# Basis Data Lanjut

# Pemrograman T-SQL dan Error Handling

Dosen Pengampu: Irsyad Arif Mashudi, S.Kom., M.Kom.



**Nama : Maulana Bintang Irfansyah NIM : 2041720132**

**Kelas : TI-2H**

**POLITEKNIK NEGERI MALANG JURUSAN TEKNOLOGI INFORMASI**

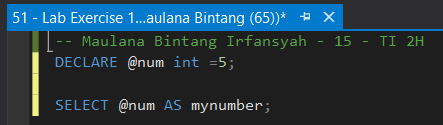
**PROGRAM STUDI D-IV TEKNIK INFORMATIKA KOTA MALANG**

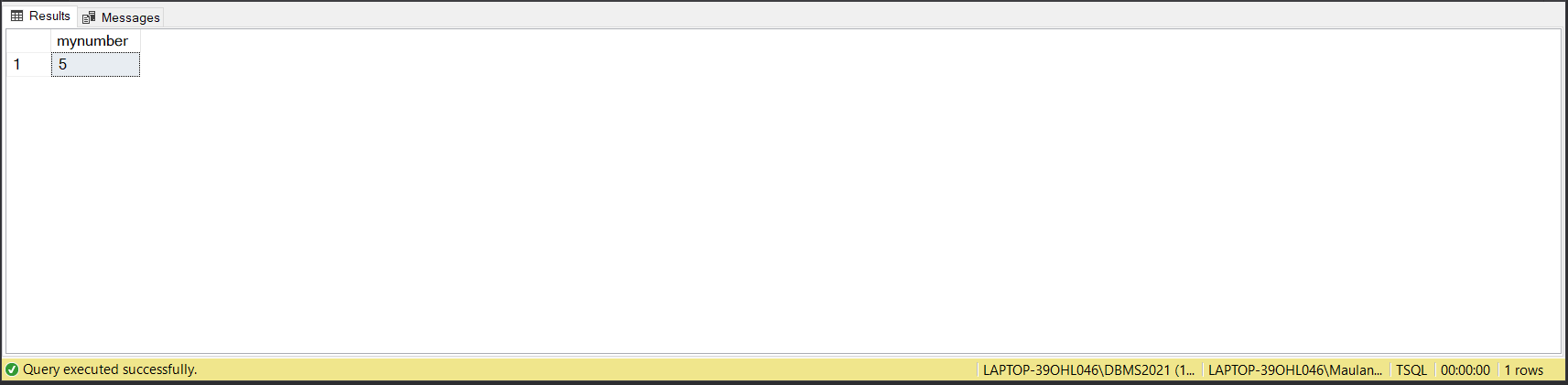
**2021**

**Praktikum 1: DEKLARASI VARIABEL & BATCH: Mendeklarasikan variabel dan mendapatkan nilai variabel**

1. Buatlah sebuah kode T-SQL dengan mendeklarasikan sebuah variable bernama @num yang bertipe data integer bernilai 5. Tampilkan nilai variabel tersebut dengan menggunakan alias mynumber lalu eksekusi.

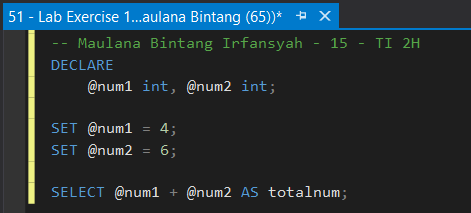
**Jawaban:**

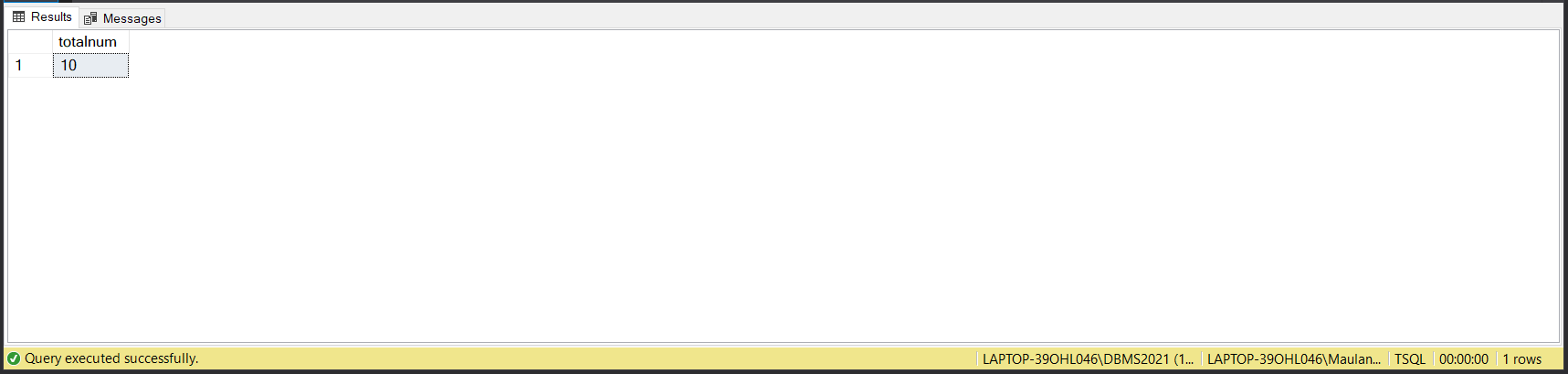




1. Dari skrip T-SQL **[Soal-1]** di atas, tambahkan batch delimiter (GO) setelahnya, lalu buatlah skrip T-SQL baru yang mendefinisikan 2 variabel bernama @num1 dan @num2 yang sama-sama bertipe data integer. Set nilainya masing-masing 4 dan 6. Tulis sebuah query SELECT yang menampilkan jumlah kedua variable tersebut sebagai totalnum, lalu eksekusi.

**Jawaban:**



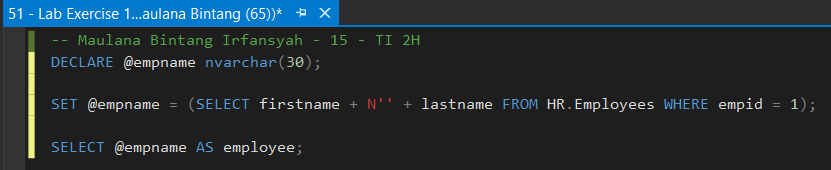


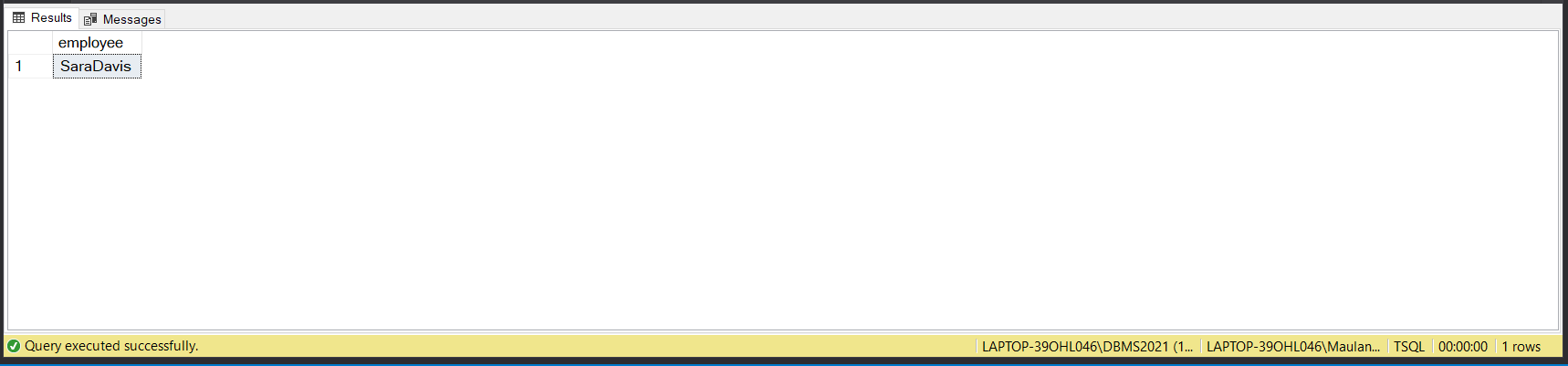
**Praktikum 2: DEKLARASI VARIABEL & BATCH: Memberi nilai terhadap variabel menggunakan query SELECT**

1. Buatlah sebuah skrip T-SQL dengan mendefinisikan variabel @empname yang bertipe data nvarchar(30). Selanjutnya, set nilai variabel tersebut sebagai hasil query SELECT terhadap tabel **HR.Employees**, yang menggabungkan kolom *firstname* dan *lastname* dengan dipisahkan spasi, dimana nilai empid-nya sama dengan 1.

Terakhir, tampilkan nilai variabel @empname dengan menggunakan query SELECT dan beri nama alias sebagai employee. Eksekusi skrip tersebut.

**Jawaban:**





1. Apakah yang terjadi apabila query SELECT yang dihasilkan lebih dari 1 baris? Lakukan uji coba misalnya dengan menghilangkan filter WHERE “empid = 1”.

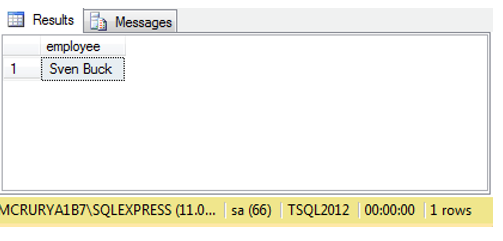
**Jawaban:**

Apabila *query* SELECT yang dihasilkan lebih dari 1 baris akan mendapatkan pesan *error* “*Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression.”* Karena pernyataan SET diharuskan untuk menggunakan *subquery* scalar untuk *pull data* dari tabel. Dimana *subquery* scalar akan gagal dijalankan ketika mengembalikan lebih dari satu nilai.

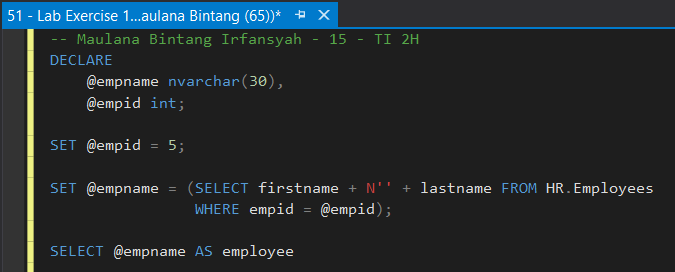
**Praktikum 3: DEKLARASI VARIABEL & BATCH: Menggunakan sebuah variabel dalam klausa WHERE**

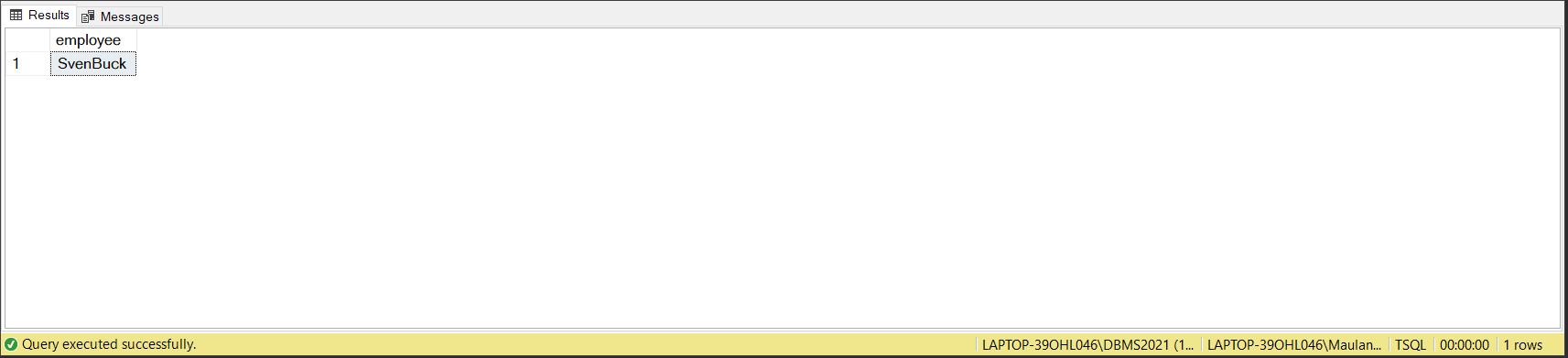
1. Salinlah skrip T-SQL dari **[Soal- 3]** di atas dan lakukan modifikasi dengan mendefinisikan sebuah variabel baru bernama @empid yang bertipe data integer bernilai 5. Lalu, gunakan variabel baru ini dalam klausa WHERE sebagai nilai dari kolom *empid*. Eksekusi skrip tersebut.

Hasil yang benar ditunjukkan pada tampilan berikut:



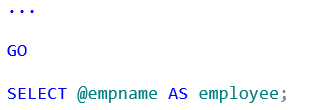
**Jawaban:**





**Praktikum 4: DEKLARASI VARIABEL & BATCH: Menambah sebuah batch delimiter**

1. Salinlah skrip T-SQL dari [Soal-5] di atas dengan menambahkan batch delimiter GO sebelum query SELECT seperti di bawah ini:



Setelah mengeksekusi skrip tersebut, apakah yang terjadi? Mengapa demikian?

**Jawaban:**

Setelah menambahkan *batch delimiter* GO sebelum *query* SELECT di atas, yang terjadi adalah *error* yang memberikan pesan “*Must declare the scalar variable "@empname*”. *Error* tersebut disebabkan karena variabel lokal untuk *batch* yang didefinisikan, dimana jika ingin merujuk ke variabel yang didefinisikan di *batch* lain akan mendapatkan kesalahan bahwa variabel tidak didefinisikan. Informasi tambahan untuk GO merupakan perintah *client* dan bukan perintah server T-SQL.

**Praktikum 5: CONTROL OF FLOW: Membuat conditional logic sederhana**

1. Tuliskan pernyataan EXECUTE untuk memanggil stored prosedur sys.sp\_helptext, kemudian lewatkan stored prosedur Sales.GetTopCustomers yang disimpan sebagai parameter, dengan hasil yang sesuai tampilan di bawah ini!

Buatlah sebuah skrip T-SQL dengan mendeklarasikan variabel @result bertipe nvarchar(20) dan variabel @i bertipe integer bernilai 8.

Tambahkan statement IF yang memenuhi *logic* di bawah ini:

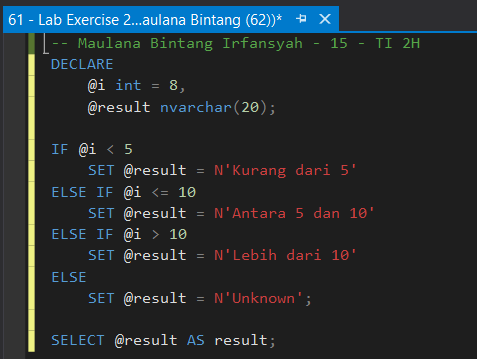
* Jika variabel @i bernilai kurang dari 5, set nilai variabel @result menjadi “Kurang dari 5”
* Jika variabel @i bernilai antara 5 dan 10, set nilai variabel @result menjadi “Antara 5 dan 10”
* Jika variabel @i bernilai lebih dari 10, , set nilai variabel @result menjadi “Lebih dari 10”
* Selain dari itu, , set nilai variabel @result menjadi “Unknown”

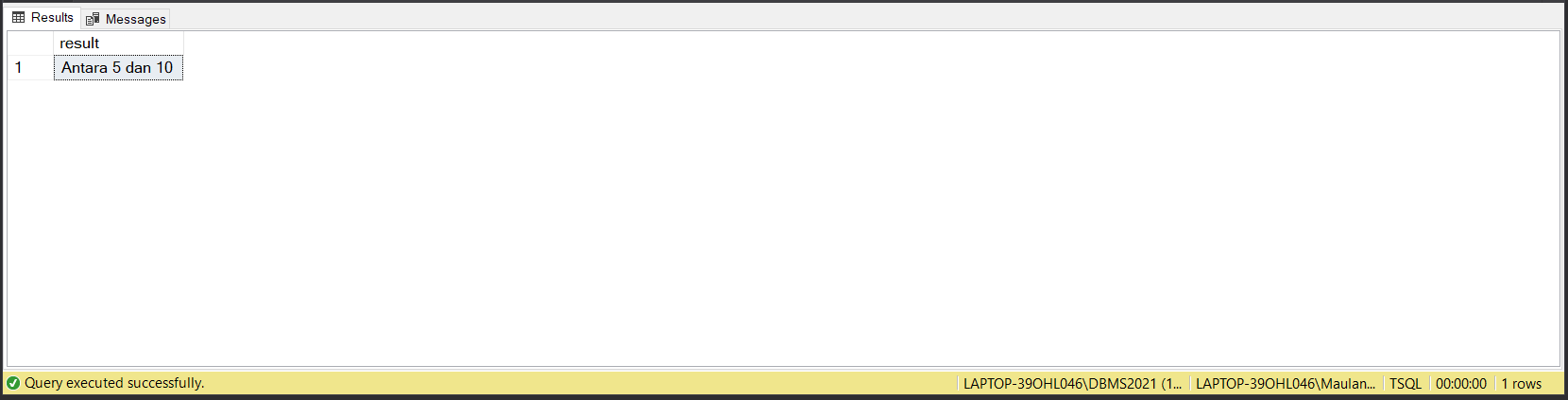
Di bagian akhir, tambahkan sebuah query SELECT untuk menampilkan nilai variabel @result dengan memberi alias result.

Eksekusi skrip yang sudah dibuat dan bandingkan dengan hasil berikut ini:



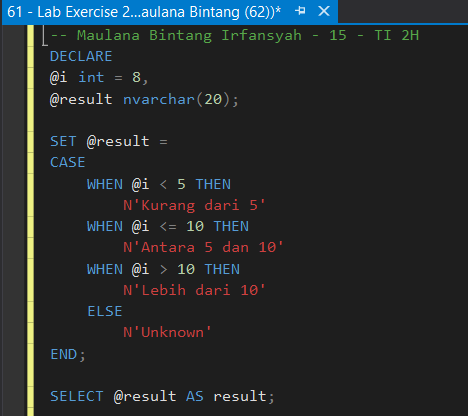
**Jawaban:**

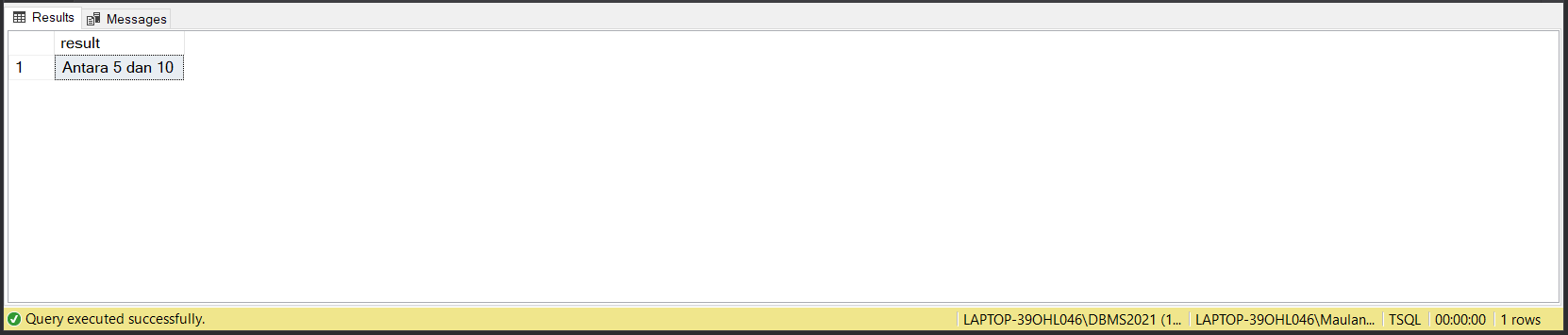




1. Modifikasilah skrip T-SQL dari [Soal-7] di atas dengan mengganti statement IF menjadi ekspresi CASE dan pastikan hasilnya sama.

**Jawaban:**



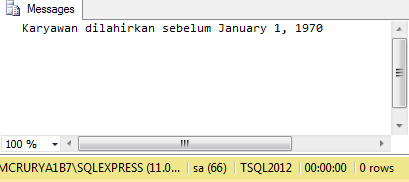


**Praktikum 6: CONTROL OF FLOW: Mengecek tanggal lahir karyawan**

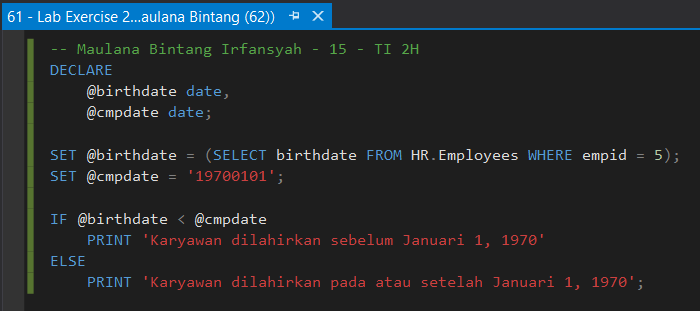
1. Ikuti langkah berikut ini untuk membuat kode T-SQL yang mengecek tanggal lahir karyawan:

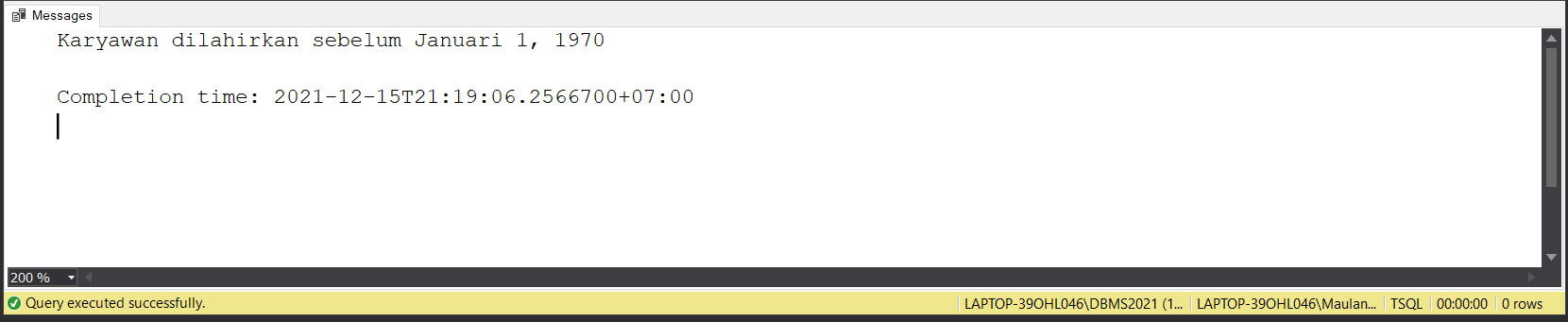
* Pertama, deklarasikan 2 variabel, yakni @birthdate dan @cmpdate (keduanya bertipe data date).
* Set nilai variabel @birthdate sebagai hasil dari query SELECT terhadap kolom birthdate dari tabel **HR.Employees**, dimana empid-nya adalah 5.
* Set variabel @cmpdate berisi tanggal January 1, 1970
* Buatlah pernyataan kondisional IF dengan membandingkan nilai @birthdate dan @cmpdate. Apabila @birthdate lebih kecil dari @cmpdate, gunakan perintah PRINT untuk menampilkan pesan “Karyawan dilahirkan sebelum Januari 1, 1970”. Selain itu, tampilkan pesan “Karyawan dilahirkan pada atau setelah Januari 1, 1970”.
* Eksekusi keseluruhan skrip T-SQL di atas.

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**



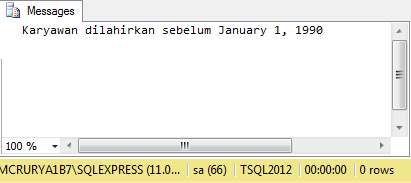


**Praktikum 7: CONTROL OF FLOW: Membuat dan mengeksekusi stored procedure**

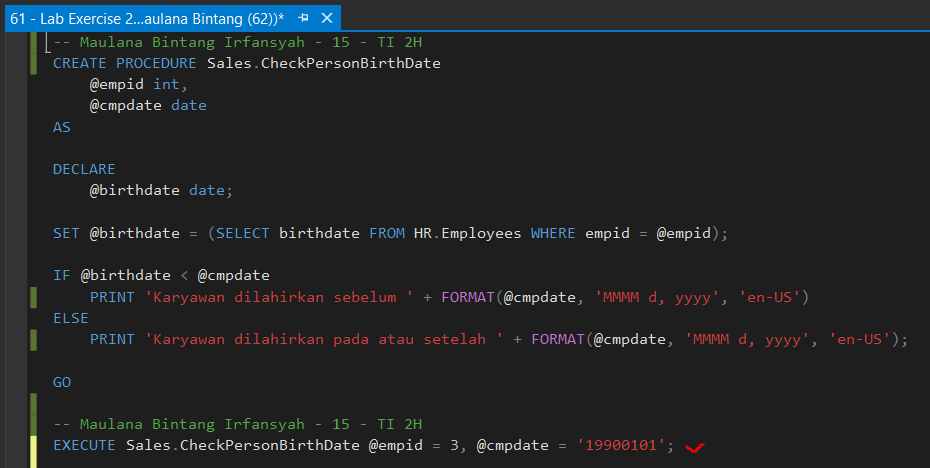
1. Stored procedure bernama **Sales.CheckPersonBirthDate** pada Langkah 1 di atas mempunyai 2 parameter, yakni @empid (untuk menentukan ID karyawan) dan @cmpdate (untuk perbandingan tanggal).

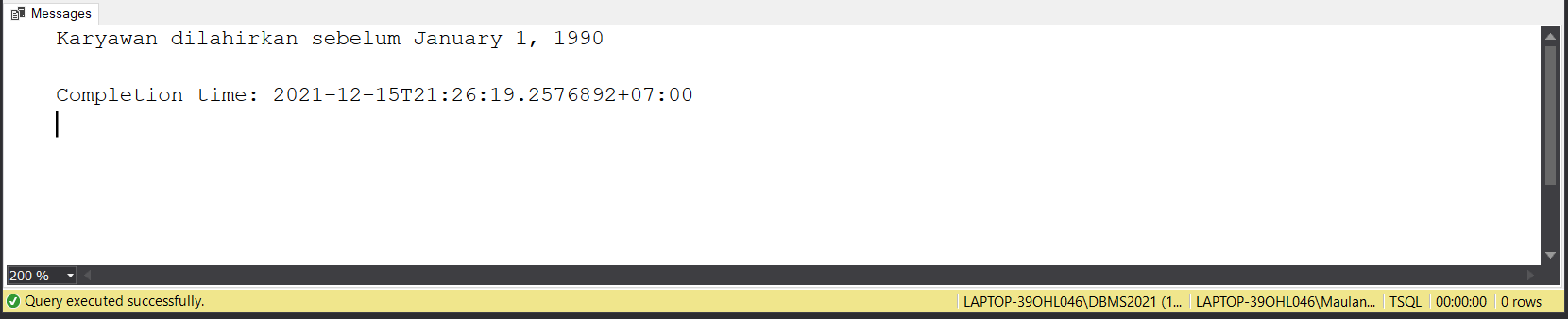
Lakukan perintah EXECUTE pada stored procedure tersebut dengan memasukkan parameter @empid = 3 dan @cmpdate yang di-set ke tanggal January 1, 1990.

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**



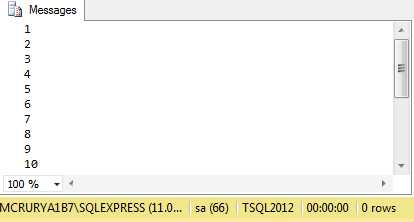


**Praktikum 8: CONTROL OF FLOW: Melakukan loop/ pengulangan menggunakan pernyataan WHILE**

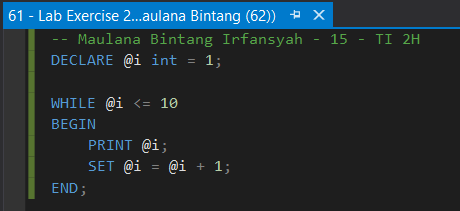
1. Buatlah sebuah skrip T-SQL yang berisi looping/ pengulangan dengan mengikuti langkah berikut:

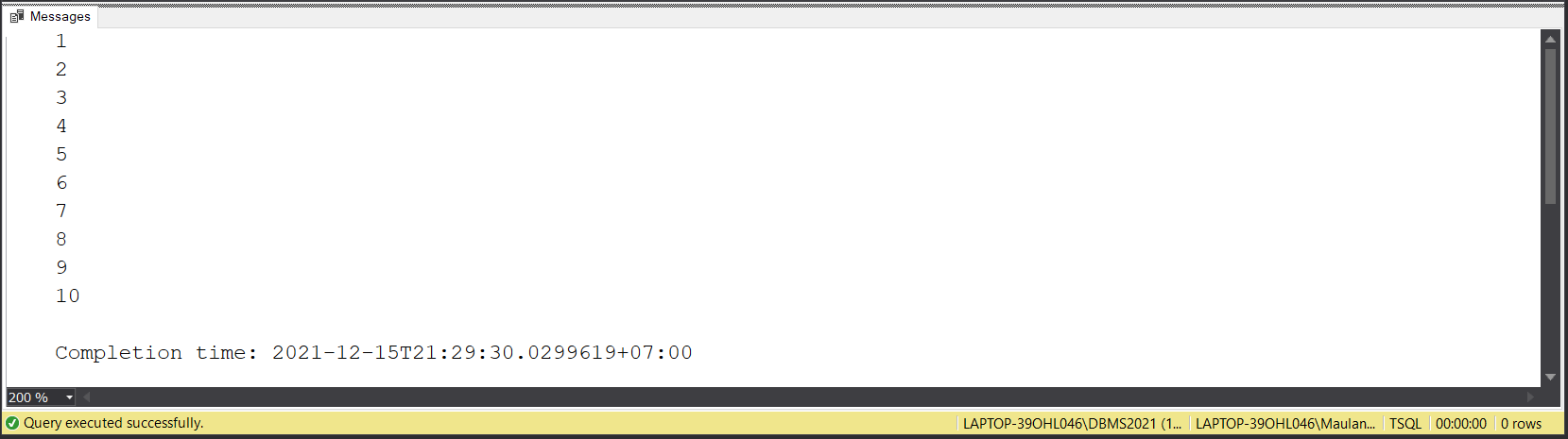
* Pertama, deklarasikan sebuah variabel @i yang bertipe data integer bernilai 1
* Lalu buatlah sebuah pengulangan dengan menggunakan pernyataan WHILE, dimana selama nilai variabel @i kurang dari 10, tampilkan/ cetak variabel @i dan tambahkan nilai @i secara incremental dengan menambah 1 (@i+1).

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**





**Praktikum 9: DYNAMIC SQL: Membuat SQL dinamis tanpa parameter**

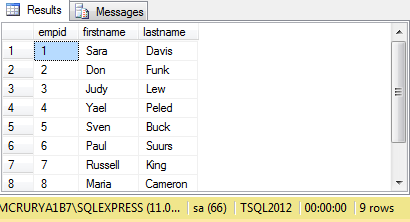
1. Buatlah skrip T-SQL dengan mendeklarasikan variabel bernama @SQLstr bertipe data nvarchar(200). Lalu, isikan dengan string berupa statement SELECT yang mengambil kolom *empid*, *firstname*, dan *lastname* dari tabel **HR.Employees**.

* **Perhatikan** bahwa yang diambil adalah string statement-nya, bukan hasil query-nya. Contoh: SET @SQLstr = N’SELECT....’;

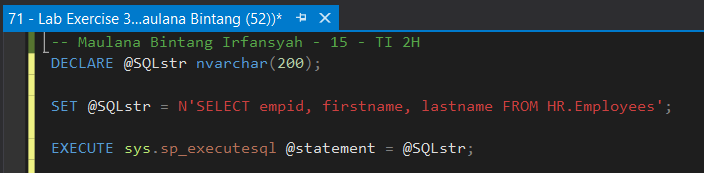
Bukan SET @SQLstr = (SELECT....);

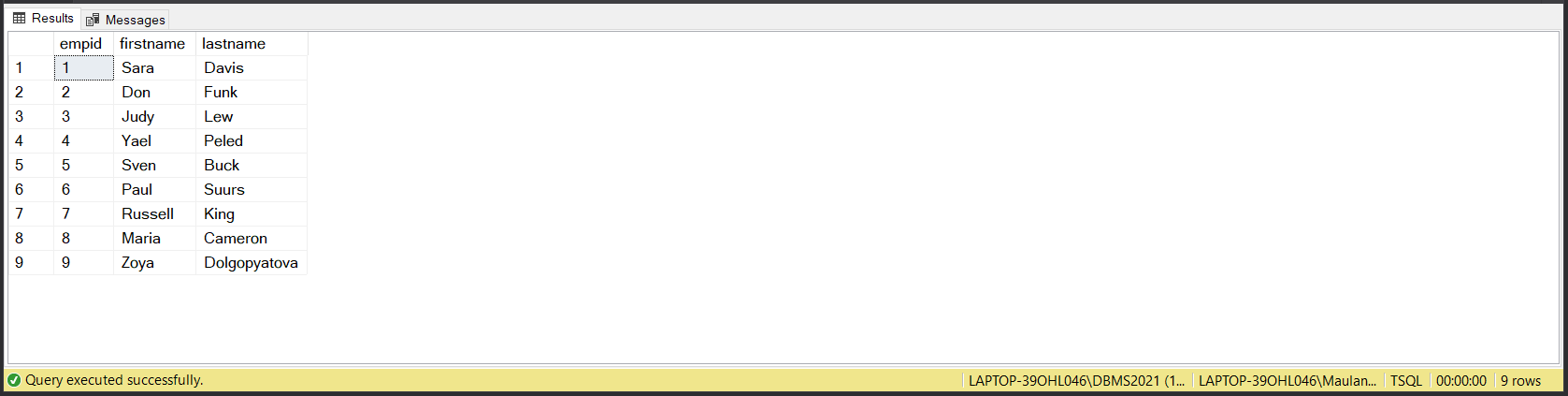
* Lakukan perintah EXECUTE yang mengeksekusi pernyataan query yang sudah di-set dalam variabel @SQLstr. (Petunjuk: gunakan sp\_executesql)

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**

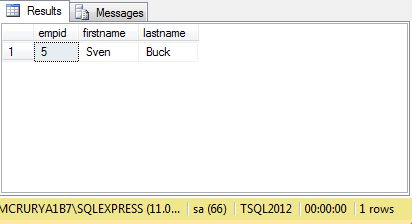




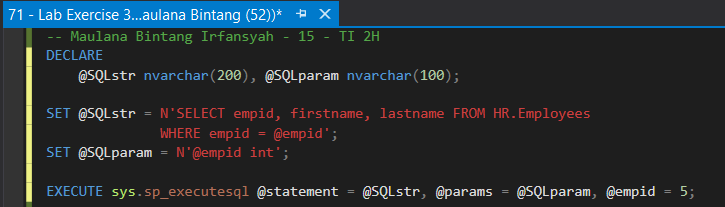
1. Salinlah skrip T-SQL dari **[Soal-12]** di atas, lakukan modifikasi dengan mengikuti langkah berikut:

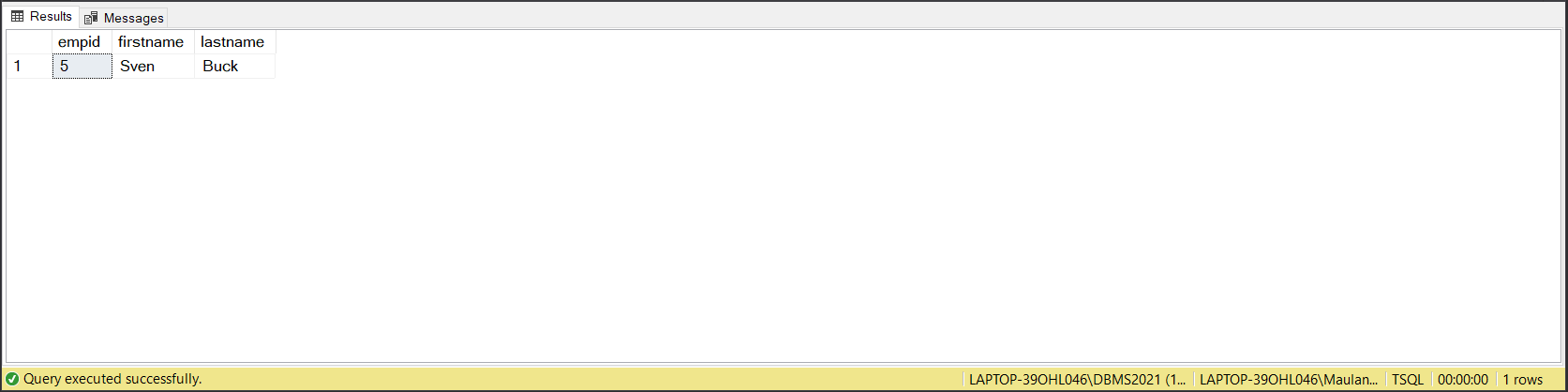
* Tambahkan filter WHERE dimana empid = @empid pada string pernyataan SELECT yang disimpan pada variabel @SQLstr
* Deklarasikan variabel baru bernama @SQLParam bertipe data nvarchar(100) untuk menyimpan string definisi dari @empid yang bertipe data int
* Seperti halnya pada **[Soal-13]**, lakukan perintah EXECUTE dengan menambahkan parameter @empid = 5

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**

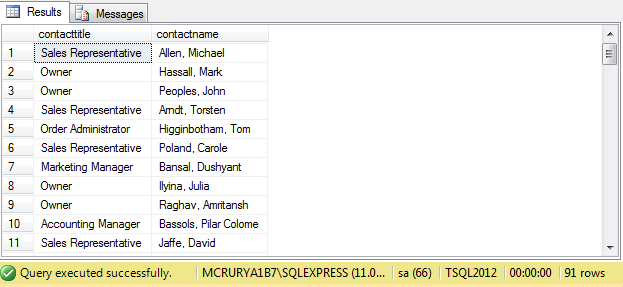




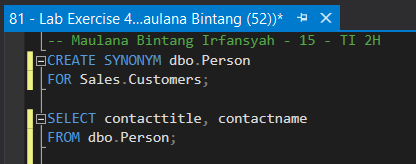
1. Tulislah sebuah skrip T-SQL yang membuat SYNONYM bernama **dbo.Pelanggan** dari tabel **Sales.Customers** dalam database TSQL. Eksekusi skrip tersebut.

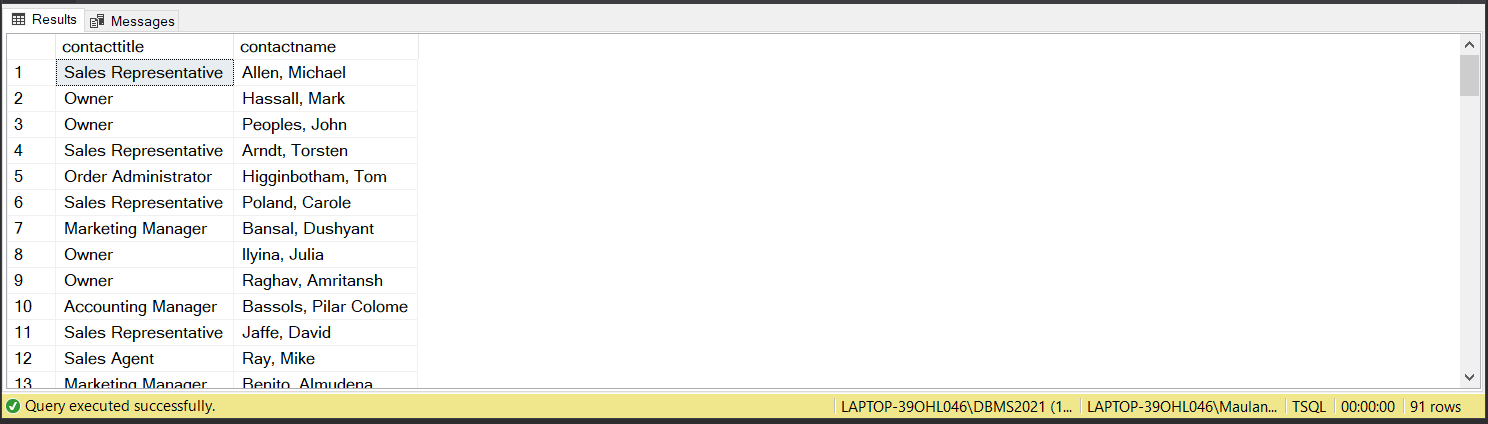
Kemudian, buatlah query SELECT terhadap synonym **dbo.Pelanggan** yang mengambil kolom *contacttitle* dan *contactname*. Eksekusi kembali skrip T-SQL ini.

Hasil yang benar ditunjukkan pada tampilan berikut:



**Jawaban:**



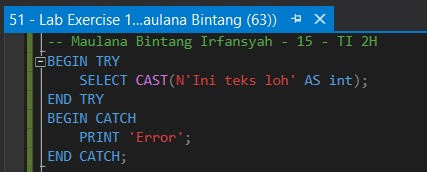


**Praktikum 11: TRY / CATCH: Membuat blok TRY / CATCH sederhana**

1. Buatlah konstruksi TRY / CATCH dengan menempatkan query pada Langkah 1 di atas dalam blok TRY. Sedangkan dalam blok CATCH, isikan perintah untuk menampilkan teks “Error”. Jalankan skrip T-SQL tersebut, lalu pada tab Messages, akan menampilkan pesan:



**Jawaban:**

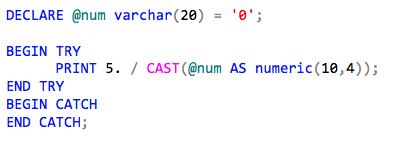




**Praktikum 12: TRY / CATCH: Menampilkan kode & pesan error**

1. Buatlah konstruksi TRY / CATCH dengan menempatkan query pada Langkah 1 di atas dalam blok TRY. Sedangkan dalam blok CATCH, isikan perintah untuk menampilkan teks “Error”. Jalankan skrip T-SQL tersebut, lalu pada tab Messages, akan menampilkan pesan:

Salin dan eksekusi skrip T-SQL berikut ini dan perhatikan hasilnya:



Pada langkah di atas, jika memperhatikan nilai variabel @num, semestinya dihasilkan error “division by zero”, tetapi nyatanya tidak. Mengapa demikian?

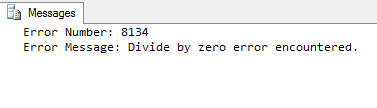
**Jawaban:**

Pada langkah di atas, tidak menghasilkan *error* karena telah ditangani oleh TRY / CATCH *construct*.

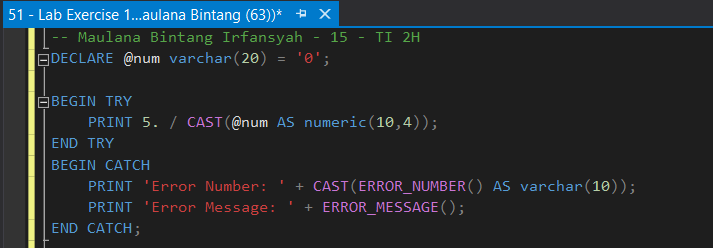
1. Modifikasilah skrip T-SQL pada Langkah 1 di atas dengan menambahkan 2 (dua) pernyataan PRINT pada bagian blok CATCH. Pernyataan yang pertama untuk menampilkan nomer error dengan menggunakan fungsi ERROR\_NUMBER dan pernyataan kedua untuk menampilkan pesan error dengan memakai fungsi ERROR\_MESSAGE.

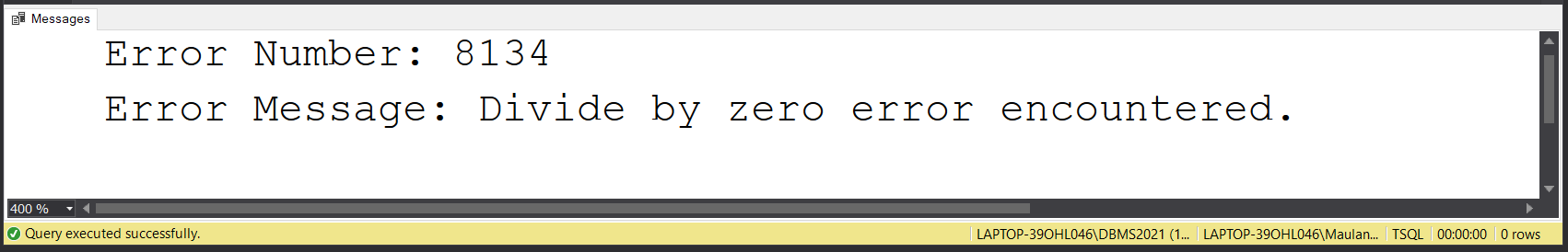
Untuk memperjelas, tambahkan string label “Error Number:” pada pesan pertama dan string label “Error Message:” pada pesan kedua.

Eksekusi dan bandingkan hasilnya dengan tampilan berikut:



**Jawaban:**



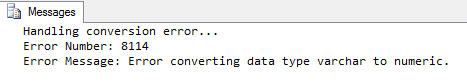


**Praktikum 13: TRY / CATCH: Menambahkan conditional logic pada blok CATCH**

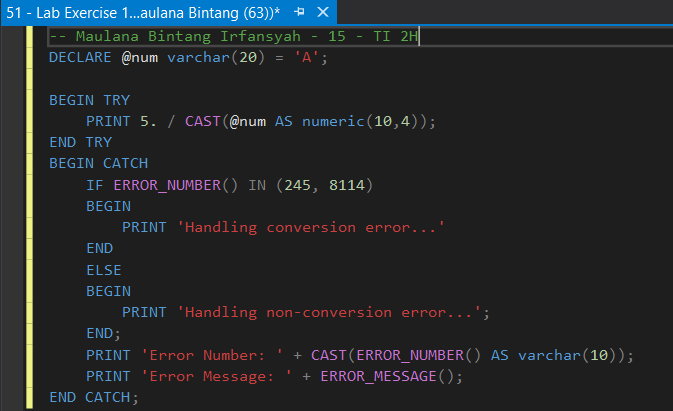
1. Dengan tetap menggunakan skrip T-SQL pada Bagian 12 Langkah 1, lakukan modifikasi dengan menambahkan pernyataan IF pada bagian blok CATCH sebelum pernyataan PRINT.

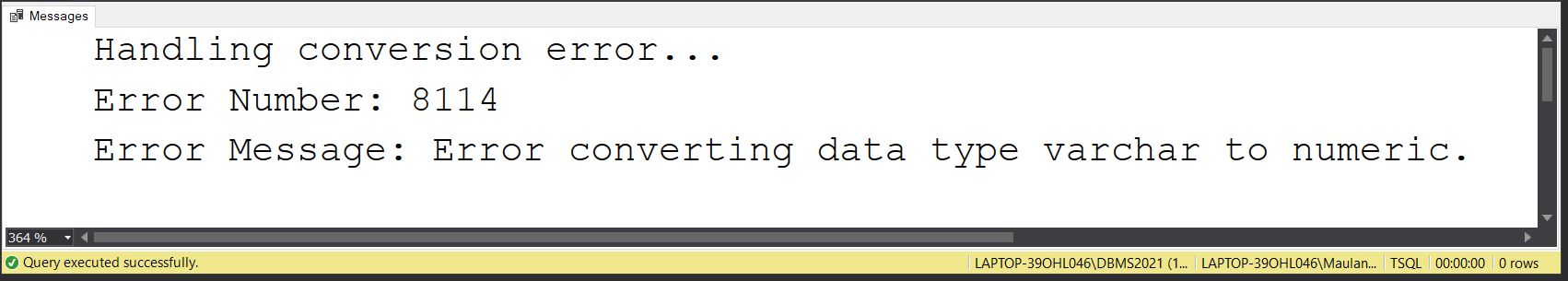
* Pernyataan IF tersebut untuk mengecek apakah nomer error = 245 atau 8114.
* Apabila kondisi ini terpenuhi, tampilkan pesan “Handling conversion error...” dengan perintah PRINT.
* Jika tidak sama dengan 245 atau 8114,
* tampilkan pesan “Handling NON conversion error...”.
* Terakhir, set nilai variabel @num sebagai ‘A’, lalu eksekusi skrip T-SQL tersebut.

Bandingkan hasilnya dengan tampilan berikut:



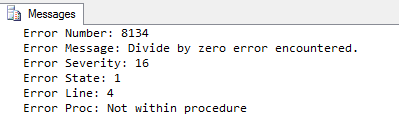
**Jawaban:**



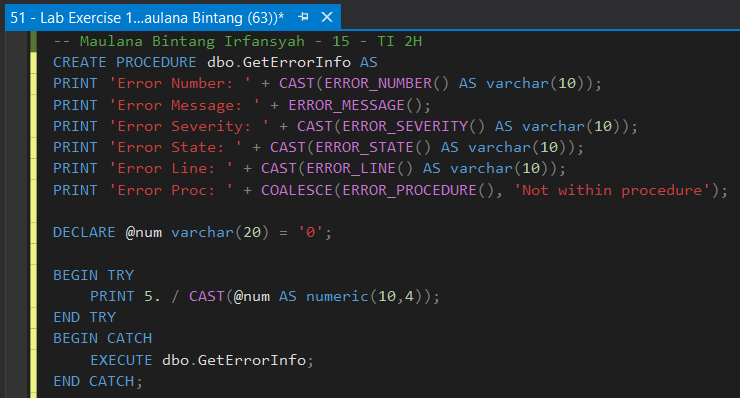


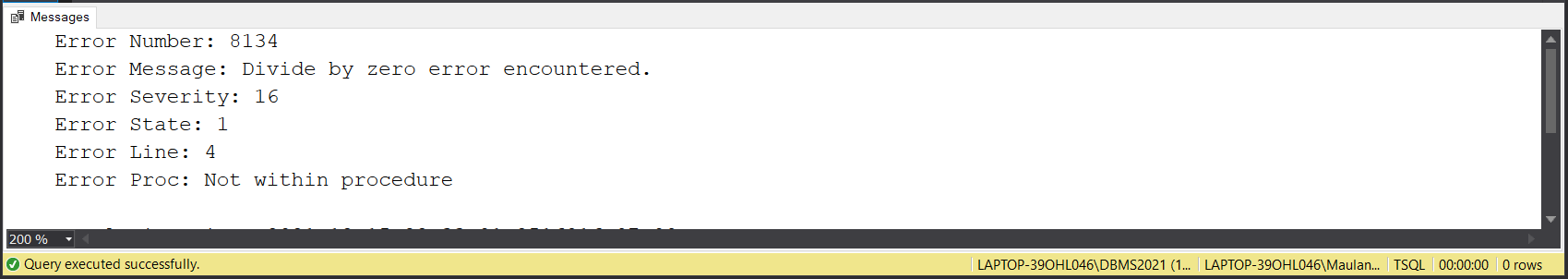
**Praktikum 14: TRY / CATCH: Mengeksekusi stored procedure pada blok CATCH**

1. Buatlah sebuah konstruksi TRY / CATCH, dimana pada bagian blok CATCH, lakukan eksekusi stored procedure yang telah dibuat pada Langkah 1 di atas, lalu jalankan.



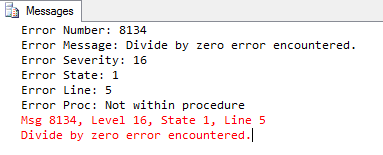
**Jawaban:**



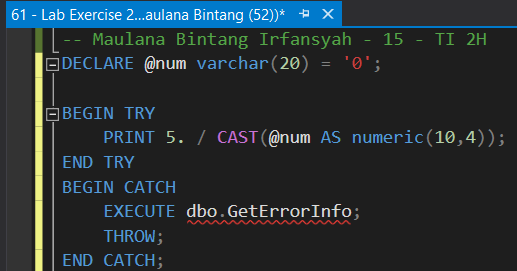


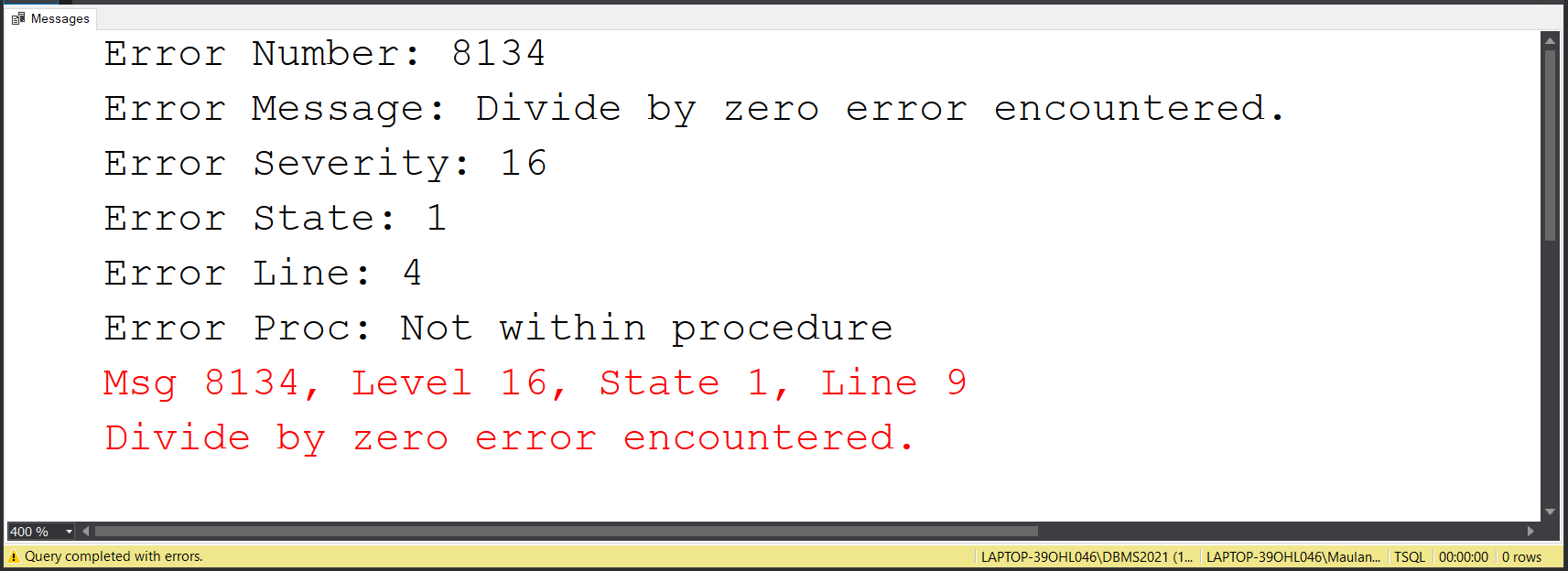
**Praktikum 15: THROW: Menggunakan THROW untuk mengirimkan kembali pesan error**

1. Modifikasi skrip T-SQL dari Soal-19 di atas dengan menambahkan perintah THROW yang ditempatkan setelah pernyataan eksekusi stored procedure.



**Jawaban:**

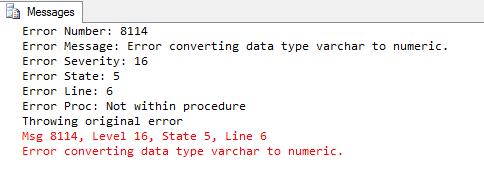




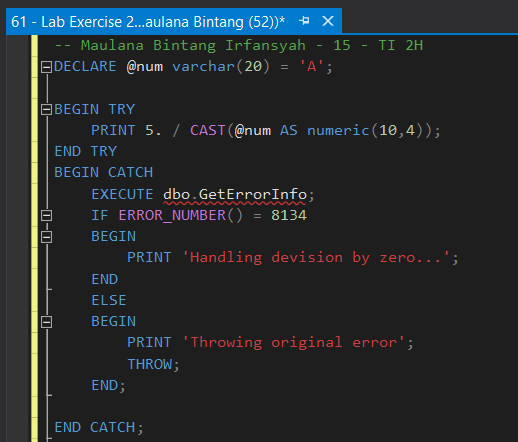
1. Modifikasi skrip T-SQL dari Soal-20 di atas dengan mengganti perintah THROW dengan pernyataan IF.

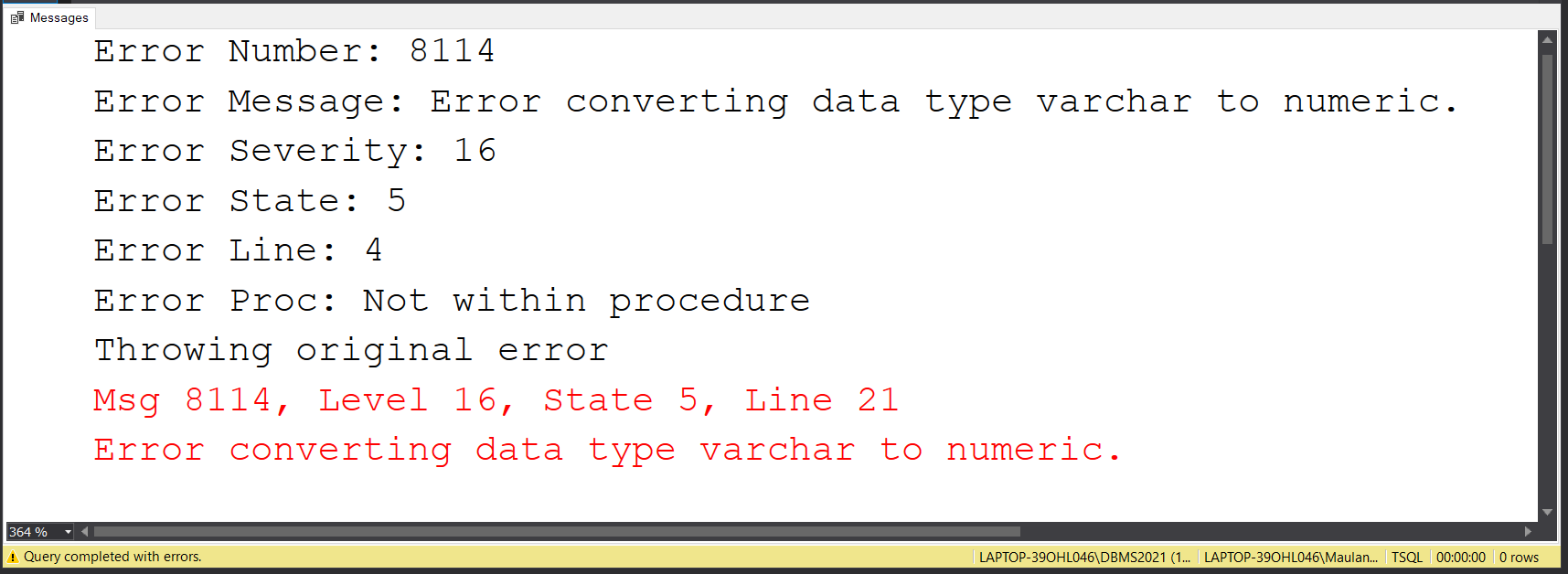
* Pernyataan IF tersebut untuk mengecek apakah nomer error = 8114.
* Apabila kondisi ini terpenuhi, tampilkan pesan “Handling division by zero...” dengan perintah PRINT.
* Jika tidak, tampilkan pesan “Throwing original error...”.
* Terakhir, set nilai variabel @num sebagai ‘A’, lalu eksekusi skrip T-SQL tersebut.

Bandingkan hasilnya dengan tampilan berikut:

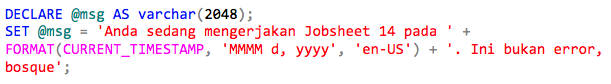


**Jawaban:**





1. Salinlah skrip T-SQL berikut ini:



Lalu, tambahkan pernyataan THROW setelah skrip di atas. Sebagai argumen pertama, isikan 50001, untuk argumen kedua, isikan variabel @msg,sedangkan argumen ketiga, isikan nilai 1.

Bandingkan hasilnya dengan tampilan berikut:



**Jawaban:**

