IMPROVING QUERY PERFORMANCE & ERROR HANDLING

ADVANCED DATABASE

Outline

Faktor Penentu Performa Query

Menulis query yang berkinerja baik

Indexing pada SQL Server

Menampilkan Data Performa Query

Execution Plan

Query Statis

Outline

- Pemrograman TSQL
  - BATCH
  - Variabel
  - Synonyms
  - IF…Else
  - While
- Error Handling
  - TRY…CATCH
  - THROW

Faktor Penentu Performa Query

Menulis query yang berkinerja baik (1)

Tentukan Tipe Data yang tepat

- Sebagai contoh, kita perlu mengetahui kapan kita menggunakan tipe data char atau varchar. Keduanya menampung karakter, bedanya char menyediakan ukuran penyimpanan yang tetap (*fixed-length*), sedangkan varchar menyediakan ukuran penyimpanan sesuai dengan isi data (*variable-length*).

Menulis query yang berkinerja baik (1)

Hanya ambil data yang diperlukan

- Pada query SELECT hindari penggunaan *, hanya gunakan kolom yang dibutuhkan
- Gunakan Klausa where, untuk memfilter baris yang dibutuhkan

Menulis query yang berkinerja baik (3)

- **Batasi Printah *ORDER BY dan GROUP BY***
  - Penggunaan *ORDER BY* yang berfungsi untuk mengurutkan data, ternyata memiliki konsekuensi menambah beban query, karena akan menambah satu proses lagi, yaitu proses sort.
  - Penggunaan *GROUP BY* yang berfungsi untuk mengelompokkan data juga menambah beban query karena menambah proses grouping data
  - gunakan *ORDER BY/GROUP BY* hanya jika benar-benar dibutuhkan oleh aplikasi tersebut

Menulis query yang berkinerja baik (4)

- ***Subquery* Atau *JOIN***
  - Adakalanya sebuah instruksi dapat dituliskan dalam bentuk subquery atau perintah *JOIN*, disarankan memprioritaskan penggunaan *JOIN* karena dalam kasus yang umum akan menghasilkan performa yang lebih cepat.
  - Walaupun demikian, mengolah query merupakan suatu seni, selalu ada kemungkinan ternyata subquery bekerja lebih cepat dibandingkan *JOIN*, misalnya dalam kondisi penggunaan *JOIN* yang terlalu banyak, ataupun logika query yang belum optimal.

Menulis query yang berkinerja baik (5)

- **Kecepatan Akses Operator**
  - *WHERE 1=1* dan *WHERE 0 <> 1* sama-sama merupakan kondisi yang menghasilkan nilai true. Tetapi, dalam hal ini lebih baik digunakan *WHERE 1=1* daripada *WHERE 0 <> 1*.
  - operator **=** diproses lebih cepat dibandingkan dengan operator **<>**. Dari sisi kinerja, urutan operator yang diproses paling cepat adalah:
    - 1. =
    - 2. >, >=, <. <=
    - 3. LIKE
    - 4. <>

Menulis query yang berkinerja baik (6)

- **Membatasi Jumlah Record**
  - Bayangkan apabila akan ditampilkan isi sebuah tabel dengan menggunakan *SELECT*, dan ternyata tabel tersebut memiliki jutaan record yang sangat tidak diharapkan untuk tampil seluruhnya.
  - Skenario yang lebih buruk masih dapat terjadi, yaitu query tersebut diakses oleh ratusan pengguna lain dalam waktu bersamaan. Oleh karena itu, perlu dibatasi jumlah record yang berpotensi mengembalikan record dalam jumlah besar (kecuali memang benar-benar dibutuhkan), pada SQL Server, Anda dapat menggunakan operator *TOP* di dalam perintah *SELECT*.
    - Contohnya : *SELECT TOP 100* nama... (akan menampilkan 100 record teratas field nama).

SQL Server Indexing

Index dapat meningkatkan kecepatan pencarian pada record yang diinginkan. Tetapi, yang harus diperhatikan adalah bagaimana memilih field yang digunakan untuk kunci index, karena tidak semua field memerlukannya.

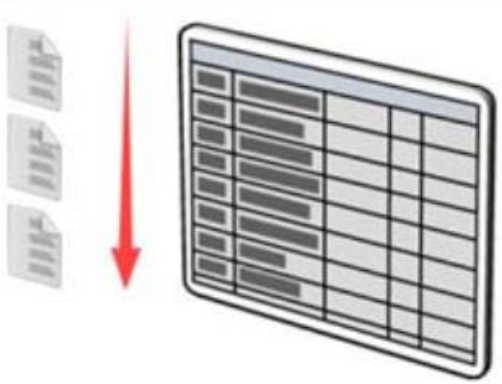
Ibaratnya membaca buku, proses pencarian akan membaca dari awal hingga akhir halaman. pada field yang di-index, pencarian dilakukan secara index scan, atau membaca pada index, tidak langsung pada tabel yang bersangkutan.

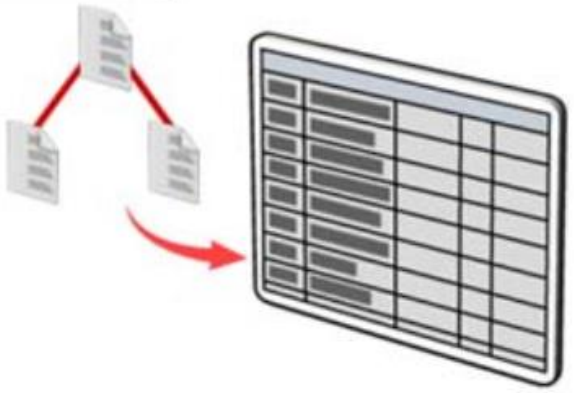
Cara mengindex table:	https://docs.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?view=sql-server-ver15

SQL Server Index

- SQL Server accesses data by using indexes or by scanning all rows in a table
- Indexes also supports ordering operations such as grouping, joining, and ORDER BY clauses

Table scan: SQL Server reads all table rows

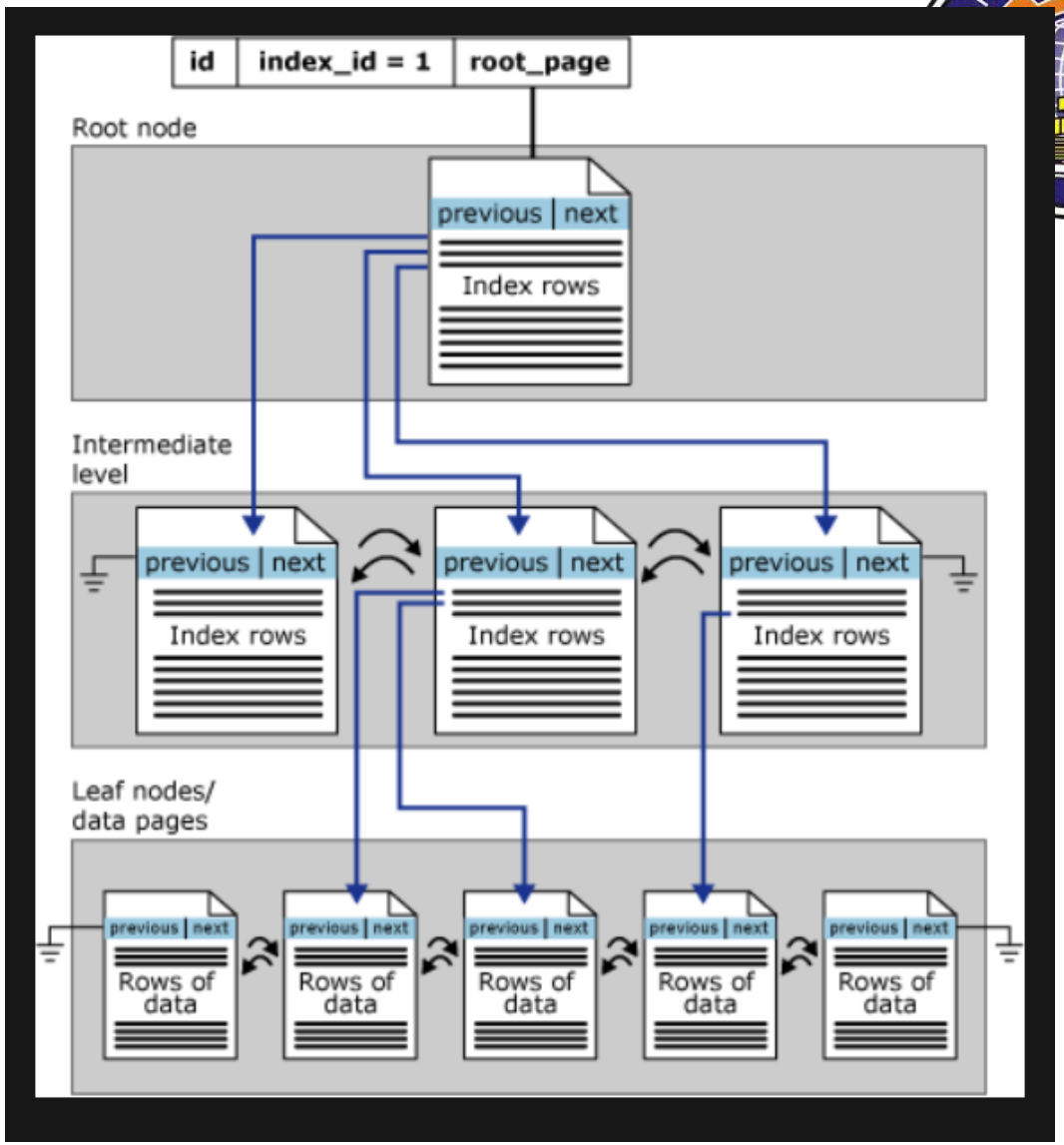




Index seek/scan: SQL Server uses indexes to find rows

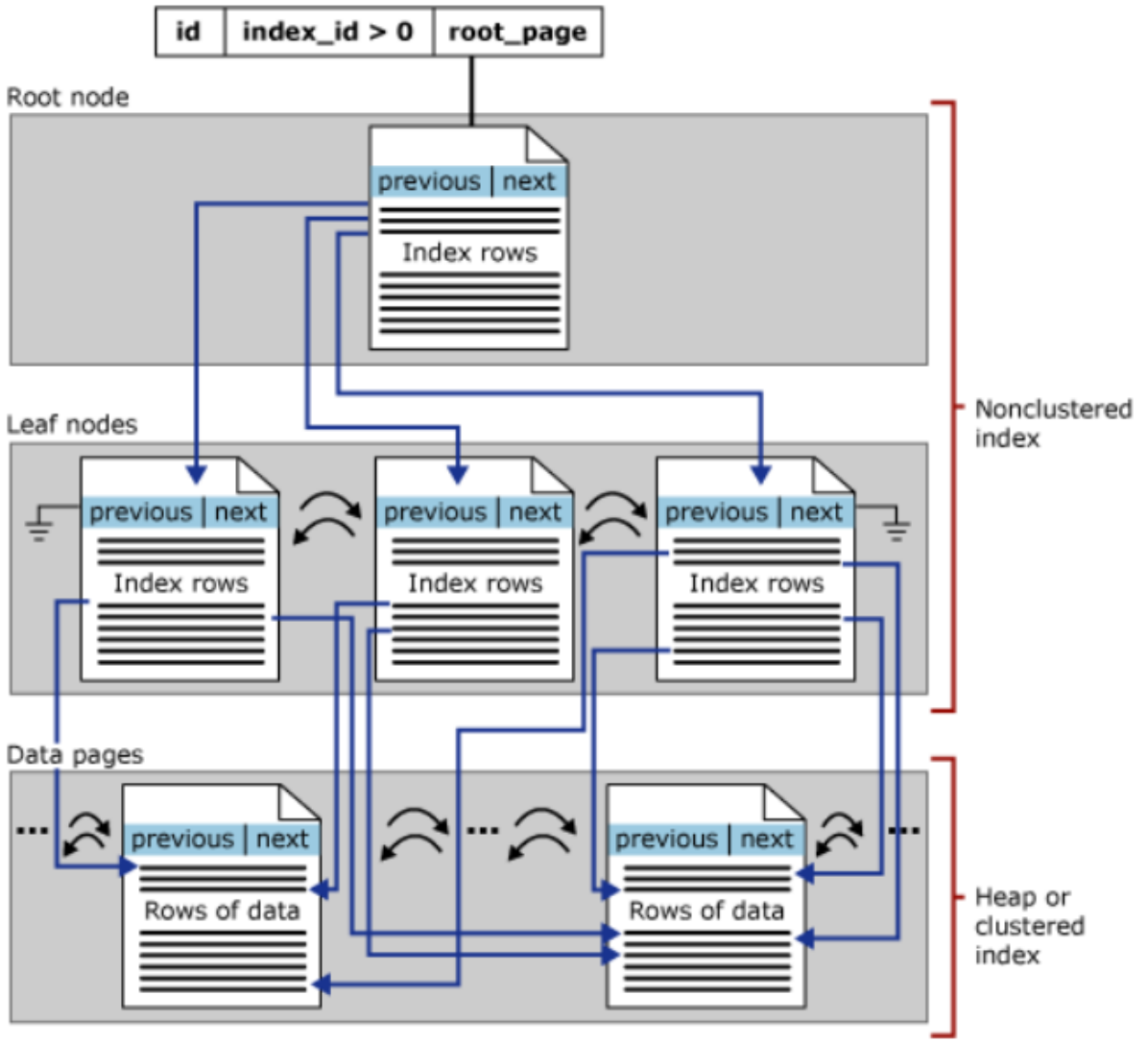
Clustered Index

- pada gambar adalah gambar index clustered, terlihat struktur standard index B+Tree yang mana leaf node nya mengarah langsung ke data.



Non-Clustered Index

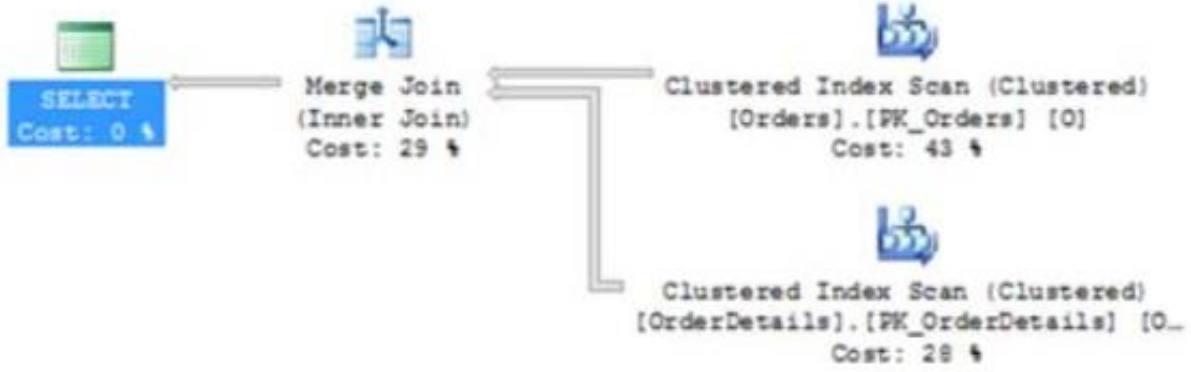
- Dari gambar tersebut terlihat struktur terpisah menjadi 2 yaitu (root node + leaf node) dan data pages.
- Untuk (root node + leaf node) merupakan struktur non clustered indexnya sendiri dan untuk data pages ini bisa langsung data (heap) atau bahkan clustered index.



Menampilkan Data Performa Query

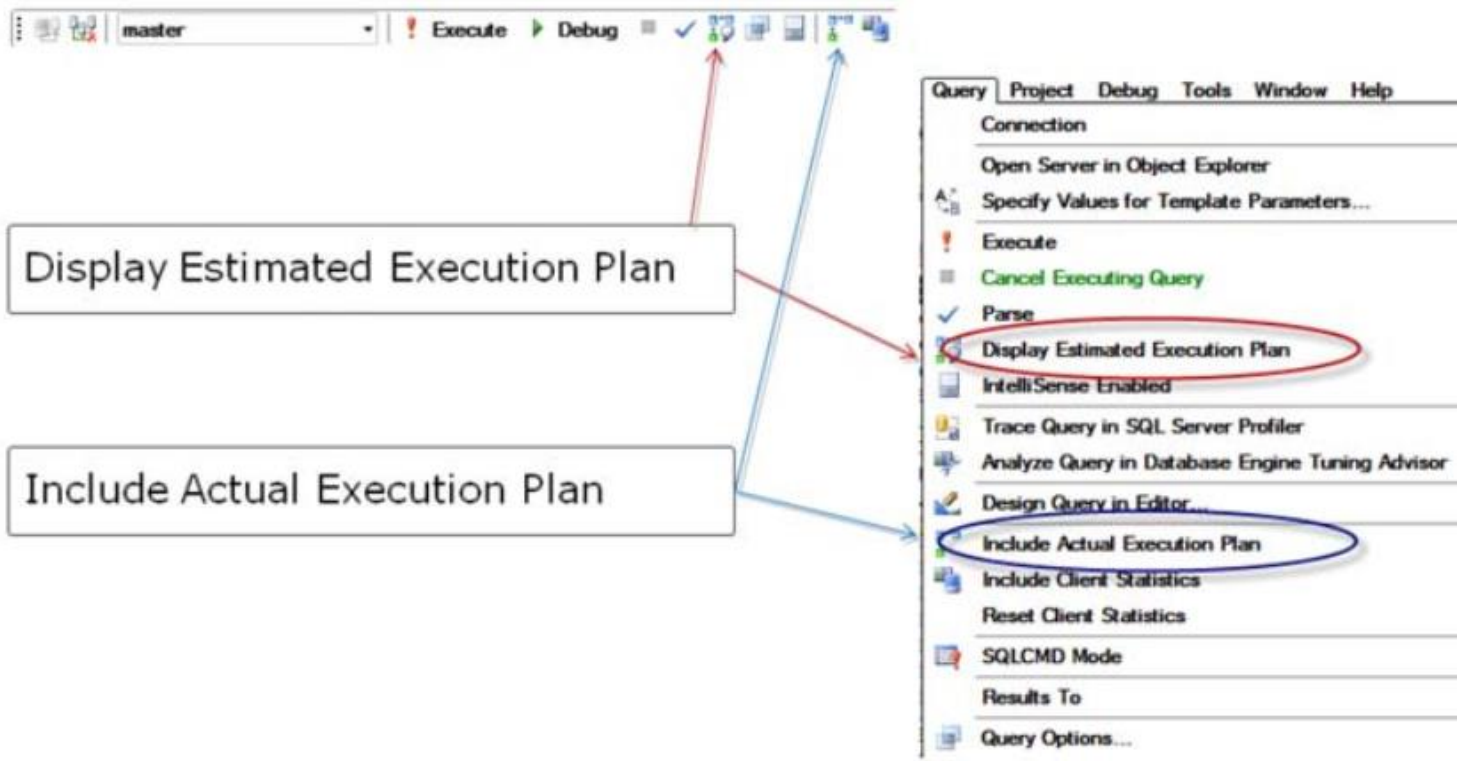
Excecution Plan

- Execution plans graphically represent the methods that SQL Server uses to execute the statements in a T-SQL query
- SSMS provides access to two forms of execution plans:
  - **Estimated** execution plans do not execute the query. Instead, they display the plan that SQL Server would likely use if the query were run.
  - **Actual** execution plans are returned the next time the query is executed. They display the plan that was actually used by SQL Server



Excecution Plan

- Enable execution plan viewers in SSMS



Query Statistics

- SQL Server provides detailed runtime information about the execution of a query
- STATISTICS TIME will show time spent parsing and compiling a query

```
SET STATISTICS TIME ON;
```

- STATISTICS IO will show amount of disk activity generated by a query

```
SET STATISTICS IO ON;
```

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(830 row(s) affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 107 ms.
```

```
(830 row(s) affected)
Table 'Orders'. Scan count 1, logical reads 21, physical reads 3
```

Elemen pemrograman T-SQL

Batch T-SQL

- Kumpulan dari satu atau lebih pernyataan T-SQL yang dikirim ke SQL Server sebagai unit untuk penguraian, optimisasi
- Eksekusi Batch diakhiri dengan klausa GO
- Batch juga batas untuk ruang lingkup variabel
- Beberapa statement (mis., CREATE FUNCTION, CREATE PROCEDURE, CREATE VIEW) tidak dapat digabungkan dengan yang lain dalam Batch yang sama

```
CREATE VIEW HumanResources.EmployeeList
AS
SELECT BusinessEntityID, JobTitle, HireDate, VacationHours
FROM HumanResources.Employee;
GO
```

Batch T-SQL (2)

- Batch dianggap sebagai satu unit
- Jika satu pernyataan di dalam Batch eror, maka semua pernyataan dalam Batch akan ditolak
- Batch dapat berisi kode Error Handling

```
--Valid batch
INSERT INTO Production.UnitMeasure (Name, UnitMeasureCode,
ModifiedDate)
VALUES (N'Square Footage', N'F4', GETDATE()),
       (N'Square Inches', N'I2', GETDATE());
GO
--Invalid batch
INSERT INTO dbo.t1 VALUE(1,2,N'abc');
INSERT INTO dbo.t1 VALUES(2,3,N'def');
GO
```

Variabel pada TSQL

- Objek yang memungkinkan penyimpanan nilai untuk digunakan dalam batch yang sama
- Didefinisikan dengan keyword DECLARE dan dimulai dengan symbol **@**
  - Dideklarasikan dan diinisialisasi dalam pernyataan yang sama
- Bersifat lokal pada batch saat pendeklarasian hingga batch berakhir
-

```
--Declare,initialize, and use a variable
DECLARE @SalesPerson_id INT = 5;
SELECT OrderYear, COUNT(DISTINCT CustomerID) AS CustCount
FROM (
SELECT YEAR(OrderDate) AS OrderYear, CustomerID
FROM Sales.SalesOrderHeader
WHERE SalesPersonID = @SalesPerson_id
) AS DerivedYear
GROUP BY OrderYear;
```

Variabel pada TSQL(2)

- Nilai dapat diisikan dengan perintah SET atau SELECT statement
- SET hanya dapat mengisi nilai satu variabel pada satu waktu. SELECT dapat mengisi banyak variabel sekaligus
- Saat menggunakan SELECT untuk mengisi nilai, pastikan tepat satu baris dihasilkan oleh query

```
--Declare and initialize variables
DECLARE @numrows INT = 3, @catid INT = 2;

--Use variables to pass parameters to
procedure
EXEC Production.ProdsByCategory
      @numrows = @numrows, @catid = @catid;
GO
```

Synonyms

- Alias atau link yang merujuk ke object baik pada SQL Server yang sama maupun pada server yang terhubung
  - Synonyms dapat merujuk ke table, view, procedures dan fungsi
- Dapat digunakan untuk merujuk objek remote atau untuk memberikan nama alternatif ke objek lokal lainnya
- Gunakan command CREATE, ALTER dan DROP untuk menggunakan synonyms

```
-- Create a synonym for the Product table in AdventureWorks CREATE
SYNONYM dbo.MyProduct
FOR AdventureWorks.Production.Product;
GO
-- Query the Product table by using the synonym.
SELECT ProductID, Name
FROM MyProduct
WHERE ProductID < 5;
GO
```

TSQL Control of Flow

TSQL Control of Flow

- Elemen TSQL tambahan yang mengontrol aliran dan eksekusi statement T-SQL dalam batch, stored procedure, dan fungsi
- Elemen control of flow memungkinkan pengguna untuk menentukan statement yang perlu dieksekusi dalam urutan yang ditentukan
- Defaultnya, statement dieksekusi secara berurutan, namun bisa juga menggunakan IF ... ELSE, BEGIN ... END, WHILE, RETURN, dan lainnya untuk mengontrol aliran batch atau stored procedure

```
IF OBJECT_ID (‘Production.Product', 'U') IS NOT NULL
   PRINT 'I am here and contain data, so don’t delete me’
```

IF…Else

- IF…ELSE menggunakan kondisi untuk menentukan aliran kode
  - Kode dalam blok IF dieksekusi jika kondisi bernilai true
  - Kode dalam blok ELSE dieksekusi jika kondisi bernilai FALSE atau UNKNOWN
- Bisa menggunakan operator EXIST untuk seleksi kondisi

```
IF OBJECT_ID (‘Production.Product', 'U') IS NOT NULL
   PRINT 'I am here and contain data, so don’t delete me’
ELSE
    PRINT ‘Table not found, so feel free to create one’
GO
```

Perulangan WHILE

- Statement dieksekusi di blok WHILE selama kondisi terpenuhi / bernilai TRUE dan tidak berhenti mengeksekusi sampai kondisi tidak terpenuhi atau bernilai FALSE atau UNKNOWN
- Eksekusi dapat diubah dengan menggunakan BREAK atau CONTINUE

```
DECLARE @BusinessEntID AS INT = 1, @Title AS
NVARCHAR(50);
WHILE @BusinessEntID <=10
    BEGIN
        SELECT @Title = JobTitle FROM
HumanResources.Employee
                WHERE BusinessEntityID =
@BusinessEntID;
        PRINT @Title;
        SET @BusinessEntID += 1;
    END;
```

Error handling

Handling Exception

- Exception handling memungkinkan respons terpusat terhadap runtime error
  - TRY digunakan untuk menjalankan blok statement dan CATCH untuk menangkap setiap error yang terjadi
  - Eksekusi bergerak ke blok statement CATCH ketika error terjadi
  - Tidak perlu memeriksa setiap statement untuk melihat apakah ada error yang terjadi

Error yang sering terjadi

Property	Function to Query	Description
Number	ERROR_NUMBER	Unique number assigned to the error
Message	ERROR_MESSAGE	Error message text
Severity	ERROR_SEVERITY	Severity class (1-25)
Procedure Name	ERROR_PROCEDURE	Name of the procedure or trigger that raised the error
Line Number	ERROR_LINE	Number of the line that raised the error in the batch, procedure, trigger, or function

Demo (fungsi error object)

```
BEGIN TRY
	SELECT 1/0; --generate error
END TRY
BEGIN CATCH
	SELECT
		ERROR_NUMBER() AS errnum,
		ERROR_MESSAGE() AS errmsg,
		ERROR_SEVERITY() AS errsev,
		ERROR_PROCEDURE() AS errproc,
		ERROR_LINE() AS errline;
END CATCH;
```

```
errnum      errmsg                                  errsev      errproc errline
----------- --------------------------------------- ----------- ------- -----------
8134        Divide by zero error encountered. 16          NULL    2
```

Membuat blok try catch

- Blok TRY yang didefinisikan menggunakan statement BEGIN TRY ... END TRY
  - Tempatkan semua kode yang mungkin menimbulkan error
  - Tidak ada kode yang ditempatkan antara END TRY dan BEGIN CATCH
  - Blok TRY dan CATCH bisa dibuat bersarang
- Blok CATCH yang didefinisikan menggunakan statement BEGIN CATCH ... END CATCH
  - Eksekusi berpindah ke blok CATCH ketika terjadi error yang ditangkap dalam blok TRY

Membuat blok try catch

```
BEGIN TRY
   -- Generate a divide-by-zero error.
 SELECT 1/0;
END TRY
BEGIN CATCH
SELECT
       ERROR_NUMBER() AS ErrorNumber
      ,ERROR_SEVERITY() AS ErrorSeverity
      ,ERROR_STATE() AS ErrorState
      ,ERROR_PROCEDURE() AS ErrorProcedure
      ,ERROR_LINE() AS ErrorLine
      ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

ErrorNumber	ErrorSeverity	ErrorState	ErrorProcedure	ErrorLine	ErrorMessage
8134	16	1	NULL	3	Divide by zero error encountered.

Blok try catch

- Tidak semua error dapat ditangkap oleh TRY / CATCH

```
BEGIN TRY
   -- Table does not exist; object name resolution
   -- error not caught.
SELECT * FROM IDontExist;
END TRY
BEGIN CATCH
   SELECT
       ERROR_NUMBER() AS ErrorNumber
      ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

```
Msg 208, Level 16, State 1, Line 4
Invalid object name ‘IDontExist'.
```

Statement THROW

- SQL Server menyediakan statement THROW
  - Pada versi sebelumnya menggunakan statement RAISERROR
  - Tidak perlu mendefinisikan error dalam tabel sys.messages
- THROW memungkinkan pilihan saat menangani error:
  - Menangani error secara spesifik di blok CATCH lokal
  - Mengirimkan error ke proses yang lain
- Gunakan THROW:
  - Dengan parameter untuk mengirimkan error yang ditentukan pengguna
  - Tanpa parameter untuk menggunakan error yang asli(harus berada dalam blok CATCH)

```
BEGIN TRY
        SELECT 100/0 AS 'Problem';
END TRY
BEGIN CATCH
        PRINT 'Code inside CATCH is beginning'
        PRINT 'MyError: ' + CAST(ERROR_NUMBER()
                AS VARCHAR(255));
        THROW;
END CATCH
```

Demo (user defined error messages)

- Metode 1 penggunaan THROW

```
THROW 55000, 'The object does not exist.', 1;
```

```
Msg 55000, Level 16, State 1, Line 1
The object does not exist.
```

Demo (original error messages)

- Metode 2 penggunaan THROW

```
BEGIN TRY
   SELECT 100/0;--generate an error
END TRY
BEGIN CATCH
   PRINT 'Code inside CATCH is beginning'
   PRINT 'Error: ' + CAST(ERROR_NUMBER() AS VARCHAR(255));
   THROW;
END CATCH
```

-
```
Code inside CATCH is beginning
Error: 8134
Msg 8134, Level 16, State 1, Line 2
Divide by zero error encountered.
```

References

- https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15
- https://msdn.microsoft.com/en-us/library/ms177443.aspx
- https://msdn.microsoft.com/en-us/library/ms177484.aspx
- Querying Microsoft® SQL Server® 2012
- http://dinus.ac.id/repository/docs/ajar/optimasi-SQL