pengalihan. Ini berbeda dengan menggunakan halaman yang tidak valid untuk nama domain yang ada dan menanggapi permintaan sebagai server web. Dalam kasus terakhir, pesan **404 page not found** akan dihasilkan dan ini tidak akan secara otomatis menyebabkan proses autentikasi gagal.

Untuk tujuan pengembangan, kita akan menggunakan Layanan Informasi Internet lokal dan kita dapat merujuk ke server web ini sebagai localhost. Jelas, ini tidak akan cukup untuk konektor kustom produksi karena tidak semua orang memiliki server web lokal yang terpasang dan dikonfigurasi secara default. Skenario produksi akan mengharuskan pengembang untuk menyiapkan server web yang tersedia secara umum dan URI pengalihan yang sesuai. Skenario produksi semacam ini berada di luar cakupan buku ini, tetapi ini bisa semudah menyiapkan situs blog WordPress dengan atau tanpa domain kustom.

Langkah selanjutnya adalah menginstal dan mengonfigurasi Discord, jadi mari kita mulai!

4. Menyiapkan Discord

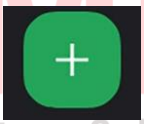
Untuk menyiapkan Discord agar dapat digunakan dengan Discord API, kita memerlukan hal berikut:

- Klien Discord
- Server Discord
- Aplikasi Discord (app)
- Konfigurasi OAuth

Mari kita tinjau cara mendapatkan dan mengonfigurasi masing-masing item ini.

Discord Client

Untuk mengunduh dan memasang klien Discord, gunakan peramban web untuk membuka https://discord.com. Klik tombol **Download for Windows** di beranda, lalu jalankan penginstalan untuk memasang Discord. Atau, Anda dapat menggunakan tombol **Open Discord in your browser**.



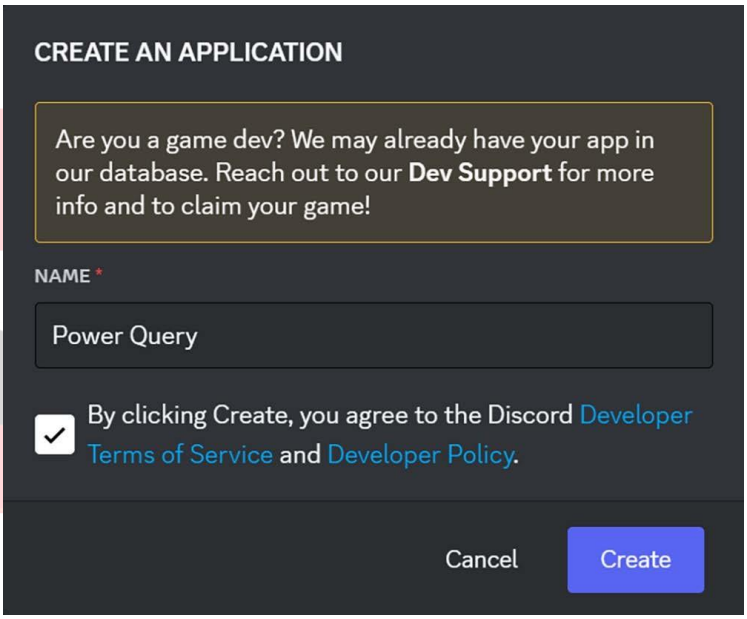
Gambar 4. 3 Tambahkan server Discord

Pilih jenis server yang akan dibuat dan berikan nama untuk server Anda. Klik tombol **Create** untuk membuat server. Setelah dibuat, server akan muncul di panel navigasi kiri klien Discord. Langkah berikutnya adalah membuat aplikasi Discord.

Aplikasi Discord

Untuk menggunakan API Discord, kita perlu membuat aplikasi Discord. Untuk membuat aplikasi Discord, ikuti langkah-langkah berikut:

1) Gunakan peramban web dan navigasikan ke halaman web berikut: https://discord.com/developers/applications.
2) Klik tombol **New Application**.
3) Berikan nama untuk aplikasi tersebut, lalu klik tombol **Create**:



Gambar 4. 4 *Create* aplikasi Discord

4) Klik tombol ***Create*** untuk membuat aplikasi.

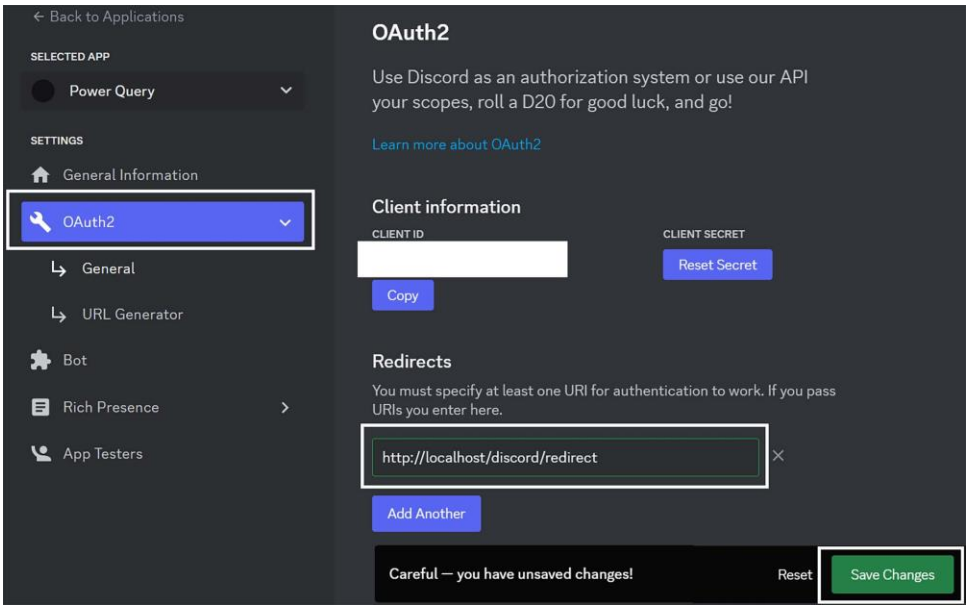
Setelah mengklik tombol **Create**, Anda akan dibawa ke halaman pengaturan aplikasi. Di sinilah kita dapat melakukan langkah berikutnya, mengonfigurasi OAuth, jadi mari kita lakukan itu selanjutnya. Aplikasi ini dapat ditambahkan ke server Discord meskipun untuk tujuan kita hal itu tidak diperlukan.

Konfigurasi OAuth Discord

Untuk menggunakan API guna mengirim permintaan ke Discord API atas nama pengguna, pengguna harus terlebih dahulu mengotorisasi aplikasi. Konfigurasi untuk ini dilakukan di halaman pengaturan aplikasi dan memerlukan langkah-langkah berikut:

1) Klik tautan **OAuth2** di panel navigasi kiri halaman pengaturan aplikasi.

2) Catat **CLIENT ID** di bawah **Client information**. Salin dan simpan ID klien ini karena kita akan membutuhkannya nanti saat membuat konektor khusus.
3) Selanjutnya, klik tombol Atur **Reset Secret** yang ditunjukkan pada gambar berikut:



Gambar 4. 5 Tambahkan pengalihan aplikasi OAuth2

4) Pastikan untuk menggunakan tombol **Copy** untuk menyalin kunci rahasia klien Anda dan menyimpan kunci ini. Anda akan membutuhkannya saat membuat konektor kustom.
5) Sekarang klik tombol **Add Redirect** dan tambahkan pengalihan http://localhost/discord/redirect, seperti yang ditunjukkan pada Gambar 16.5.
6) Terakhir, klik tombol **Save Changes** untuk menyimpan pengalihan.

Halaman web http://localhost/discord/redirect sebenarnya tidak ada. Bahkan, URI apa pun yang menggunakan localhost dapat digunakan untuk langkah ini. Yang penting adalah localhost merujuk ke PC lokal tempat server web lokal berjalan sebagaimana dikonfigurasi di bagian Menyiapkan Layanan Informasi Internet.

Kita sekarang telah selesai menyiapkan lingkungan. Sekarang mari beralih ke pembuatan konektor atau ekstensi kustom.

E. Membuat Konektor Khusus

Dengan Visual Studio Code, Power Query SDK terinstal, lingkungan lokal kita dikonfigurasi dengan IIS, dan elemen Discord dibuat dan dikonfigurasi, kita sekarang dapat membuat konektor kustom untuk Power Query.

Kode lengkap untuk konektor kustom untuk buku ini tersedia di repositori GitHub yang ditemukan di sini: https://github.com/PacktPublishing/The-Definitive-Guide-to-Power-Query-M-/tree/main/Chapter%2016. Anda dapat mengunduh file TDGTPQM_Discord.zip dari Bab 14 di repositori, mengekstrak file, menyalin dan menempel folder konektor di suatu tempat di PC Anda, lalu membuka folder di Visual Studio Code (**File | Open Folder…**), atau Anda dapat mengikuti bagian selanjutnya di mana kita membangun konektor dari awal.

Dalam kedua kasus tersebut, Anda akan terbantu dengan membaca penjelasan kode di seluruh bagian ini. Bagian ini mengasumsikan bahwa Anda membuat konektor kustom dari awal. Langkah-langkah untuk membuat konektor adalah sebagai berikut:

1) Buat proyek ekstensi.
2) Konfigurasikan autentikasi.
3) Konfigurasikan navigasi dan konten.

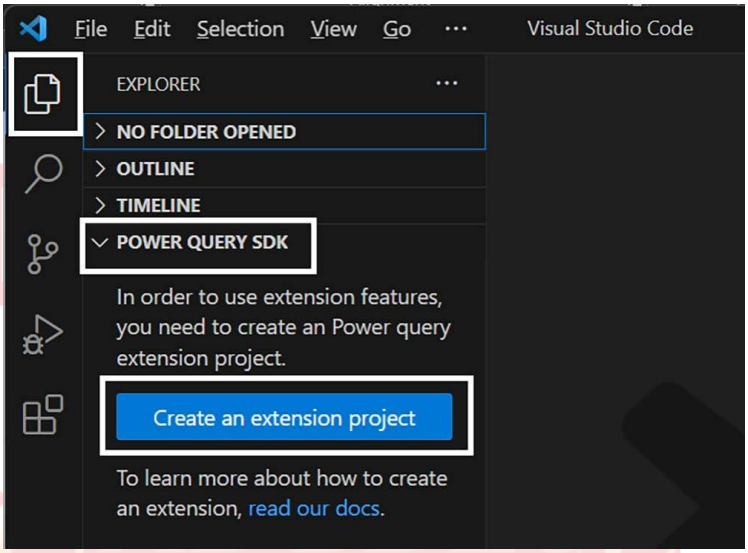
Mari kita lihat langkah pertama: membuat proyek ekstensi.

1. Membuat Proyek Ekstensi

Seperti yang telah disebutkan sebelumnya, kita akan membuat konektor kustom di Visual Studio Code, pendekatan yang saat ini disukai Microsoft sejak buku ini diterbitkan. Untuk membuat proyek ekstensi, lakukan hal berikut:

1) Luncurkan Visual Studio Code jika belum berjalan.

2) Akses **EXPLORER**, perluas bagian **POWER QUERY SDK**, lalu buat proyek ekstensi baru dengan mengeklik tombol **Create an extension project**:



Gambar 4. 6 Buat proyek ekstensi

3) Saat diminta, masukkan *TDGTPQM_Discord* sebagai **New project name** dan tekan tombol Enter.
4) Saat diminta folder untuk ruang kerja, buat folder baru bernama *TDGTPQM_Discord*, pilih folder tersebut, dan tekan **tombol Select workspace**.

Struktur file dan folder berikut dibuat:

- *TDGTPQM_Discord*
  - *.vscode*
    - *settings.json*
  - *bin*
    - *AnyCPU*
    - *Debug*
    - *TDGTPQM_Discord.mez*
  - *resources.resx*
  - *TDGTPQM_Discord.pq*

- *TDGTPQM_Discord.proj*
- *TDGTPQM_Discord.query.pq*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*
- *TDGTPQM_Discord.png*

Mari kita lihat lebih dekat berkas yang dibuat:

- *settings.json*: Berkas **JavaScript Open Notation (JSON)** yang menyimpan pengaturan ruang kerja.
- *TDGTPQM_Discord.mez*: Berkas ini terdapat dalam direktori *bin\AnyCPU\Debug* dan merupakan ekstensi sebenarnya yang dibuat oleh Power Query SDK yang Anda terapkan dan gunakan sebagai konektor kustom.
- *resources.resx*: Berkas XML yang utamanya menyimpan string yang digunakan dalam ekstensi.
- *TDGTPQM_Discord.pq*: Ini adalah berkas kode utama untuk konektor kustom. Hanya satu berkas *.pq* yang diizinkan dan formatnya adalah bagian M.
- *TDGTPQM_Discord.proj*: Berkas XML yang berisi pengaturan proyek.
- *TDGTPQM_Discord.query.pq*: Terutama digunakan sebagai metode untuk membuat kueri pengujian. Berkas ini hanya berisi kode berikut:

```
let
    result = TDGTPQM_Discord.Contents()
in
    result
```

Berkas ini hanya merujuk pada suatu fungsi (*TDGTPQM_Discord.Contents*) yang didefinisikan dalam *TDGTPQM_Discord.pq*, seperti yang akan kita lihat nanti. Kode ini cukup untuk keperluan kita dan tidak akan berubah.

- *.png files*: Ikon yang digunakan oleh konektor.

Untuk menulis konektor kustom kita, kita akan bekerja dengan berkas *TDGTPQM_Discord.pq*, jadi mari kita bahas berkas ini secara lebih rinci.

TDGTPQM_Discord.pq

Berkas ini berisi logika kode utama untuk konektor kustom dan merupakan berkas yang akan kita modifikasi untuk membuat konektor kustom. Seluruh konektor kustom diimplementasikan sebagai bagian sesuai kode ini di bagian atas berkas:

```
// This file contains your Data Connector logic
[Version = "1.0.0"]
section TDGTPQM_Discord;
```

Seperti yang dijelaskan sebelumnya, dokumen *section* menunjukkan bagaimana ekstensi diimplementasikan dalam M. Di bawah definisi bagian ini, kita menemukan kode berikut:

```
[DataSource.Kind="TDGTPQM_Discord", Publish="TDGTPQM_Discord.Publish"]
shared TDGTPQM_Discord.Contents = (optional message as text) =>
    let
        _message = if (message <> null) then message else "Hello World",
        a = "Hello from HelloWorld: " & _message
    in
        a;
```

TDGTPQM_Discord.Contents adalah fungsi utama yang akan kita gunakan untuk mengembalikan konten konektor kita. Perhatikan kata kunci shared sebelum definisi fungsi. Ini berarti bahwa fungsi ini dapat dirujuk di luar bagian tersebut.

Perhatikan juga definisi untuk *DataSource.Kind* dan *Publish.DataSource.Kind* didefinisikan sebagai *TDGTPQM_Discord*.

Menggunakan kata kunci shared bersama dengan definisi atribut literal *DataSource.Kind* untuk fungsi tersebut mengaitkan fungsi tersebut dengan sumber data tertentu, sumber data *TDGTPQM_Discord*.

Blok kode berikutnya adalah record definisi untuk sumber data terkait, *TDGTPQM_Discord*:

```
// Data Source Kind description
TDGTPQM_Discord = [
    Authentication = [
        // Key = [],
        // UsernamePassword = [],
        // Windows = [],
        Anonymous = []
    ]
];
```

Record definisi sumber data menentukan jenis autentikasi yang didukung sumber data. Dalam kasus ini, hanya autentikasi anonim atau implisit yang didukung sebagaimana yang ditentukan saat ini.

Selain atribut literal *DataSource.Kind*, ada juga atribut literal Publish yang dikaitkan dengan fungsi *TDGTPQM_Discord.Contents*. Blok kode di bawah record definisi sumber data adalah definisi record Publish:

```
// Data Source UI publishing description
TDGTPQM_Discord.Publish = [
    Beta = true,
    Category = "Other",
    ButtonText = { Extension.LoadString("ButtonTitle"), Extension.
LoadString("ButtonHelp") },
    LearnMoreUrl = "https://powerbi.microsoft.com/",
    SourceImage = TDGTPQM_Discord.Icons,
    SourceTypeImage = TDGTPQM_Discord.Icons
];
```

Record Publikasikan digunakan oleh **User Interface (UI)** Power Query (misalnya, editor Power Query) untuk menampilkan ekstensi sumber data melalui dialog **Get Data**. Pengaturan ini dapat disesuaikan untuk mengubah perilaku elemen UI tertentu, seperti kategori tempat konektor sumber data ditampilkan, apakah konektor dalam versi Beta atau tidak, dan tautan **Learn More** untuk konektor.

Record Publikasikan ini merujuk ke record tambahan, *TDGTPQM_Discord.Icons*, yang menyimpan referensi ke file gambar *.png*:

```
TDGTPQM_Discord.Icons = [
    Icon16 = { Extension.Contents("TDGTPQM_Discord16.png"), Extension.
Contents("TDGTPQM_Discord20.png"), Extension.Contents("TDGTPQM_Discord24.png"),
Extension.Contents("TDGTPQM_Discord32.png") },
    Icon32 = { Extension.Contents("TDGTPQM_Discord32.png"), Extension.
Contents("TDGTPQM_Discord40.png"), Extension.Contents("TDGTPQM_Discord48.png"),
Extension.Contents("TDGTPQM_Discord64.png") }
];
```

Ini adalah ikon yang ditampilkan dalam UI Power Query untuk konektor. Ini melengkapi eksplorasi terperinci kita tentang file *TDGTPQM_Discord.pq*. Anda mungkin terkejut dengan sedikitnya kode yang diperlukan untuk membuat konektor kustom; jangan salah, proyek ekstensi default yang dibuat oleh SDK Power Query sudah merupakan konektor kustom yang berfungsi. Meski begitu, konektor ini saat ini tidak benar-benar melakukan apa pun selain pada dasarnya mengembalikan padanan dari *Hello World*.

Dengan proyek ekstensi yang dibuat, sekarang kita dapat melanjutkan untuk menulis kode kustom untuk autentikasi.

2. Mengonfigurasi Otentikasi

Setelah proyek ekstensi dasar kita dibuat dan dijelaskan, sekarang saatnya untuk mengalihkan perhatian kita ke penyesuaian ekstensi untuk memenuhi kebutuhan kita dalam menghubungkan ke Discord. Tugas pertama kita dalam hal ini adalah membuat kode dan mengonfigurasi autentikasi untuk konektor.

Discord menggunakan protokol keamanan OAuth untuk autentikasi. Tersedia beberapa jenis autentikasi lain:

- Anonim
- AAD
- Nama pengguna dan kata sandi
- Windows
- Kunci

Masing-masing mekanisme autentikasi ini unik. Informasi selengkapnya tentang jenis autentikasi tambahan ini dapat ditemukan di sini: *https://learn.microsoft.com/en-us/power-query/handling-authentification*.

Autentikasi OAuth sejauh ini merupakan metode autentikasi paling rumit untuk ekstensi Power Query dan itulah sebabnya metode ini dipilih untuk buku ini sebagai contoh. Seperti yang disebutkan sebelumnya, OAuth adalah singkatan dari Open Authentication dan merupakan standar terbuka yang memungkinkan pengguna memberikan akses terbatas ke situs web atau layanan kepada aplikasi pihak ketiga. Dalam kasus kita, pengguna memberikan akses terbatas kepada konektor kustom pihak ketiga kita ke informasi Discord mereka.

Protokol *OAuth* memungkinkan akses tanpa pengguna harus membagikan kredensial mereka (nama pengguna dan kata sandi).

OAuth menyediakan cara yang aman dan terstandarisasi bagi pengguna untuk mengotorisasi aplikasi eksternal untuk mengakses data mereka atau melakukan tindakan atas nama mereka.

Alih-alih kredensial seperti nama pengguna dan kata sandi, OAuth menggunakan pertukaran rahasia klien dan token akses antara pengguna, aplikasi yang meminta akses, dan layanan yang menyediakan sumber daya.

Dengan menerapkan OAuth, penyedia layanan dapat memastikan bahwa kredensial pengguna tetap terlindungi dan tidak terekspos ke aplikasi eksternal. OAuth memungkinkan pengguna untuk mengontrol dan mencabut akses ke sumber daya mereka kapan saja, memberi mereka keamanan dan kontrol yang lebih baik atas data mereka.

Secara keseluruhan, OAuth memainkan peran penting dalam memungkinkan akses yang aman dan terkendali ke sumber daya, mempromosikan interoperabilitas, dan meningkatkan privasi pengguna di berbagai aplikasi web dan seluler. Anda kemungkinan besar pernah melihat

permintaan otorisasi OAuth di masa lalu, meskipun Anda mungkin tidak mengenalinya.

Untuk menerapkan autentikasi OAuth, kita perlu melakukan langkah-langkah berikut:

1) Tambahkan ID klien dan file rahasia klien.
2) Tambahkan pengaturan konfigurasi.
3) Buat fungsi OAuth.
4) Ubah definisi sumber data
5) Tambahkan kredensial.
6) Uji koneksi.

Menambahkan *Client ID* Klien dan *Secret Files*

Langkah pertama dalam mengonfigurasi autentikasi untuk konektor kustom kita adalah menambahkan file untuk menyimpan ID klien dan rahasia klien aplikasi kita. Untuk melakukannya, lakukan langkah-langkah berikut:

1) Dengan proyek ekstensi terbuka, pilih File | File Baru dari menu Visual Studio Code.
2) Saat diminta, masukkan client_id sebagai nama file, tekan tombol Enter, lalu tekan tombol Buat File.
3) File client_id secara otomatis dibuka di Visual Studio Code. Pada baris 1, cukup tempel ID klien yang Anda simpan sebagai bagian dari bagian konfigurasi Discord OAutb.
4) Ulangi langkah-langkah ini tetapi, kali ini, gunakan client_secret untuk nama file Anda dan tempel rahasia klien Anda yang disimpan sebagai bagian dari bagian konfigurasi Discord OAutb.

File yang baru saja Anda buat akan digunakan sebagai bagian dari pengaturan konfigurasi OAuth untuk konektor kustom kita, jadi mari tambahkan pengaturan konfigurasi tersebut sekarang.

Menambahkan Pengaturan Konfigurasi

Pengaturan konfigurasi hanyalah variabel yang didefinisikan dalam konektor kustom kita. Variabel ini dapat dianggap sebagai variabel global dalam arti bahwa setiap ekspresi atau fungsi yang didefinisikan dalam konektor kustom kita dapat mengakses dan menggunakan variabel ini. Untuk menambahkan pengaturan konfigurasi, lakukan hal berikut:

- Di bagian atas berkas Anda, Anda akan melihat baris berikut:

```
Section TDGTPQM_Discord;
```

- Tempel kode berikut tepat di bawah baris sebelumnya:

```
// OAuth Configuration Settings
client_id = Text.FromBinary(Extension.Contents("client_id"));
client_secret = Text.FromBinary(Extension.Contents("client_secret"));
redirect_uri = "http://localhost/discord/redirect";
authorize_uri = "https://discord.com/api/oauth2/authorize?";
token_uri = "https://discord.com/api/oauth2/token";

// Static variables
api_uri = "https://discord.com/api";

// Connector window
windowWidth = 1200;
windowHeight = 1000;
```

Untuk menjelaskan variabel yang dibuat dengan lebih baik, pertimbangkan hal berikut:

- *client_id*: Ini membaca file *client_id* yang dibuat di bagian Menambahkan *client ID* dan file rahasia klien dan menyimpan nilainya.
- *client_secret*: Ini membaca file *client_id* yang dibuat di bagian Menambahkan ID klien dan file rahasia klien dan menyimpan nilainya.
- *redirect_uri*: Ini menyimpan URI pengalihan yang digunakan saat mengonfigurasi Discord OAuth di bagian konfigurasi *Discord OAuth*. Ini harus merupakan URI pengalihan yang sama yang digunakan dalam konfigurasi tersebut.
- *authorize_uri*: Ini menyimpan nilai untuk URI otorisasi OAuth Discord.

- *token_uri*: Ini menyimpan nilai untuk URI pemberian token Discord.
- *api_uri*: Ini menyimpan URI API Discord dasar.
- *WindowWidth*: Ini menentukan lebar jendela untuk dialog autentikasi.
- *WindowHeight*: Ini menentukan tinggi jendela untuk dialog autentikasi.

Dengan variabel/pengaturan konfigurasi ini, kita sekarang dapat melanjutkan untuk menerapkan fungsi yang diperlukan yang ditentukan oleh jenis autentikasi OAuth Power Query, jadi mari kita lakukan sekarang.

Membuat Fungsi *OAuth*

Jenis autentikasi *OAuth* Power Query memerlukan penerapan dua bidang, *StartLogin* dan *FinishLogin*. Ada juga dua bidang opsional yang dapat diterapkan, *Refresh* dan *Logout*. Masing-masing bidang ini harus menentukan fungsi yang ditetapkan dalam konektor kustom kita. Kita akan menerapkan bidang/fungsi yang diperlukan tetapi bukan bidang/fungsi opsional sebagai bagian dari konektor kustom ini.

Di antara definisi record sumber data, *TDGTPQM_Discord*, dan definisi record *TDGTPQM_Discord.Publish*, masukkan kode berikut:

```
//
// OAuth2 flow definition
//
StartLogin = (resourceUrl, state, display) =>
    let
        AuthorizeUrl = authorize_uri
            & Uri.BuildQueryString(
                [
                    client_id = client_id,
                    redirect_uri = redirect_uri,
                    response_type = "code",
                    scope = "identify email connections guilds guilds.members.
read"
                ]
            )
    in
        [
```

```
            LoginUri = AuthorizeUrl,
            CallbackUri = redirect_uri,
            WindowHeight = windowHeight,
            WindowWidth = windowWidth,
            Context = null
        ];

FinishLogin = (context, callbackUri, state) =>
    let
        Parts = Uri.Parts(callbackUri)[Query]
    in
        TokenMethod(Parts[code]);

TokenMethod = (code) =>
    let
        Response = Web.Contents(
            token_uri,
            [
                Content = Text.ToBinary(
                    Uri.BuildQueryString(
                        [
                            client_id = client_id,
                            client_secret = client_secret,
                            grant_type = "authorization_code",
                            code = code,
                            redirect_uri = redirect_uri
                        ]
                    )
                ),
                Headers = [#"Content-type" = "application/x-www-form-
urlencoded", #"Accept" = "application/json"]
            ]
        ),
        Parts = Json.Document(Response)
    in
        Parts;
```

Perlu dicatat bahwa cara kerja internal tipe autentikasi OAuth Power Query agak tidak jelas. Penjelasan atau dokumentasi yang diberikan oleh Microsoft di luar contoh kode implementasi sangatlah terbatas. Kode di sini didasarkan pada contoh konektor GitHub milik Microsoft, yang dapat ditemukan di sini: *https://github.com/microsoft/DataConnectors/tree/master/samples/Github*.

Definisi fungsi *StartLogin* dan *FinishLogin* akan bervariasi tergantung pada spesifikasi masing-masing API. Fungsi-fungsi ini menerapkan informasi dan struktur yang dibutuhkan oleh API Discord. Mari kita mulai dengan melihat lebih dekat fungsi *StartLogin*.

Fungsi *StartLogin* memulai proses autentikasi OAuth dengan membangun URI otorisasi yang diharapkan oleh API Discord. Nilai ini disimpan oleh ekspresi *AuthorizationUrl*. Pada dasarnya, ekspresi ini hanya membangun URI dengan parameter string kueri tertentu. URI aktual yang dibangun adalah sebagai berikut: *https://discord.com/api/oauth2/authorize?client_id=1128020929558098050&redirect_uri=http%3A%2F%2Flocalhost%2Fdiscord%2Fredirect&response_type=code&scope=identify%20guilds%20guilds.members.read%20email%20connections*.

Perhatikan bahwa URI otorisasi dasar berasal dari variabel *authorize_uri* yang ditetapkan di bagian Menambahkan pengaturan konfigurasi. String kueri berisi *client_id* (milik Anda akan berbeda) beserta *redirect_uri*, *response_type* kode, dan parameter scope. Jenis respons kode memberi tahu API Discord jenis respons otorisasi yang ingin kita terima, dan scope menentukan izin yang ingin kita berikan kepada konektor kustom pihak ketiga. Fungsi *StartLogin* mengembalikan record yang berisi kolom berikut:

- *LoginUri*: Ini adalah ekspresi *AuthorizeUrl*, yang pada akhirnya adalah URI yang disediakan sebelumnya: *https://discord.com/api/oauth2/authorize?client_id=1128020929558098050&redirect_uri=http%3A%2F%2Flocalhost%2Fdiscord%2Fredirect&response_type=code&scope=identify%20guilds%20guilds.members.read%20email%20 connections*
- *CallbackUri*: Variabel/pengaturan konfigurasi *redirect_uri* kita
- *WindowHeight*: Variabel/pengaturan konfigurasi *WindowHeight* kita
- *WindowWidth*: Variabel/pengaturan konfigurasi *WindowWidth* kita

- *Context: null*

Untuk mendapatkan pemahaman yang lebih baik tentang cara kerjanya, cukup tempel URI yang disediakan sebelumnya (*AuthorizeUrl*) ke dalam peramban web. Prompt otorisasi Discord muncul yang menunjukkan bahwa aplikasi pihak ketiga mencari izin berikut:

- Mengakses nama pengguna, avatar, dan banner Anda.
- Mengakses alamat email Anda.
- Mengakses koneksi pihak ketiga Anda.
- Mengetahui server tempat Anda berada.
- Membaca info anggota Anda (nama panggilan, avatar, peran, dll.) untuk server tempat Anda berada.

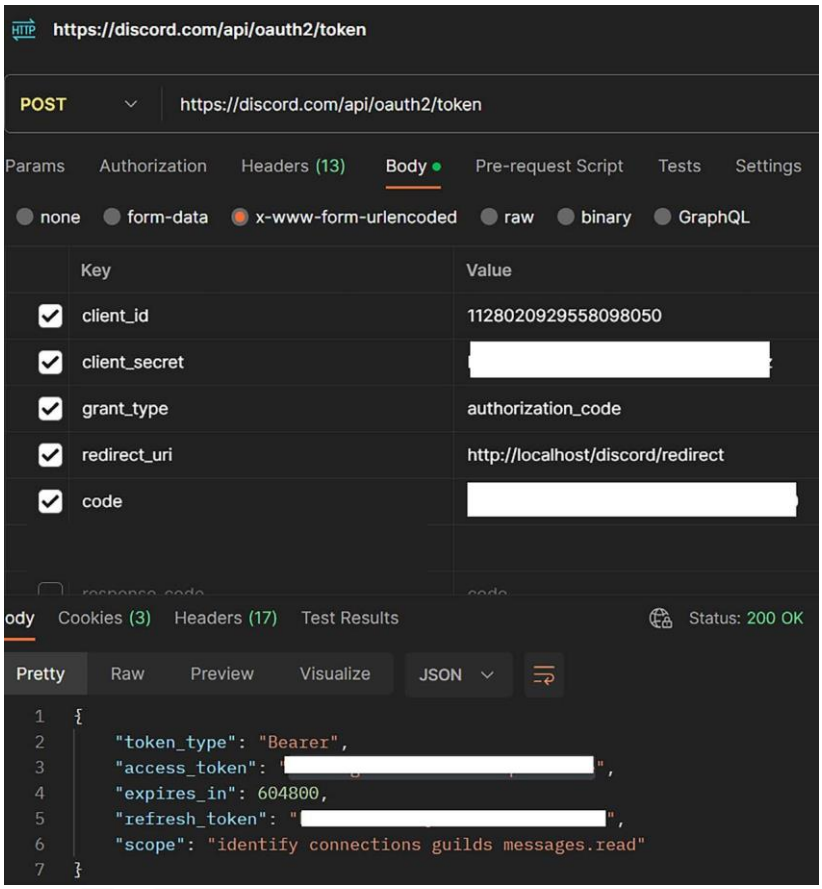
List izin ini berasal dari parameter string kueri *scope* URI. Klik tombol **Authorize**. Anda sekarang diarahkan ke *redirect_uri* yang ditentukan, *http://localhost/discord/redirect*. Halaman ini sebenarnya tidak ada, jadi server web lokal Anda menampilkan halaman 404 Tidak Ditemukan. Namun, perhatikan lebih dekat URI di browser; URI akan seperti ini: *http://localhost/discord/redirect?code=WV9s0kMY3Rg6AsPvCzj6CgbgnUvwz0*.

Perhatikan parameter kode dalam string kueri. Karena kita meminta kode *response_type*, ini adalah kode yang dikembalikan oleh server Discord. Kita kemudian menggunakan kode ini dalam fungsi *FinishLogin* untuk membuat token akses dan menyelesaikan proses otorisasi kita. Ini dilakukan dengan memasukkan kode (*Parts[code]*) ke dalam fungsi *TokenMethod*.

Sebenarnya, metode *FinishLogin* mengambil respons dari redirect_uri kita dan memasukkan nilai parameter string kueri kode ke fungsi *TokenMethod*. Fungsi *TokenMethod* menyelesaikan proses pengambilan token akses *OAuth* kita.

Untuk lebih memahami proses ini, kita dapat menggunakan Postman (postman.com) untuk meniru perilaku tersebut, seperti yang ditunjukkan pada

tangkapan layar berikut. Postman adalah platform API yang mencakup alat yang umumnya digunakan untuk meniru interaksi API yang kompleks seperti autentikasi. Di sini, dalam permintaan Postman ini, kita telah memilih operasi *POST* (sebagai lawan dari *GET*):



Gambar 4. 7 Mengambil token akses OAuth melalui Postman

Konektor kustom kita berperilaku sama karena kita telah menyertakan elemen **Headers** dalam panggilan fungsi *Web.Contents* kita dalam fungsi *TokenMethod*. Dalam Gambar 4.7, kita telah menetapkan posting ke URI *https://discord.com/api/oauth/token*, yang sama dengan *token_uri* yang kita gunakan dalam konektor kustom kita.

Dalam **Body** permintaan Postman kita, kita telah menetapkan body sebagai *x-www-form-urlencoded*, sama seperti yang diterapkan dalam **Header**

kita untuk Content-type dalam fungsi *TokenMethod*. Dalam body tersebut, kita mengodekan informasi berikut:

- *client_id*: Ini adalah ID klien kita yang dibuat di bagian konfigurasi Discord *OAutb*.
- *client_secret*: Rahasia klien yang dibuat di bagian konfigurasi Discord *OAutb*.
- *grant_type*: Ini ditetapkan ke authorization_code sebagaimana diwajibkan oleh Discord *OAuth* API.
- *redirect_uri*: URI pengalihan kita yang ditetapkan di bagian konfigurasi Discord *OAutb, http:// localhost/discord/redirect*.
- *code*: Ini adalah kode yang dikembalikan dari panggilan otorisasi Discord API awal. Di konektor kustom kita, ini adalah kode yang dibuat dan dimasukkan ke dalam string kueri dari *redirect_uri* kita. Jika Anda menempelkan contoh URI (*AuthorizationUrl*), kodenya adalah: *http://localhost/discord/re direct?code=WV9s0kMY3Rg6AsPvCzj6CgbgnUvwz0*.

Setiap informasi ini juga dikodekan ke dalam permintaan yang dibuat dalam fungsi *TokenMethod*.

Pada Gambar 16.7, Anda juga dapat melihat respons yang diberikan oleh Discord. URI token Discord merespons dengan informasi berikut dalam format JSON:

- *token_type*: Jenis token yang dikembalikan, dalam hal ini, *Bearer*
- *access_token*: Token akses *Bearer*
- *expires_in*: Saat token kedaluwarsa
- *refresh_token*: Token untuk menyegarkan token kita
- *scope*: Cakupan token akses

Perhatikan dalam fungsi *TokenMethod* kita bahwa kita mengharapkan *Json.Document* dikembalikan sebagai respons (*Response*).

Meskipun spesifikasi parameter yang tepat yang diteruskan dalam permintaan ini bervariasi antara implementasi API OAuth, secara keseluruhan,

kode M yang diberikan seharusnya berfungsi dengan modifikasi minimal untuk sebagian besar skenario autentikasi OAuth. Beruntung bagi kita, jenis autentikasi OAuth Power Query menangani banyak pekerjaan berat dari sini dan seterusnya.

Misalnya, jika kita melanjutkan contoh kita di Postman, untuk setiap permintaan API berikutnya (misalnya, mendapatkan list server untuk pengguna), kita perlu menyertakan token akses dalam parameter yang disebut Otorisasi dengan nilai "*Bearer <access_token>*". Namun, tipe autentikasi OAuth Power Query menangani penyertaan header yang diperlukan secara otomatis untuk kita selama permintaan API berikutnya.

Sekarang setelah kita menerapkan fungsi tipe autentikasi OAuth yang diperlukan, kita dapat mengubah definisi record sumber data kita untuk mengubah tipe autentikasi dari **Anonymous** menjadi **OAuth**.

Mengubah Definisi *Record* Sumber Data

Setelah fungsi OAuth yang diperlukan diterapkan, kini kita dapat mengubah record definisi Sumber Data untuk mengubah autentikasi **Anonymous** menjadi autentikasi **OAuth**. Untuk melakukannya, ikuti langkah-langkah berikut:

1) Temukan record definisi Sumber Data, *TDGTPQM_Discord*.
2) Ganti definisi record dengan kode berikut. Perhatikan bahwa sekarang kita menentukan jenis autentikasi OAuth yang diberikan sebagai record yang menentukan fungsi *StartLogin* dan *FinishLogin* untuk bidang jenis autentikasi *OAuth* yang sesuai:

```
// Data Source Kind description
TDGTPQM_Discord = [
    TestConnection = (dataSourcePath) => {"TDGTPQM_Discord.Contents"},
    Authentication = [
        OAuth = [
            StartLogin = StartLogin,
            FinishLogin =
            FinishLogin
        ]
    ]
];
```

Setelah jenis autentikasi OAuth kita didefinisikan, sekarang saatnya menguji autentikasi OAuth untuk konektor kustom kita. Untuk melakukannya, pertama-tama kita harus menambahkan kredensial.

Menambahkan Kredensial

Menambahkan kredensial dalam Visual Studio Code meniru perilaku UI Power Query (seperti editor Power Query) saat Anda masuk atau mengautentikasi untuk terhubung ke sumber data. Untuk menambahkan kredensial, ikuti langkah-langkah berikut:

1) Jika Anda belum melakukannya, pilih File | Save All dari menu Visual Studio Code.
2) Luaskan **POWER QUERY SDK** di panel **Explorer**.
3) Klik **Run TestConnection function**. Perhatikan bahwa Anda menerima galat:

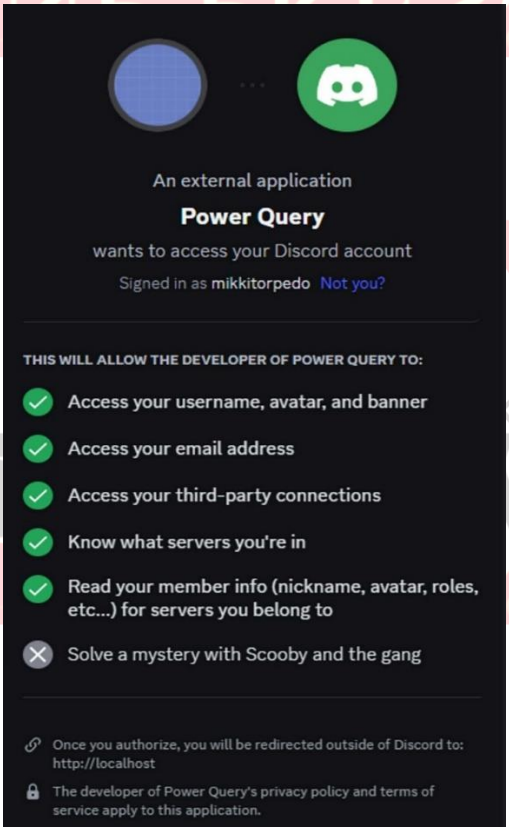
   Failed to run the query due to Error: Failed to run the query due to Error: Unable to find credential for 'TDGTPQM_Discord/TDGTPQM_Discord'
4) Klik *Evaluate current file*. Perhatikan bahwa Anda kembali menerima pesan galat:

   Credentials are required to connect to the TDGTPQM_Discord source. (Source at TDGTPQM_Discord.)
5) Klik **Set credential**.

6) Saat diminta **Choose the data source kind**, pilih *TDGTPQM_Discord* dari list.
7) Saat diminta **Choose a query/test file**, pilih *TDGTPQM_Discord.query.pq* dari list.
8) Saat diminta **Choose an authentication method**, pilih **OAuth** dari list.

> Perhatikan bahwa jika Anda hanya melihat **Anonymous** yang tercantum, ini adalah salah satu dari banyak "keanehan" Visual Studio Code. Coba simpan pekerjaan Anda dan jalankan Langkah 3 dan 4 lagi. Jika tidak berhasil, **File | Close Folder** dari menu, simpan pekerjaan Anda jika diminta, lalu buka kembali folder menggunakan **File | Open Folder….**

9) Setelah folder dibuka kembali, perluas bagian **POWER QUERY SDK** pada panel **Explorer** dan lanjutkan dari Langkah 5.
10) Kotak dialog seperti berikut akan ditampilkan:



Gambar 4. 8 Dialog otorisasi Discord

11) Klik tombol **Authorize**.

Anda akan menerima pesan **New OAuth credential has been generated successfully**.

Menguji Koneksi

Meskipun konektor kustom kita masih belum melakukan banyak hal selain merespons dengan teks "*Hello World*" yang ada di mana-mana, pengujian koneksi pada tahap ini memvalidasi bahwa kredensial OAuth kita ada. Untuk menguji koneksi, ikuti langkah-langkah berikut:

1) Luaskan bagian **POWER QUERY SDK** di panel **Explorer**.
2) Klik **Run TestConnection function**. Kali ini, tidak ada kesalahan yang ditampilkan.
3) Pastikan bahwa file *TDGTPQM.pq* dipilih di panel **EXPLORER** lalu klik **Evaluate current file**. Kali ini, tab hasil *PQTest* muncul dengan Nilai yang dimulai dengan *Hello from TDGTPQM_Discord*:.

Setelah koneksi kustom kita berhasil diuji, mengonfirmasi bahwa kredensial OAuth kita dibuat dan ada dalam Visual Studio Code, kita sekarang dapat melanjutkan untuk membuat konektor kustom kita mengambil data dari API Discord.

3. Mengonfigurasi Navigasi dan Konten

Sekarang saatnya untuk mulai membuat konektor kustom kita benar-benar mengambil dan menyajikan data dari Discord. Jika Anda mempertimbangkan pengalaman pengguna di editor Power Query saat menghubungkan ke sumber data, Anda mengikuti pengalaman ini:

1) Pilih **Get Data** lalu pilih konektor sumber data.
2) Masuk atau autentikasi sesuai dengan opsi autentikasi yang disajikan.
3) Sering kali, dialog **Navigator** ditampilkan tempat Anda dapat menavigasi folder dan tabel untuk mengambil data pratinjau.

4) Anda memilih tabel data dalam dialog **Navigator** dan menekan tombol **OK** untuk mengambil data.

Kita telah menerapkan jenis autentikasi kita, OAuth. Sehubungan dengan penerapan langkah-langkah lainnya dalam kode M, pada dasarnya kita perlu menerapkan proses ini dalam urutan terbalik dengan melakukan hal berikut:

1) **Tambahkan fungsi pengambilan data panggilan API**: Fungsi-fungsi ini akan menerapkan pengambilan data aktual dari API Discord. Beginilah cara data akan diambil dan ditampilkan dalam panel **Preview** dialog **Navigator** serta cara data akan diambil dan dimuat ke editor Power Query dan akhirnya ke model data.
2) **Tambahkan fungsi navigasi**: Fungsi-fungsi ini akan menyajikan dialog **Navigator** (panel navigasi sebelah kiri) dan menautkan navigasi ini ke operasi pengambilan data yang sesuai (fungsi panggilan API) yang diterapkan pada Langkah 1.
3) **Ubah fungsi TDGTPQM.Contents**: Selesaikan penerapan dialog **Navigator** dengan mengubah fungsi *TDGTPQM.Contents* untuk menginisialisasi navigasi.

Mari kita lihat penerapan langkah pertama dalam proses ini: membuat kode M untuk melakukan panggilan API.

Menambahkan Fungsi Panggilan API Untuk Pengambilan Data

Untuk konektor Discord kustom kita, kita ingin menerapkan tiga operasi pengambilan data berbeda yang terkait dengan hal berikut:

- Informasi yang terkait dengan identitas pengguna
- Informasi yang terkait dengan server atau guild tempat pengguna berada
- Informasi yang terkait dengan identitas pengguna dalam setiap server/guild

Informasi yang diambil di sini cukup mendasar dan mungkin tidak memiliki banyak nilai analitis. Namun, konektor dasar ini dapat diperluas

untuk mengambil jumlah pesan yang diposting ke setiap server/guild, dan seterusnya.

Untuk menerapkan operasi pengambilan data ini, lakukan hal berikut. Tepat di atas definisi jenis sumber data, *TDGTPQM_Discord*, masukkan tiga fungsi berikut:

```
// ** Identity
TDGTPQM_Discord.GetIdentity = () as table =>
    let

        apiCall = Json.Document(
            Web.Contents(

                api_uri, [

                    RelativePath = "users/@me"

                ]

            )

        ),

        output = Table.FromRecords({apiCall})
    in

        output;


// ** Servers (Guilds)
TDGTPQM_Discord.GetGuilds = () as table =>
    let

        apiCall = Json.Document(
            Web.Contents(

                api_uri,
                [

                    RelativePath = "users/@me/guilds"

                ]

            )

        ),

        output = Table.FromList(apiCall, Splitter.SplitByNothing(), null,
null, ExtraValues.Error)
```

```
    in
        output;

// ** Member User
TDGTPQM_Discord.GetGuildMember = (guildid as text) as table =>
    let
        apiCall =
            Json.Document(
            Web.Contents(
                api_uri,
                [
                    RelativePath = "users/@me/guilds/" & guildid &
                    "/member"
                ]
            )
        ),
        output = Table.FromRecords({apiCall})
    in
        output;
```

Fungsi-fungsi ini sangat mirip. Mari kita lihat fungsi *TDGTPQM_Discord.GetIdentity* terlebih dahulu. Ini adalah fungsi yang mengambil informasi yang terkait dengan identitas pengguna. Seperti yang dapat kita lihat dalam definisi fungsi, fungsi ini mengembalikan tipe tabel (*as table*) dan tidak memiliki parameter. Dengan membaca ekspresi *apiCall* secara terbalik, kita membuat URI yang terdiri dari *api_uri* dasar seperti yang dibuat dalam Menambahkan pengaturan konfigurasi dan jalur relatif *users/@me*. URI ini kemudian menjadi *https://discord.com/api/users/@me*.

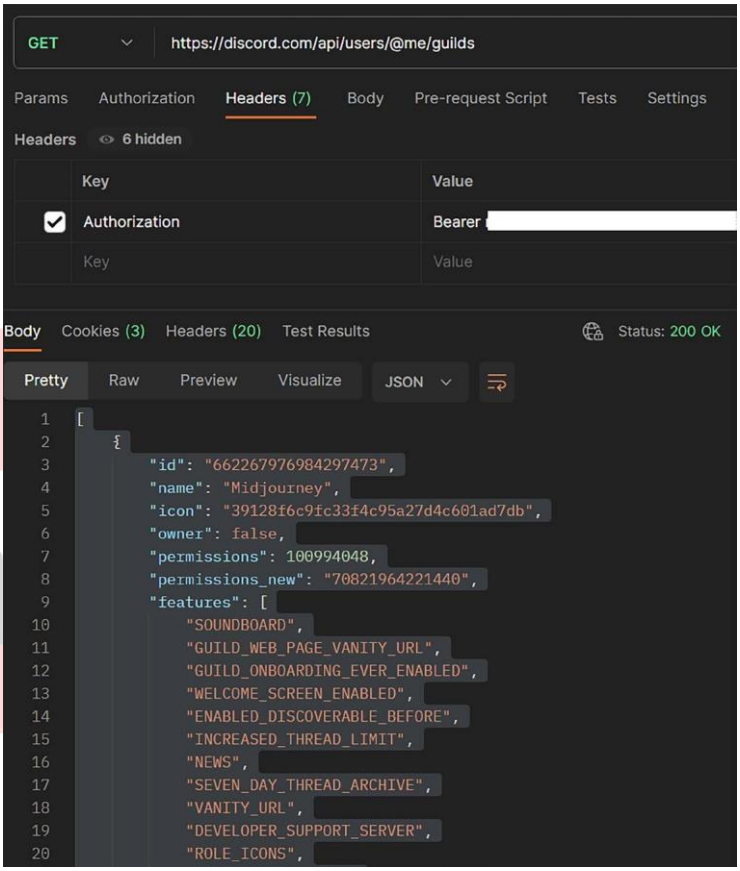
Kita menggunakan *Web.Contents* untuk mengambil respons dari panggilan URI ini dan menentukan bahwa konten ini adalah *Json.Document*. Ekspresi keluaran kemudian mengambil informasi dalam dokumen JSON ini menggunakan fungsi *Table.FromRecords*.

Fungsi berikutnya, *TDGTPQM.GetGuilds*, mengambil informasi yang terkait dengan server, atau guild dalam istilah API Discord, tempat pengguna berada. Fungsi ini juga tidak mengambil parameter dan mengembalikan tabel. Panggilan API dilakukan ke URI berikut: *https://discord.com/api/users/@me/guilds*.

Ekspresi outputnya sedikit berbeda. Alih-alih memproses record, kita memproses list menggunakan fungsi *Table.FromList*.

Anda mungkin bertanya-tanya bagaimana kita dapat mengetahui cara memproses respons dari panggilan API, dan apakah akan menggunakan fungsi *Table.FromRecords* atau fungsi *Table.FromList*. Jawabannya adalah dengan membaca dokumentasi yang disediakan oleh API yang kita panggil dengan saksama. Namun, trik yang bagus adalah menggunakan Postman untuk membuat dan melihat respons. Anda dapat melakukannya dengan mengikuti langkah-langkah berikut:

1) Tempel *AuthorizationUrl* ke browser web dan tekan tombol Enter. Sebagai referensi, URI ini adalah *https://discord.com/api/oauth2/authorize?client_id=112802092955809805 0&redirect_uri=http%3A%2F%2Flocalhost%2Fdiscord%2Fredirect&response_type=code&scope=identify%20guilds%20guilds.members.read%20email%20 connections*
2) Salin nilai parameter string kueri kode yang dikembalikan sebagai bagian dari *redirect_uri*.
3) Seperti yang ditunjukkan pada Gambar 16.7, gunakan Postman untuk meminta token akses menggunakan kode sebagai salah satu parameter yang dikodekan.
4) Gunakan token akses ini dalam permintaan Postman lain ke panggilan API:



Gambar 4. 9 Panggilan API Discord menggunakan Postman

Seperti yang dapat Anda lihat pada Gambar 4.9, kita telah memulai permintaan GET ke titik akhir API Discord di *https://discord.com/api/users/@me/guilds*. Kita telah menyertakan parameter *Authorization* di **Headers** kita dengan nilai "*Bearer <access token>*". Di panel bawah, kita dapat melihat respons yang dikembalikan dari panggilan API.

5) Salin respons yang dikembalikan dan tempel data ini ke dalam file.
6) Gunakan UI editor Power Query untuk mengambil dan memperluas/memproses informasi dalam file.
7) Lihat kode M yang dihasilkan di **Advanced Editor**.
8) Salin kode M yang relevan dari **Advanced Editor** dan tempel kode ini sebagai ekspresi keluaran untuk fungsi kita.

Semoga ini membantu Anda memahami cara membuat kode M untuk fungsi pengambilan data Anda. Mari selesaikan bagian ini dengan melihat sekilas fungsi pengambilan data akhir, *TDGTPQM_Discord.GetGuildMember*.

Fungsi *TDGTPQM_Discord.GetGuildMember* mengambil informasi yang terkait dengan identitas pengguna dalam setiap *server/guild*. Fungsi ini juga mengembalikan tabel sebagai output tetapi juga mengambil satu parameter, *guildid*. Parameter ini digunakan dalam panggilan API untuk menentukan server/guild tempat mengambil informasi pengguna. Panggilan API dalam format berikut: *https://discord.com/api/users/@me/guilds/<guildid>/member*.

Ekspresi output menggunakan *Table.FromRecords* untuk mengubah respons menjadi tabel.

Ini melengkapi penjelasan kita tentang fungsi pengambilan data panggilan API. Sekarang kita dapat beralih ke fungsi yang akan menyediakan pengalaman navigasi kita dalam dialog **Navigatior** di UI editor Power Query.

Menambahkan Fungsi Navigasi

Dengan fungsi pengambilan data panggilan API yang dikodekan, kini kita dapat mengimplementasikan fungsi yang menyajikan pengalaman pengguna dalam dialog **Navigatior** dari UI editor Power Query.

Untuk mengimplementasikan fungsi navigasi, pertama-tama kita perlu mengkodekan fungsi pembantu: *Table.ToNavigationTable*. Kode untuk fungsi ini diambil langsung dari kode contoh Microsoft yang ditemukan di sini: *https://github.com/microsoft/DataConnectors/tree/master/samples/NavigationTable*. Tambahkan fungsi berikut di akhir kode konektor kustom:

```
// Navigation Tables Functions
Table.ToNavigationTable = (
    table as table,
    keyColumns as list,
    nameColumn as text,
    dataColumn as text,
    itemKindColumn as text,
    itemNameColumn as text,
    isLeafColumn as text
) as table =>
    let
        tableType = Value.Type(table),
        newTableType = Type.AddTableKey(tableType, keyColumns, true) meta [
            NavigationTable.NameColumn = nameColumn,
            NavigationTable.DataColumn = dataColumn,
            NavigationTable.ItemKindColumn = itemKindColumn,
            Preview.DelayColumn = itemNameColumn,
            NavigationTable.IsLeafColumn = isLeafColumn
        ],
        navigationTable = Value.ReplaceType(table, newTableType)
    in
        navigationTable;
```

Singkatnya, fungsi ini mengubah tabel dengan menambahkan metadata yang diharapkan oleh UI dialog **Navigator** dalam pengalaman Power Query. Metadata yang ditambahkan mencakup list berikut:

- *NavigationTable.NameColumn*
- *NavigationTable.DataColumn*
- *NavigationTable.ItemKindColumn*
- *Preview.DelayColumn*
- *NavigationTable.IsLeafColumn*

Saat ini, Anda harus menambahkan fungsi ini secara manual ke kode Anda. Namun, Microsoft telah mengindikasikan bahwa fungsi ini pada akhirnya dapat menjadi bagian dari pustaka standar M. Fungsi ini mengambil argumen berikut:

- *table*: Tabel yang ingin Anda ubah menjadi tabel navigasi.
- *keyColumns*: Kolom kunci utama (nilai unik) untuk tabel Anda.
- *nameColumn*: Kolom yang digunakan untuk nama tampilan dalam dialog **Navigator**.

- *dataColumn*: Kolom yang berisi tabel atau fungsi yang mengembalikan tabel. Ini adalah data aktual yang akan diambil jika dipilih dalam dialog **Navigator**.
- *itemKindColumn*: Kolom yang menentukan jenis ikon yang akan ditampilkan dalam dialog **Navigator**. Nilai dalam kolom ini dapat berupa salah satu dari berikut ini:
  - *Cube*
  - *CubeDatabase*
  - *CubeView*
  - *CubeViewFolder*
  - *Database*
  - *DatabaseServer*
  - *DefinedName*
  - *Dimension*
  - *Feed*
  - *Folder*
  - *Function*
  - *Record*
  - *Sheet*
  - *Subcube*
  - *Table*
  - *View*
- *itemNameColumn*: Kolom yang menentukan cara pratinjau data ditangani dalam dialog **Navigator**. Paling sering, nilai dalam *itemKindColumn* sama dengan nilai dalam *itemNameColumn*.
- *isLeafColumn*: Kolom yang menentukan apakah item dapat diperluas atau tidak. Kolom ini harus berisi nilai logika atau Boolean.

Informasi tambahan tentang fungsi ini dapat ditemukan di sini: https://learn.microsoft.com/en-us/power-query/handling-navigation-tables.

Selanjutnya, tepat di atas fungsi *TDGTPQM_Discord.GetIdentity*, tambahkan tiga fungsi berikut:

```
TDGTPQM_Discord.UsersNavigation = () as table =>
    // Navigation
    //** Users Navigation
    let
        objects = #table(
            {"Name", "Key", "Data", "ItemKind", "ItemName", "IsLeaf"},
            {
                {"Me", "me", TDGTPQM_Discord.GetIdentity(), "Record", "Record",
true}
            }
        ),
        Navigation = Table.ToNavigationTable(objects, {"Key"}, "Name", "Data",
"ItemKind", "ItemName", "IsLeaf")
    in
        Navigation;

TDGTPQM_Discord.GuildsNavigation = () as table =>
    // Navigation
    //** Guilds (Server) Navigation
    let
        objects = #table(
            {"Name", "Key", "Data", "ItemKind", "ItemName", "IsLeaf"},
            {
                {"Servers", "guilds", TDGTPQM_Discord.GetGuilds(), "Table",
"Table", true}
            }
        ),

        Navigation = Table.ToNavigationTable(objects, {"Key"}, "Name", "Data",
"ItemKind", "ItemName", "IsLeaf")
    in
        Navigation;

TDGTPQM_Discord.MemberNavigation = () as table =>
    // Navigation
    //** Membership Navigation
```

```
    let
        servers = Table.ExpandRecordColumn(TDGTPQM_Discord.GetGuilds(),
"Column1", {"name", "id"}, {"Name", "Key"}),
        addDataColumn = Table.AddColumn(servers, "Data", each TDGTPQM_Discord.
GetGuildMember([Key])),
        addItemKindColumn = Table.AddColumn(addDataColumn, "ItemKind", each
"Record"),
        addItemNameColumn = Table.AddColumn(addItemKindColumn, "ItemName", each
"Function"),
        addIsLeafColumn = Table.AddColumn(addItemNameColumn, "IsLeaf", each
true),
```

```
        Navigation = Table.ToNavigationTable(addIsLeafColumn, {"Key"}, "Name",
"Data", "ItemKind", "ItemName", "IsLeaf")

    in
        Navigation;
```

Dua fungsi pertama, *TDGTPQM_Discord.UsersNavigation* dan *TDGTPQM_Discord.GuildsNavigation*, hampir identik. Dalam kedua kasus, ada ekspresi *objects* yang mendefinisikan tabel dengan kolom berikut:

- *Name*
- *Key*
- *Data*
- *ItemKind*
- *ItemName*
- *IsLeaf*

Untuk masing-masing, satu simpul daun navigasi didefinisikan. Fungsi *TDGTPQM_Discord.UsersNavigation* mendefinisikan nilai berikut untuk baris tunggal dalam tabel ini:

- *Me*
- *me*
- *TDGTPQM_Discord.GetIdentity()*
- *Record*
- *Record*
- *true*

Menurut definisi ini, kata Saya akan muncul dalam dialog **Navigator** untuk simpul ini. Nama atau kunci unik untuk simpul tersebut adalah saya. Data yang diambil dengan memilih node ini adalah data yang diambil oleh fungsi pengambilan data panggilan API kita, *TDGTPQM_Discord.GetIdentity*. Ikon dan tampilan pratinjau ditetapkan sebagai Record dan, akhirnya, nilai akhir true menetapkan bahwa ini adalah node daun (titik akhir) dan bukan folder yang dapat diperluas.

Fungsi *TDGTPQM_Discord.GuildsNavigation* hampir identik tetapi menetapkan nilai kolom Data dari fungsi pengambilan data panggilan API kita, *TDGTPQM_Discord.GetGuilds*. Selain itu, kita menetapkan bahwa ikon dan pratinjau data harus ditangani sebagai *Table*, bukan *Record*.

Dalam kedua kasus, tabel ini kemudian dimasukkan ke fungsi *Table.ToNavigationTable* dengan parameter tambahan yang merujuk ke kolom yang sesuai dalam tabel objek kita.

Fungsi *TDGTPQM_Discord.MemberNavigation* menyajikan navigasi dinamis yang lebih kompleks. Fungsi ini menggunakan fungsi pengambilan data panggilan API *TDGTPQM_Discord.GetGuilds* kita untuk mengambil list server tempat pengguna berada. Record yang dikembalikan diperluas untuk kolom nama dan id yang dikembalikan oleh panggilan API dan masing-masing dinamai ulang sebagai Nama dan Kunci. Kemudian, kita menambahkan kolom tambahan yang diperlukan oleh fungsi *Table.ToNavigationTable*, termasuk kolom Data tempat kita memanggil fungsi *TDGTPQM_Discord.GetGuildMember*, dengan memasukkan Key (id server/guild) sebagai satu-satunya parameternya. Kita juga menentukan kolom *ItemKind* tempat kita menentukan nilai *Record* dan kolom *IsLeaf* yang menyimpan nilai logika true.

Untuk kolom *ItemName*, alih-alih menentukan nilai yang sama yang digunakan untuk kolom *ItemKind (Record)*, kita menentukan nilai *Function*. Ini diperlukan karena ini adalah tabel navigasi yang dibuat secara dinamis.

Ini melengkapi penjelasan kita tentang fungsi tabel navigasi. Kita sekarang siap untuk menyelesaikan konektor kustom kita dengan memodifikasi fungsi *TDGTPQM_Discord.Contents*, jadi mari kita lakukan itu selanjutnya.

Mengubah Fungsi Konten

Saatnya untuk menyelesaikan kode konektor kustom kita. Modifikasi terakhir yang diperlukan adalah memodifikasi fungsi

TDGTPQM_Discord.Contents untuk menampilkan level teratas navigasi kita yang akan muncul di dialog **Navigator** pada UI Power Query. Alih-alih menampilkan satu string teks, kita dapat menampilkan dialog **Navigator** yang memungkinkan pengguna untuk memilih data yang ingin diambil dari Discord. Untuk melakukannya, ikuti langkah-langkah berikut:

1) Temukan fungsi *TDGTPQM_Discord.Contents* di dalam kode konektor kustom Anda.
2) Ganti fungsi *TDGTPQM_Discord.Contents* dengan yang berikut:

```
[DataSource.Kind="TDGTPQM_Discord", Publish="TDGTPQM_Discord.Publish"]
shared TDGTPQM_Discord.Contents = () =>

    let
        objects = #table(
            {"Name", "Key", "Data", "ItemKind", "ItemName", "IsLeaf"},
            {
                {"Users", "usersFolder", TDGTPQM_Discord.
UsersNavigation(), "Folder", "usersFolder", false},
                {"Servers", "guildsFolder", TDGTPQM_Discord.
GuildsNavigation(), "Folder", "guildsFolder", false},
                {"Membership", "membershipFolder", TDGTPQM_Discord.
MemberNavigation(), "Folder", "membershipFolder", false}
            }
        ),
        Navigation = Table.ToNavigationTable(objects, {"Key"}, "Name",
"Data", "ItemKind", "ItemName", "IsLeaf")
    in
        Navigation;
```

Seperti yang dapat Anda lihat, kode untuk ini sangat mirip dengan dua fungsi navigasi pertama kita, *TDGTPQM_Discord.UserNavigation* dan *TDGTPQM_Discord.GuildsNavigation*. Kita sekali lagi mendefinisikan ekspresi objek yang mengembalikan tabel dengan kolom yang diharapkan oleh fungsi *Table.ToNavigationTable*.

Setiap baris dalam tabel ini berisi nama (*Name*) dan kunci unik (*Key*) serta spesifikasi *ItemKind* dari *Folder*, *ItemName* dari *Folder*, dan *IsLeaf* dari *false*. Untuk kolom *Data*, kita tentukan nilai fungsi tabel navigasi terkait sebagaimana mestinya, yaitu:

- *TDGTPQM_Discord.UserNavigation*
- *TDGTPQM_Discord.GuildsNavigation*
- *TDGTPQM_Discord.MemberNavigation*

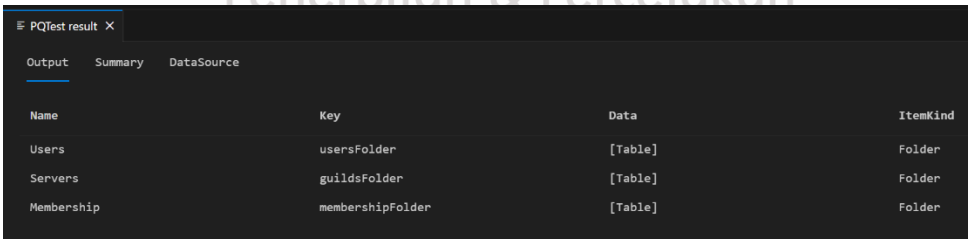
Mirip dengan fungsi navigasi, kita masukkan tabel *objects* ini ke fungsi *Table.ToNavigationTable* sebagai output dari fungsi *TDGTPQM_Discord.Contents*.

Sekarang setelah kita menyelesaikan pengodean konektor kustom kita, kita akhirnya siap untuk memasang dan mulai menggunakan konektor ini dalam Power BI Desktop.

F. Memasang dan Menggunakan Konektor Khusus

Untuk melakukannya, ikuti langkah-langkah berikut:

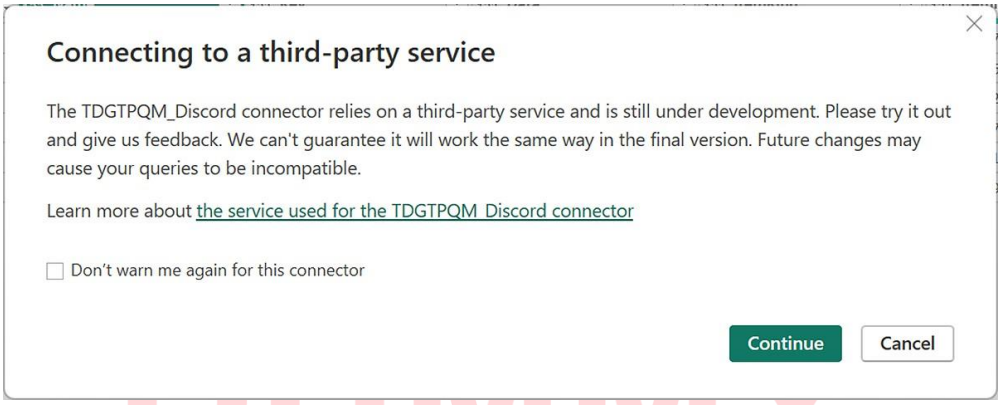
1) Dengan proyek Anda terbuka di Visual Studio Code, simpan pekerjaan Anda dengan mengeklik **File | Save All** di menu.
2) Luaskan bagian **POWER QUERY SDK** di panel **EXPLORER**.
3) Klik **Run TestConnection function** dan pastikan tidak ada kesalahan yang ditampilkan.
4) Pastikan file *TDGTPQM.pq* dipilih di panel Explorer, lalu klik **Evaluate current file**. Kali ini, tab **PQTest result** akan menampilkan informasi berikut:



Gambar 4. 10 Hasil evaluasi *TDGTPQM.pq*

Seperti yang dapat Anda lihat, ini adalah level teratas dari tampilan navigasi kita dalam dialog **Navigator** dari UI Power Query dan secara langsung sesuai dengan spesifikasi ekspresi *objects* kita dari fungsi *TDGTPQM_Discord.Contents*.

5) Klik tautan folder *bin\AnyCPU\Debug* di panel **EXPLORER** untuk memperluas folder ini.
6) Klik kanan file *TDGTPQM_Discord.mez* dan pilih **Reveal in File Explorer**.
7) Salin file *TDGTPQM_Discord.mez*.
8) Saat berada di File Explorer, navigasikan ke direktori **Documents** Anda.
9) Buat folder bernama *Microsoft Power BI Desktop*.
10) Navigasi ke folder *Microsoft Power BI Desktop*.
11) Buat folder bernama *Custom Connectors*.
12) Navigasi ke folder *Custom Connectors*.
13) Tempel file *TDGTPQM_Discord.mez*.
14) Buka Power BI Desktop.
15) Di pita, navigasikan ke **File | Options and settings | Options**.
16) Di dialog **Options**, pilih **Security** di navigasi kiri.
17) Di bawah **Data Extensions**, pilih tombol radio untuk **(Not Recommended) Allow any extension to load without validation or warning"**.
18) Klik tombol **OK**.
19) Mulai ulang Power BI Desktop.
20) Dari tab **Home** di pita, pilih **Transform Data** untuk membuka editor Power Query.
21) Dari tab **Home** di pita editor Power Query, pilih **New Source | More**.
22) Di bilah pencarian, masukkan *tdgtpqm*.
23) Pilih konektor **TDGTPQM_Discord (Beta)** lalu tekan tombol **Connect**. Dialog berikut ditampilkan:



Gambar 4. 11 Pemberitahuan koneksi layanan pihak ketiga

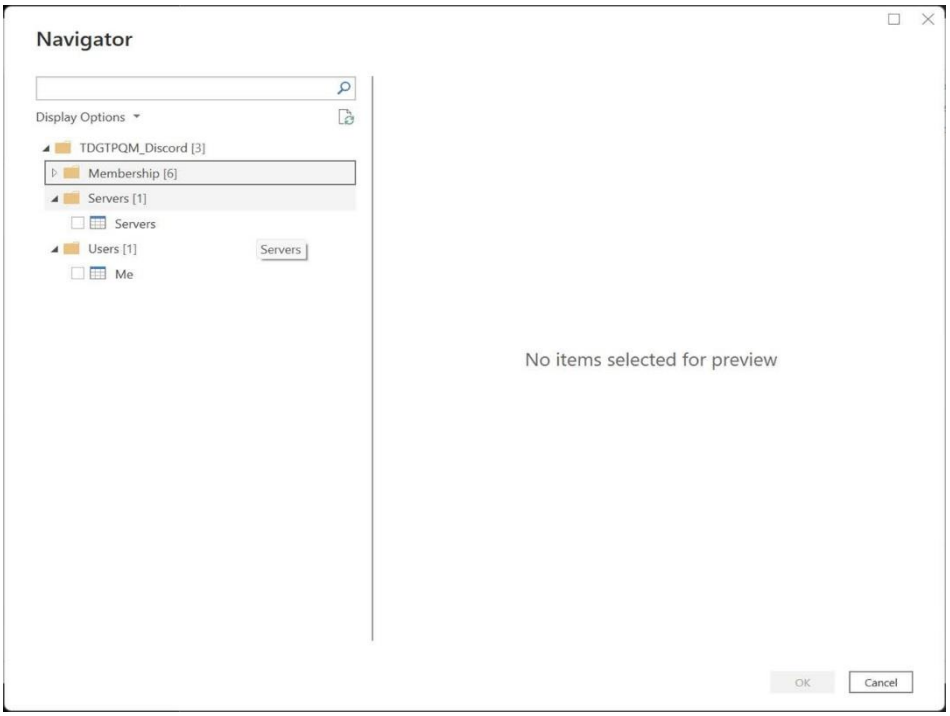
24) Tekan tombol **Continue**.

25) Klik tombol **Sign-in**.

26) Otorisasi aplikasi dengan mengklik tombol **Authorize** di kotak dialog.

27) Klik tombol **Connect**.

28) Dialog **Navigator** ditampilkan:



Gambar 4. 12 *TDGTPQM._Dialog* Navigator konektor kustom Discord di editor Power Query

Selamat! Anda telah berhasil membuat konektor kustom untuk Power Query yang terhubung ke dan mengambil data dari Discord API. Anda dapat menggunakan konektor ini dengan cara yang sama seperti konektor default yang disertakan dengan Power BI Desktop.

Perlu diingat bahwa ini hanyalah salah satu contoh pembuatan konektor kustom untuk mengambil data dari API. Ada ratusan, bahkan ribuan, API kustom tempat Anda dapat mengambil data. Dengan mempelajari cara membuat kode konektor kustom untuk Power Query, Anda dapat dengan leluasa menentukan jumlah dan jenis sumber data yang dapat Anda hubungkan.

G. Ringkasan

Dalam bab ini, kita menyelami dunia perluasan kemampuan Power Query dengan menambahkan pustaka fungsi kustom dan membuat konektor kustom. Power Query adalah alat transformasi dan integrasi data yang hebat, tetapi potensinya yang sebenarnya dapat dilepaskan dengan mengembangkan konektor Anda sendiri untuk mengakses sumber data atau API khusus. Ini memungkinkan Anda untuk memasukkan data dari sistem atau layanan unik yang tidak didukung secara asli oleh Power Query.

Kita mulai dengan menyiapkan lingkungan pengembangan, termasuk mengonfigurasi Visual Studio Code, Power Query SDK, dan elemen Discord, sumber data target kita. Selanjutnya, kita membuat dan menjelaskan templat dasar untuk ekstensi Power Query kustom. Kita kemudian membuat kode dan mengonfigurasi autentikasi OAuth untuk konektor kustom kita. Selain itu, kita menulis kode M kustom untuk mengambil data dari Discord API dan menyajikan pengalaman navigasi untuk dialog Navigator yang ditemukan di UI Power Query. Terakhir, kita menunjukkan cara memasang dan menggunakan konektor kustom di Power BI Desktop.

Sepanjang bab ini, kita telah menjelaskan secara menyeluruh proses pengembangan dan anatomi konektor kustom, menguraikan komponen dan

kodenya untuk menjelaskan tujuannya. Kita meneliti cara menentukan metadata konektor, membuat koneksi, menangani autentikasi, dan menerapkan logika pengambilan dan transformasi data.

Anda telah mencapai akhir buku ini, yang menurut kita merupakan sumber tunggal paling definitif untuk bahasa M yang ada di mana pun. Jika Anda telah membaca seluruh buku ini, Anda telah mempelajari banyak informasi tentang bahasa M, dari elemen dasar bahasa M hingga konsep abstrak yang mendasari M dan segala hal di antaranya. Sepanjang jalan, kita telah memberikan banyak contoh praktis tentang cara menggunakan M untuk memecahkan masalah transformasi data yang menantang, serta praktik terbaik untuk mengodekan M. Kita sangat berharap Anda menikmati perjalanan ini dan merasa berdaya untuk memanfaatkan semua kemampuan bahasa M.

H. Penutup

1. Tes Formatif

No	Soal	Bobot
1.	Jelaskan konsep ekstensi dalam Power Query M. Apa saja keuntungan yang dapat diperoleh dengan menggunakan ekstensi dalam pengolahan data?	10
2.	Deskripsikan langkah-langkah untuk mengaktifkan ekstensi dalam Power Query. Apa saja pertimbangan yang perlu diperhatikan saat mengembangkan ekstensi?	10
3.	Diskusikan bagaimana fungsi Expression.Evaluate dapat digunakan untuk menciptakan kueri dinamis. Berikan contoh kasus penggunaan yang relevan dalam konteks bisnis.	10
4.	Apa itu lingkungan global dalam Power Query M? Jelaskan peran dan pentingnya lingkungan global dalam mengelola variabel dan fungsi dalam kueri.	10
5.	Analisis bagaimana penggunaan bagian (sections) dapat meningkatkan manajemen kueri dalam Power Query. Apa perbedaan antara bagian global dan lokal dalam konteks ini?	10
6.	Diskusikan tantangan yang mungkin dihadapi saat bekerja dengan ekstensi di Power Query. Bagaimana cara mengatasi tantangan tersebut untuk memastikan integritas dan efisiensi dalam proses transformasi data?	10

7.	Jelaskan bagaimana metadata dapat dimanfaatkan dalam Power Query untuk meningkatkan tata kelola data. Apa saja informasi yang bisa dilacak melalui metadata?	10
8.	Buatlah contoh skenario di mana penggunaan ekstensi dan fungsi Expression.Evaluate berkolaborasi untuk menyelesaikan masalah pengolahan data yang kompleks.	10
9.	Jelaskan bagaimana mekanisme penanganan kesalahan bekerja dalam ekstensi Power Query. Apa yang dapat dilakukan pengembang untuk memastikan bahwa ekstensi dapat menangani kesalahan dengan efektif?	10
10.	Diskusikan pentingnya interoperabilitas antara Power Query dan sistem eksternal lainnya. Bagaimana ekstensi dapat membantu dalam menciptakan integrasi yang lebih baik antara Power Query dan aplikasi lain?	10

DUMMY
Penerbitan & Percetakan
UNP PRESS

DAFTAR PUSTAKA

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer (2015). Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15): https://proceedings.neurips.cc/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf.

Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In 3rd International Conference on Learning Representations. https://arxiv.org/pdf/1409.0473.pdf

Thang Luong, Hieu Pham, and Christopher D. Manning (2015). Effective Approaches to Attention- based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. https://aclanthology.org/D15-1166/

André F. T. Martins, Ramón Fernandez Astudillo (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In Proceedings of the 33rd International Conference on Machine Learning. http://proceedings.mlr.press/v48/martins16.html

Ben Peters, Vlad Niculae, André F. T. Martins (2019). Sparse Sequence-to-Sequence Models. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. https://aclanthology.org/P19-1146/

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). Attention is All you Need. In Advances in Neural Information Processing Systems. https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). https://aclanthology. org/N19-1423/

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein (2018). Visualizing the Loss Landscape of Neural Nets. In Advances in Neural Information Processing Systems. https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf

Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath (2021). An Attentive Survey of Attention Models. ACM Trans. Intell. Syst. Technol. 12, 5, Article 53 (October 2021). https://doi.org/10.1145/3465055

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. (2020). The M4 Competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting, Volume 36, Issue 1. Pages 54-74. https://doi.org/10.1016/j.ijforecast.2019.04.014.

Slawek Smyl. (2018). M4 Forecasting Competition: Introducing a New Hybrid ES-RNN Model. https://www.uber.com/blog/m4-forecasting-competition/.

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

Kin G. Olivares and Cristian Challu and Grzegorz Marcjasz and R. Weron and A. Dubrawski. (2022). Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx. International Journal of Forecasting, 2022. https://www.sciencedirect.com/science/article/pii/S0169207022000413.

Cristian Challu and Kin G. Olivares and Boris N. Oreshkin and Federico Garza and Max Mergenthaler-Canseco and Artur Dubrawski. (2022). N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting. arXiv preprint arXiv: Arxiv-2201.12886. https://arxiv.org/ abs/2201.12886.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Lukasz, and Polosukhin, Illia. (2017). Attention is All you Need. Advances in Neural Information Processing Systems. https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. (2019). Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel. N Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4,344–4,353. https://aclanthology. org/D19-1443/.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. (2021). Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. Thirty-Fifth {AAAI} Conference on Artificial Intelligence, {AAAI} 2021. https://ojs.aaai.org/index.php/AAAI/article/view/17325.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. (2021). Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021. https://proceedings.neurips.cc/paper/2021/hash/bcc0d400288793e8bdcd7c19a8ac0c2b-Abstract.html.

Bryan Lim, Sercan Ö. Arik, Nicolas Loeff, and Tomas Pfister. (2019). Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. International Journal of Forecasting, Volume 37, Issue 4, 2021, Pages 1,748-1,764. https://www.sciencedirect.com/science/article/pii/S0169207021000637.

David Salinas, Valentin Flunkert, and Jan Gasthaus. (2017). DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. International Journal of Forecasting, 2017. https://www.sciencedirect.com/science/article/pii/S0169207019301888.

Taieb, S.B., Bontempi, G., Atiya, A.F., and Sorjamaa, A. (2012). A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. Expert Syst. Appl., 39, 7067-7083: https://arxiv.org/pdf/1108.3259.pdf

Li Zhang, Wei-Da Zhou, Pei-Chann Chang, Ji-Wen Yang, Fan-Zhang Li. (2013). Iterated time series prediction with multiple support vector regression models. Neurocomputing, Volume 99, 2013: https://www.sciencedirect.com/science/article/pii/ S0925231212005863

Taieb, S.B. and Atiya, A.F. (2016). A Bias and Variance Analysis for Multistep-Ahead Time Series Forecasting. in IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 1, pp. 62-76, Jan. 2016: https://ieeexplore.ieee.org/document/7064712.

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.

DUMMY
Penerbitan & Percetakan
UNP PRESS

INDEKS

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

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 "**Panduan Definitif untuk Power Query (M) – Jilid 4**" melanjutkan eksplorasi mendalam tentang kemampuan bahasa M dalam mengelola dan mengoptimalkan transformasi data. Jilid ini membahas topik-topik lanjutan seperti iterasi dan rekursi, penerapan pola data yang bermasalah, strategi optimasi kinerja termasuk query folding dan firewall, hingga teknik pengembangan ekstensi Power Query untuk membangun konektor khusus.

Disusun secara sistematis dengan contoh kasus, tes formatif, glosarium, dan lampiran, buku ini dirancang untuk membantu pembaca memahami konsep lanjutan sekaligus praktik implementasi di dunia nyata. Kehadiran jilid keempat ini menjadikan seri buku Power Query (M) semakin lengkap sebagai rujukan bagi mahasiswa, dosen, peneliti, dan praktisi data yang ingin menguasai transformasi data secara komprehensif dan profesional.

LAMPIRAN

1. Kartu Rencana Studi (KRS)

	KEMENTERIAN PENDIDIKAN, KEBUDAYAAN, RISET DAN TEKNOLOGI UNIVERSITAS NEGERI PADANG – DIREKTORAT AKADEMIK – SUBDIT. INOVASI PEMBELAJARAN DAN MBKM					
	RENCANA PEMBELAJARAN SEMESTER					
MATA KULIAH (MK)	**KODE**	**Rumpun MK**	**BOBOT (sks)**		**SEMESTER**	**Tgl Penyusunan**
Praktikum Basis Data	INF1.62.4008	Mata kuliah Wajib Program Studi	1 SKS	Praktek	4	30 Januari 2024
OTORISASI / PENGESAHAN	**Dosen Pengembang RPS**		**Koordinator RMK**		**Koordinator PRODI**	
Direktorat Akademik, UNP **Subdit. Inovasi Pembelajaran dan MBKM (Dr. Nofrion, M. Pd)**	**1. Dr. Phil. Dony Novaliendry, S. Kom., M. Kom**		**Dr. Phil. Dony Novaliendry, S. Kom., M. Kom** **NIP. 197511042006041002**		**Dr. Yeka Hendriyani, S.Kom., M.Kom** **NIP. 198405202010122003**	

Capaian Pembelajaran	**CPL-PRODI yang dibebankan pada MK**	
	S5	Bertaqwa kepada Tuhan Yang Maha Esa dan mampu menunjukkan sikap religious dan Menunjukkan sikap bertanggungjawab atas pekerjaan di bidang keahliannya secara mandiri
	P2	Memahami konsep dasar matematika, ilmu listrik dan elektronika dalam bidang komputer
	KU5	mampu mengambil keputusan secara tepat dalam konteks penyelesaian masalah di bidang keahliannya, berdasarkan hasil analisis informasi dan data.
	KK 2	Kemampuan menguasai dasar pemograman phyton,metode komputasi Gauss dan komputasi metode LU Decomposition
	Capaian Pembelajaran Mata Kuliah (CPMK)	
	CPMK 1	Menguasai konsep bahasa pemrograman.
	CPMK 2	Mengidentikasi model- model bahasa pemrograman.
	CPMK 3	Membandingkan berbagai solusi.
	CPMK 4	Menguasai konsep-konsep basis data dan mampu membangun basis data untuk pngembangan sistem berbasis komputer
	Kemampuan akhir tiap tahapan belajar (Sub-CPMK)	
	SUB CPMK 1	Mahasiswa mampu memahami dan menganalisa konsep Bahasa pemrograman
	SUB CPMK 2	Mahasiswa mampu merumuskan konsep model-model Bahasa pemrograman
	SUB CPMK 3	Mahasiswa mampu merumuskan latar belakang konsep membandingkan berbagai solusi terkait Bahasa pemrograman.
	SUB CPMK 4	Mahasiswa mampu memahami, menganalisa, dan membangun konsep basis data, perancangan basis data, perancangan tabel-tabel, entry data pada tabel, relasi antar tabel, dan pembuatan report.

Peta CPL – CP MK

	Sub-CPMK1	Sub-CPMK2	Sub-CPMK3	Sub-CPMK4
CPMK 1	√			
CPMK 2		√		
CPMK 3			√	
CPMK 4				√

Deskripsi Singkat Mata Kuliah	Mata kuliah ini mempelajari dan menguasai konsep dan mengimplementasi pembuatan aplikasi basis data menggunakan salah satu DBMS dan bahasa pemrograaman basis data dengan urutan: perancangan tabel-tabel, entry data pada tabel, pembuatan form (untuk entry data dan untuk tampilan menu), pembuatan query, pembuatan report.
Bahan Kajian: Materi pembelajaran	1. Aplikasi DBMS, MariaDB, MySQL, PostgreSQL, phpMyAdmin 2. Bahasa pemograman sql, DDL 3. DML (Data Manipulation Language) 4. Arithmetic Operator, Agregate function, String function, Numeric function, Date/Time Function 5. Group by, group by with order, having 6. Fungsi agregat 7. Database Relation 8. Database Relation Join 9. Union, Intersect, Except 10. View dan control flow function 11. Create Procedure

Pustaka Buku 10 tahun terakhir Artikel 5 tahun terakhir Kecuali buku atau artikel yang memuat grand theory.	**Utama:** 1. Modul Praktikum Sistem Basis Data: Tim Dosen Program Studi Informatika 2. C. J. Date. 2006. An Introduction to Database Systems 8th. Pearson Education
	Pendukung: 1. Churcher, C., 2007, Beginning Database Design: From Novice to Professional (ebook available) 2. Oppel, A. & Sheldon, R., 2009, SQL: A Beginner's Guide- 3. Taylor, A.G., 2011, SQL Essential-All in One for Dummies
Dosen Pengampu	Dr. Phil. Dony Novaliendry, S. Kom., M. Kom NIP. 197511042006041002
Mata kuliah syarat	-

Mg Ke-	SUB-CPMK (Kemampuan Akhir Yang Diharapkan)	Penilaian		Bentuk Pembelajaran, Metode Pembelajaran, Penugasan Mahasiswa [Estimasi Waktu]		Materi Pembelajaran [Rujukan]	Bobot Penilaian (%)
		Indikator	Bentuk & Kriteria	Luring (Tatap Muka)	Daring (Online)		
1	Mahasiswa mengenal dan dapat menggunakan Aplikasi DBMS	1. Ketepatan menjelaskan aplikasi Xampp (MariaDB) 2. Ketepatan menjelaskan aplikasi MySQL 3. Ketepatan menjelaskan aplikasi PostgreSQL 4. Ketepatan menjelaskan aplikasi Oracle	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Mengenal Aplikasi DBMS - MariaDB - MySQL - PostgreSQL - Oracle	**10%**
2	Mahasiswa mampu menjelaskan perintah-perintah dasar SQL dan kelompok pernyataan SQL untuk pendefinisian basis data	1. Ketepatan menjelaskan DDL (Data Defenition Language)	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	SQL: - Pengenalan SQL - DDL (Data Definition Language)	**5%**
3	Mahasiswa mampu menggunakan perintah-perintah DML	1. Ketepatan menjelaskan Data Manipulation Language	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	DML (Data Manipulation Language)	**5%**
4	Mahasiswa Memahami perintah-perintah SQL untuk mengambil data dan kemudian melakukan perhitungan-perhitungan aritmatika dari datatersebut	1. Ketepatan menjelaskan Arithmetic Operators 2. Ketepatan menjelaskan agregate function 3. Ketepatan menjelaskan aString function 4. Ketepatan menjelaskan	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Operator, Agregate Function, String Function, Numeric Function, Date/Time Function.	**10%**

		5. Ketepatan menjelaskan Date Function					
5	Mahasiswa mampu menggunakan perintah SQL untuk menyatukan dua atau lebih grup data kedalam suatu fungsi data tunggal	1. Ketepatan menjelaskan Group By 2. Ketepatan menjelaskan group by with order 3. Ketepatan menjelaskan having	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Group By, Group By with Order By, Having.	**10%**
6	Mahasiswa mampu menjalankan perintah-perintah fungsi Agregat pada tabel	1. Ketepatan menjelaskan fungsi agreagat	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Fungsi Agregat	**10%**
7	Mahasiswa mampu merelasikan tabel	1. Ketepatan menjelaskan relasi 2 tabel 2. Ketepatan menjelaskan relasi 3 tabel 3. Ketepatan menjelaskan lebih dari 3 tabel	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Database Relation - Relasi 2 tabel - Relasi 3 tabel - Relasi Lebih dari 3 tabel	**10%**
8	**Ujian Tengah Semester (UTS) = 10%**						
9	Mahasiswa mampu merelasikan tabel dengan menggunakan perintah Join	1. Ketepatan menjelaskan relasi antar tabel menggunakan join	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Relasi Antar Tabel - Join	**10%**
10	Mahasiswa mampu menjalankan perintah-perintah Union, Intersect, Except.	1. Ketepatan menjelaskan Union 2. Ketepatan menjelaskan Intersect 3. Ketepatan menjelaskan except	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	- Union - Intersect - Except	**10%**
11-12	Mahasiswa Memahami dan mampu menggunakan View dan control flow function	1. Ketepatan menjelaskan view di database 2. Ketepatan menjelaskan control flow function	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	View dan control flow function.	**10%**
13-15	Mahasiswa memahami dan mampu menggunakan Store Procedure dan Function yang merupakan perintah-perintah SQL yang diletakkan di dalam server database.	1. Ketepatan menjelaskan create procedure 2. Ketepatan menjelaskan create function	Menggunakan Rubrik Penilaian	1. Presentasi 2. Praktek **TM : 1x (1 x 100 Menit)** 3. Penugasan Terstruktur **BM+BT : 1x(1x70 Menit)**	Bahan ajar materi dapat diunduh melalui LMS UNP pada tautan : https://elearning.unp.ac.id/my/	Create Procedure dan Create Function.	**15%**
16	**Ujian Akhir Semester (UAS) = 10%**						