1. _ Indexes_ Basics and Best Practices

${toc}
Index are method by which sql server sort and organize the rows in the table to make it easier to find the desire row

What out it sql will perform something called `table scane` in which sql will scan the whole table till it find the desire row

With index sql server can find the desire row more quickly

We call the operation where sql server will check the index to find the wanted row `seek`

Index types

there are two type

- cluster index
- non-clustered index

Clustered Index

In this method sql server will sort the actual data in table , they can be only one clustered index

But what if we already sort the table by personal id column , and we want to sorting it again base on family name
While we cannot use another cluster index we cannot have two index sorting the actual table this will create mismatch and conflict

in this case we will use non-clustered index

Non-Clustered Index

Non-clustered index create separate object and it will hold sorted column we desire

So if we created clustered-index on family name we get copy if a subset of column from table sorted by those copied column

This give us freedom to have many clustered index

For instead we can have copy of column sorted by age , another one by salary and so on

Advantages and Disadvantages of both types

The clustered index essentially is the table. It contains all of the columns of the table rather than just a subset of columns. This means that queries that use the clustered index to perform their seek will be able to return any and every column they need as they are all stored on the index. On the downside, any updates to the column(s) that make up the clustered index may cause the entire row to have to be moved to another place in the clustered index. On a wide table (many columns) that can be a very resource intensive operation.

A non-clustered index contains only a subset of columns so a query that uses it for a seek will quickly find the row(s) desired, but may have to make a second query, called a key lookup, against the clustered index to return any columns requested that aren't part of the index. The idea of a second query happening sounds scary, but that all happens automatically within the SQL Server engine and requires no effort on the part of the developer.

Since the non-clustered index has fewer columns than the clustered index, updates to it that require the row copy to be moved within the non-clustered index are not as resource intensive.

Use Cases for SQL Server Clustered and Non-Clustered Indexes

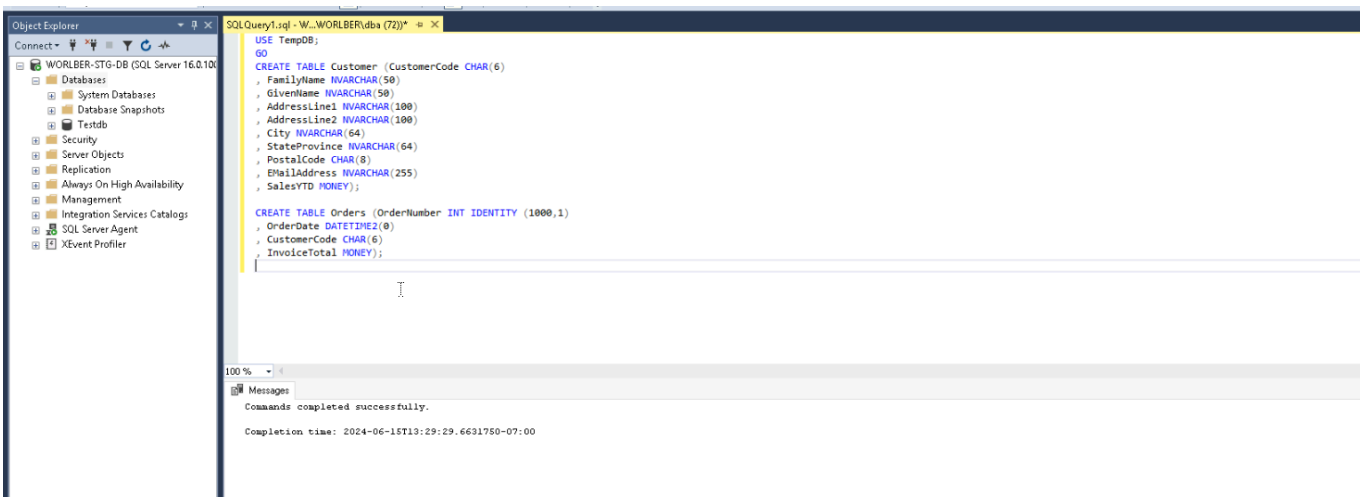
- Every table must have a clustered index.
- You can only have one clustered index, so choose the column you want to sort carefully.
- If you need to sort other columns, use non-clustered indexes.
- When selecting a column for a clustered index, ensure the column values are unique and unlikely to change in the future.
- When creating a non-clustered index, choose columns that are likely to be searched.

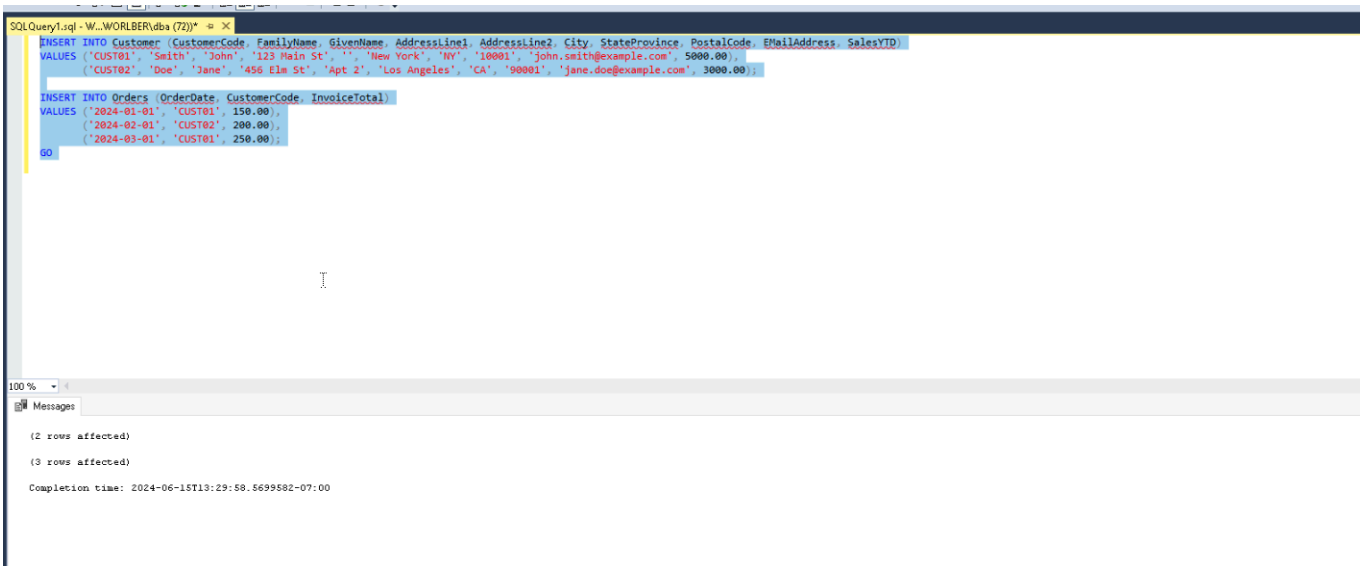
Simulation

- We will create two tables in `tempdb`, then create a clustered index and a non-clustered index on them. Before that, we will run some queries to observe how indexes improve the speed of SQL Server.

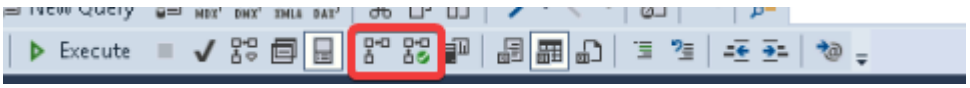
```
USE TempDB;
GO
CREATE TABLE Customer (CustomerCode CHAR(6)
```

```
, FamilyName NVARCHAR(50)
, GivenName NVARCHAR(50)
, AddressLine1 NVARCHAR(100)
, AddressLine2 NVARCHAR(100)
, City NVARCHAR(64)
, StateProvince NVARCHAR(64)
, PostalCode CHAR(8)
, EMailAddress NVARCHAR(255)
, SalesYTD MONEY);

CREATE TABLE Orders (OrderNumber INT IDENTITY (1000,1)
, OrderDate DATETIME2(0)
, CustomerCode CHAR(6)
, InvoiceTotal MONEY);


INSERT INTO Customer (CustomerCode, FamilyName, GivenName, AddressLine1,
AddressLine2, City, StateProvince, PostalCode, EMailAddress, SalesYTD)
VALUES ('CUST01', 'Smith', 'John', '123 Main St', '', 'New York', 'NY',
'10001', 'john.smith@example.com', 5000.00),
       ('CUST02', 'Doe', 'Jane', '456 Elm St', 'Apt 2', 'Los Angeles', 'CA',
'90001', 'jane.doe@example.com', 3000.00);

INSERT INTO Orders (OrderDate, CustomerCode, InvoiceTotal)
VALUES ('2024-01-01', 'CUST01', 150.00),
       ('2024-02-01', 'CUST02', 200.00),
       ('2024-03-01', 'CUST01', 250.00);
GO
```

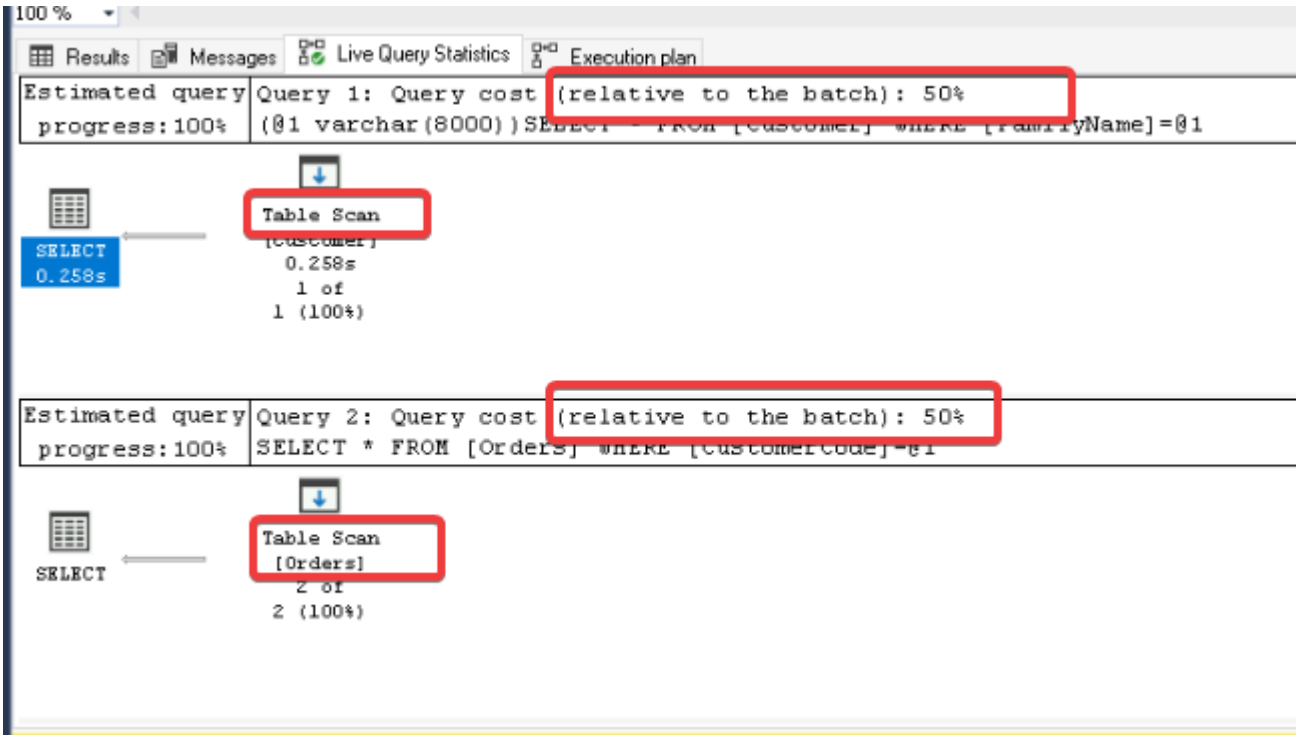




now we will run below query make sure to toggle `include live execution plan`



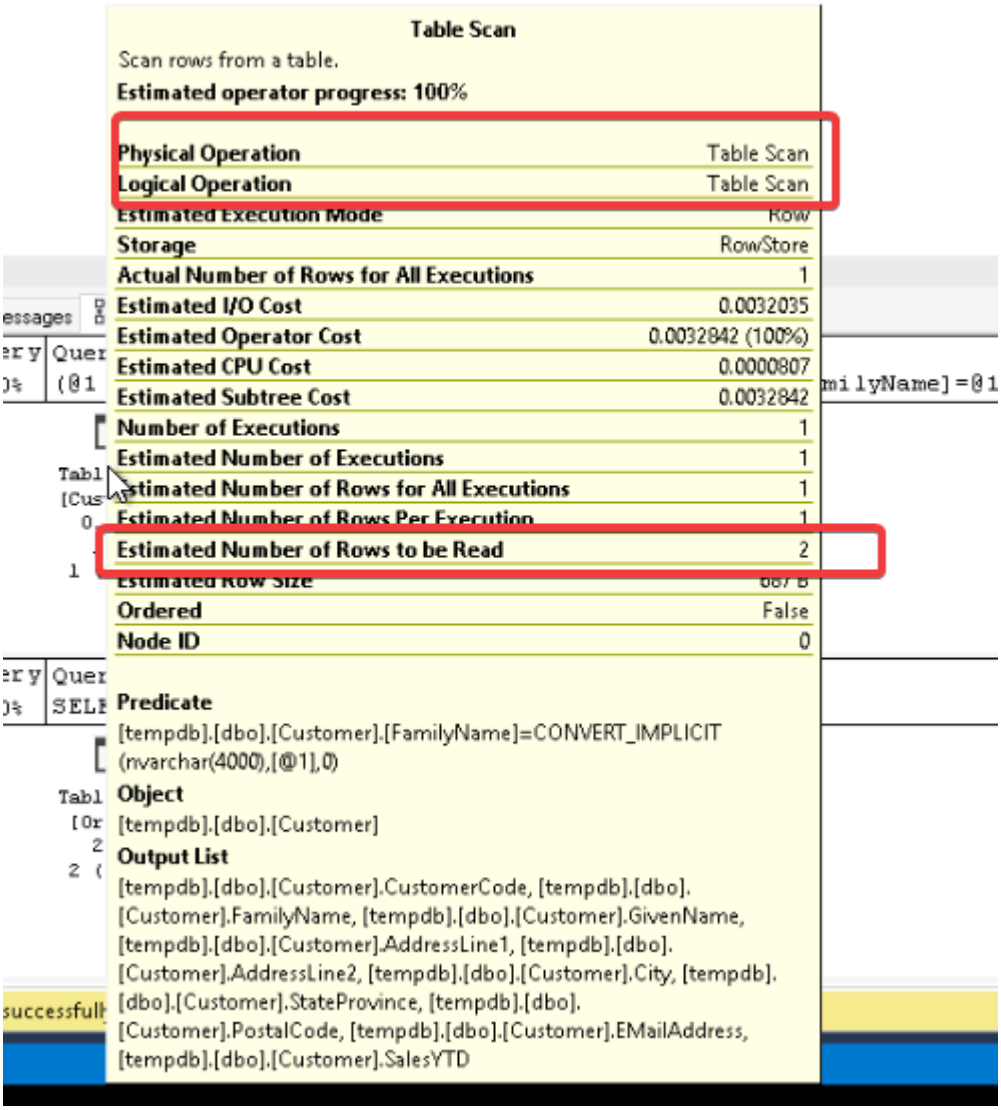

```
-- Step 3: Run Test Queries Without Indexes
-- Query to observe performance without indexes
SELECT * FROM Customer WHERE FamilyName = 'Smith';
SELECT * FROM Orders WHERE CustomerCode = 'CUST01';
GO
```



You can see from result there is table scan done till we find 'smith' for example which not good thing

This is small table but in case the table has let's say million of column this will cause CPU usage to increasing reducing performance of sql server

Also if you move curser to table scan you can see more details



you can see that Physical Operation: **Table Scan** Logical Operation: **Table Scan** which not good indecation

also if you see Estimated Number of Rows to be Read equal 2 , the table has two row and query need to read both row in order to retrive the data

creating index

we will first create index but first i will show the syntax for creating clustred index :

`CREATE UNIQUE CLUSTERED INDEX [index name you want ] ON [table-name] (column name );`

`CREATE UNIQUE CLUSTERED INDEX CX_CUSTOMER ON Customer (CustomerCode);`

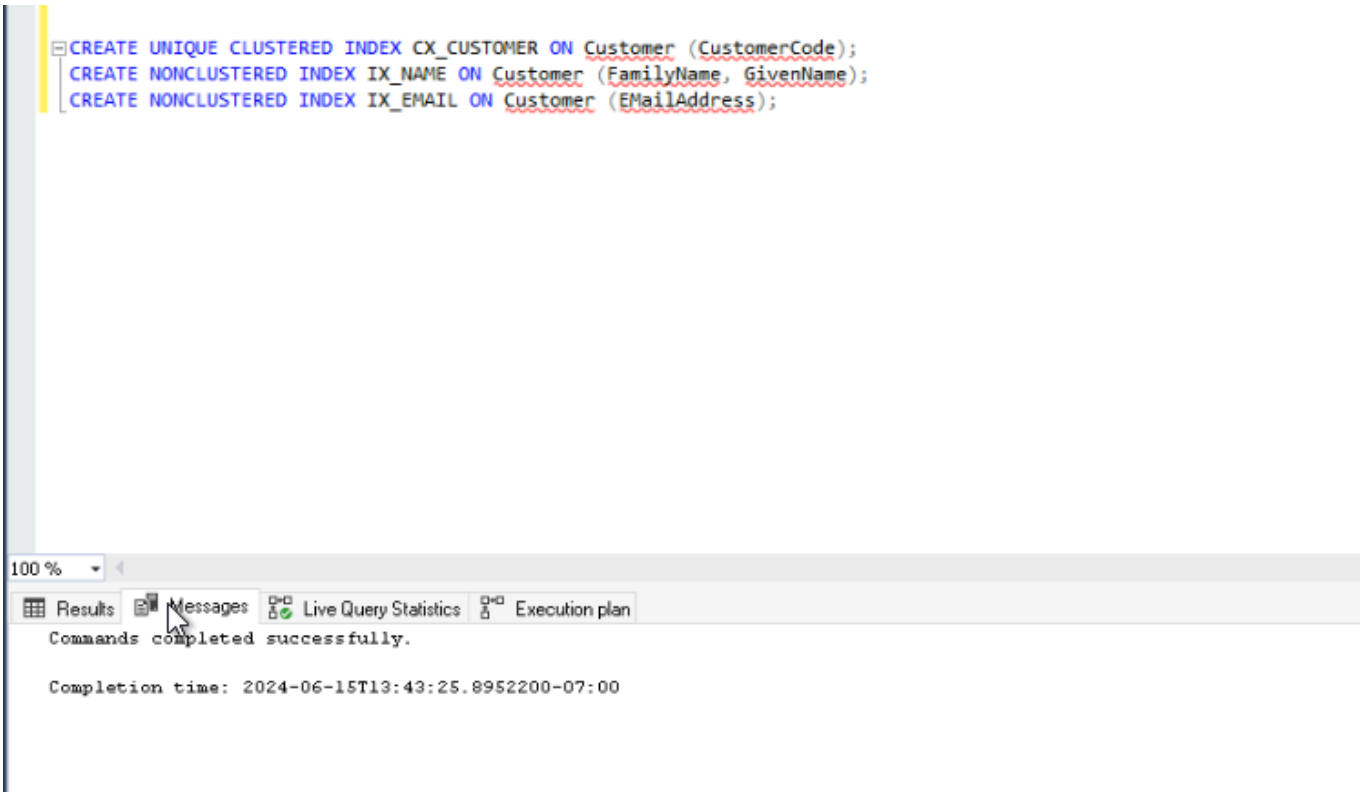
will show the syntax for creating non-clustred index :

`CREATE NONCLUSTERED INDEX [any name for index]ON [table-name] (column 1 , comn 2 )`

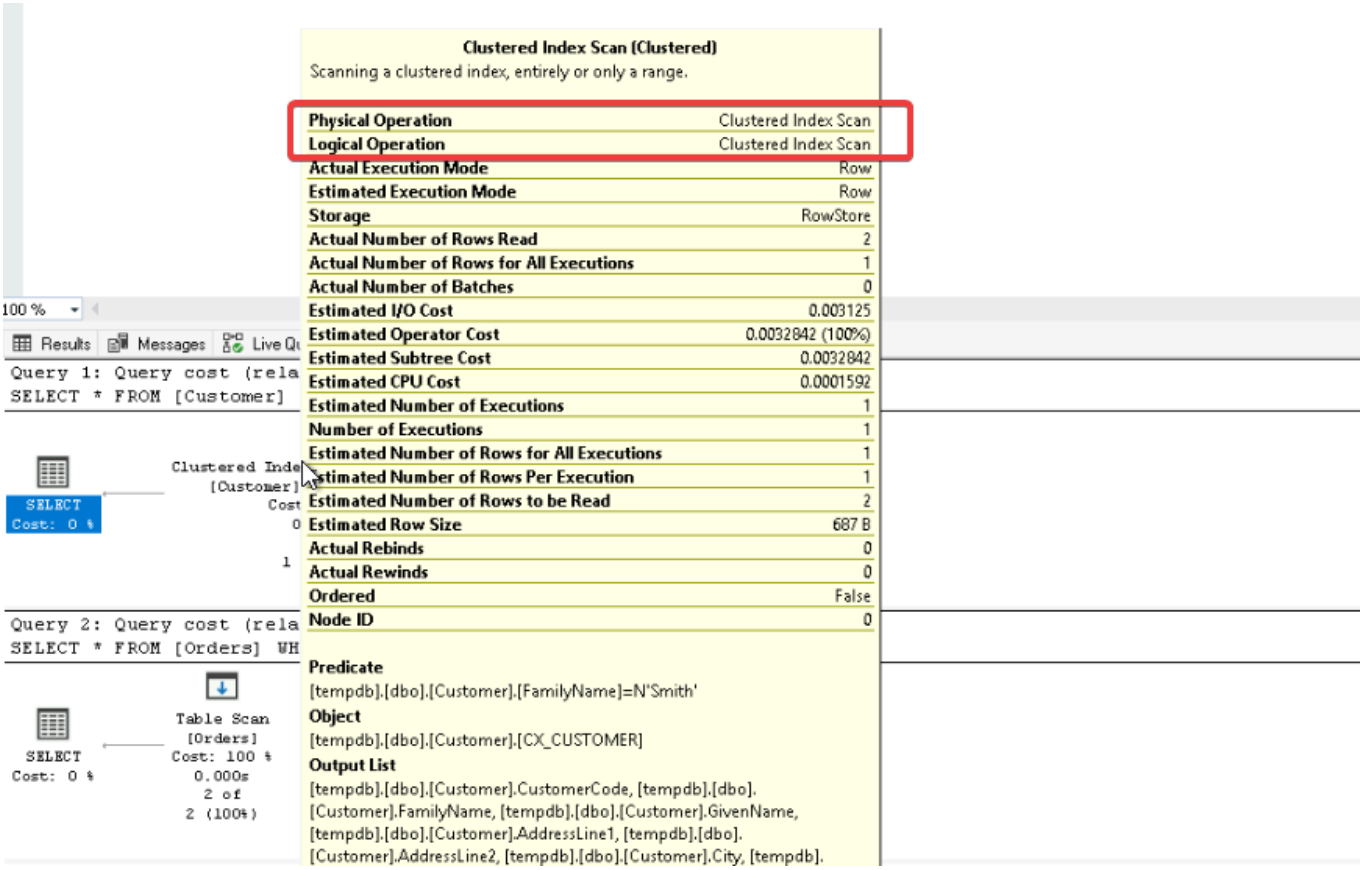
`CREATE NONCLUSTERED INDEX IX_NAME ON Customer (FamilyName, GivenName);`

full script

```
CREATE UNIQUE CLUSTERED INDEX CX_CUSTOMER ON Customer (CustomerCode);
CREATE NONCLUSTERED INDEX IX_NAME ON Customer (FamilyName, GivenName);
CREATE NONCLUSTERED INDEX IX_EMAIL ON Customer (EMailAddress);
```



now we will run the query and see the improvement



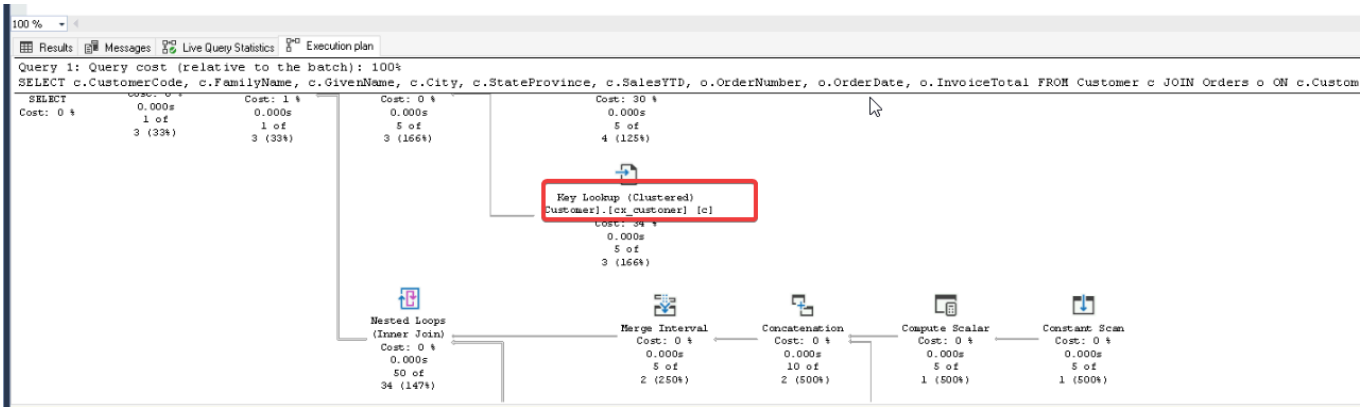
The execution plan shows a "Clustered Index Scan" operation instead of a "Table Scan." This indicates that the query is now scanning the clustered index CX_CUSTOMER on the Customer table

Using a clustered index scan is generally more efficient than a full table scan, especially for larger tables.

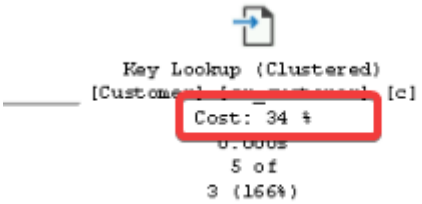
we are looking to get to point where query will do something called **Index Seek** which better preform than index scan

key lookup

the query execution plan showed that there is keylookup done on index



This action of going back and forward will cause the query to run slower
As you can see key lookup action cost is very high

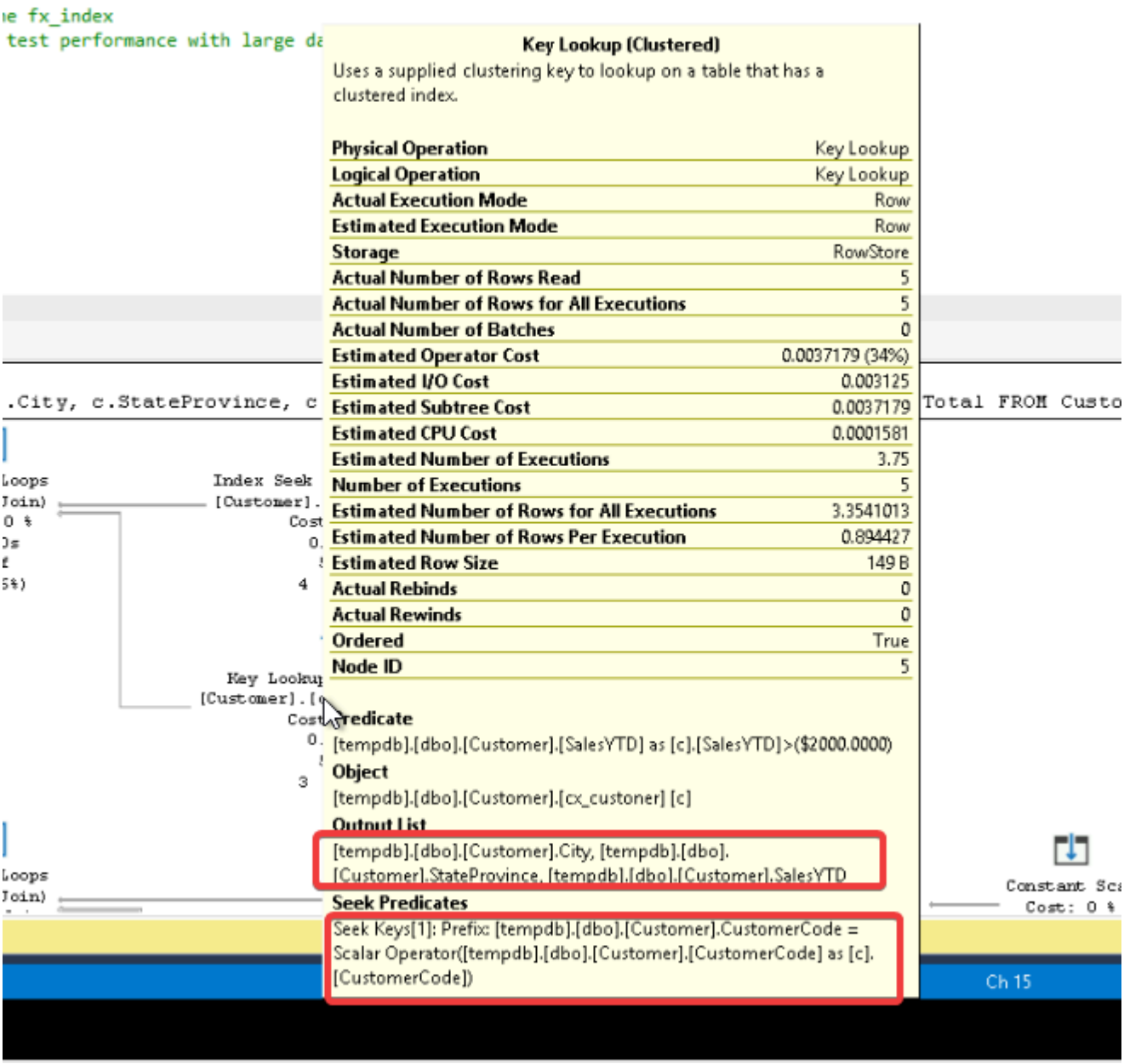


Imagine a query that runs thousands of times per minute that includes one or more key lookups. This can result in tremendous overhead

Having key lookup will indicate that our index missing some column

And need to be recheck non clustered index

If we highly the key lookup we will find



The output list differs from the seek predicate, indicating that StateProvince and SalesYTD columns are being fetched from the clustered index. To optimize this, we will update our non-clustered index to include these columns. This requires dropping the existing index and recreating it with the added columns.

we will drop the index and recreate it again

```
CREATE NONCLUSTERED INDEX fx_index ON Customer (FamilyName, GivenName) INCLUDE (StateProvince, SalesYTD); GO
```

`INCLUDE` clause adds `StateProvince` and `SalesYTD` as included columns. Included columns are not part of the index key but are included in the `leaf` level of the index to cover queries that require these columns.

Index Maintenance and Optimization

A significant task in index maintenance is reducing fragmentation. As data is added, deleted, or modified, fragmentation can occur. Our goal is to keep the data contiguous (in sequence) to reduce I/O operations.

Below is an example of a zero-fragmented index.

Page 1
Row 1
Row 2
Row 3
Row 4

Page 2
Row 5
Row 6
Row 7
Row 8

Below is an example of a index with pages and rows that are out of order.

Page 1
Row 1
Row 2

Page 3
Row 5
Row 6
Row 7
Row 8

Page 2
Row 3
Row 4

Page 4
Row 9
Row 10

As the matter of fact, they're actually in order for now but let's say that we need to insert a new record into this table. The SQL Server would look for an empty spot and the updated pages would look like this:
This below show clear fragmented index

Page 1
Row 1
Row 2
Row 11
Row 12

Page 3
Row 5
Row 6
Row 7
Row 8

Page 2
Row 13
Row 3
Row 14
Row 4

Page 4
Row 15
Row 9
Row 10
Row 16

Rebuild and Reorganize

to solve this fragmentation and and also to avoid we have to lean mor about rebuild and reorganize

rebuild it create brand new index , rebuild is cleaner and easier
rebuild will sometime cause locking so its best practices to do during of peak hour
rebuild consume more resource

reorganize : it fixes physical order and compact the pages , reorganize causes less locking

general idea is is rebuild when possible and reorganize when fragmentation is low

Microsoft suggest to rebuild when fragmentation is at 30% our high

and reorganize it between 5 % and 30 %

Rebuild/Reorganize indexes weekly and use maintenance plan to automate the process
locate and remove unuse index

Manual Rebuild and Reorganize

If in case we need to manually rebuild and reorganize and index we can do that by using we can use ola hallengren script and create job that will do the task , we can also schedule the job for weakly maintenance of the index

Index and Statistics Maintenance

First create stored procedure

By copy content of the script and executing on sql server

note you have to execute commandExecute and CommandLog First before executing IndexOptimize.Sql script

All the script must run on master database

The script will create stored procedure that we can use either to directly run to rebuild and reorganize or to schedule it in job

Run Stored Procedure for Rebuild and Reorganize

Rebuild or Reorganize All Indexes with Fragmentation on All User Databases

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium =
'INDEX_REORGANIZE,INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationHigh = 'INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationLevel1 = 5,
@FragmentationLevel2 = 30
```

Rebuild or reorganize all indexes with fragmentation and update modified statistics on all user databases

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium =
'INDEX_REORGANIZE,INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationHigh = 'INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationLevel1 = 5,
@FragmentationLevel2 = 30,
@UpdateStatistics = 'ALL',
@OnlyModifiedStatistics = 'Y'
```

Update statistics on all user databases

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium = NULL,
@FragmentationHigh = NULL,
@UpdateStatistics = 'ALL'
```

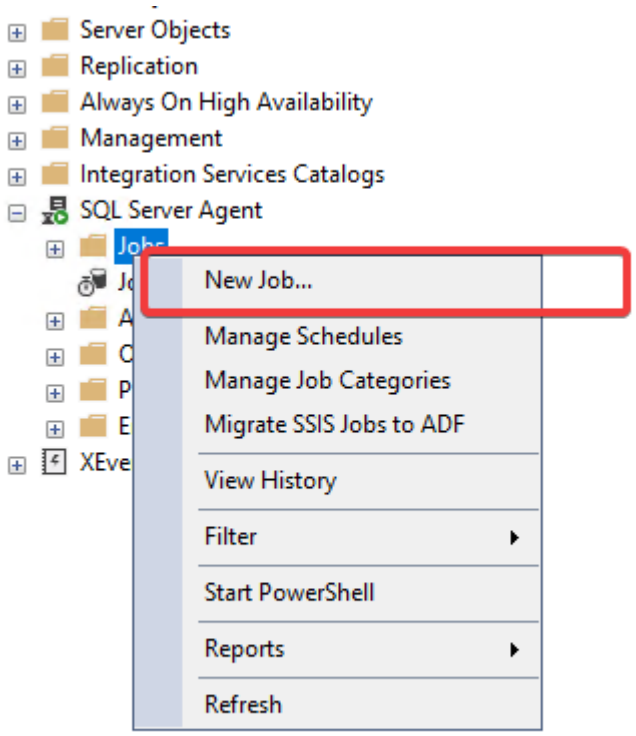
more examble can be find in the following link https://ola.hallengren.com/sql-server-index-and-statistics-maintenance.html

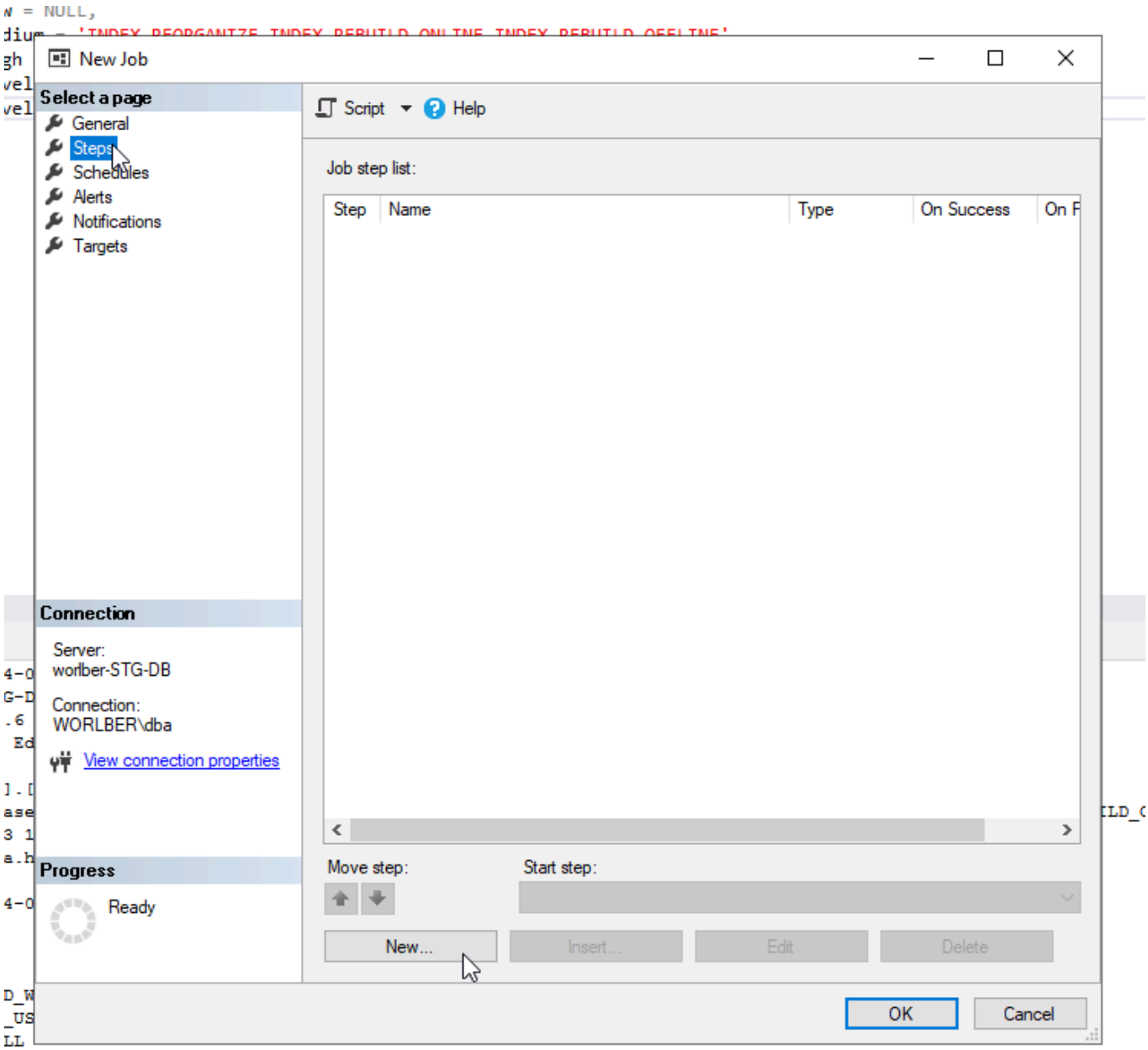
Schedule Job

We will use this script

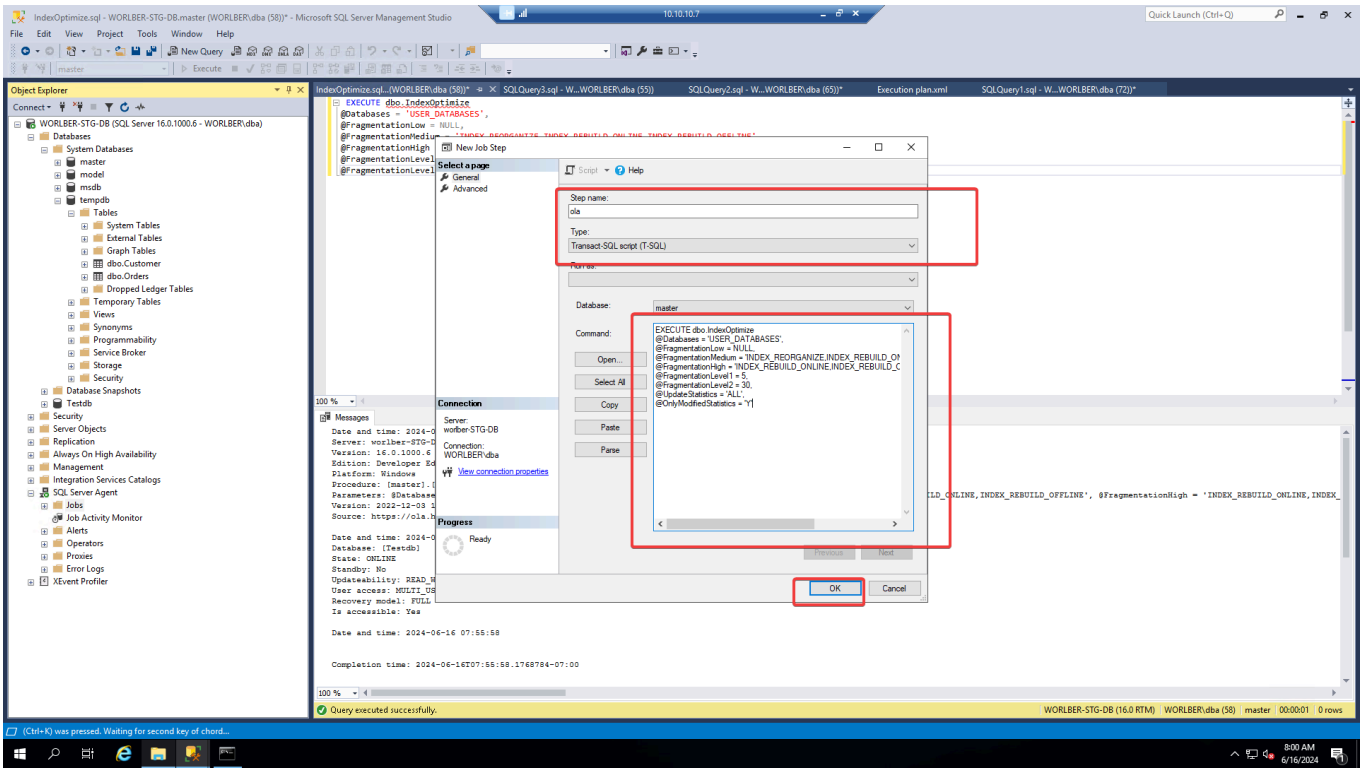
```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium =
'INDEX_REORGANIZE,INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationHigh = 'INDEX_REBUILD_ONLINE,INDEX_REBUILD_OFFLINE',
@FragmentationLevel1 = 5,
@FragmentationLevel2 = 30,
@UpdateStatistics = 'ALL',
@OnlyModifiedStatistics = 'Y'
```

Now we will create new job

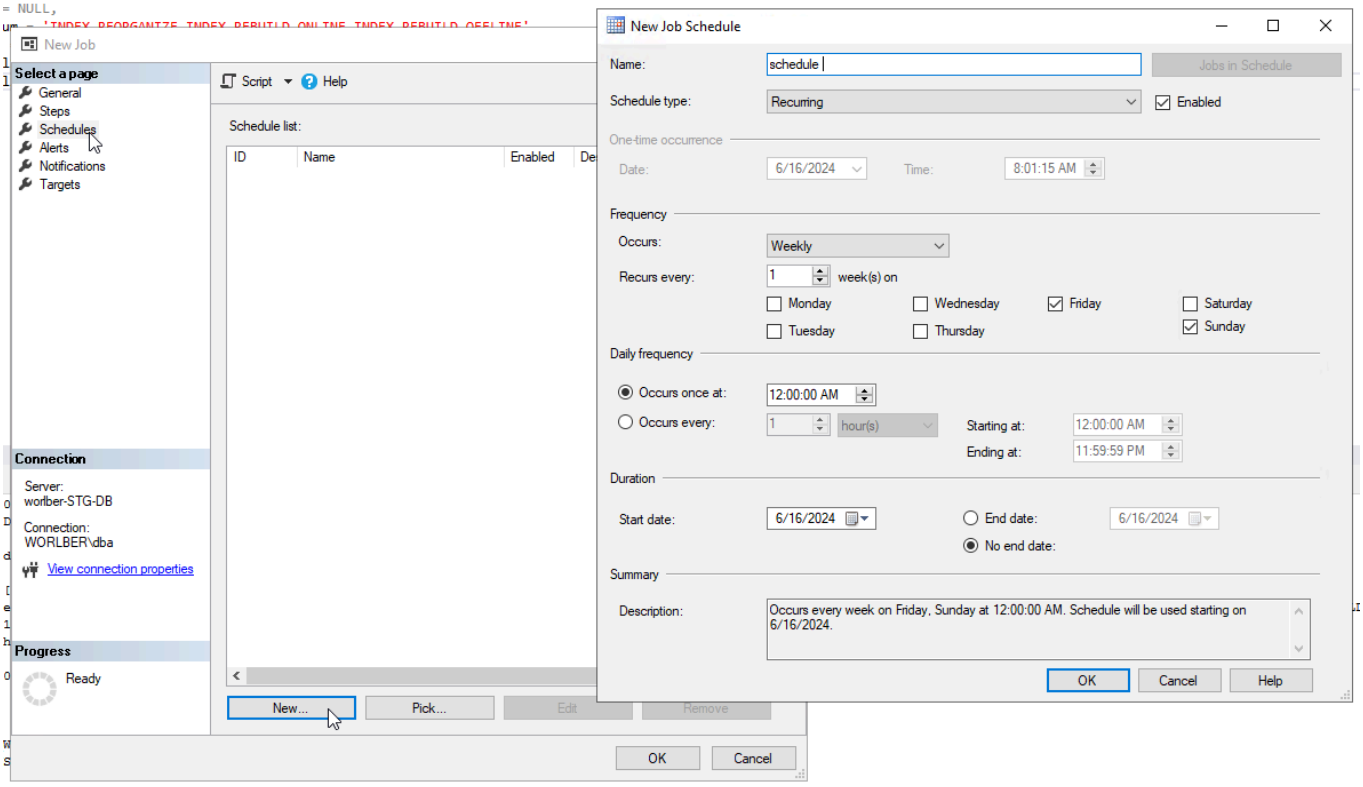




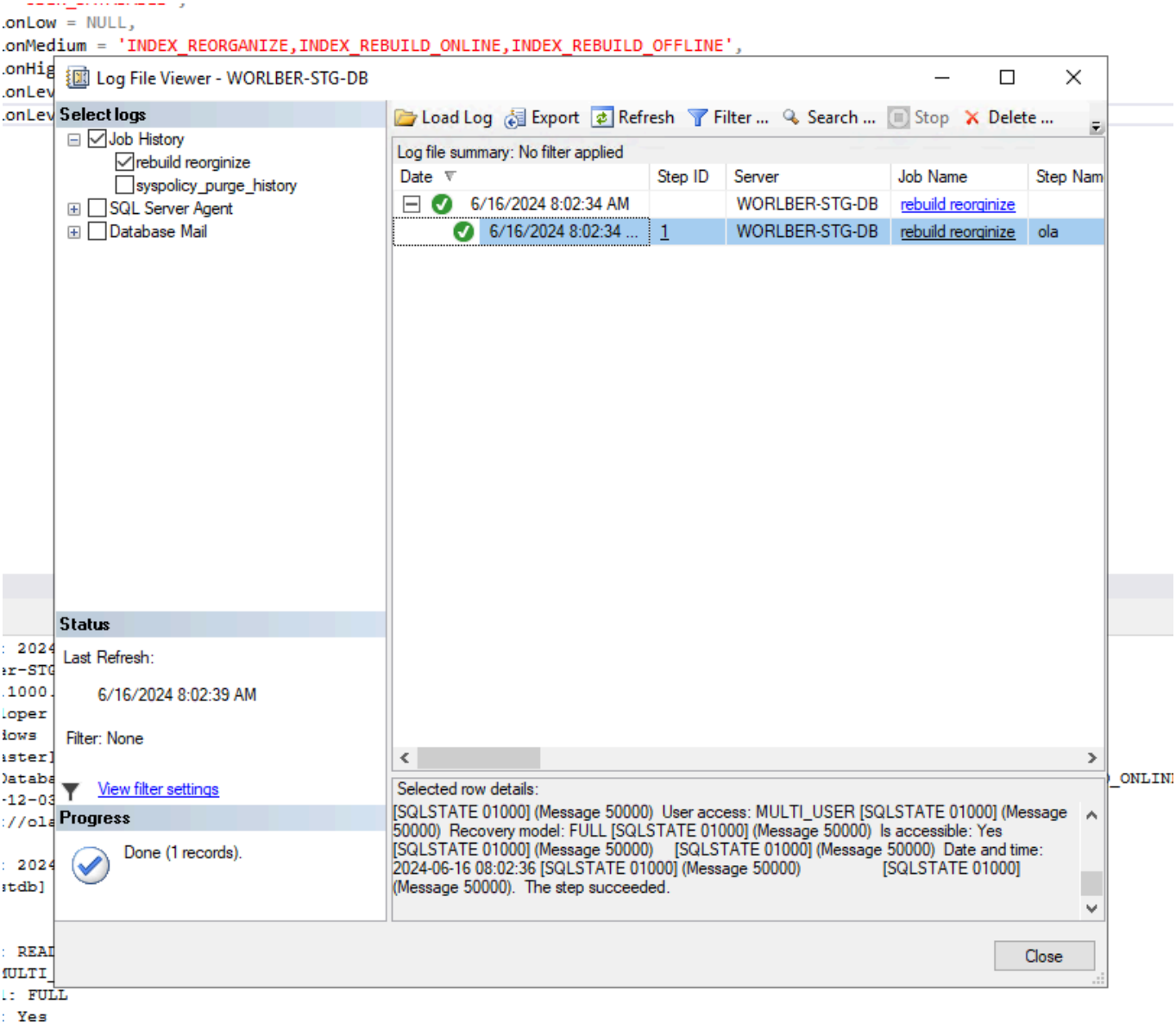
Past the scrip make sure that it uses master database then save



Click ok then make schedule for the job as follow



Then click ok then start job to see if it run successfully



Dmv for Index

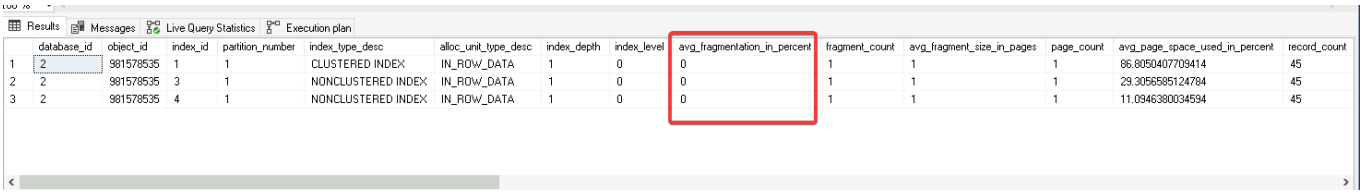
Get Index Fragmentation

`sys.dm_db_index_physical_stats` is dmv that will show fragmentation for index in specific table

syntax :

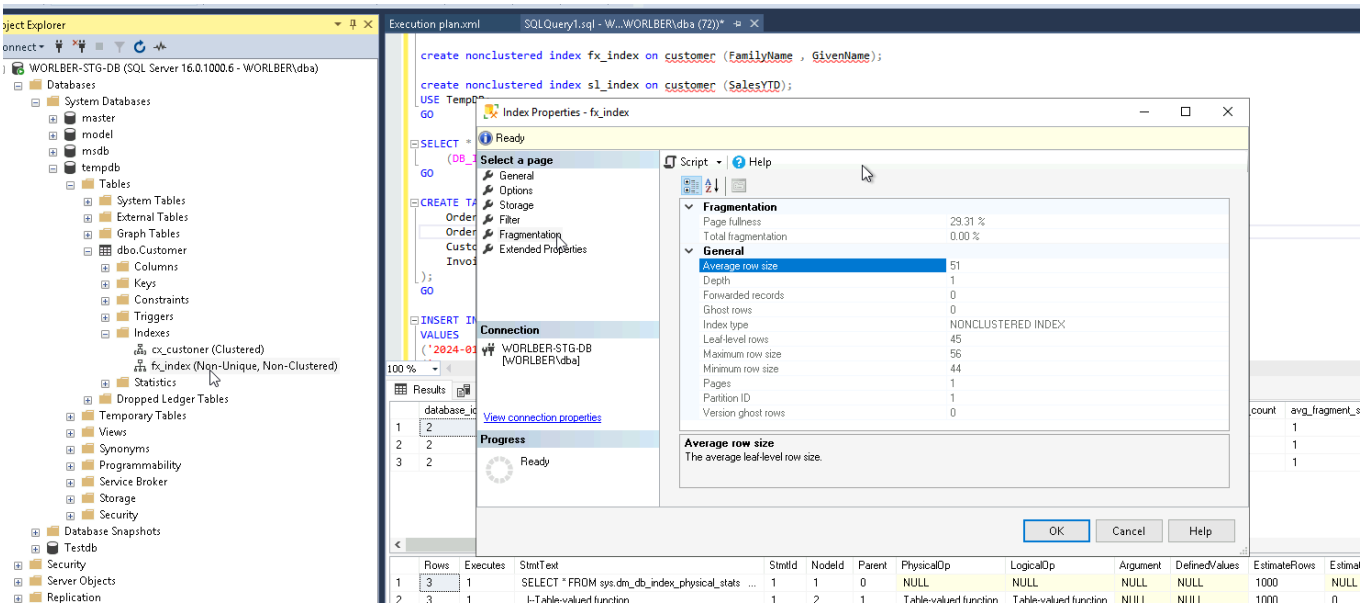
```
SELECT * FROM sys.dm_db_index_physical_stats (DB_ID(N'database - name '), OBJECT_ID(N'table name '), NULL, NULL , 'DETAILED'); GO
```

```
SELECT * FROM sys.dm_db_index_physical_stats (DB_ID(N'tempdb'), OBJECT_ID(N'Customer'), NULL, NULL , 'DETAILED'); GO
```



	database_id	object_id	index_id	partition_number	index_type_desc	alloc_unit_type_desc	index_depth	index_level	avg_fragmentation_in_percent	fragment_count	avg_fragment_size_in_pages	page_count	avg_page_space_used_in_percent	record_count
1	2	981578535	1	1	CLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	86.8050407709414	45
2	2	981578535	3	1	NONCLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	29.3056585124784	45
3	2	981578535	4	1	NONCLUSTERED INDEX	IN_ROW_DATA	1	0	0	1	1	1	11.0946380034594	45

You can also check the fragmentation from SSMS by righ click on index and select properties going to fragmentation tab




```
-- Set the values for the scalar variables
SET @fragmentation = 15 ; -- Set the desired fragmentation percentage
threshold
SET @pages = 100; -- Set the desired minimum page count

-- Your query to retrieve index fragmentation information
SELECT  '[' + DB_NAME() + '].[' + OBJECT_SCHEMA_NAME(ddips.[object_id],
DB_ID()) + '].['
        + OBJECT_NAME(ddips.[object_id], DB_ID()) + ']' AS [Object],
        i.[name] AS [Index],
        ddips.[index_type_desc] AS [Index Type],
        ddips.[partition_number] AS [Partition Number],
        ddips.[alloc_unit_type_desc] AS [Allocation Unit Type],
        ddips.[index_depth] AS [Index Depth],
        ddips.[index_level] AS [Index Level],
        CAST(ddips.[avg_fragmentation_in_percent] AS SMALLINT) AS [Average
Fragmentation (%)],
        CAST(ddips.[avg_fragment_size_in_pages] AS SMALLINT) AS [Average
Fragment Size (pages)],
        ddips.[fragment_count] AS [Fragments],
        ddips.[page_count] AS [Pages]
```

```
FROM    sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'limited')
ddips
        INNER JOIN sys.[indexes] i ON ddips.[object_id] = i.[object_id]
                                      AND ddips.[index_id] = i.[index_id]
WHERE   ddips.[avg_fragmentation_in_percent] > @fragmentation
        AND ddips.[page_count] > @pages
ORDER BY ddips.[avg_fragmentation_in_percent],
        OBJECT_NAME(ddips.[object_id], DB_ID()),
        i.[name];
```

Know if Your Index Is Used or Not

Let's say you created index and then the user complains the query is still slow

You want to know whether this index is used or not

We can achieve that by using dmv `sys.dm_db_index_usage_stats`

```
SELECT   OBJECT_NAME(S.[OBJECT_ID]) AS [OBJECT NAME],
         I.[NAME] AS [INDEX NAME],
         USER_SEEKS,
         USER_SCANS,
         USER_LOOKUPS,
         USER_UPDATES
FROM     SYS.DM_DB_INDEX_USAGE_STATS AS S
         INNER JOIN SYS.INDEXES AS I
           ON I.[OBJECT_ID] = S.[OBJECT_ID]
              AND I.INDEX_ID = S.INDEX_ID
WHERE    OBJECTPROPERTY(S.[OBJECT_ID],'IsUserTable') = 1
```

```
SELECT   OBJECT_NAME(S.[OBJECT_ID]) AS [OBJECT NAME],
         I.[NAME] AS [INDEX NAME],
         USER_SEEKS,
         USER_SCANS,
         USER_LOOKUPS,
         USER_UPDATES
FROM     SYS.DM_DB_INDEX_USAGE_STATS AS S
         INNER JOIN SYS.INDEXES AS I
           ON I.[OBJECT_ID] = S.[OBJECT_ID]
              AND I.INDEX_ID = S.INDEX_ID
WHERE    OBJECTPROPERTY(S.[OBJECT_ID],'IsUserTable') = 1


INSERT INTO Orders (OrderDate, CustomerCode, InvoiceTotal)
```

Results | Messages | Live Query Statistics | Execution plan

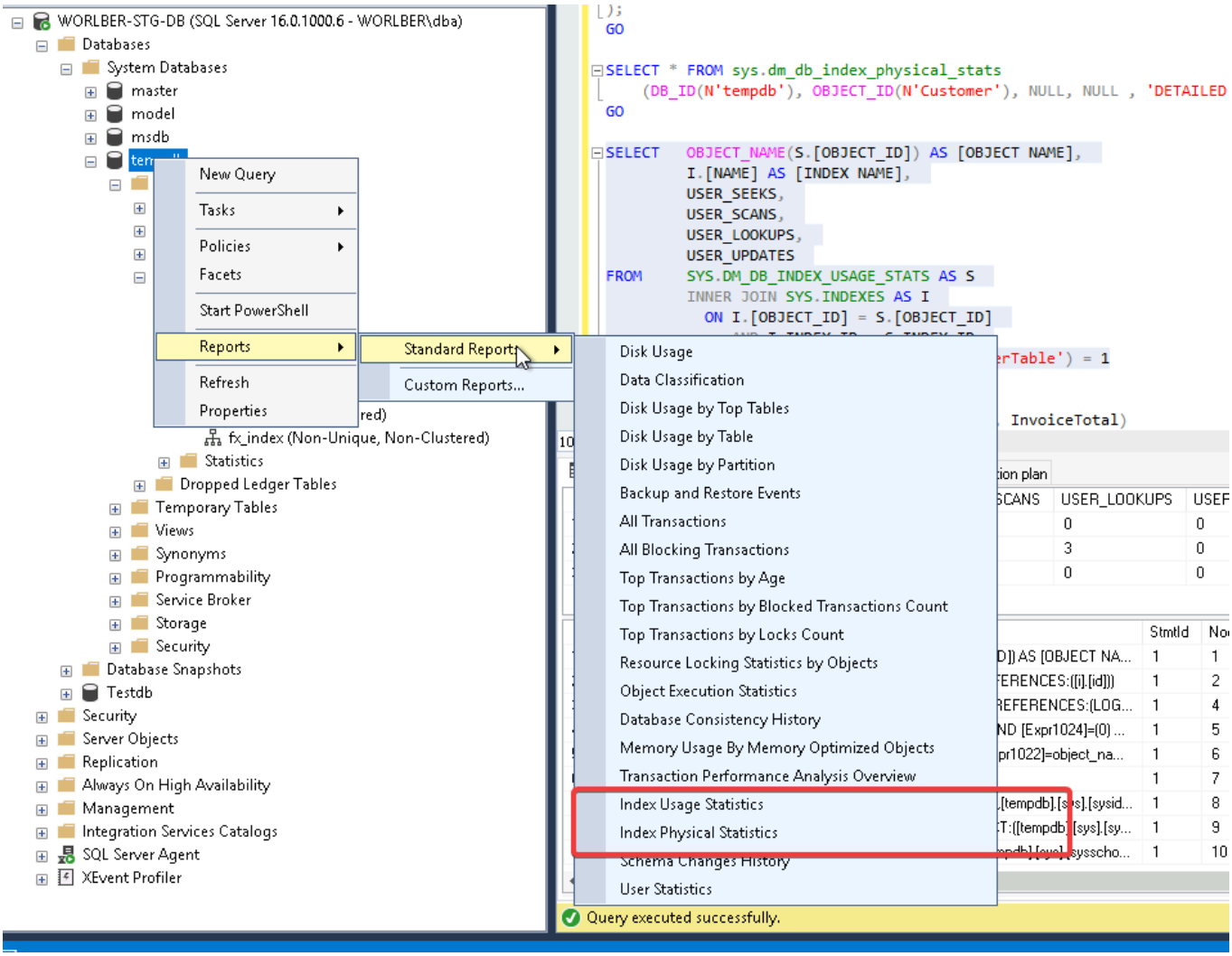
OBJECT NAME	INDEX NAME	USER_SEEKS	USER_SCANS	USER_LOOKUPS	USER_UPDATES
Customer	fx_index	1	0	0	0
Customer	cx_custoner	0	4	3	0
Orders	ordate_index	2	0	0	0

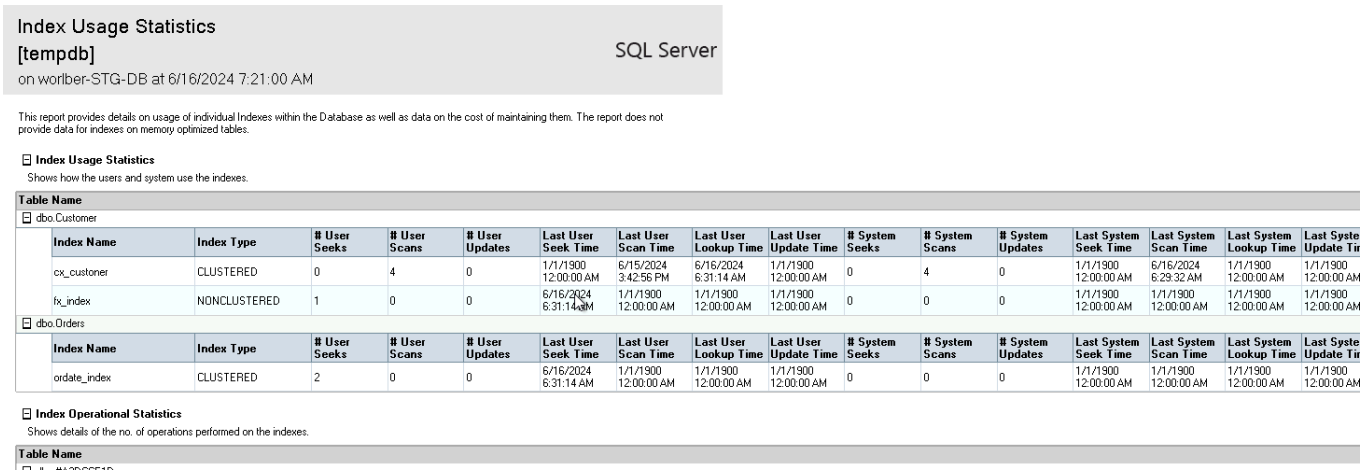
But other non-clustered index we can see good amount of seek

This indicate that index was used

Check Index Usage From SSMS

another way to check usage is to generate index usage report from SSMS





Get Table with Missing Index

If we want to locate table that are missing index we can use the dmv `sys.dm_db_missing_index_details`

```
select * from SELECT * FROM sys.dm_db_missing_index_details
```

Also we can use the below script that will return table without clustered index

```
SELECT o.name
FROM sys.objects o
WHERE o.type='U'
  AND NOT EXISTS(SELECT 1 FROM sys.indexes i
                 WHERE o.object_id = i.object_id
                   AND i.type_desc = 'CLUSTERED')
```

Get Column with Missing Index

The dmv `sys.dm_db_missing_index_columns` is very useful when we want to get a column that is missing an index, which might be helpful when deciding which column we need for an index.

2.Statistics and Their Importance

${toc}

- Statics is one of SQL Server key inputs that used by a query optimizer during a generating query plan.
- Statistics are used by optimizers to determine how many rows will be returned from a query, so it calculates the cost of a query plan using this estimation.

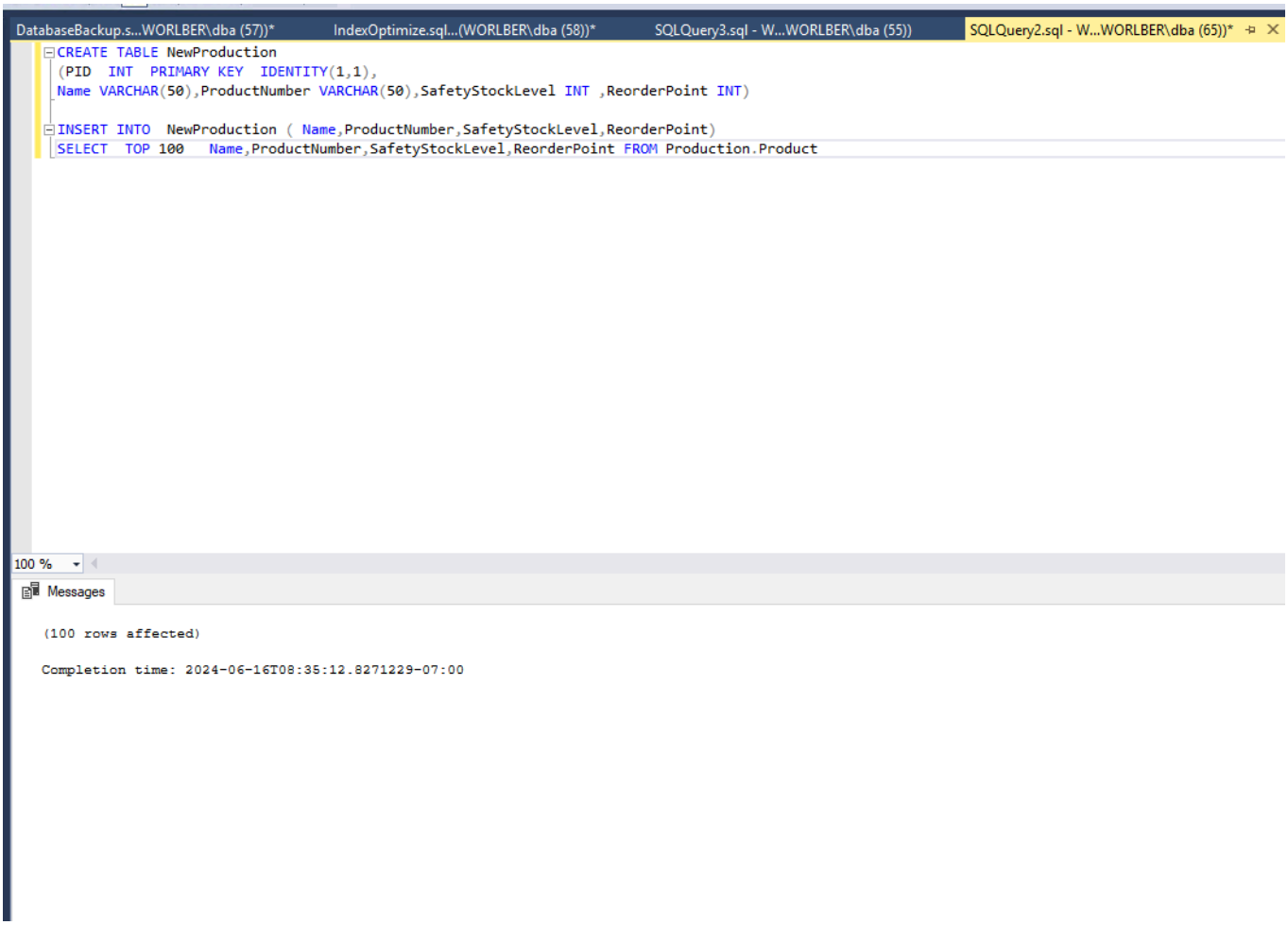
CPU, I/O memory requirements are made according to the estimate.

- In this case, accurate up-to-date statistics play a key role in creating more effective query plans.
- SQL Server automatically creates statistics when we create an index.

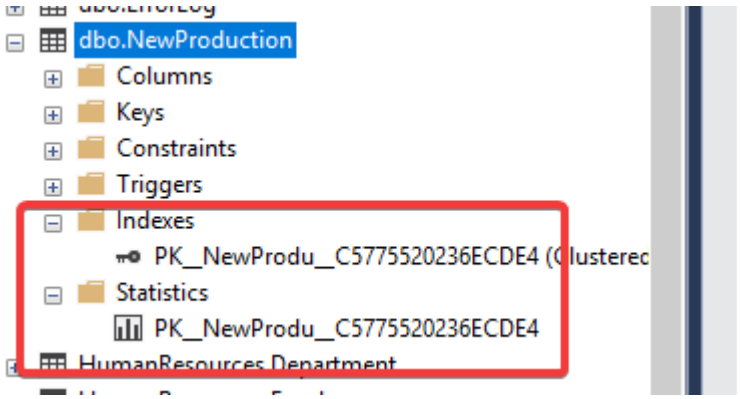
Simulation

We will use a short copy of the Product table of the Adventureworks database. We will use the following T-SQL script to create the NewProduction table and populate it with data from the Product table.

```
CREATE TABLE NewProduction
(PID  INT  PRIMARY KEY  IDENTITY(1,1),
Name VARCHAR(50),ProductNumber VARCHAR(50),SafetyStockLevel INT
,ReorderPoint INT)

INSERT INTO  NewProduction (
Name,ProductNumber,SafetyStockLevel,ReorderPoint)
SELECT  TOP 100   Name,ProductNumber,SafetyStockLevel,ReorderPoint FROM
Production.Product
```



As we stated if we check properties of new table we created we can see that there statistics been created one clustered index is created



Dmv to Check Statistics Details

the dmv `DBCC SHOW_STATISTICS` will show meta data details of statistics

```
DBCC SHOW_STATISTICS ('NewProduction','PK__NewProdu__C57755203D35D630')
```

```
DBCC SHOW_STATISTICS ('Person.Person','IX_Person_LastName_FirstName_MiddleName')
```

100 %

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	IX_Person_LastName_FirstName_MiddleName	May 8 2023 12:07PM	19972	19972	200	0.5642176	28.23934	YES	NULL	19972	0

	All density	Average Length	Columns
1	0.0008291874	11.2126	LastName
2	5.124001E-05	23.02013	LastName, FirstName
3	5.05025E-05	24.23934	LastName, FirstName, MiddleName
4	5.00701E-05	28.23934	LastName, FirstName, MiddleName, BusinessEntityID

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	Abbas	0	1	0	1
2	Adams	10	86	6	1.666667
3	Alexander	28	123	15	1.866667
4	Allen	0	82	0	1
5	Alonso	1	93	1	1
6	Alvarez	5	99	4	1.25
7	Anand	2	74	2	1
8	Andersen	0	95	0	1
9	Anderson	0	85	0	1

Query executed successfully. | WORLBER-STG-DB (16.0 RTM) | WORLBER\dba (65)

The result show the statics number of rows
And it shows destiny factor and histogram

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	IX_Person_LastName_FirstName_MiddleName	Oct 27 2017 2:33PM	19972	19972	199	0,5505741	28,23934	YES	NULL	19972	0

Statistic Header

	All density	Average Length	Columns
1	0,0008291874	11,2126	LastName
2	5,124001E-05	23,02013	LastName, FirstName
3	5,05025E-05	24,23934	LastName, FirstName, MiddleName
4	5,00701E-05	28,23934	LastName, FirstName, MiddleName, BusinessEntityID

Density Vector

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	Abbas	0	1	0	1
2	Adams	10	86	6	1,666667
3	Alexander	28	123	15	1,866667
4	Allen	0	82	0	1
5	Alonso	1	93	1	1
6	Alvarez	5	99	4	1,25
7	Anand	2	74	2	1
8	Andersen	0	95	0	1
9	Anderson	0	85	0	1
10	Arthur	9	24	8	1,125
11	Arun	0	58	0	1
12	Ashe	0	30	0	1
13	Bailey	9	95	7	1,285714
14	Baker	0	84	0	1
15	Barnes	13	77	11	1,181818
16	Beck	17	36	13	1,307692
17	Bell	31	62	2	15,5
18	Bennett	4	68	3	1,333333

Histogram

What Is a Histogram?

A histogram is the classification of the values in a data group. Histograms allow us to see the frequency distribution of a data set. For example, the following histogram data is created based

on the age distribution of people who are visiting a museum per day. Histogram charts are generally used to virtualize histogram data.

Age	Visitor Number
10-14	10
15-19	11
20-24	11
25-29	4
30-34	8
35-39	7
40-44	7
45-49	10
50-54	10
55-59	11

SQL Server statistics stores the distribution of the column data in a histogram and stores unique values ratio in the density vector. These two meta-data are used by the query optimizer to calculate how many rows will return a query. Understanding SQL Server statistics concepts with the help of examples will be more beneficial than continuing with theoretical information to learn better about real-world issues.

Returning to our sinario we will nsert 5000 row and see if the stattsics are updated or not

```
INSERT INTO  NewProduction (
Name,ProductNumber,SafetyStockLevel,ReorderPoint)
SELECT    Name,ProductNumber,SafetyStockLevel,ReorderPoint FROM
Production.Product
GO 1000
SELECT * FROM NewProduction WHERE PID > 24
```

```
INSERT INTO  NewProduction ( Name,ProductNumber,SafetyStockLevel,ReorderPoint)
SELECT    Name,ProductNumber,SafetyStockLevel,ReorderPoint FROM Production.Product
GO 1000
SELECT * FROM NewProduction WHERE PID > 24
```

100 %

Results | Messages

	PID	Name	ProductNumber	SafetyStockLevel	ReorderPoint
1	25	Flat Washer 9	FW-3400	1000	750
2	26	Flat Washer 4	FW-3800	1000	750
3	27	Flat Washer 3	FW-5160	1000	750
4	28	Flat Washer 8	FW-5800	1000	750
5	29	Flat Washer 5	FW-7160	1000	750
6	30	Flat Washer 7	FW-9160	1000	750
7	31	Fork Crown	FC-3654	800	600
8	32	Front Derailleur Cage	FC-3982	800	600
9	33	Front Derailleur Linkage	FL-2301	800	600
10	34	Guide Pulley	GP-0982	800	600
11	35	LL Grip Tape	GT-0820	800	600
12	36	ML Grip Tape	GT-1209	800	600
13	37	HL Grip Tape	GT-2908	800	600
14	38	Thin-Jam Hex Nut 9	HJ-1213	1000	750
15	39	Thin-Jam Hex Nut 10	HJ-1220	1000	750

Now we will check the statics

```
DBCC SHOW_STATISTICS ('NewProduction','PK__NewProdu__C57755203D35D630')
```

```
DBCC SHOW_STATISTICS ('NewProduction','PK__NewProdu__C5775520236ECDE4')
```

100 %

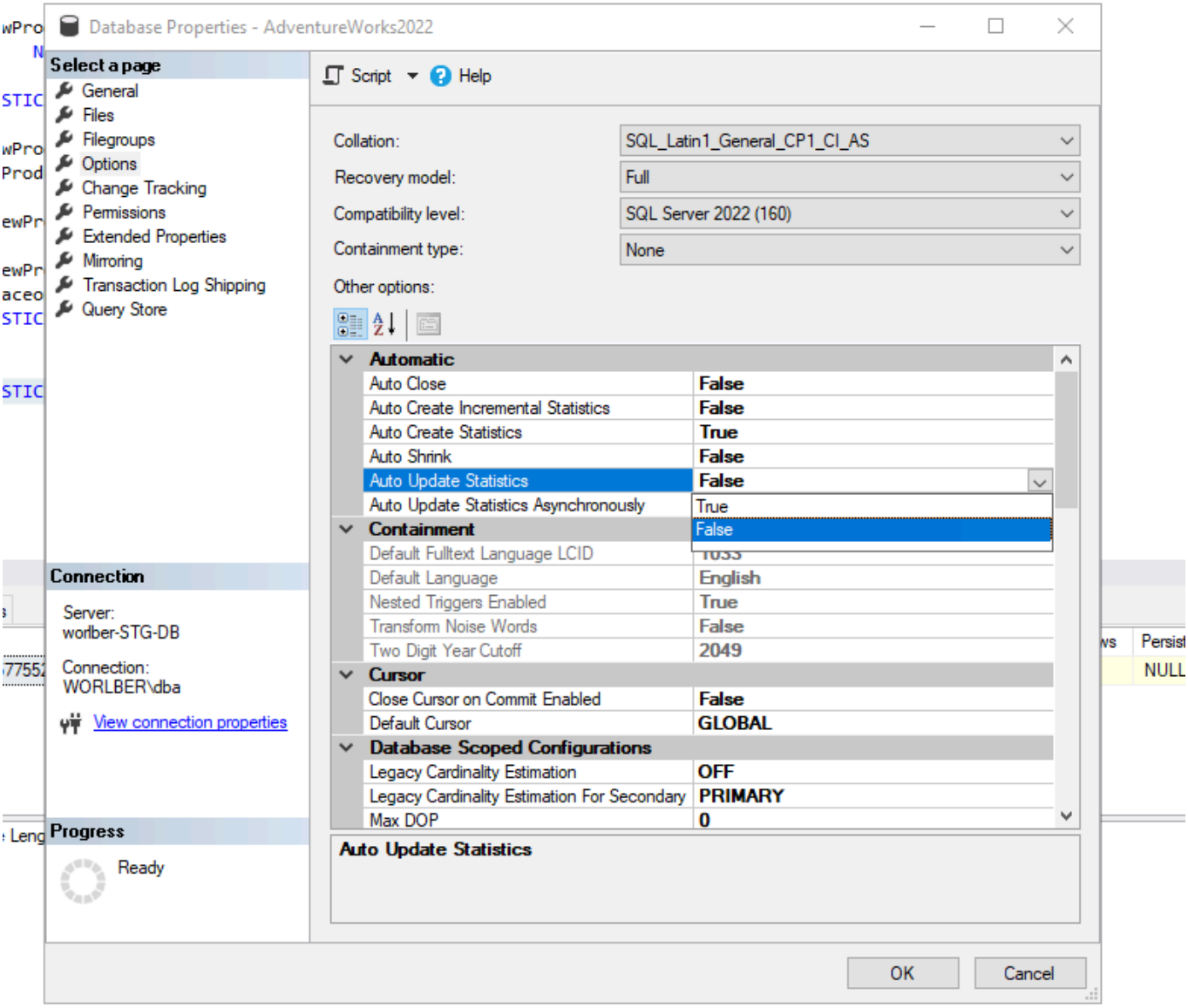
Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	PK__NewProdu__C5775520236ECDE4	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

No data because auto update statics is disabled

I will toggled it to be enable and we will check the status again

```
),ProductNumber VARCHAR(50),SafetyStockLevel INT ,ReorderPoint INT)
```



RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS

```
DBCC SHOW_STATISTICS ('NewProduction','PK__NewProdu__C5775520236ECDE4')
```

100 %

Results | Messages

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	PK__NewProdu__C5775520236ECDE4	Jun 16 2024 8:48AM	504100	143706	176	0.9941197	4	NO	NULL	504100	0

	All density	Average Length	Columns
1	2.293478E-06	4	PID

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	1	0	1	0	1
2	2	0	1	0	1
3	3779	3187.926	1	3166	1.006867
4	5008	1797.597	1	1228	1.463841
5	8516	3588.172	1	3507	1.023146
6	8517	0	1	0	1
7	10723	3588.172	1	2205	1.627289
8	12185	1794.086	1	1461	1.227985
9	14000	1794.086	1	1784	1.005502
10	15593	1794.086	1	1592	1.126938

Query executed successfully. | WORLBER-STG-DB (16.0 RTM) | WORLBER\dba (65) | AdventureW

Maintenance

It important to make auto update statics to be true

Also creating maintenance plan for update statics on daily basics is actually advisable

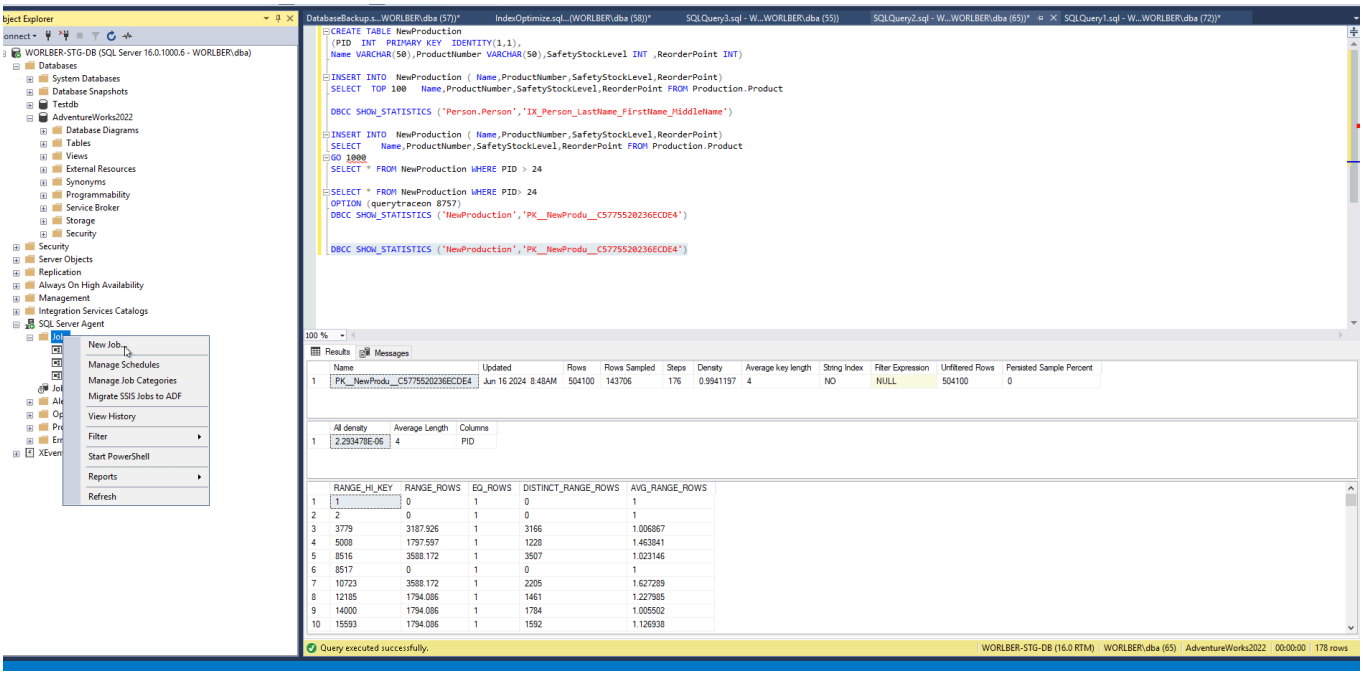
We will use ola hallengren scrip to create job that will update statics on database

link to download script

note you have to execute CommandExecute and CommandLog First before executing IndexOptimize.Sql script

We can use the below script and schedule

```
EXECUTE dbo.IndexOptimize
@Databases = 'USER_DATABASES',
@FragmentationLow = NULL,
@FragmentationMedium = NULL,
@FragmentationHigh = NULL,
@UpdateStatistics = 'ALL',
@OnlyModifiedStatistics = 'Y'
```



3.Identifying Slow Running Queries

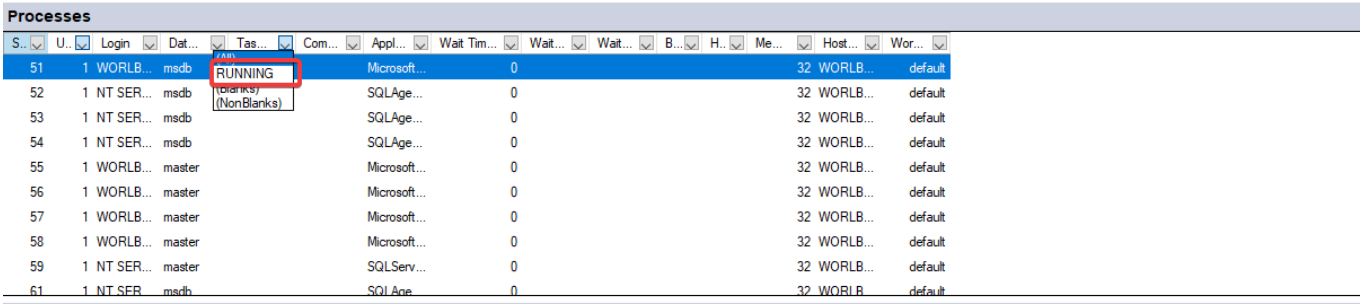
${toc}

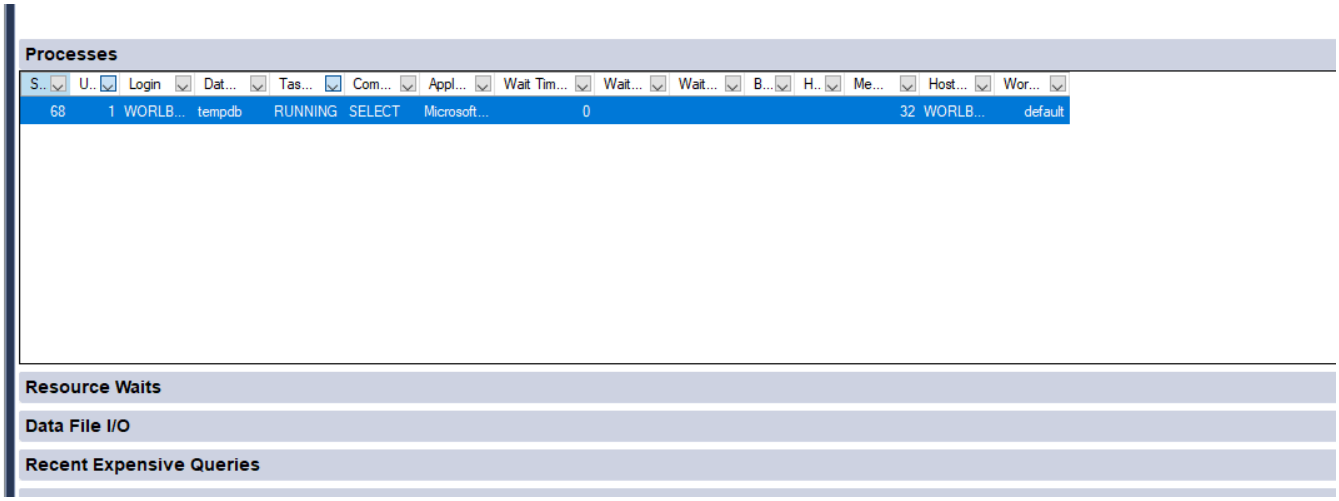
1. : Indexes: Basics and Best Practices

Using Activity Monitor

Activity monitory is gui tool that help check overall performance status of sql Server

when check activity monitor you need to focus on processes time which will give overall view of processes running





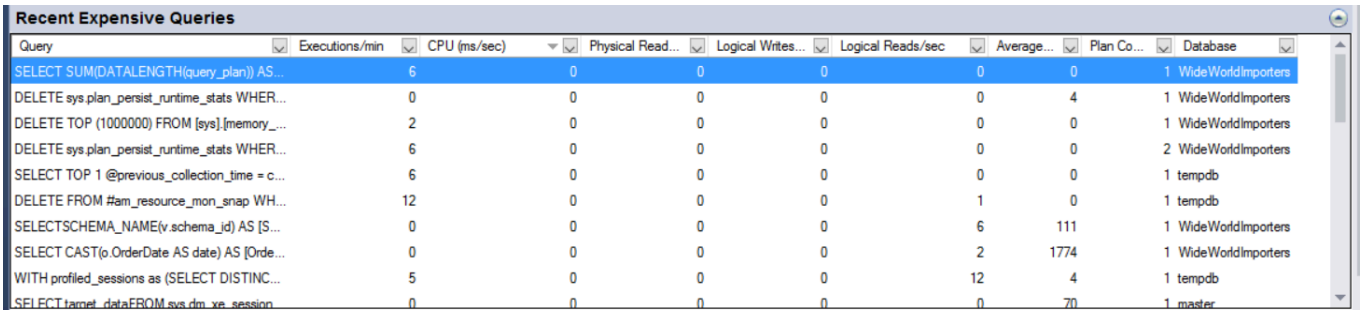
Focus on tap such as wait type and wait time https://Www.Sqlshack.Com/sql-server-wait-types/

This means query is blocked because is waiting for something

It might not be because lock on resource it could other factor

If there wait type on process you can use link which will give you More information about the wait

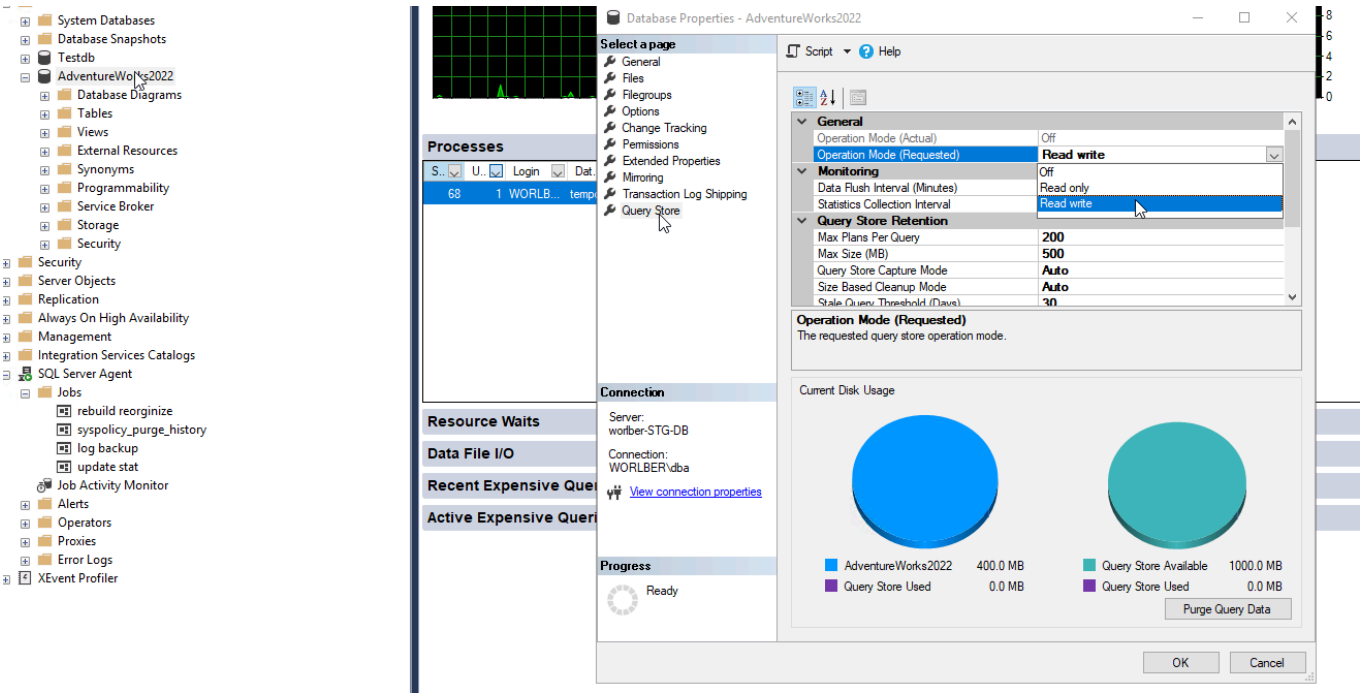
Another section is The Active Expensive and Recent Expensive queries will give you information about the queries which have high CPU, Logical Reads or High Elapsed time.



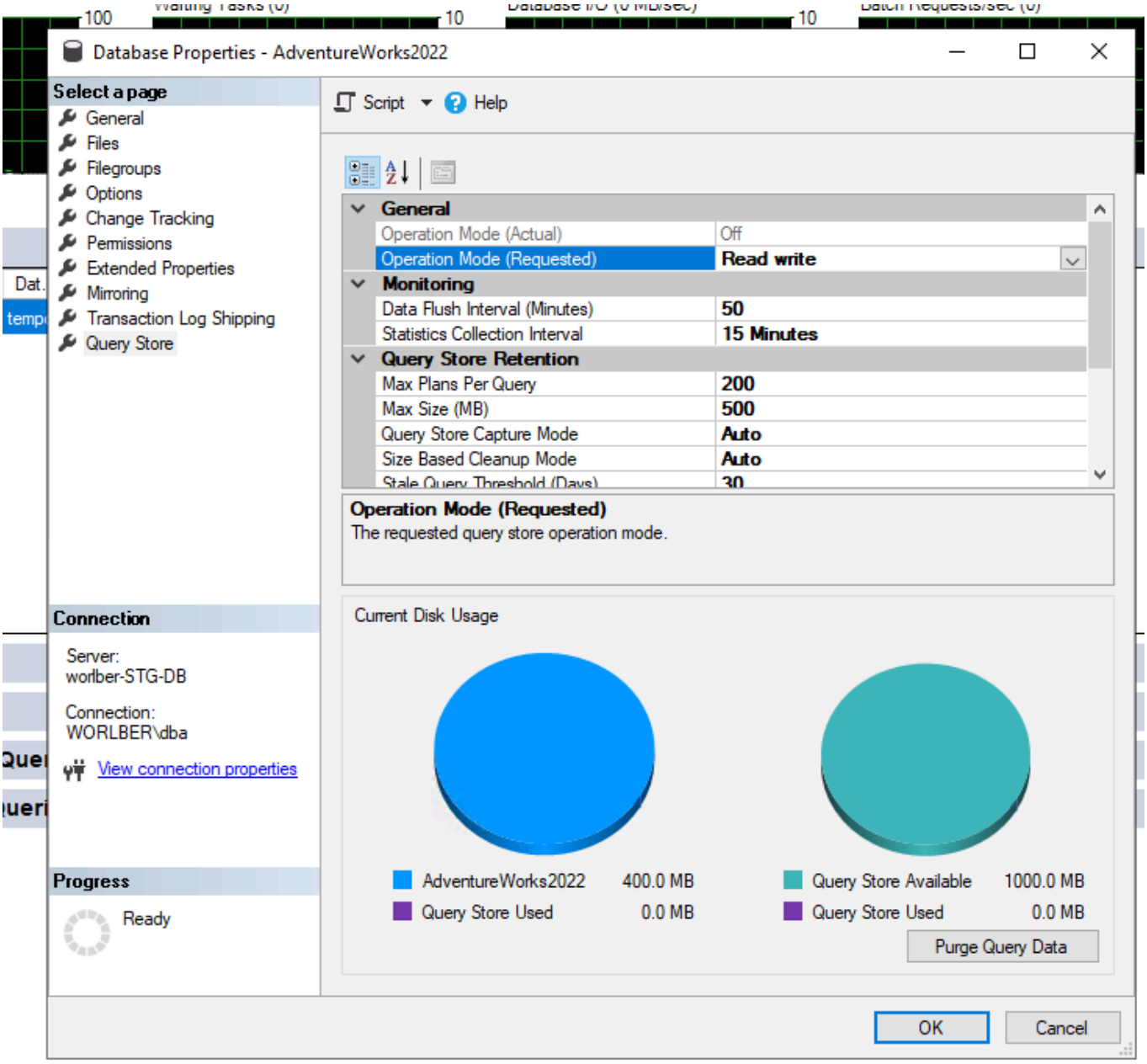
Query Store

Query store is feature that introduced in sql server 2016 it capture query performance and store them so you can check again if there any slow query's

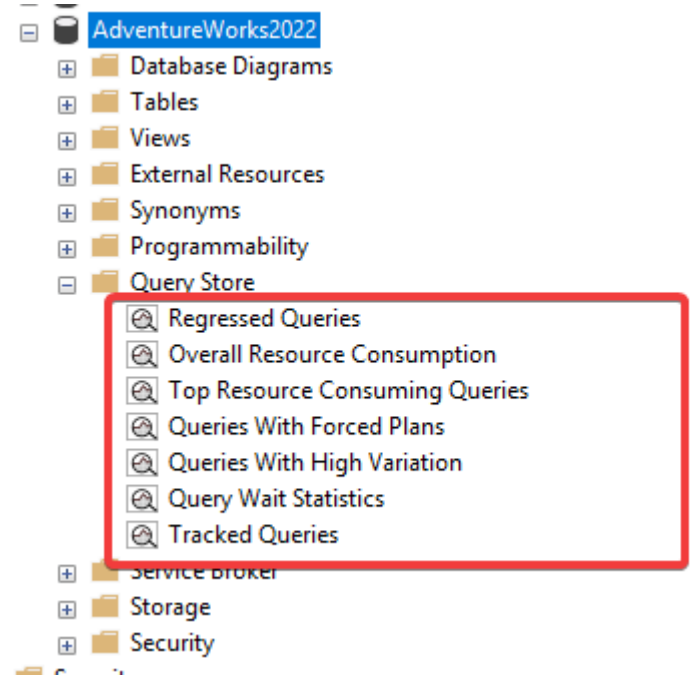
To enable it right click on database and then select query store

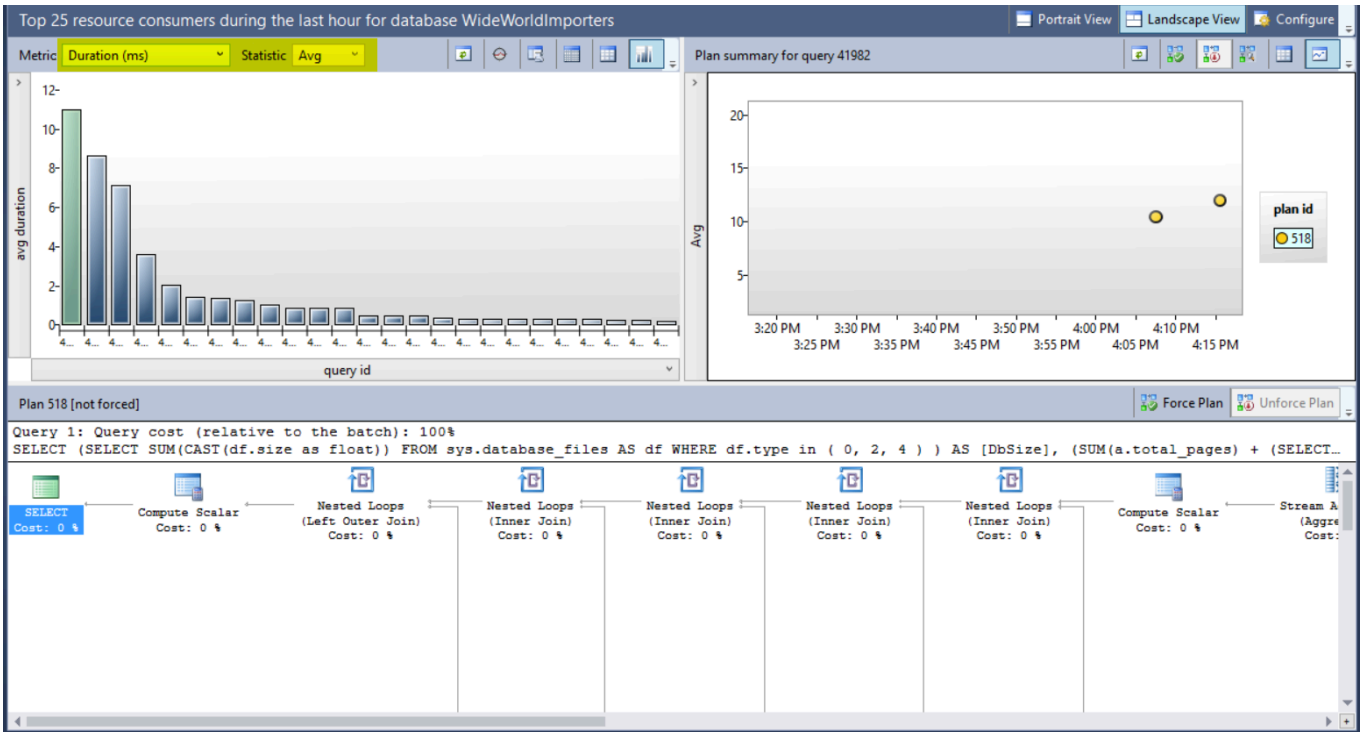


below is best config for query store



Give the Query Store a day or two to capture the production load so that you can easily work on it with real load.





dmv

cpu-intensive query's

Below query will return top 10 cpu intensive query

```
DECLARE @count INT;
SET @count = 10;
SELECT TOP (@count)
```

```
        RANK() Over (ORDER BY deqs.total_worker_time DESC) As [Rank],
        CONVERT(decimal(38,2), CONVERT(float, total_worker_time) / 1000) AS
[Total CPU Time (ms)],
        execution_count AS [Execution Count],
        CONVERT(decimal(38,2), (CONVERT(float, total_worker_time) /
execution_count) / 1000) AS [Average CPU Time (ms)] ,
        SUBSTRING(execText.text,
          -- starting value for substring
          CASE WHEN deqs.statement_start_offset = 0
             OR deqs.statement_start_offset IS NULL
               THEN 1
               ELSE deqs.statement_start_offset/2 + 1 END,
          -- ending value for substring
          CASE WHEN deqs.statement_end_offset = 0
            OR deqs.statement_end_offset = -1
            OR deqs.statement_end_offset IS NULL
               THEN LEN(execText.text)
               ELSE deqs.statement_end_offset/2 END -
                 CASE WHEN deqs.statement_start_offset = 0
                   OR deqs.statement_start_offset IS NULL
                     THEN 1
                     ELSE deqs.statement_start_offset/2  END + 1
        ) AS [Query Text],
        execText.text AS [Object Text]
FROM    sys.dm_exec_query_stats deqs
        CROSS APPLY sys.dm_exec_sql_text(deqs.plan_handle) AS execText
ORDER BY deqs.total_worker_time DESC ;
```

```sql
SELECT TOP (@count)
        RANK() Over (ORDER BY deqs.total_worker_time DESC) As [Rank],
        CONVERT(decimal(38,2), CONVERT(float, total_worker_time) / 1000) AS [Total CPU Time (ms)],
        execution_count AS [Execution Count],
        CONVERT(decimal(38,2), (CONVERT(float, total_worker_time) / execution_count) / 1000) AS [Average CPU Time (ms)] ,
        SUBSTRING(execText.text,
          -- starting value for substring
          CASE WHEN deqs.statement_start_offset = 0
             OR deqs.statement_start_offset IS NULL
               THEN 1
               ELSE deqs.statement_start_offset/2 + 1 END,
          -- ending value for substring
          CASE WHEN deqs.statement_end_offset = 0
            OR deqs.statement_end_offset = -1
            OR deqs.statement_end_offset IS NULL
               THEN LEN(execText.text)
               ELSE deqs.statement_end_offset/2 END -
                 CASE WHEN deqs.statement_start_offset = 0
                   OR deqs.statement_start_offset IS NULL
                     THEN 1
                     ELSE deqs.statement_start_offset/2  END + 1
        ) AS [Query Text],
        execText.text AS [Object Text]
FROM    sys.dm_exec_query_stats deqs
        CROSS APPLY sys.dm_exec_sql_text(deqs.plan_handle) AS execText
ORDER BY deqs.total_worker_time DESC ;
```

00 %

Results | Messages | Execution plan

	Rank	Total CPU Time (ms)	Execution Count	Average CPU Time (ms)	Query Text	Object Text
1	1	37266.33	696	53.54	SELECT target_data FROM sys.dm_xe_session_t...	SELECT target_data FROM sys.dm_xe_session_ta...
2	2	4636.72	170	27.27	WITH profiled_sessions as (SELECT DISTINCT sessi...	WITH profiled_sessions as (SELECT DISTINCT sessi...
3	3	4456.37	170	26.21	SELECT @current_total_io_mb = SUM(num_of_bytes_r...	SET NOCOUNT ON; DECLARE @previous_collectio...
4	4	4179.42	58	72.06	WITH Quartiles AS (SELECT DISTINCT ...	DECLARE @msticks bigint, @mstickstime d...
5	5	2152.97	33	65.24	WITH merged_query_stats AS (SELECT [sql...	CREATE PROC #am_get_querystats AS BEGIN -- Set...
6	6	1845.52	3	615.17	INSERT INTO @tmpIndexesStatistics (SchemaID, Sche...	CREATE PROCEDURE [dbo].[IndexOptimize] @Datab...
7	7	1149.95	10	115.00	INSERT INTO @xp_results EXECUTE master.dbo...	CREATE PROCEDURE sp_sqlagent_has_server_acce...
8	8	1043.48	168	6.21	WITH interval_waitstats AS (-- First get resource w...	CREATE PROC #am_generate_waitstats AS BEGIN ...
9	9	888.23	33	26.92	INSERT INTO #am_fingerprint_stats_snapshots SELE...	CREATE PROC #am_get_querystats AS BEGIN -- Set...
10	10	863.23	1	863.23	SELECT udf.name AS [Name], udf.object_id AS [ID], ud...	(@_msparam_0 nvarchar(4000),@_msparam_1 nvarchar(...

The table will show the folloiwng

- Total CPU Time (ms): The total CPU time consumed by the query in milliseconds.
- Average CPU Time (ms): The average CPU time per execution, calculated as the total CPU time divided by the execution count.
- Query Text: A substring of the actual SQL text executed.
- Object Text: The full SQL text of the executed statement.

4. Understanding SQL Server Waits and Queues

${toc}
🌶️

what is wait statics

When we have single or multiple CPU and we have multiple queries running that need to be executed, the cpu returns execution fast with no wait

When query is sent from application the must compete for limited CPU processor

If we have single core CPU with little transaction activity, accessing the CPU may not be a problem.

However every in very busy transactional database where you have thousands of user requesting their query to be process this become an issues

As these request compete for limited CPU recourse , waiting for CPU to process the request can become as issue

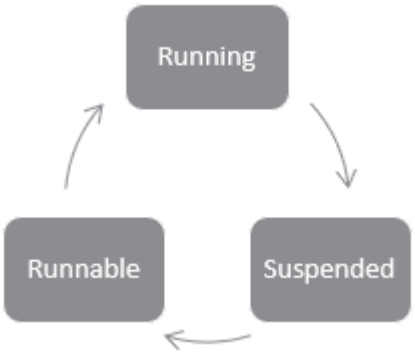
This waiting for sql to execute sql statement is what we call WAIT STATISTICS

The wait times are captured and recorded by the SQL Server and all of this captured information is called wait statistics

There can be many factor for waiting including

- locking
- poor index
- badly designed query
- and more processing
  A query must wait each time it is sent for processing by an application.

The average wait time is only slightly different in milliseconds (4 milliseconds).



we have three stats for query process state

1: **running** When a query is being processed by the CPU, it is set to be in the running stats, which state that we want our query to be
2: **Suspended** If there is a lock on a query or a missing resource the query needs, then the query moves to state suspended and moves to the queue for waiting for the resource to be obtained.
3: **runnable**: When a missing resource query is provided, it goes to runnable state and the entire cycle repeats for the query and all other queries.

DMV to view the wait statistics

If the query is forced to wait for resource sql server record that wait stats and wait type , there are many wait types and the following link will help you understand the types :
There are about 180 wait types

https://www.sqlshack.com/sql-server-wait-types/

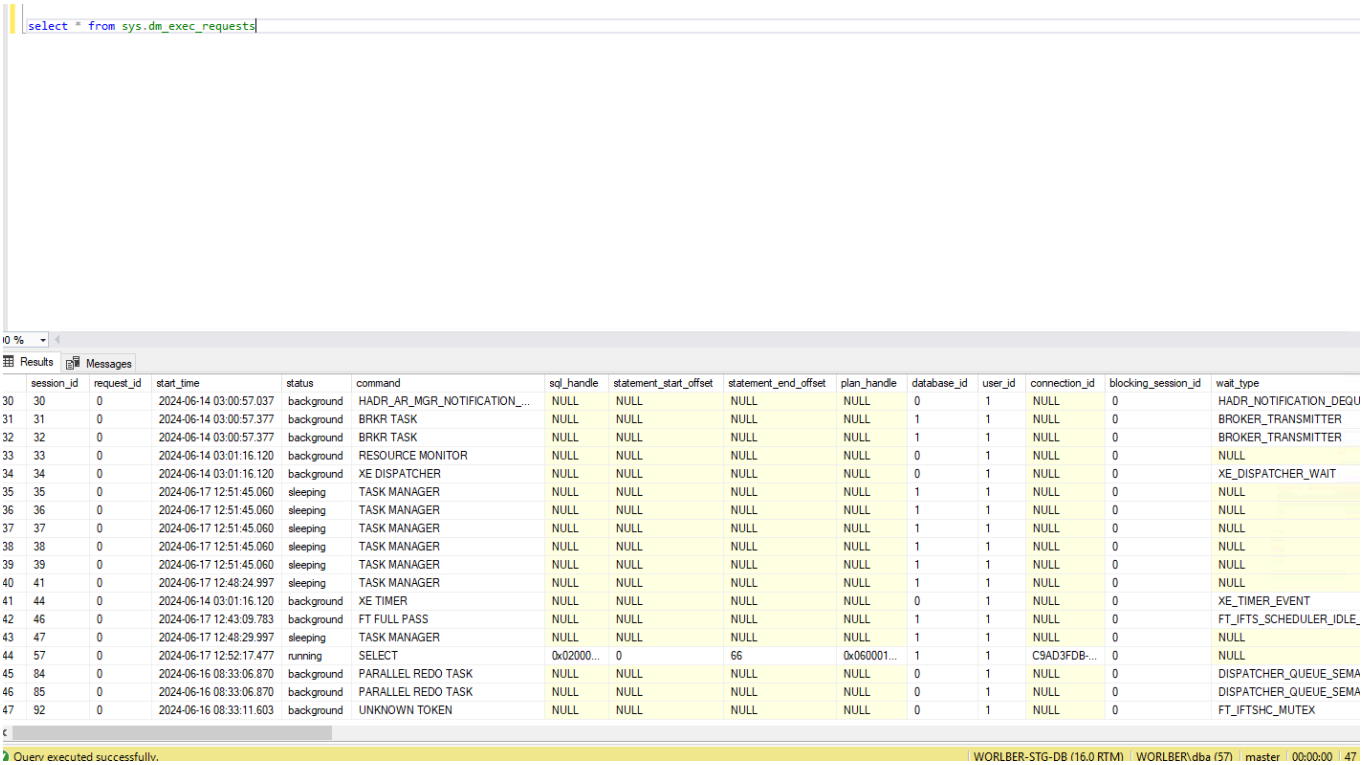
We can view the wait statistics by examining the following dmvs:

```
select * fom sys.dm_os_wait_stats

select * fom sys.dm_os_waiting_tasks

select * fom sys.dm_os_exec_requests
```

`select * from sys.dm_os_wait_stats` where waiting_tasks_count give you how many time wait type accours the during of time very useful if you need to fine tune sql
also provide wait type that then you check what is causing this wait type.

`select * from sys.dm_exec_requests`

Returns information about each request that is executing in SQL Server.



```
select * from sys.dm_exec_requests
```

Results | Messages

	session_id	request_id	start_time	status	command	sql_handle	statement_start_offset	statement_end_offset	plan_handle	database_id	user_id	connection_id	blocking_session_id	wait_type
30	30	0	2024-06-14 03:00:57.037	background	HADR_AR_MGR_NOTIFICATION_...	NULL	NULL	NULL	NULL	0	1	NULL	0	HADR_NOTIFICATION_DEQU
31	31	0	2024-06-14 03:00:57.377	background	BRKR TASK	NULL	NULL	NULL	NULL	1	1	NULL	0	BROKER_TRANSMITTER
32	32	0	2024-06-14 03:00:57.377	background	BRKR TASK	NULL	NULL	NULL	NULL	1	1	NULL	0	BROKER_TRANSMITTER
33	33	0	2024-06-14 03:01:16.120	background	RESOURCE MONITOR	NULL	NULL	NULL	NULL	0	1	NULL	0	NULL
34	34	0	2024-06-14 03:01:16.120	background	XE DISPATCHER	NULL	NULL	NULL	NULL	0	1	NULL	0	XE_DISPATCHER_WAIT
35	35	0	2024-06-17 12:51:45.060	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
36	36	0	2024-06-17 12:51:45.060	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
37	37	0	2024-06-17 12:51:45.060	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
38	38	0	2024-06-17 12:51:45.060	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
39	39	0	2024-06-17 12:51:45.060	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
40	41	0	2024-06-17 12:48:24.997	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
41	44	0	2024-06-14 03:01:16.120	background	XE TIMER	NULL	NULL	NULL	NULL	0	1	NULL	0	XE_TIMER_EVENT
42	46	0	2024-06-17 12:43:09.783	background	FT FULL PASS	NULL	NULL	NULL	NULL	1	1	NULL	0	FT_IFTS_SCHEDULER_IDLE_
43	47	0	2024-06-17 12:48:29.997	sleeping	TASK MANAGER	NULL	NULL	NULL	NULL	1	1	NULL	0	NULL
44	57	0	2024-06-17 12:52:17.477	running	SELECT	0x02000...	0	66	0x060001...	1	1	C9AD3FDB-...	0	NULL
45	84	0	2024-06-16 08:33:06.870	background	PARALLEL REDO TASK	NULL	NULL	NULL	NULL	0	1	NULL	0	DISPATCHER_QUEUE_SEMA
46	85	0	2024-06-16 08:33:06.870	background	PARALLEL REDO TASK	NULL	NULL	NULL	NULL	0	1	NULL	0	DISPATCHER_QUEUE_SEMA
47	92	0	2024-06-16 08:33:11.603	background	UNKNOWN TOKEN	NULL	NULL	NULL	NULL	0	1	NULL	0	FT_IFTSHC_MUTEX

Query executed successfully. | WORLBER-STG-DB (16.0 RTM) | WORLBER\dba (57) | master | 00:00:00 | 47

As mentioned each query when send to cpu will wait but propose of this dmv it will show current running process and what wait type

You need to compare it with `select * from sys.dm_os_wait_stats` And if wait type happen more often and the duration

5. Using SQL Profiler and Extended Events

and select path and max file size ${toc}

sql profile is deprecated and will be removed on future version

Extended Events

the goal of extended event is to capture all request that are happening on databases and log them out somewhere that way we can trace the event and investigate an error we are facing

extended event give you abiltiy to monitor and collect diffrent event and system information from sql server to be analyse later
The Extended Events architecture can be broken up into 5 main concepts. These are:

- Packages
- Targets
- Engine
- Sessions
- Tools

Extended Event package is simply a container for the objects used by Extended Events. It contains all the metadata for the objects that exist within Extended Events

Events

basically a point of execution within your program When an event fires along with the time the event occurred

he state information is also logged. The state information is stored in a versioned schema

Targets

A target is the event consumer.
extended Events provide multiple targets that can be used to direct event output to a file, memory buffer or aggregate event data

Actions

An action is a programmed response to an event. Actions must be tied to an event and each event can have its own unique set of actions

Identifying the long-running queries

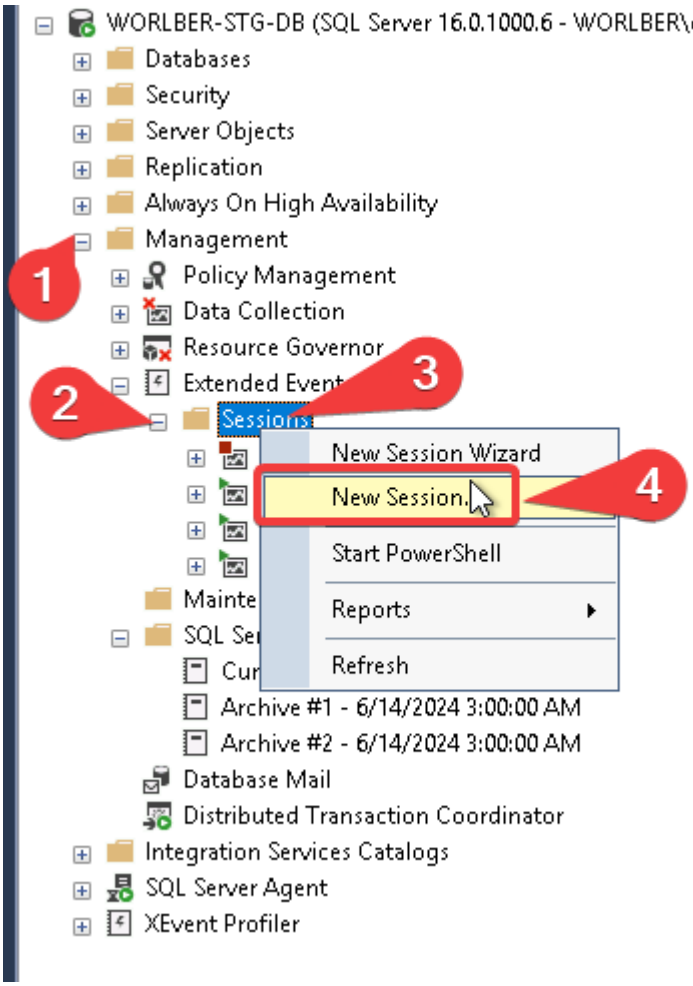
we need to detect long-running queries to troubleshoot these queries.

The sql_statement_completed event captures all queries which are executed against in the database and it involves the following metrics:

CPU time
Duration of the query execution
Number of the logical reads
Number of the physical reads
SQL text and statement
Number of the writes
Client hostname
Client application name

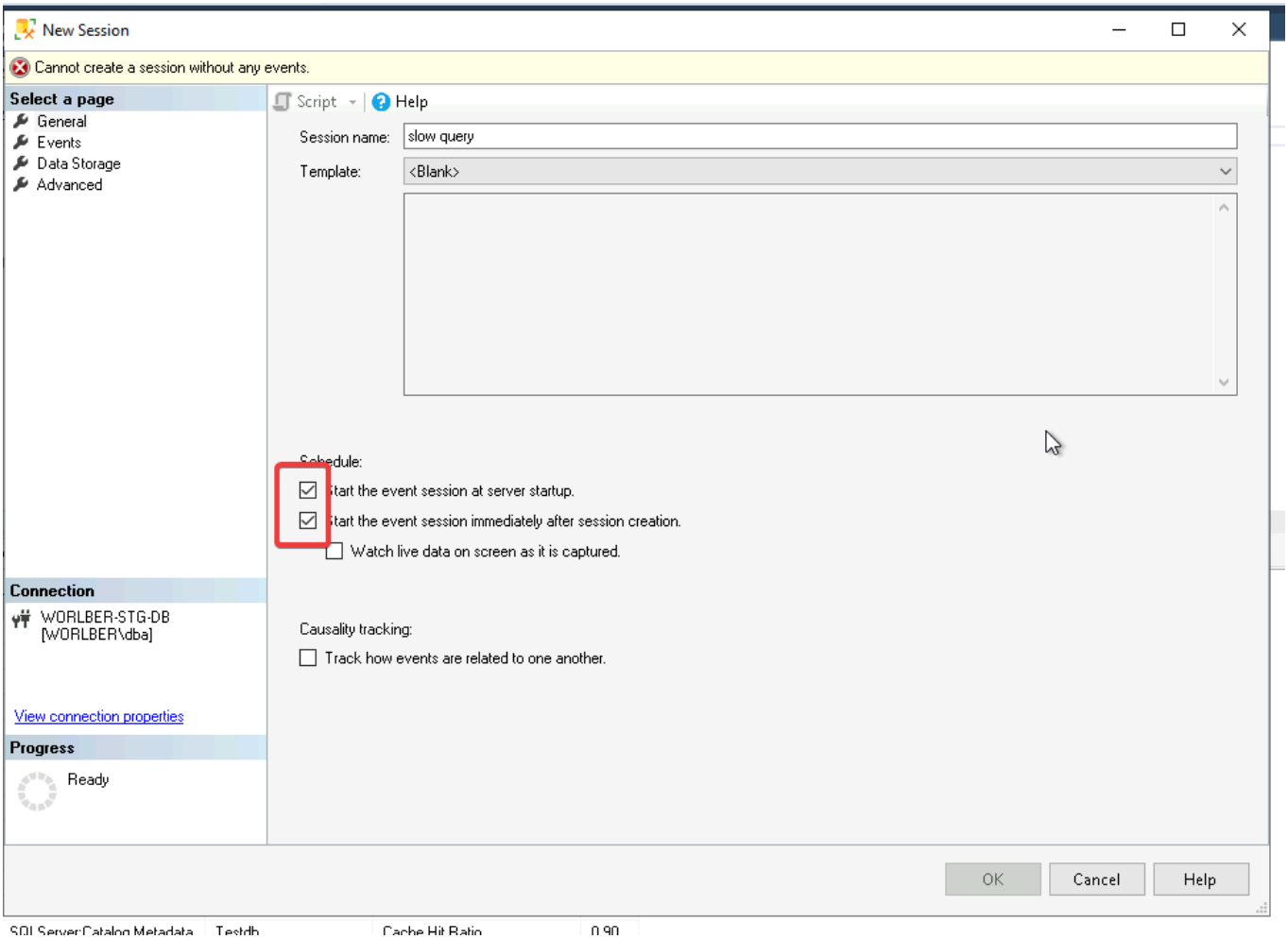
Create an extended event session to identify the long-running queries

on ssms expand management and expand extended event and right click and select new session



check mark the below options

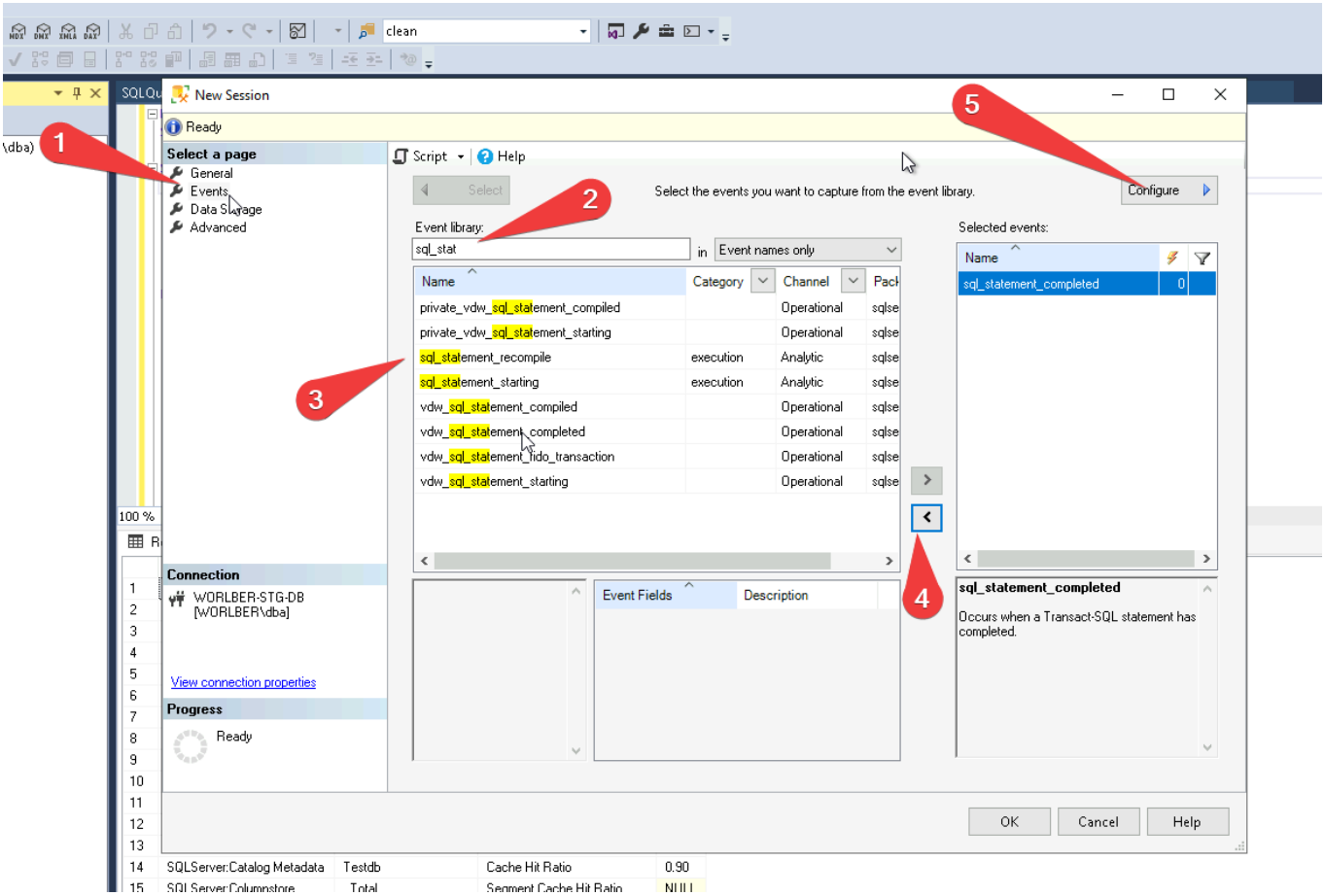
- start the event session at server startup
- start the event session immediately after session creation.
  These two options allow starting the extended event session automatically when the database service is started and also after the creation of the session it automatically begins to collect data.



click on **event** tab on left panel

on event library search for **sql_statment_completed**

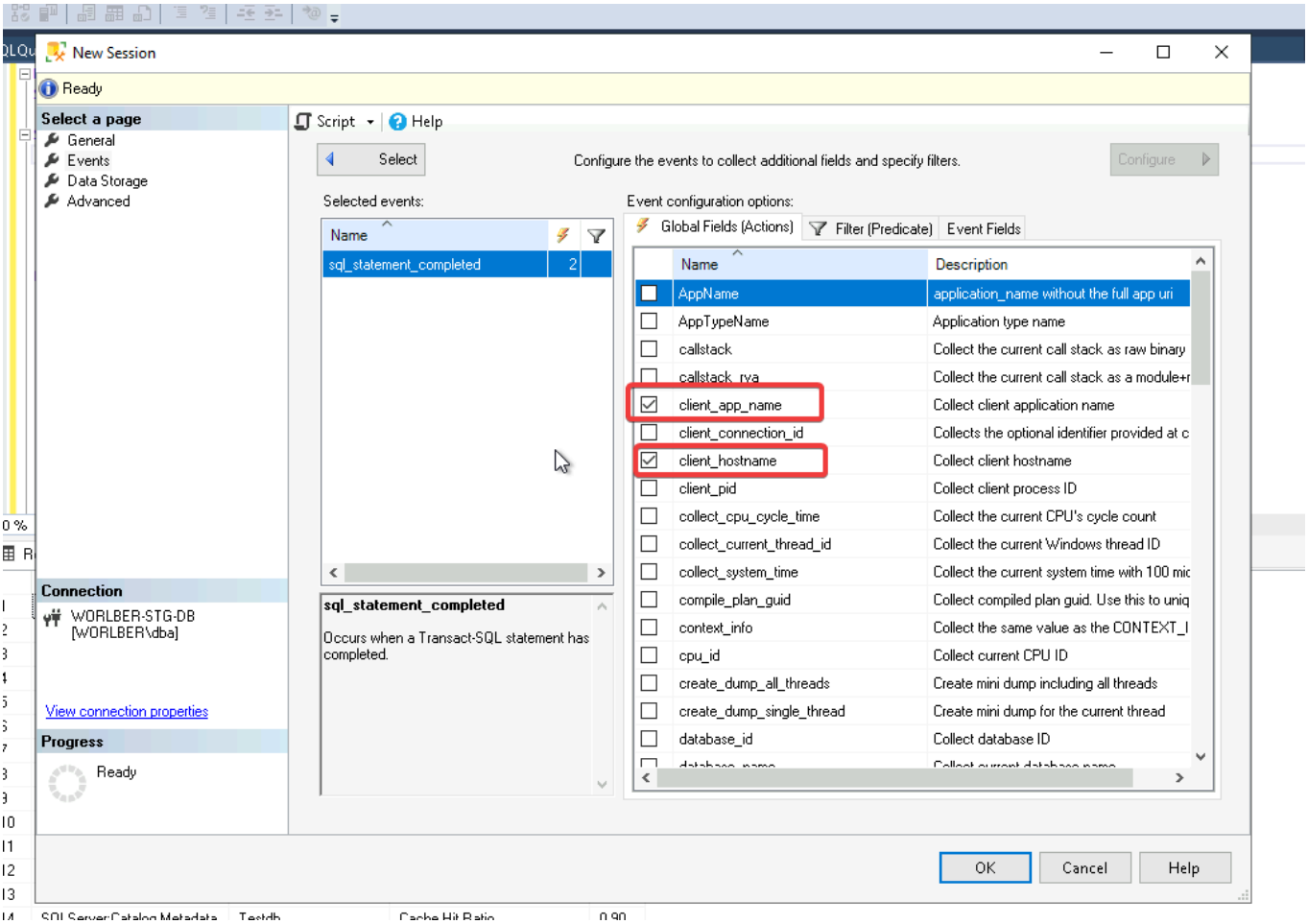
move the library to selected evets after that click **configure**



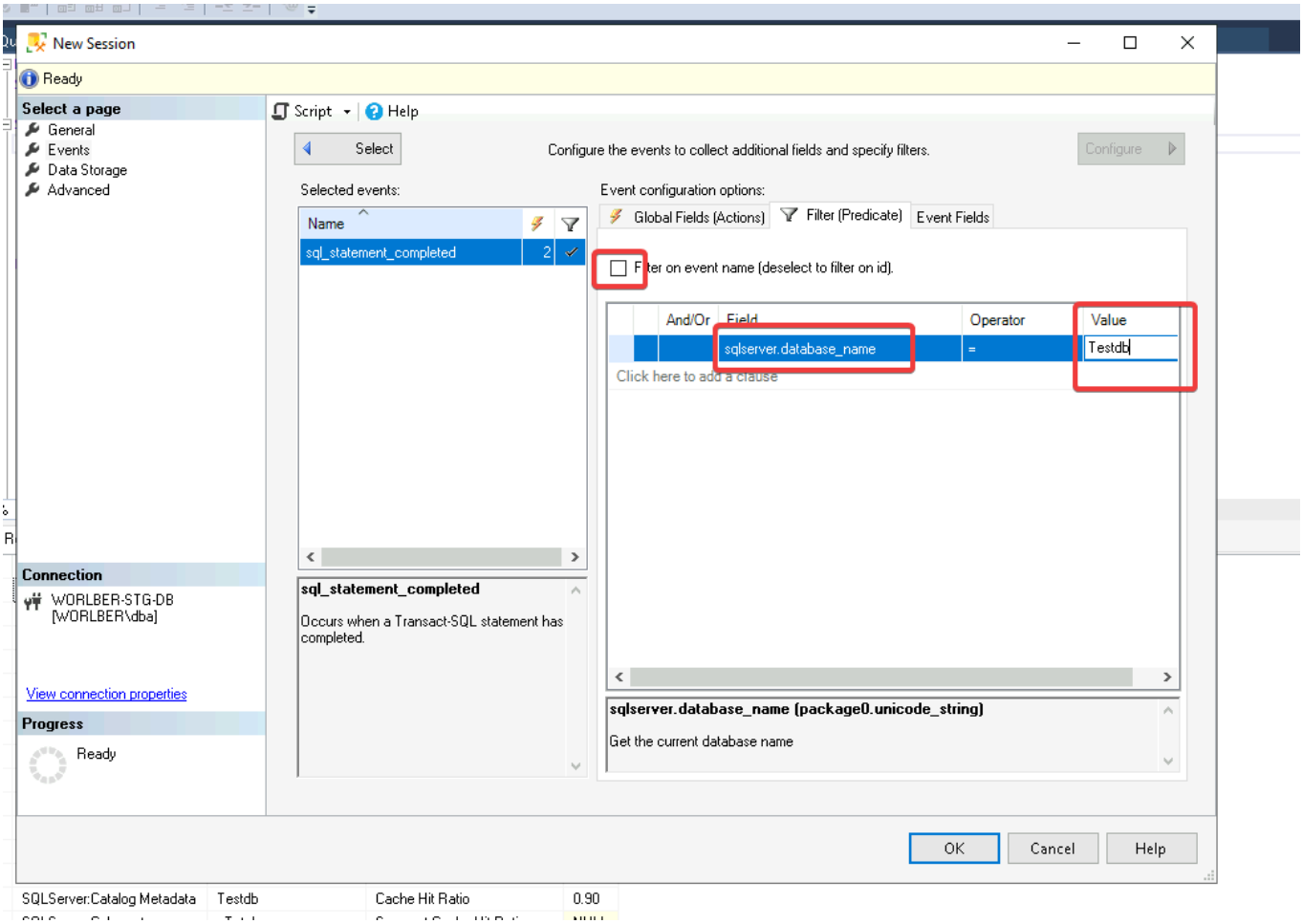
the filed will ask what action you want to record

Event fields are the elements of the events that give information about the specific part of the events.

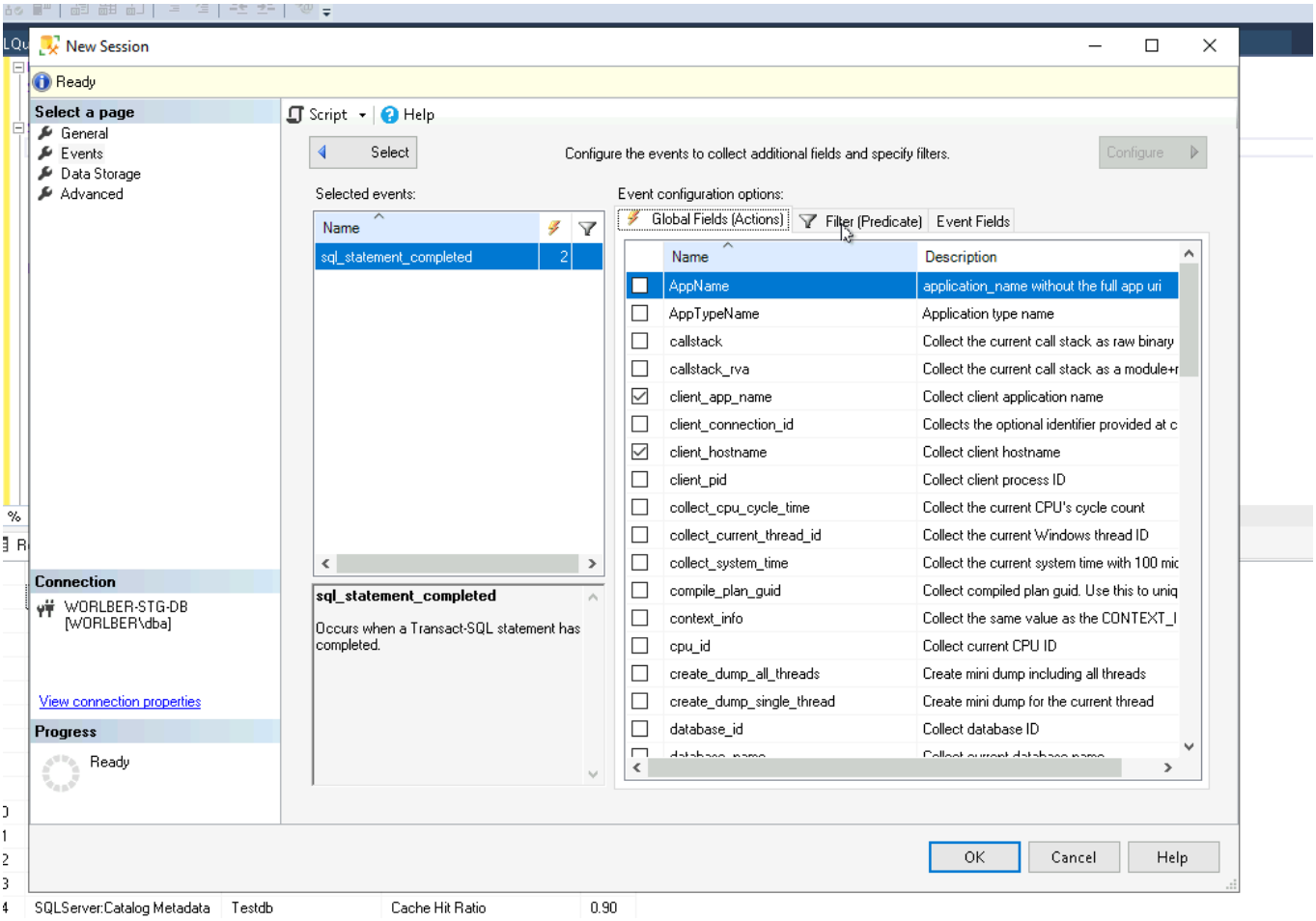
we will select **client_app_name**
and **client_hostname**



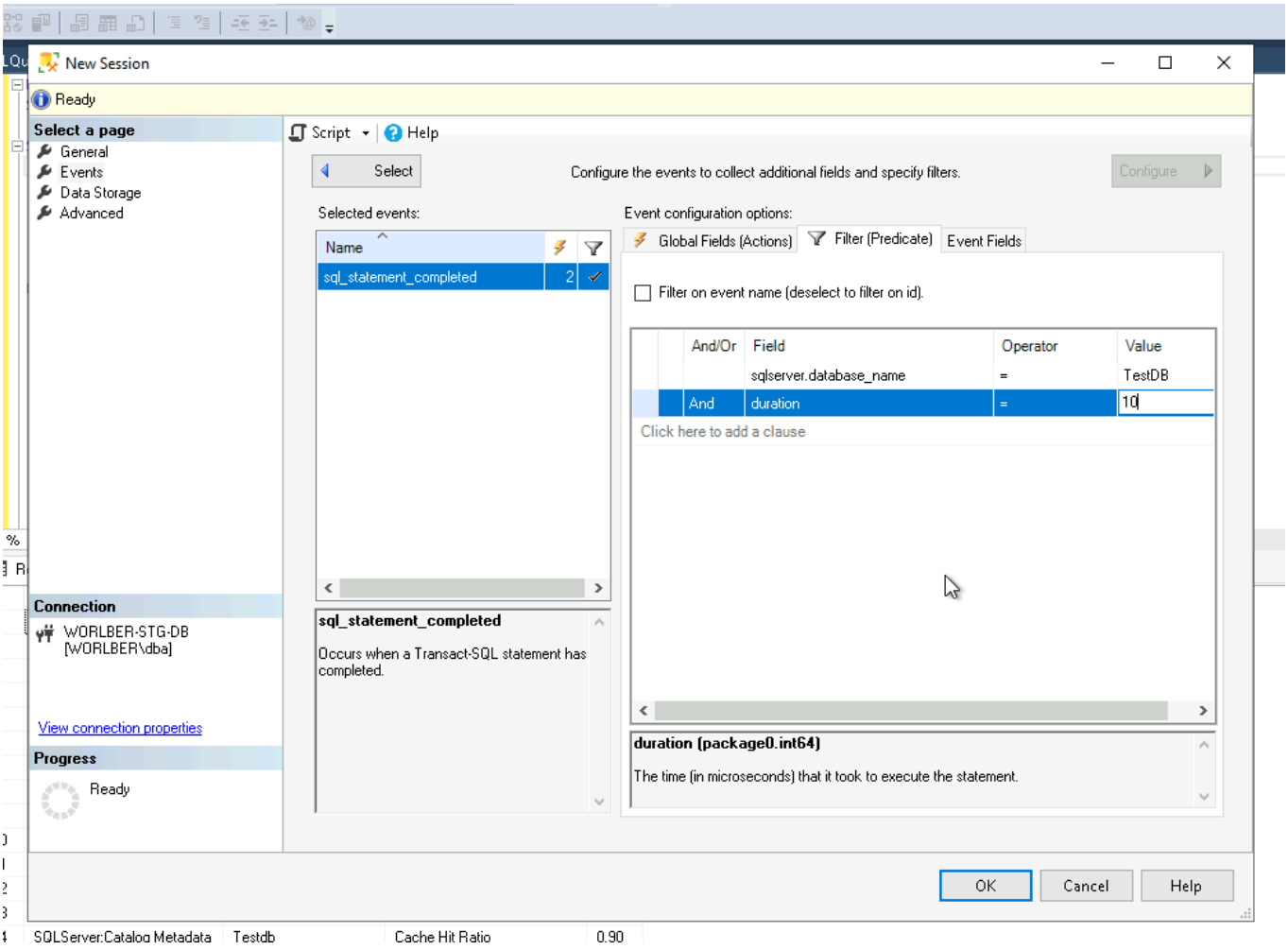
We can filter any specific event field on the Filter (Predicate) tab. For our sample SQL Server extended events session, we filter only the database name so that we will only track the SQL statements that are executed in that database.



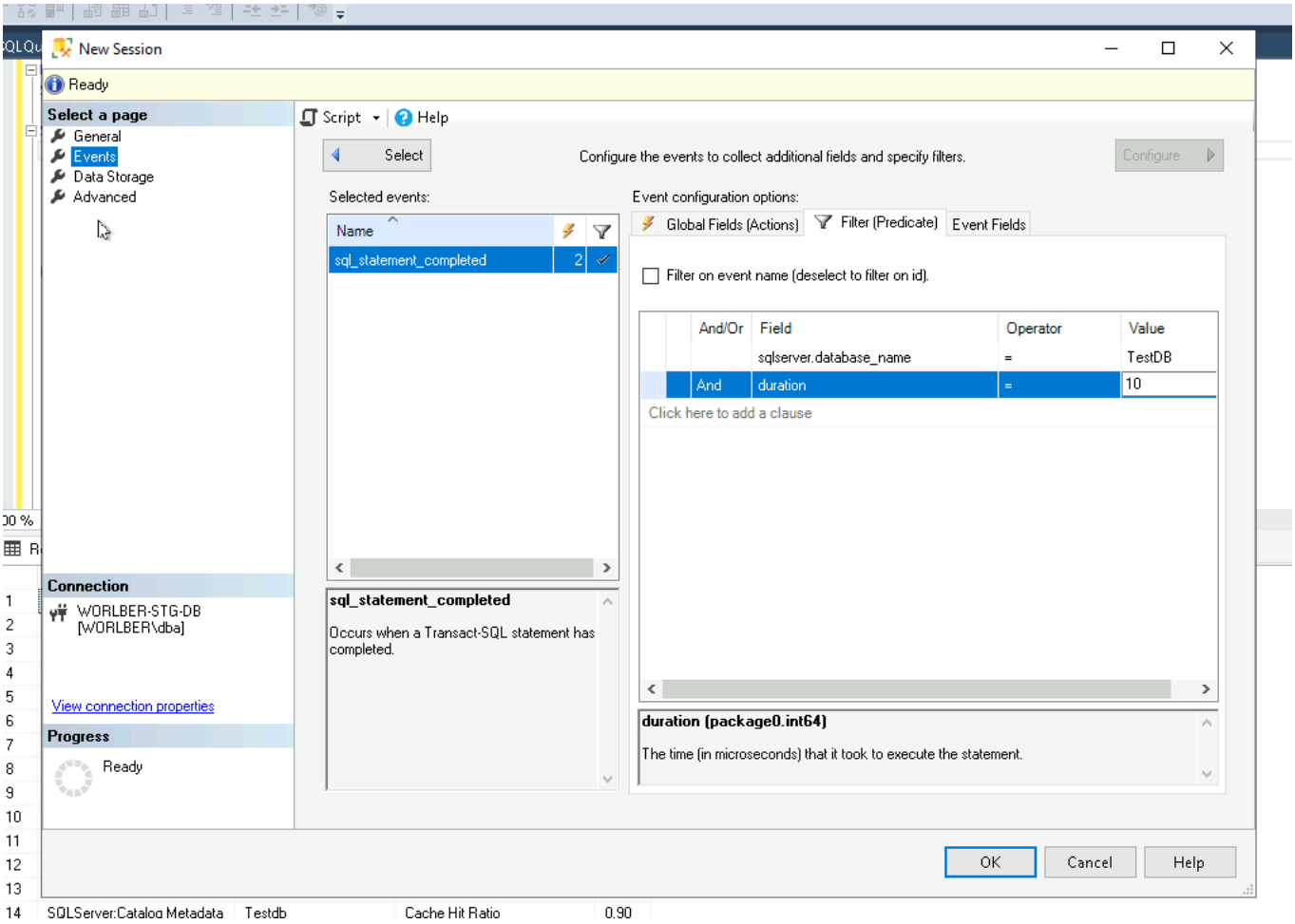
SQLServer:Catalog Metadata	Testdb	Cache Hit Ratio	0.90



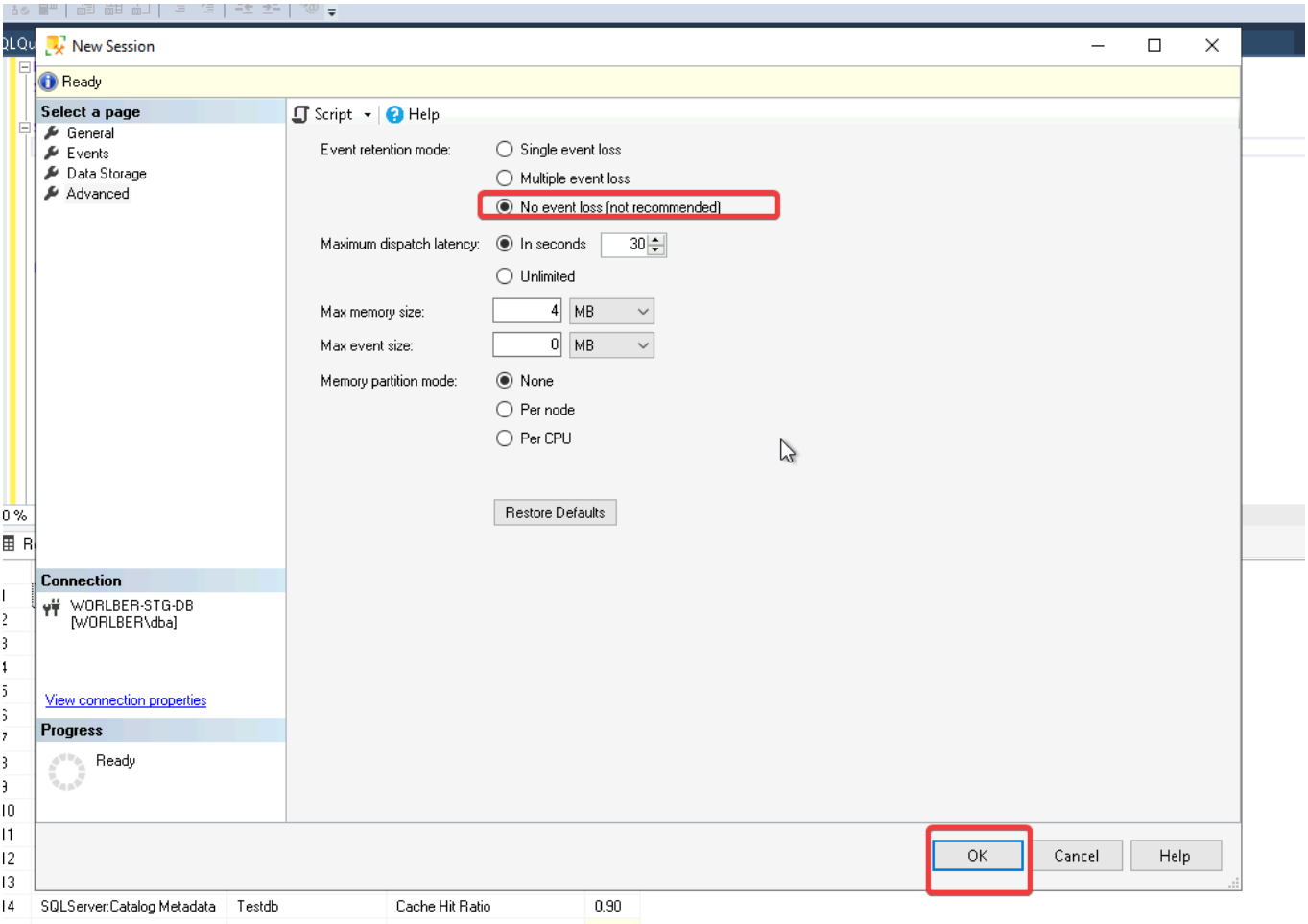
At the same time, we can filter the duration if we have already set a threshold for poor-performing queries. This option can help us troubleshoot query performance problems.



click on data storage in type select event file



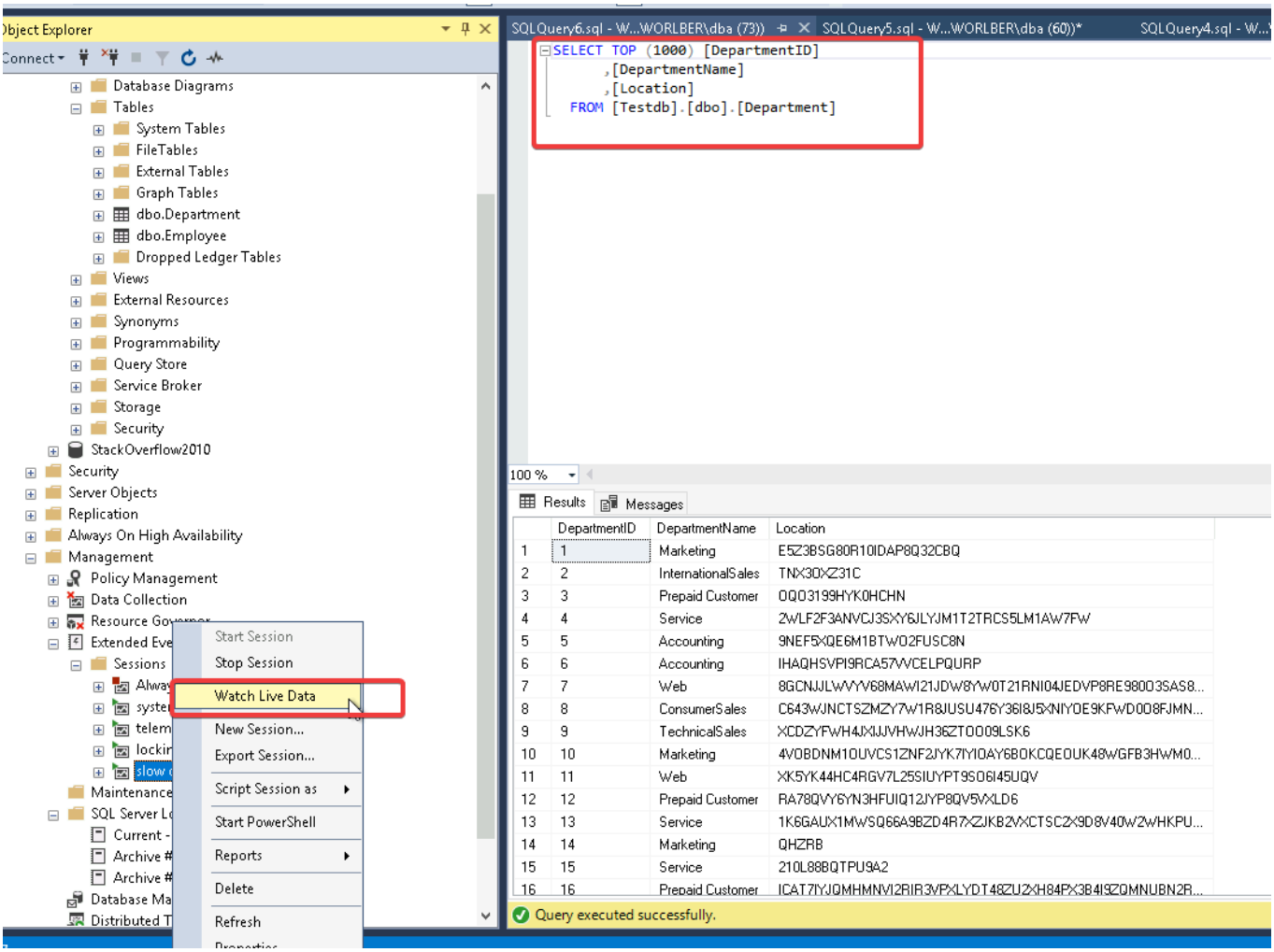
On the Advanced tab, we can find some detailed settings. We can set how much tolerance we have for data loss for the session we created in the event storage mode options. We will choose the No event loss but this option can affect the system performance negatively the databases which have heavy workloads. Another option is Maximum dispatch latency and this option specifies the maximum duration that an event can be kept in the buffer before being written to the target. Finally, we click the OK button and create the extended event.



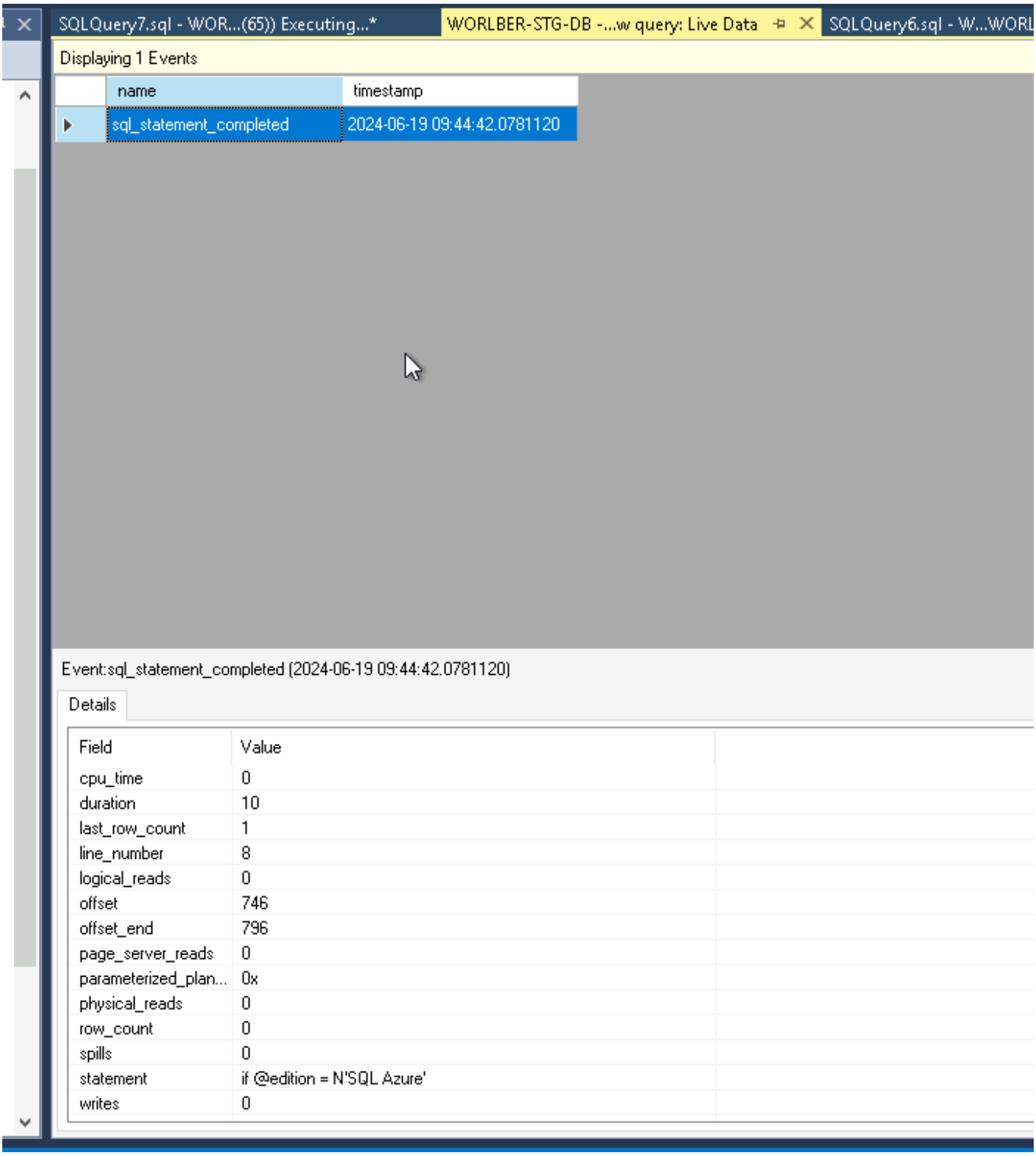
after that we it will start collecting event we can use watch live data to analyze collected data

for examble i will run select on target database

:




```
SELECT  SO.AccountNumber FROM
Sales.SalesOrderHeader SO
INNER  JOIN Sales.SalesOrderDetail SD
ON SO.ModifiedDate = SD.ModifiedDate
```



how to detect the TempDb spills with SQL Server extended events

one of the common issue that any dba will face that cause tempdb to get fill when query optimizer make inaccurate estimation and memeory grant it asked was not enugh , causing optimizer to use tempdb resulting degreed performance

you can use the below that create extedned event that monitor tempdb growth

```
CREATE EVENT SESSION [PublicToilet] ON SERVER
ADD EVENT [sqlserver].[database_file_size_change] (
    ACTION ( [sqlserver].[session_id], [sqlserver].[database_id],
    [sqlserver].[client_hostname], [sqlserver].[sql_text] )
    WHERE ( [database_id] = ( 2 )
            AND [session_id] > ( 50 ) ) ),
ADD EVENT [sqlserver].[databases_log_file_used_size_changed] (
    ACTION ( [sqlserver].[session_id], [sqlserver].[database_id],
    [sqlserver].[client_hostname], [sqlserver].[sql_text] )
    WHERE ( [database_id] = ( 2 )
            AND [session_id] > ( 50 ) ) )
ADD TARGET [package0].[asynchronous_file_target] (  SET filename =
N'c:\temp\publictoilet.xel' ,
                                                    metadatafile =
N'c:\temp\publictoilet.xem' ,
                                                    max_file_size = ( 10 ) ,
                                                    max_rollover_files = 10
)
WITH (  MAX_MEMORY = 4096 KB ,
        EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS ,
        MAX_DISPATCH_LATENCY = 1 SECONDS ,
        MAX_EVENT_SIZE = 0 KB ,
        MEMORY_PARTITION_MODE = NONE ,
        TRACK_CAUSALITY = ON ,
        STARTUP_STATE = ON );
GO

ALTER EVENT SESSION [PublicToilet] ON SERVER STATE = START;
```

we can use the below query to test the extedned event
requet database stackoverflow to be in the instance

```
use tempdb

CREATE TABLE [#Users](
  [Id] [int] NULL,
  [Age] [int] NULL,
  [DisplayName] [nvarchar](40) NULL,
  [AboutMe] [nvarchar](max) NULL
```

```
  )

DECLARE @i INT = 0;
DECLARE @rc VARCHAR(20) = 0

WHILE @i < 3
      BEGIN

            INSERT  [#Users]
                    ( [Id] ,
                      [Age] ,
                      [DisplayName] ,
                      [AboutMe] )
            SELECT TOP ( 75000 )
                    [Id] ,
                    [Age] ,
                    [DisplayName] ,
                    [AboutMe]
            FROM    [StackOverflow2010].[dbo].[Users];

            UPDATE  [u1]
            SET     [u1].[DisplayName] = SUBSTRING(CAST(NEWID() AS
NVARCHAR(MAX)), 0, 40) ,
                    [u1].[Age] = ( [u2].[Age] * 2 ) ,
                    [u1].[AboutMe] = REPLACE([u2].[AboutMe], 'a',
                                             CAST(NEWID() AS NVARCHAR(MAX)))
            FROM    [#Users] [u1]
            CROSS JOIN    [#Users] [u2];

      SET @rc = @@ROWCOUNT
      RAISERROR ('Current # rows %s', 0, 1, @rc) WITH NOWAIT

            SET @i += 1;
      RAISERROR ('Current # runs %i', 0, 1, @i) WITH NOWAIT
      END;

          ```

![df0a80e8358ee0bb10fbbcf538781812.png]
(../../_resources/df0a80e8358ee0bb10fbbcf538781812.png)
```
```

6-query plan

${toc}

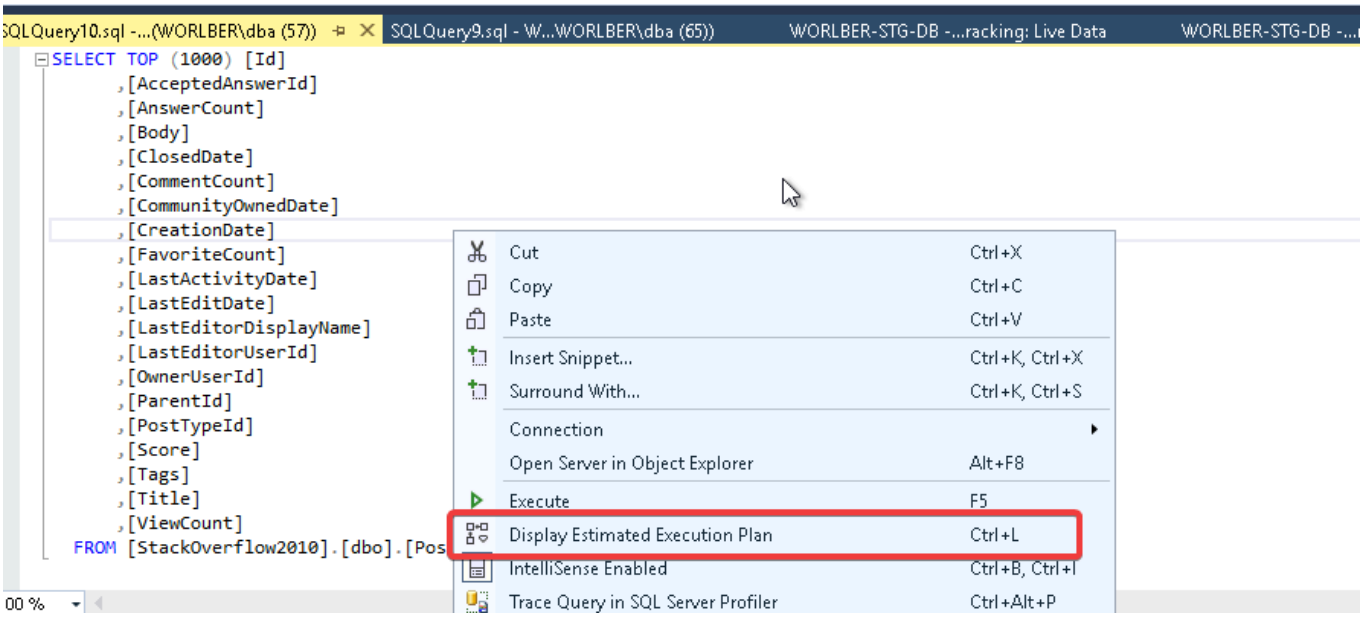
Query or execution plan come in three flavors :

- **Estimated execution plan** : the initial query plan the optimizer engine comes up with
- **Actual execution plan** : hoe the query ended up being executed
- **Live query statistics** : display in real-time how mush work is involved in each step of the querys execution

in the majority of the case both Estimated execution plan and Actual execution plan will be the same .

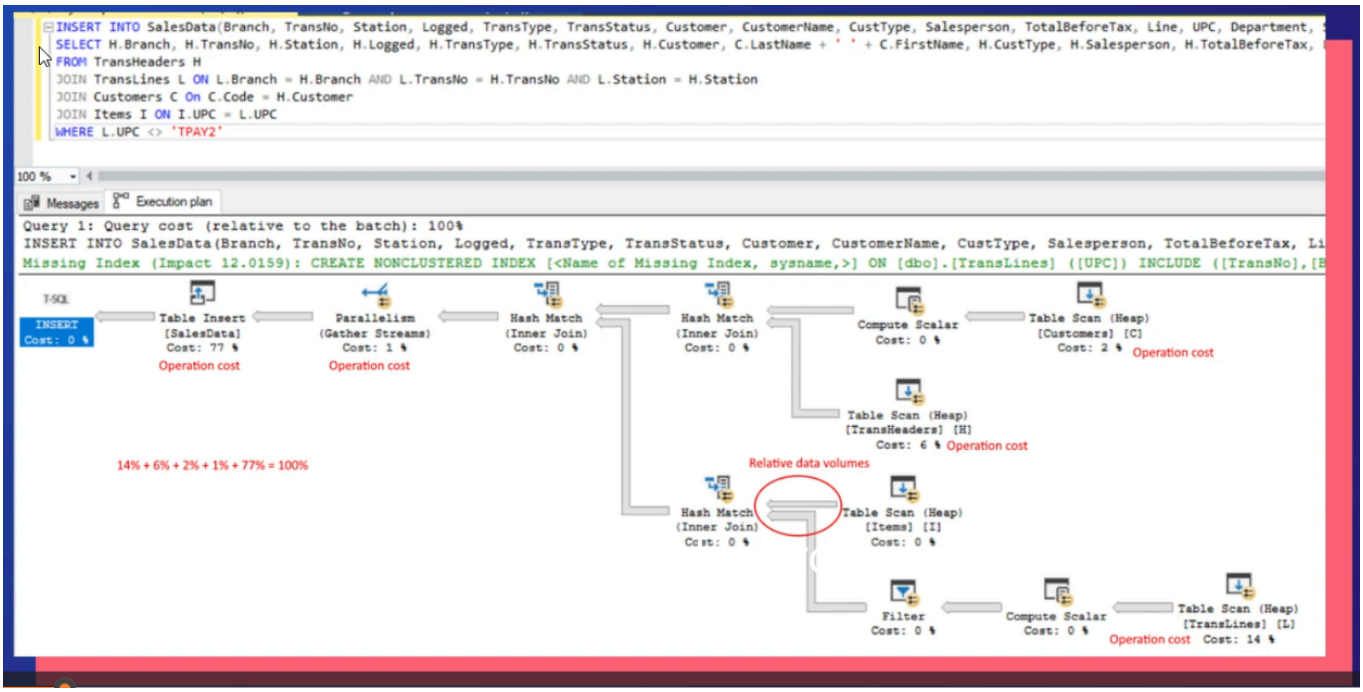
Estimated execution plan

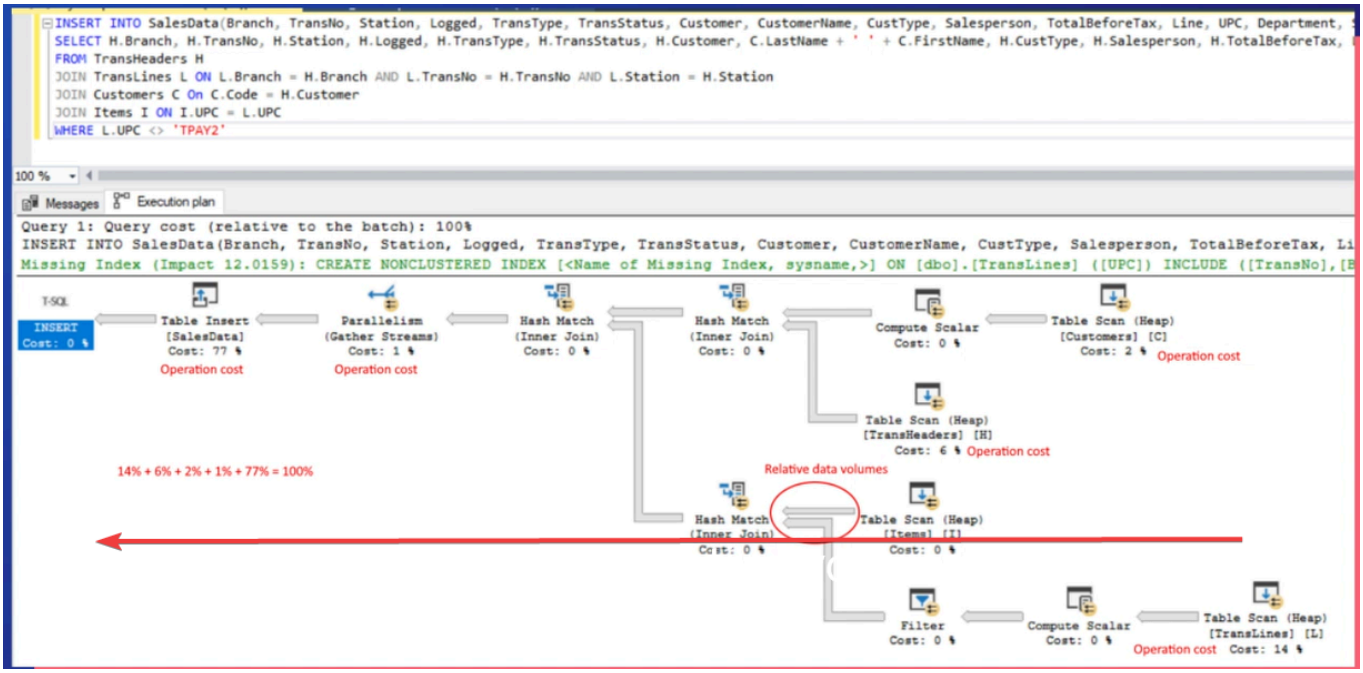
Within ssms we can right click on query and selecting **display estimated execution plan**
Or press Ctrl+L
This will veiw the plan without executing



below is example of execution plan

the plan should be read from right to left

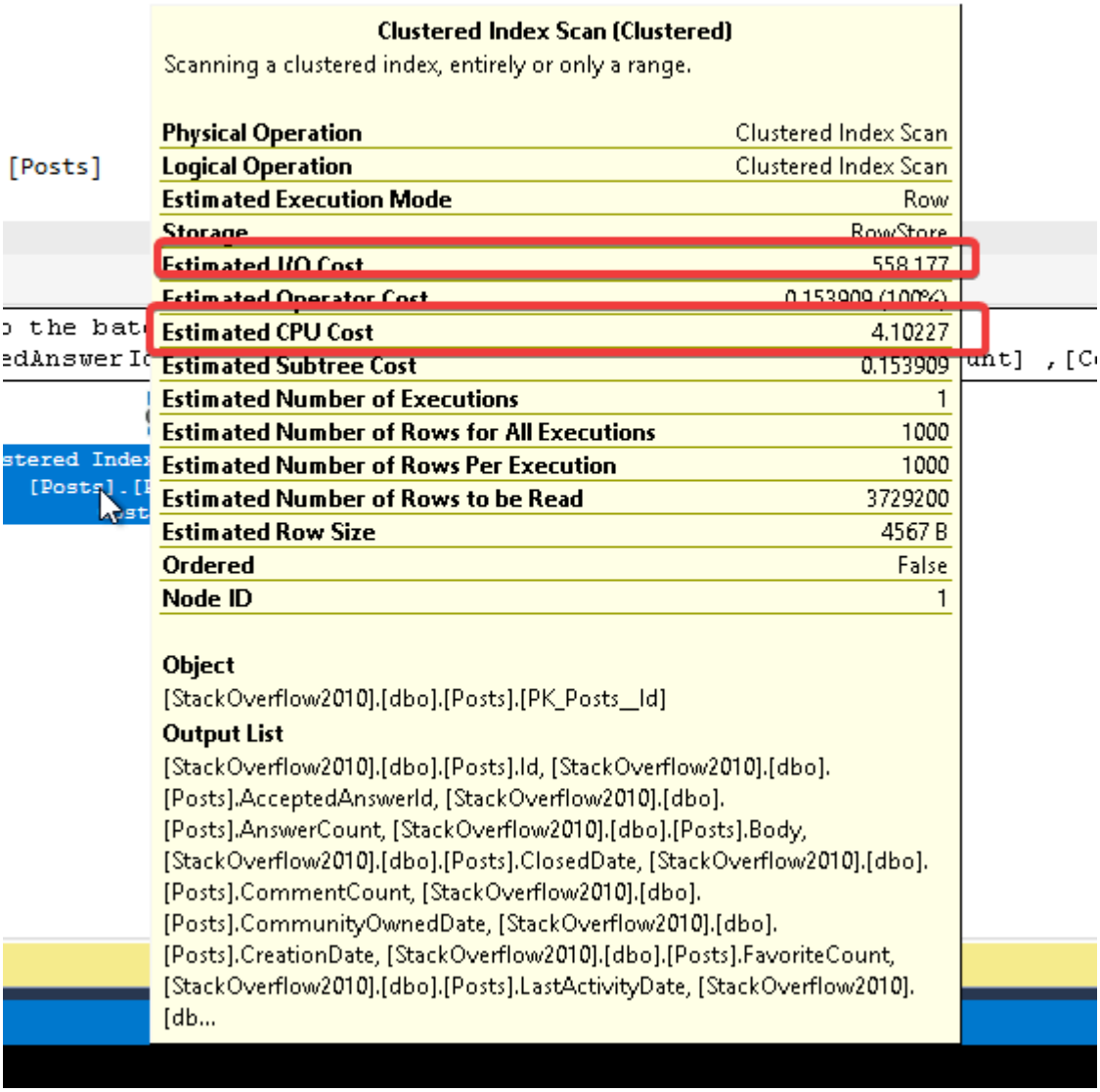




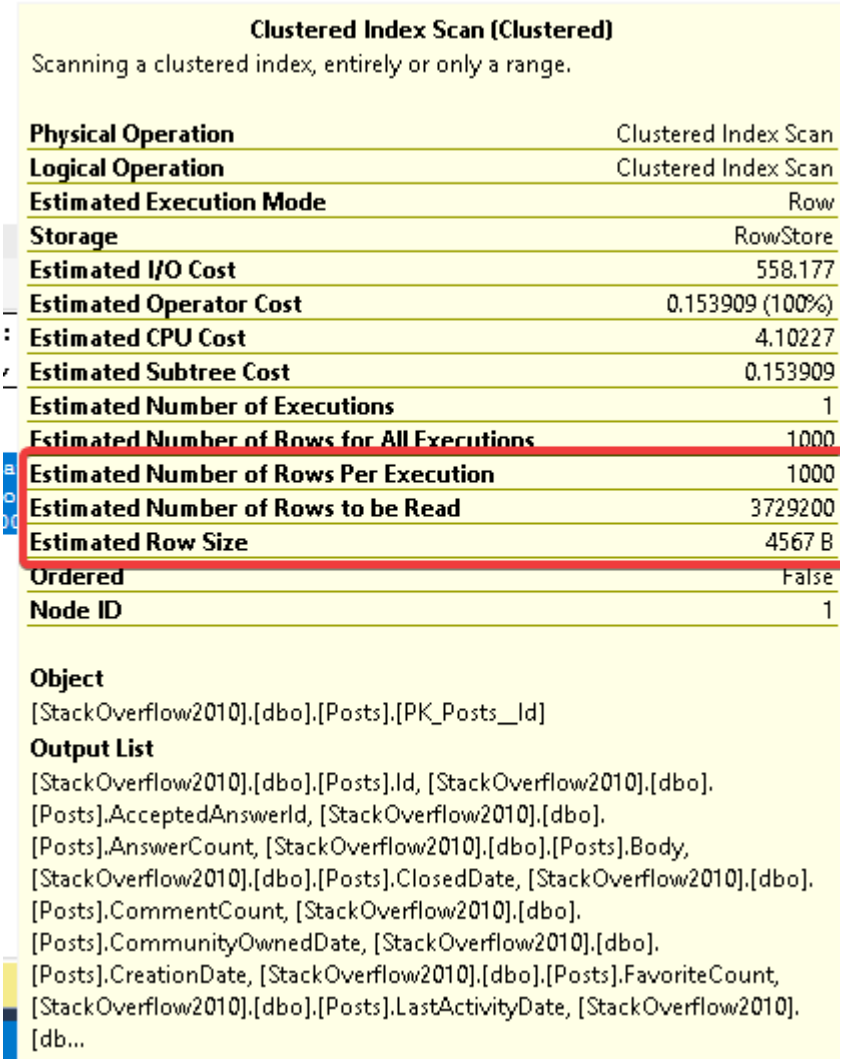
each operation is relative cost overall query execution

this help check which aspect of the query is most expensive

if you click on operation you will be able to check i/o cpu cost



also the number of rows to be read at the bottom



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	558.177
Estimated Operator Cost	0.153909 (100%)
Estimated CPU Cost	4.10227
Estimated Subtree Cost	0.153909
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	1000
Estimated Number of Rows Per Execution	1000
Estimated Number of Rows to be Read	3729200
Estimated Row Size	4567 B
Ordered	False
Node ID	1

Object

[StackOverflow2010].[dbo].[Posts].[PK_Posts__Id]

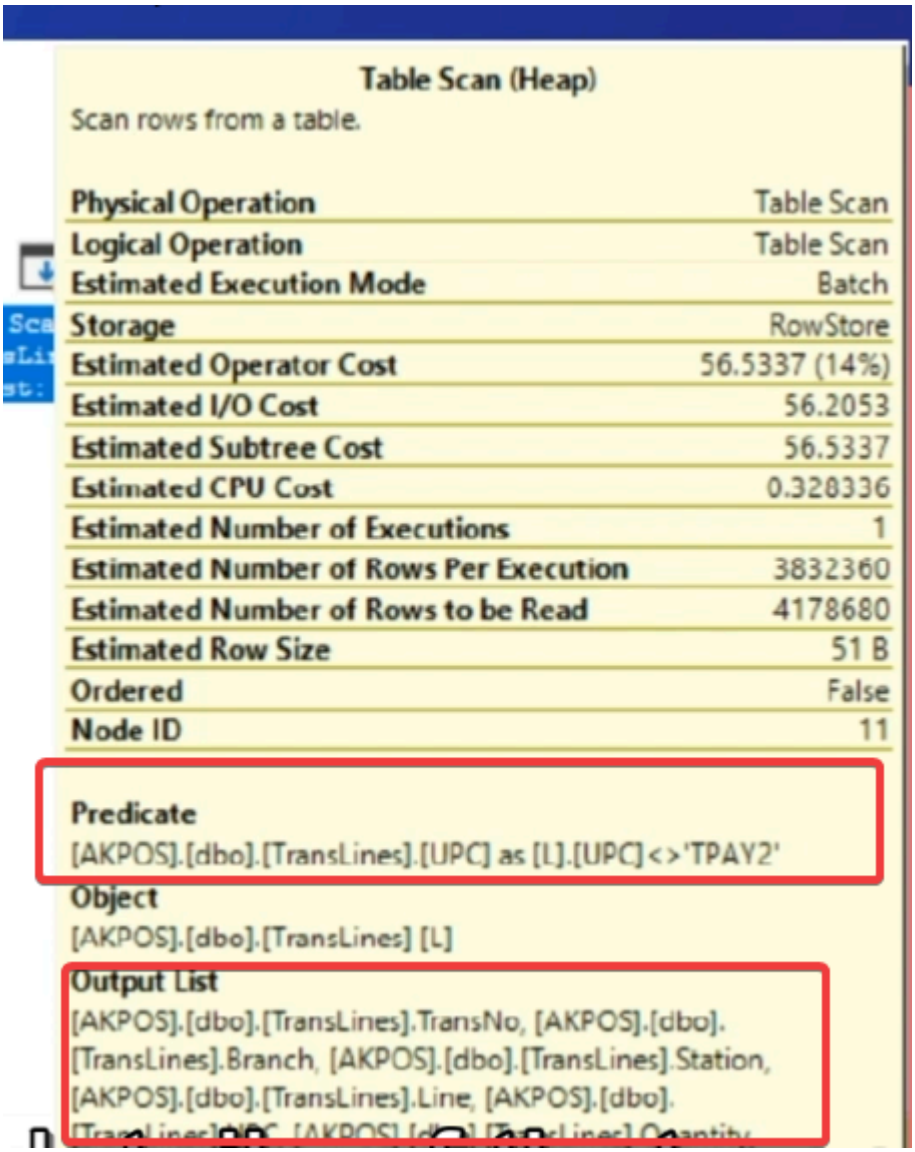
Output List

[StackOverflow2010].[dbo].[Posts].Id, [StackOverflow2010].[dbo].[Posts].AcceptedAnswerId, [StackOverflow2010].[dbo].[Posts].AnswerCount, [StackOverflow2010].[dbo].[Posts].Body, [StackOverflow2010].[dbo].[Posts].ClosedDate, [StackOverflow2010].[dbo].[Posts].CommentCount, [StackOverflow2010].[dbo].[Posts].CommunityOwnedDate, [StackOverflow2010].[dbo].[Posts].CreationDate, [StackOverflow2010].[dbo].[Posts].FavoriteCount, [StackOverflow2010].[dbo].[Posts].LastActivityDate, [StackOverflow2010].[db...

Also you will see **predicate** which related to where clause

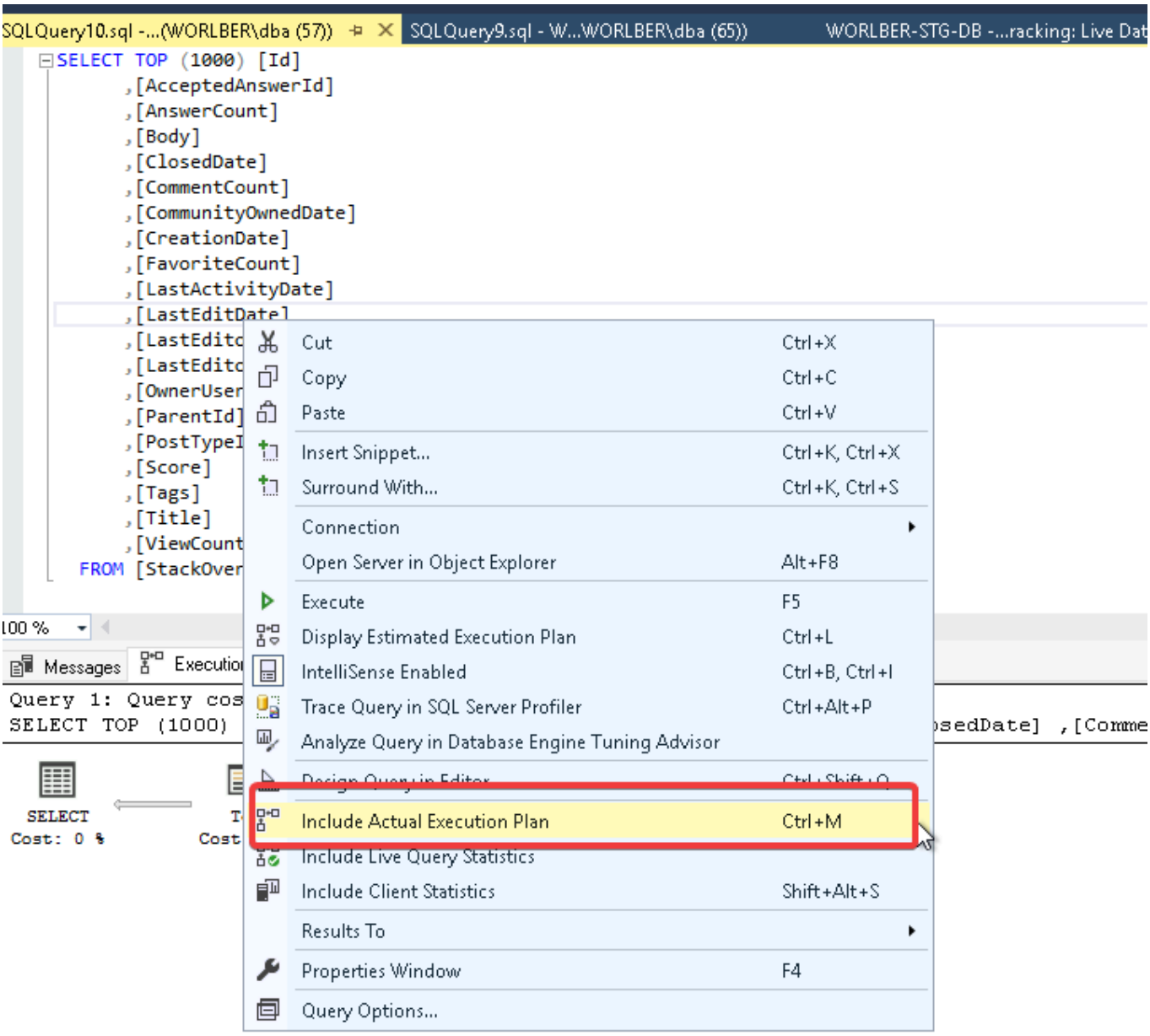
The object refer to table that used by this operation

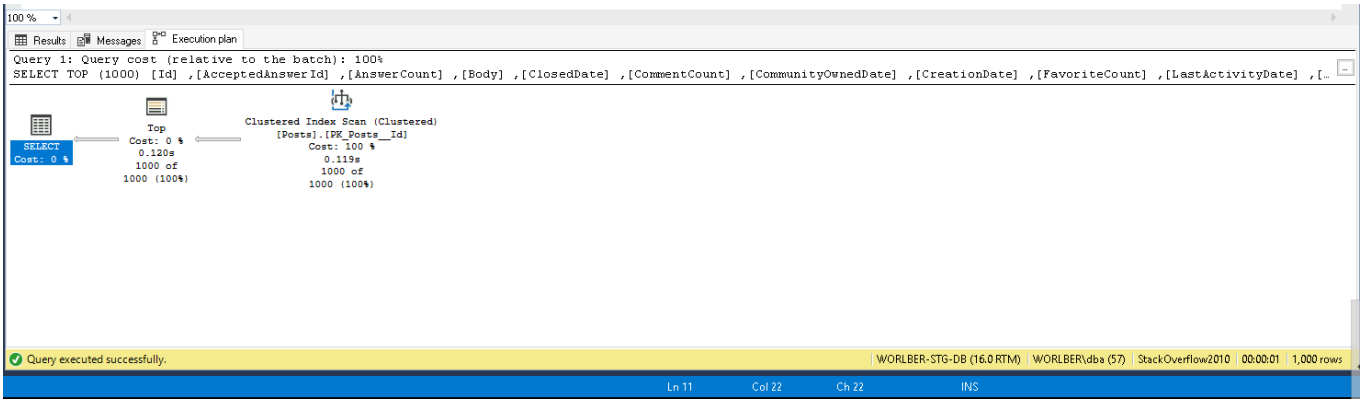
And output list of column done by operation



Actual execution plan

You can view actual execution plan either by right clicking and select **include actual execution plan**
Or click **Ctrl+M**

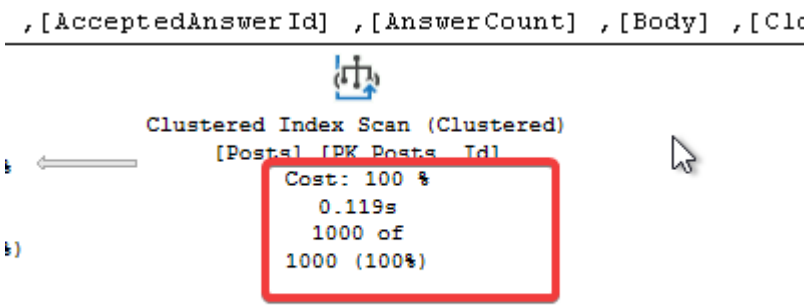




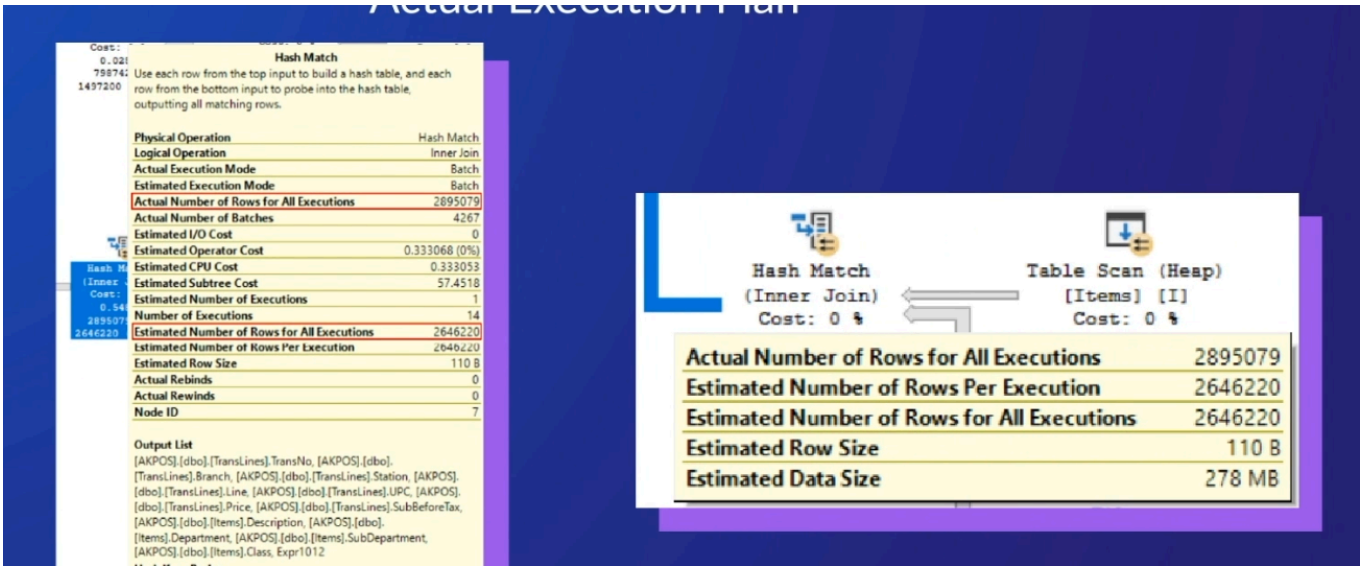
most of the time Actual execution plan is the same as estimated executing plan

When we see Actual execution plan we get the cost calculated my seconds

And shows number of row as percentage of the estimated number



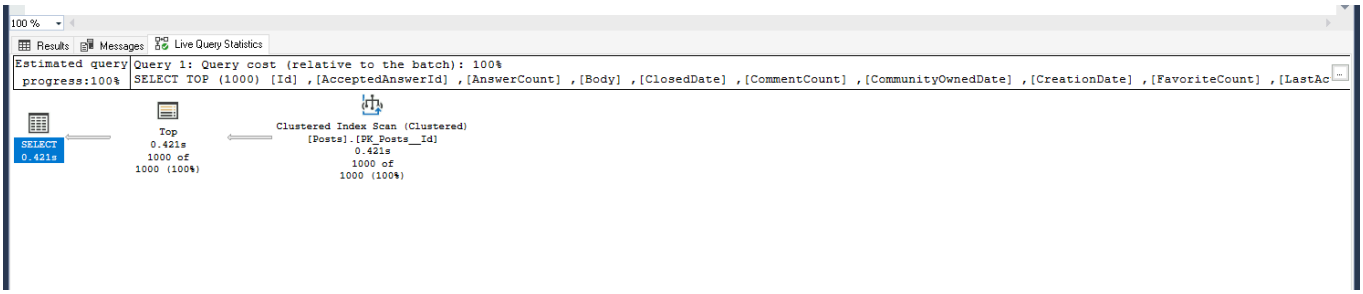
If you find significant variation between the estimated and actual rows in operation this may indicate problem with statistic



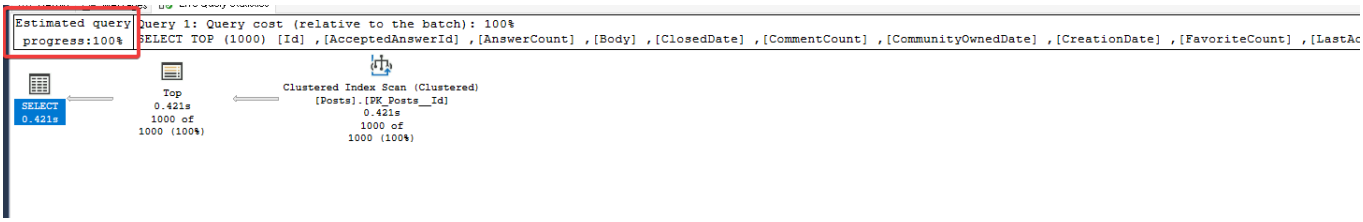
Live query statistics

to view live query statics by right clicking and select include live query statistics

the query will come alive with animation and real time update of the operation as they progress

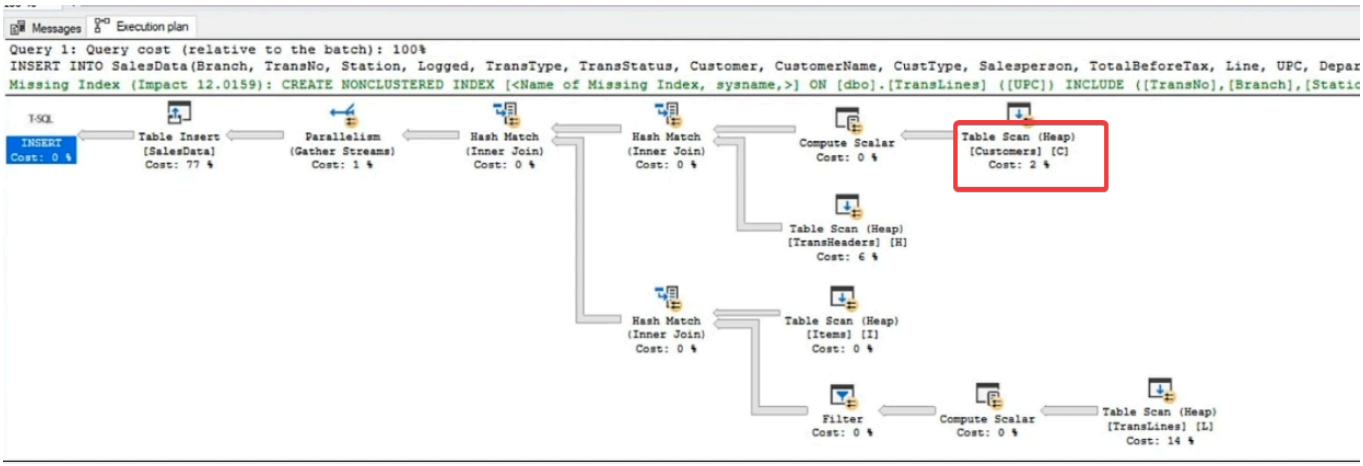


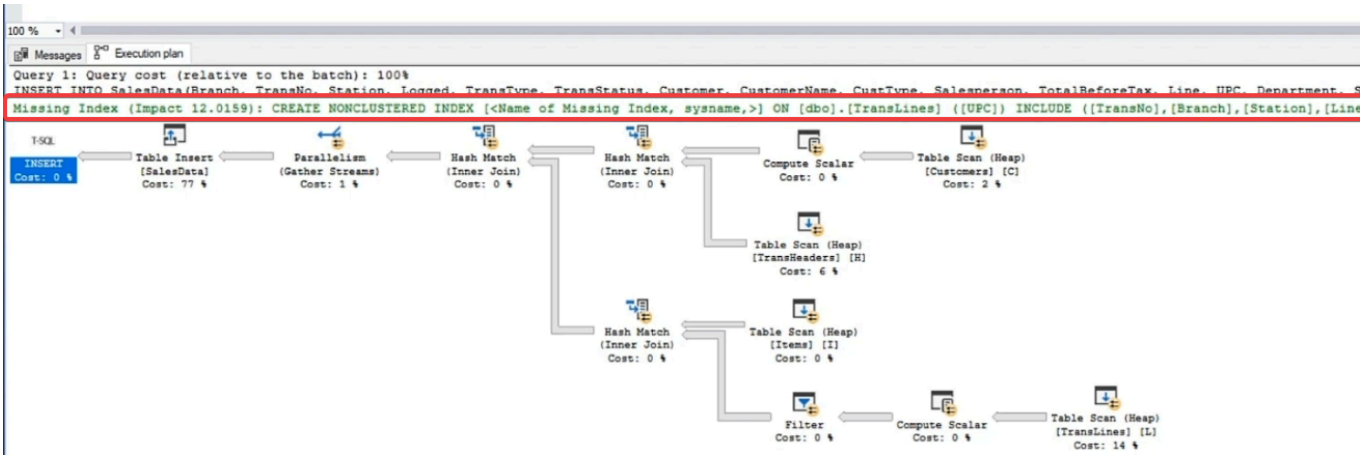
on the left you will see estimate query progress in percentage



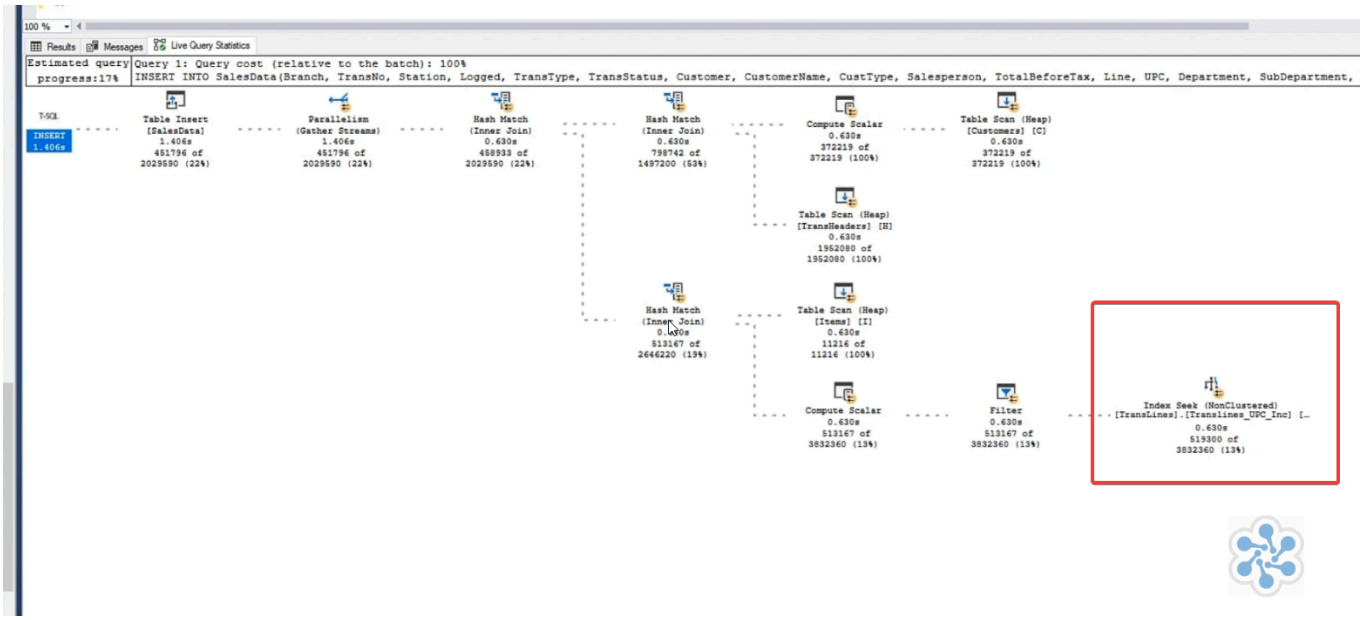
analyze query plan

if you look at the below plan one first thing you will notice is that table start with table scan meaning it going to read the whole table





if you apply the sugested index and rerun the query we notice the query start with index seek which good thing


100 %
Results
Messages
Live Query Statistics
Estimated query progress:17%
Query 1: Query cost (relative to the batch): 100%
INSERT INTO SalesData(Branch, TransNo, Station, Logged, TransType, TransStatus, Customer, CustomerName, CustType, Salesperson, TotalBeforeTax, Line, UPC, Department, SubDepartment,
T-SQL
INSERT
1.406s
Table Insert
[SalesData]
1.406s
451736 of
2029590 (22%)
Parallelism
(Gather Streams)
1.406s
451736 of
2029590 (22%)
Hash Match
(Inner Join)
0.630s
458933 of
2029590 (22%)
Hash Match
(Inner Join)
0.630s
798742 of
1497200 (53%)
Compute Scalar
0.630s
372219 of
372219 (100%)
Table Scan (Heap)
[Customers] [C]
0.630s
372219 of
372219 (100%)
Table Scan (Heap)
[TransHeaders] [H]
0.630s
1952080 of
1952080 (100%)
Hash Match
(Inner Join)
0.630s
513167 of
2646220 (19%)
Table Scan (Heap)
[Items] [I]
0.630s
11216 of
11216 (100%)
Compute Scalar
0.630s
513167 of
3832360 (13%)
Filter
0.630s
513167 of
3832360 (13%)
Index Seek (NonClustered)
[TransLines].[Translines_UPC_Inc] [...
0.630s
519300 of
3832360 (13%)

7. transaction isolation levels

SQL Server isolation levels are used to define the degree to which one transaction must be isolated from resource or data modifications made by other concurrent transactions. The different Isolation Levels are:

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable
- Snapshot
  the default one is read commeted

If you do not want the SELECT statement to acquire locks that prevent other transactions from writing data, you can set the isolation level to Read Uncommitted. In this isolation level, SELECT statements do not place shared locks on the data they read, and they do not honor exclusive locks set by other transactions.

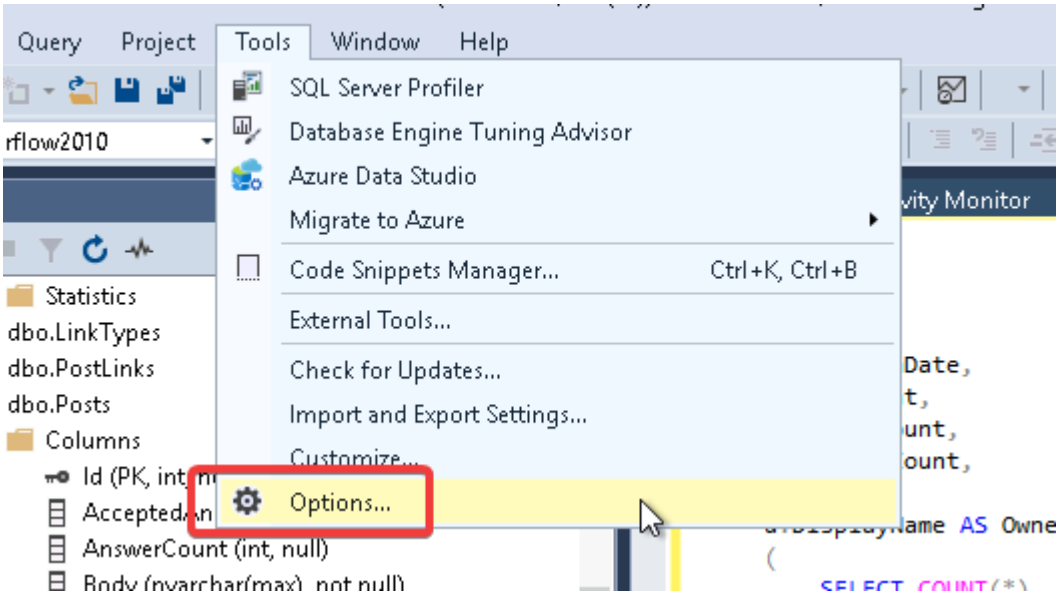
Changing the Isloation Level From Query

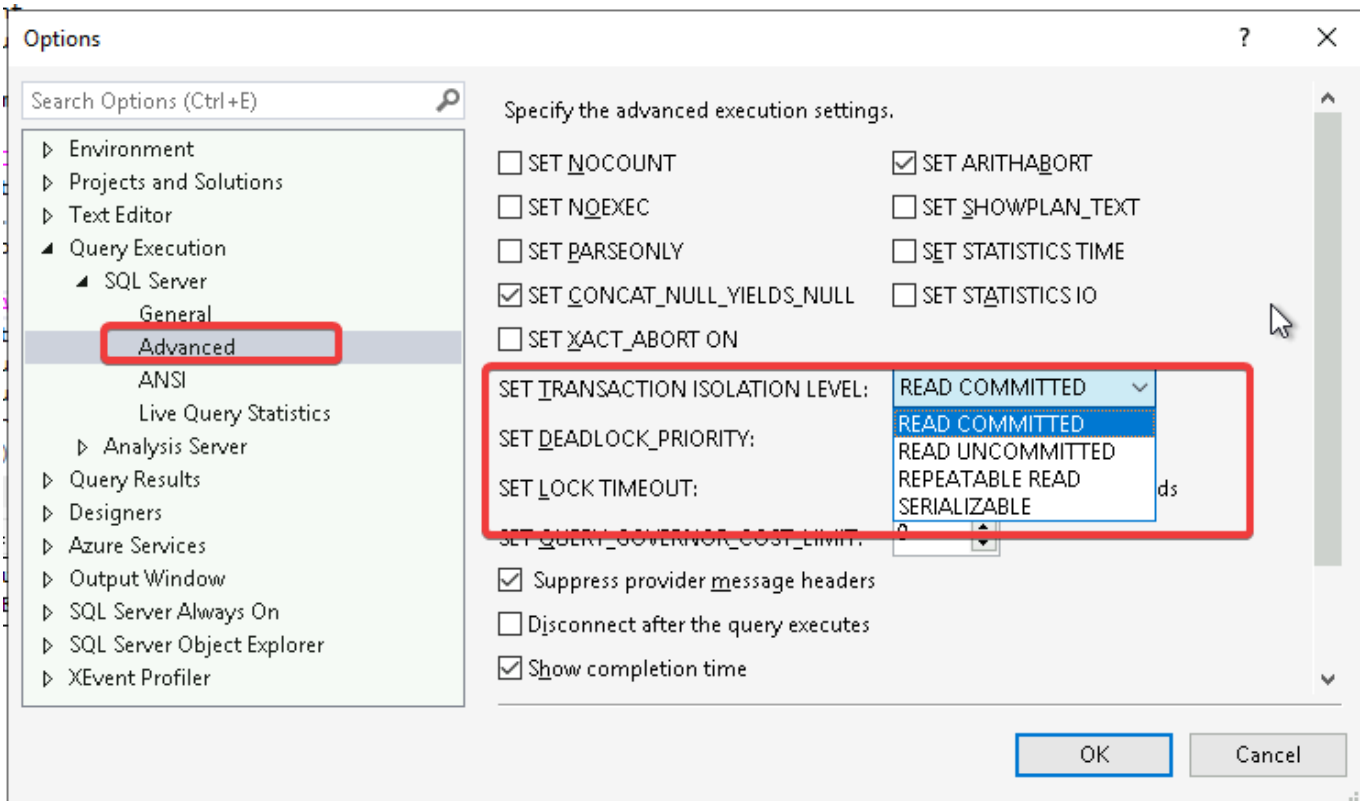
You can specify the Isloation level while performing a transaction

Syntax

```
SET TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED
    | READ COMMITTED
    | REPEATABLE READ
    | SNAPSHOT
    | SERIALIZABLE
    }
```

Or from SSMS by going to tools option the expanding query execution , then expand SQL server , then selecting advance





prerequisites

BEGIN TRANSACTION
INSERT INTO Exam

below script we will create table and insert value for tetsing porpuse

```
CREATE TABLE Dept
(
```

```
	deptId INT PRIMARY KEY,
	deptName VARCHAR(100),
	deptDesc VARCHAR(300)
)
INSERT INTO Dept
(deptId,deptName,deptDesc)
VALUES
(101,'Computer Science And Engineering','UG and PG Courses in Computer
Science And Engineering'),
(102,'Electronics And TeleCommunications','UG and PG Courses in Electronics
And TeleCommunications')
CREATE TABLE Exam
(
	examId INT PRIMARY KEY,
	examName VARCHAR(100),
	examDesc VARCHAR(300)
)
INSERT INTO Exam
(examId,examName,examDesc)
VALUES
(201,'Data Structure','Theory Paper and Lab in Data Structure'),
(202,'Database Management System','Theory Paper and Lab in Database
Management System')
--create the table
CREATE TABLE StudentMarks
(
	studentId INT IDENTITY(1,1) PRIMARY KEY,
	deptId INT,
	examId INT,
	marksObtained INT
)
SELECT COUNT(1) FROM StudentMarks
--2397616
--Insert record in StudentMarks repeatedly
INSERT INTO StudentMarks (deptId,examId,marksObtained)
VALUES
(101,201,95)
--duplicate the records to increase the count
INSERT INTO StudentMarks (deptId,examId,marksObtained)
SELECT deptId,examId,marksObtained FROM StudentMarks
```

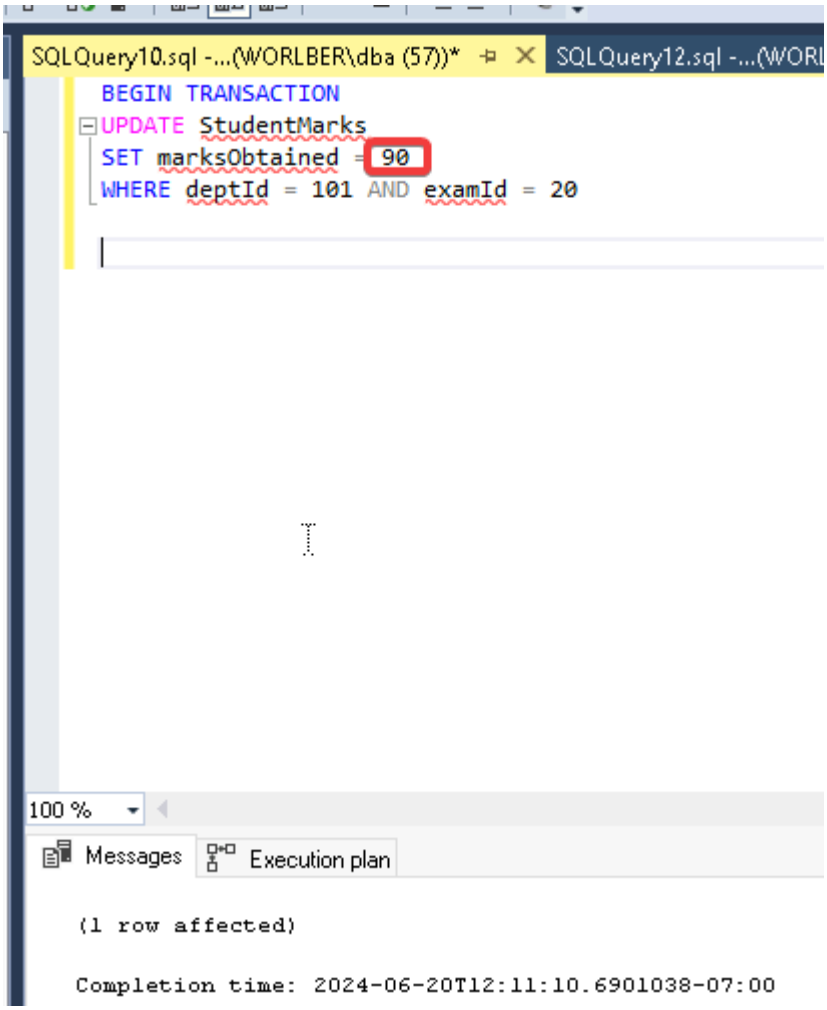
Read Committed

transactions issue exclusive locks at the time of data modification, thus not allowing other transactions to read the modified data that is not yet committed

we will have two query will do update on table another transaction for selecting and we will see the result

Transaction 1
we will update table but we will not commit the transaction at the end

```
BEGIN TRANSACTION
UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 201
```

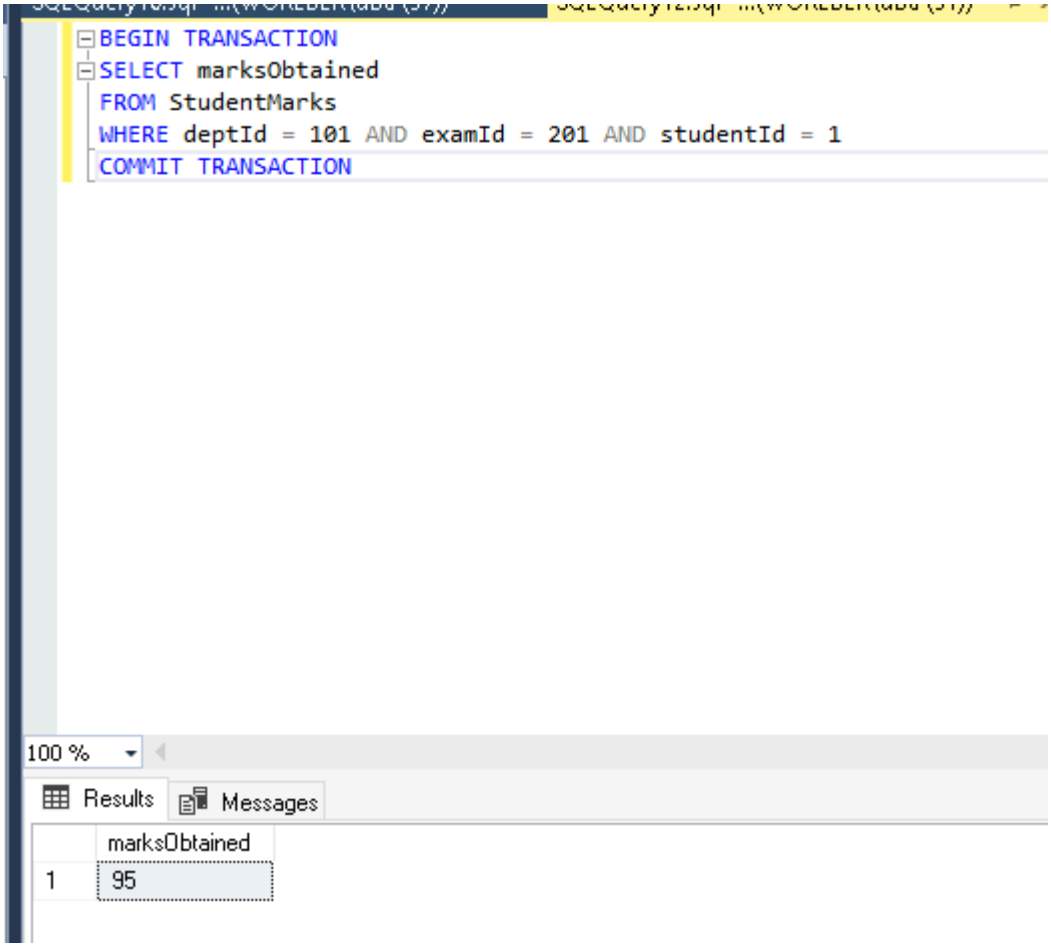


we update value to 90 and if we do select to see the new value we can see it

Transaction 2

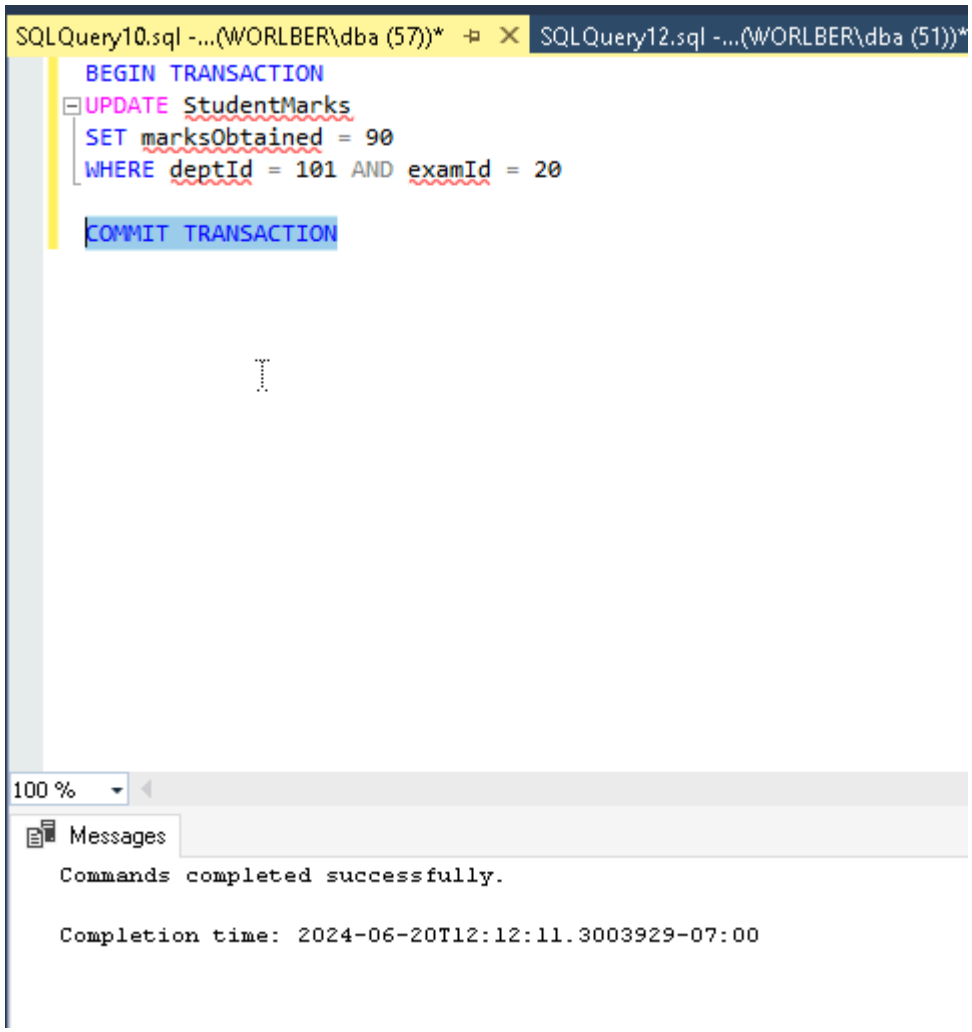
```
BEGIN TRANSACTION
SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1
COMMIT TRANSACTION
```

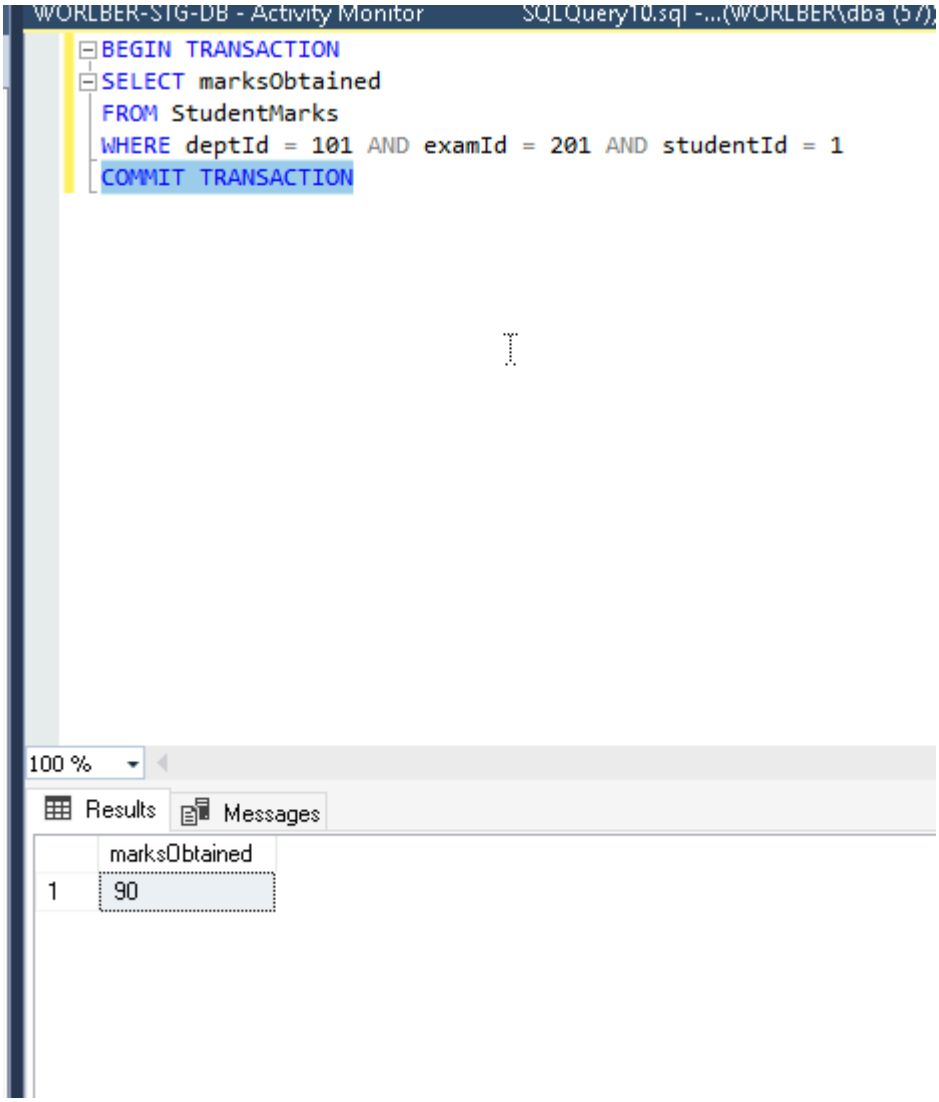
the transaction 2 is not reading the new value is still reading old 95 value because transaction is not commit and transaction 2 will only see the new value once transaction is commit



if we commit transaction on we can see the new value on transaction 2

```
COMMIT TRANSACTION
```


SQLQuery10.sql -...(WORLBER\dba (57))*
SQLQuery12.sql -...(WORLBER\dba (51))*
BEGIN TRANSACTION
UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 20
COMMIT TRANSACTION
100 %
Messages
Commands completed successfully.
Completion time: 2024-06-20T12:12:11.3003929-07:00



Read Uncommitted

Transactions running at this level do not issue shared locks to prevent other transactions from modifying data read by the current transaction. Also, transactions are not blocked by exclusive locks at the time of data modification, thus allowing other transactions to read the modified data which is not yet committed.

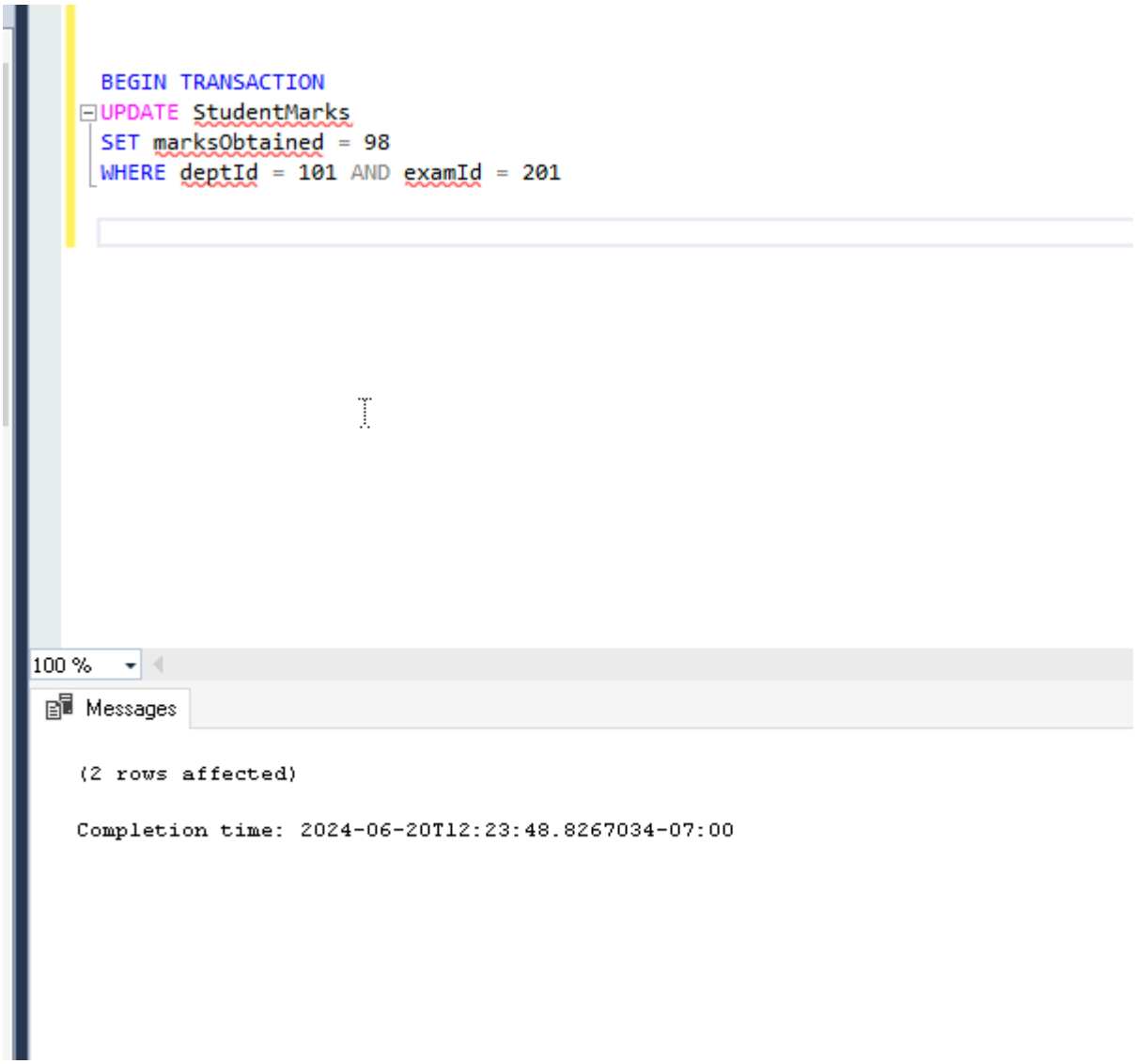
we will first use the below query which will start transaction one

but we will set transaction isolation level to READ UNCOMMITTED

Transaction 1

```
SET TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED}

BEGIN TRANSACTION
```

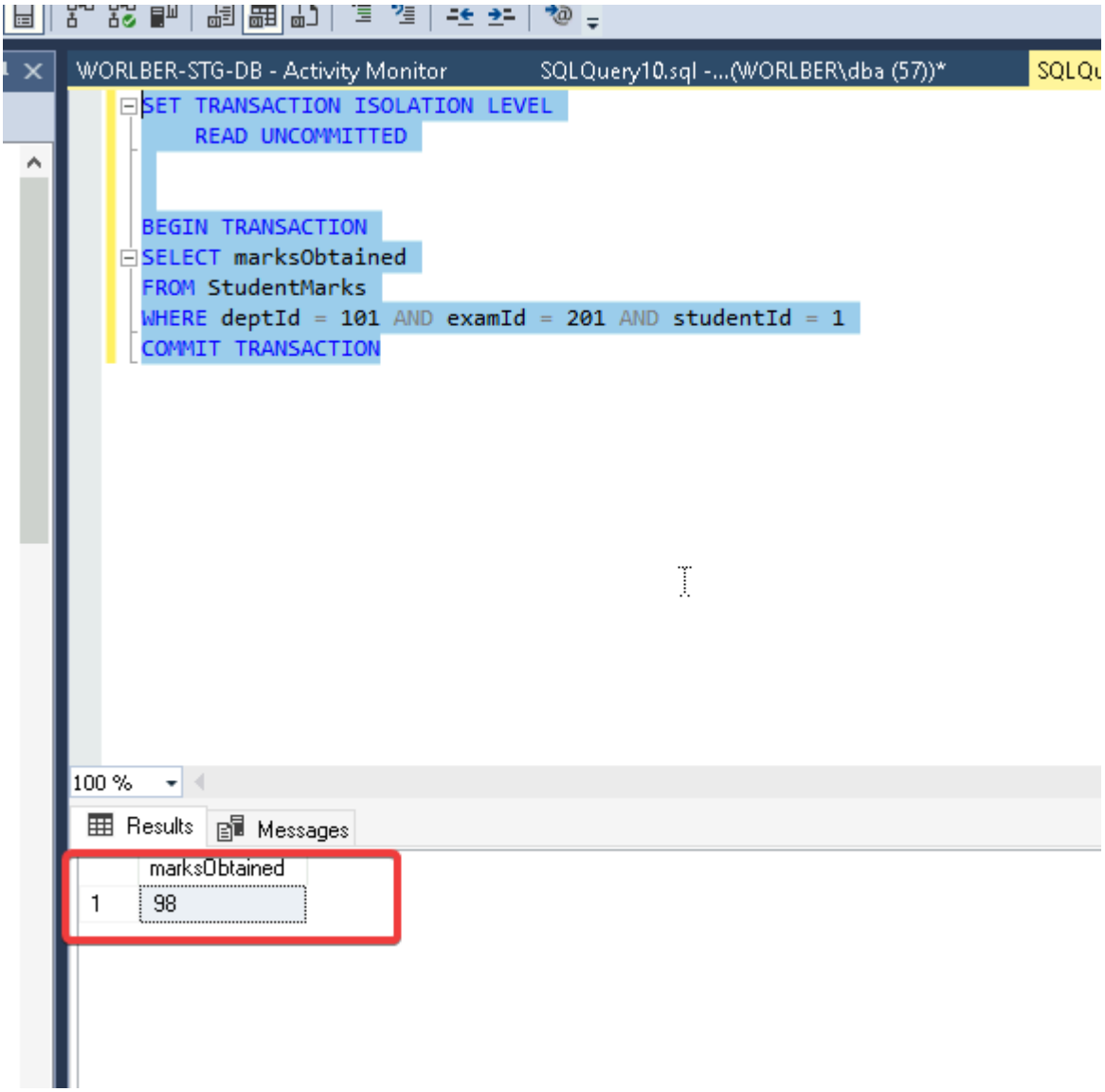
```sql
UPDATE StudentMarks
SET marksObtained = 90
WHERE deptId = 101 AND examId = 201
```



Transaction 2

transaction 2 should be able to read uncommitted value on table
and should not face any lock while doing transaction

```sql
SET TRANSACTION ISOLATION LEVEL
    READ UNCOMMITTED
BEGIN TRANSACTION
SELECT marksObtained
FROM StudentMarks
WHERE deptId = 101 AND examId = 201 AND studentId = 1
COMMIT TRANSACTION
```



Repeatable Read

- **Prevent Uncommitted Read**: A transaction cannot read data that has been changed but not yet saved by other transactions.
- **No Modification by Others**: Other transactions cannot change data that has been read by the current transaction until the current transaction finishes.
- **Shared Locks**: Shared locks are placed on all data read by the current transaction and are held until the transaction completes, preventing modifications by others.
- **Prevents Non-Repeatable Reads**: This isolation level prevents the issue where the same data read twice within a transaction might have been modified by another transaction in between.

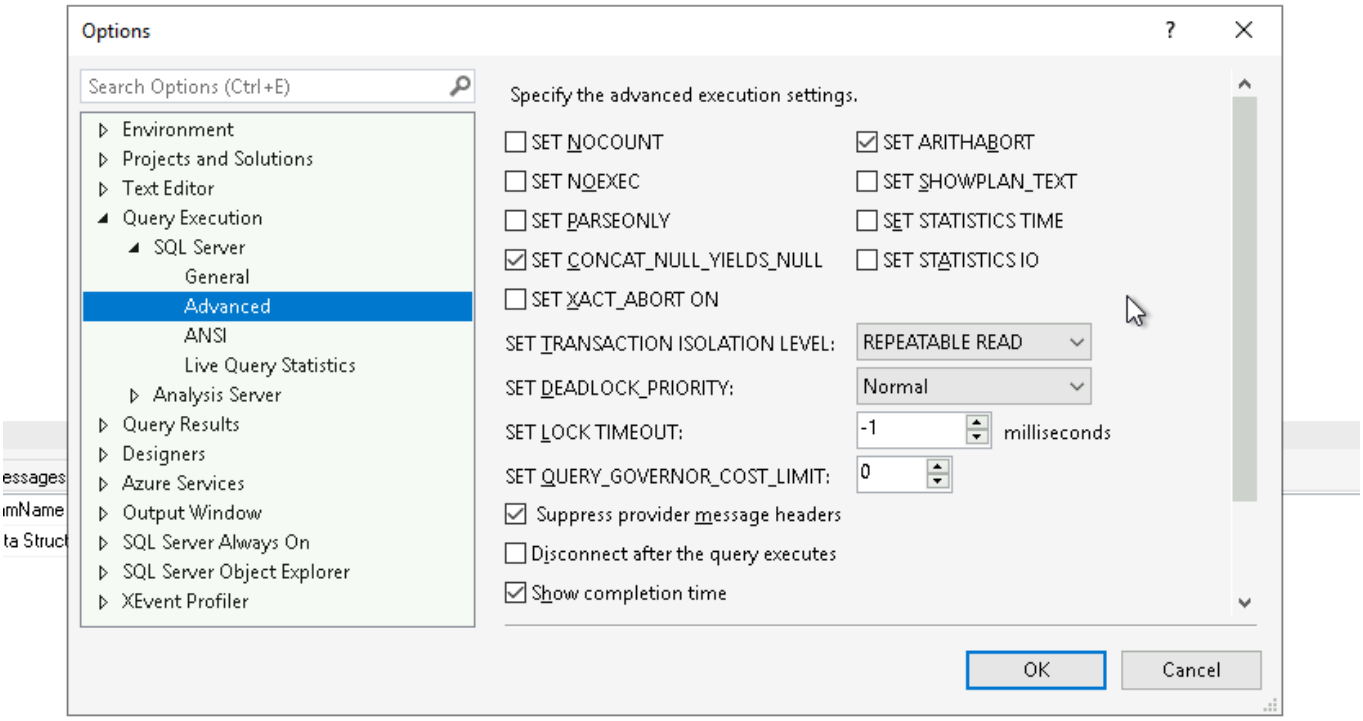
- **Allows New Inserts (Phantom Reads)**: Other transactions can insert new rows that match the search conditions of the current transaction's statements. If the current transaction re-executes the statement, it will retrieve the new rows, leading to phantom reads.

These rules help ensure that data read during a transaction remains consistent within that transaction, but new rows can still appear if other transactions insert data matching the search criteria.

Blow are transaction we will do transaction with select
Another one with update

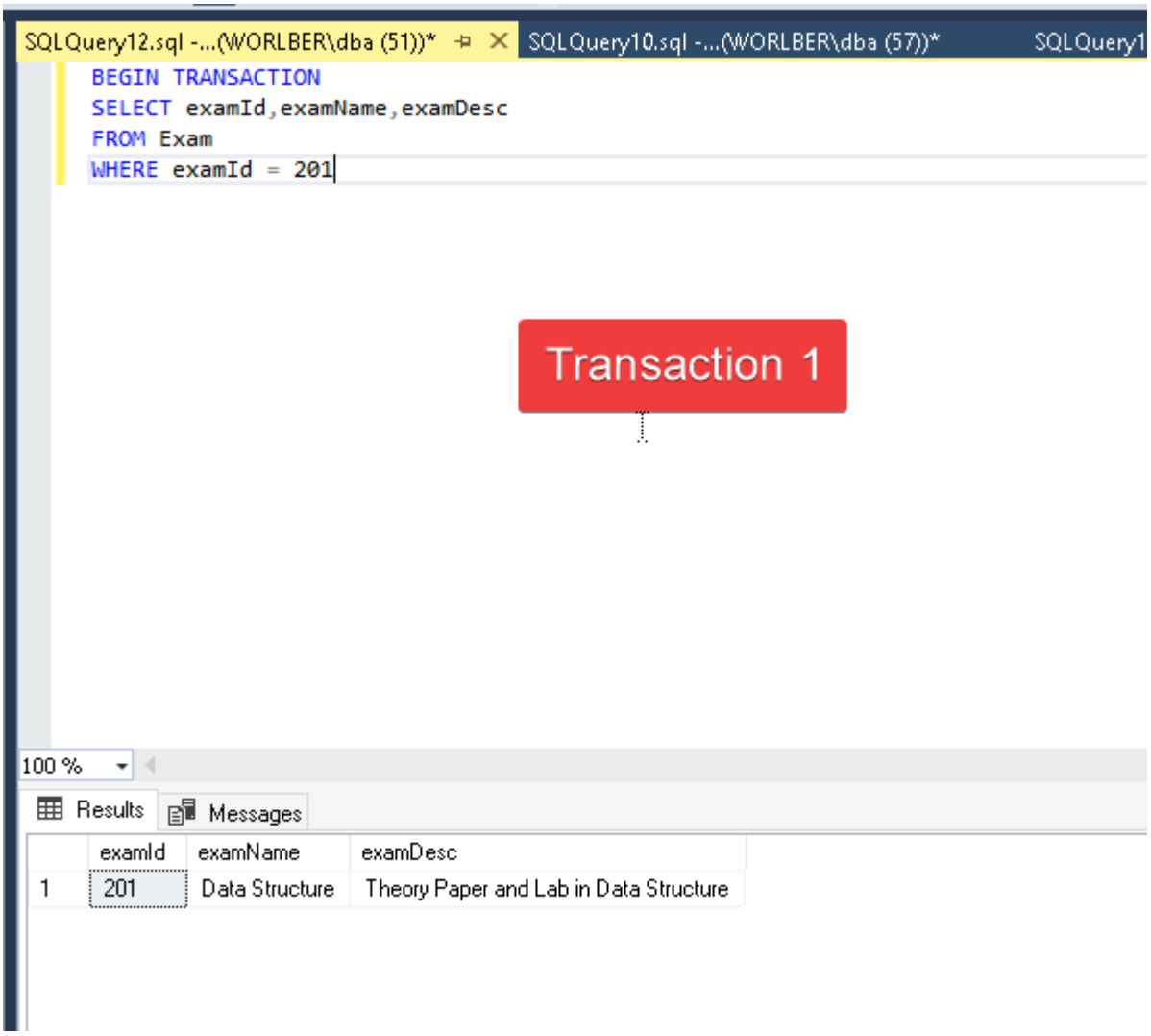
Another one with update but wont be able to complete because there is shared lock by transaction 1 and transaction 2

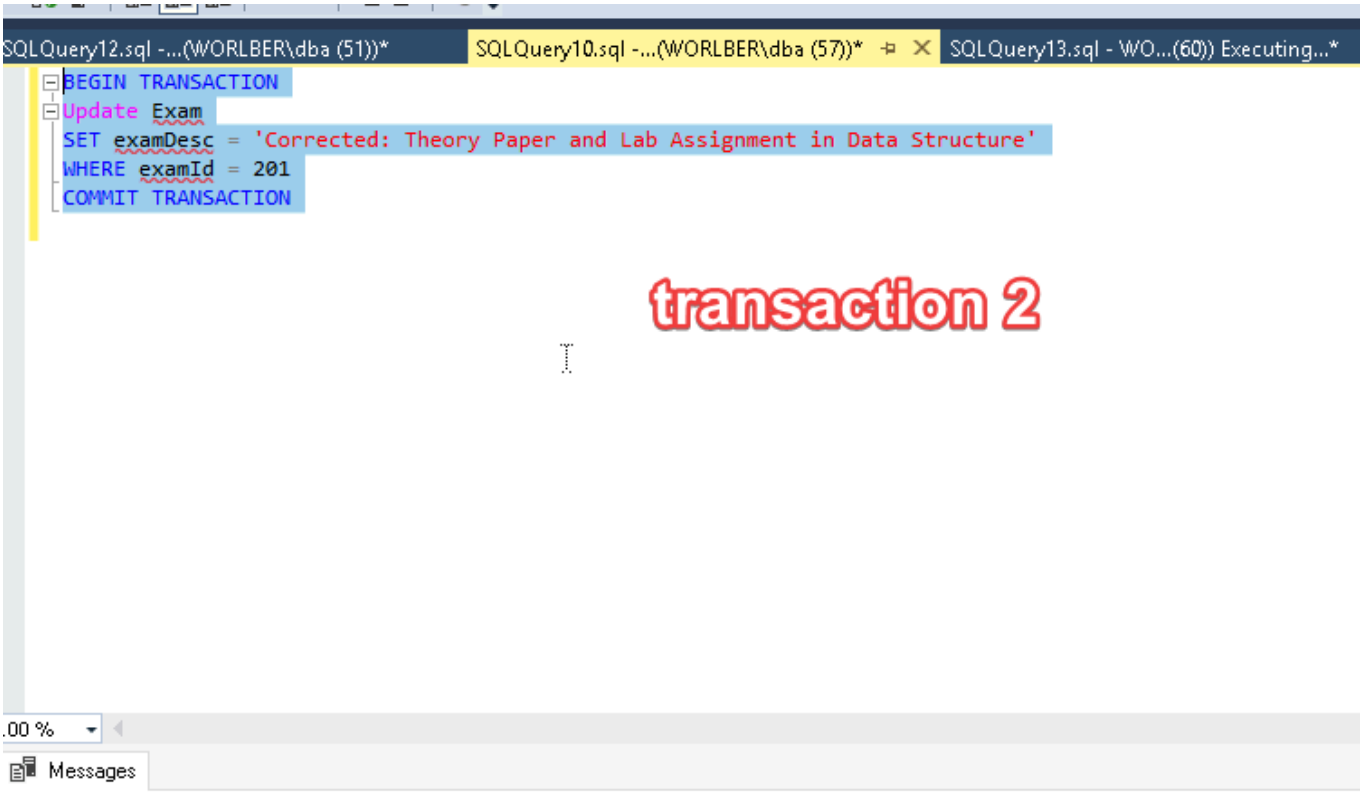
We will set the isolation to Repeatable Read from SSMS

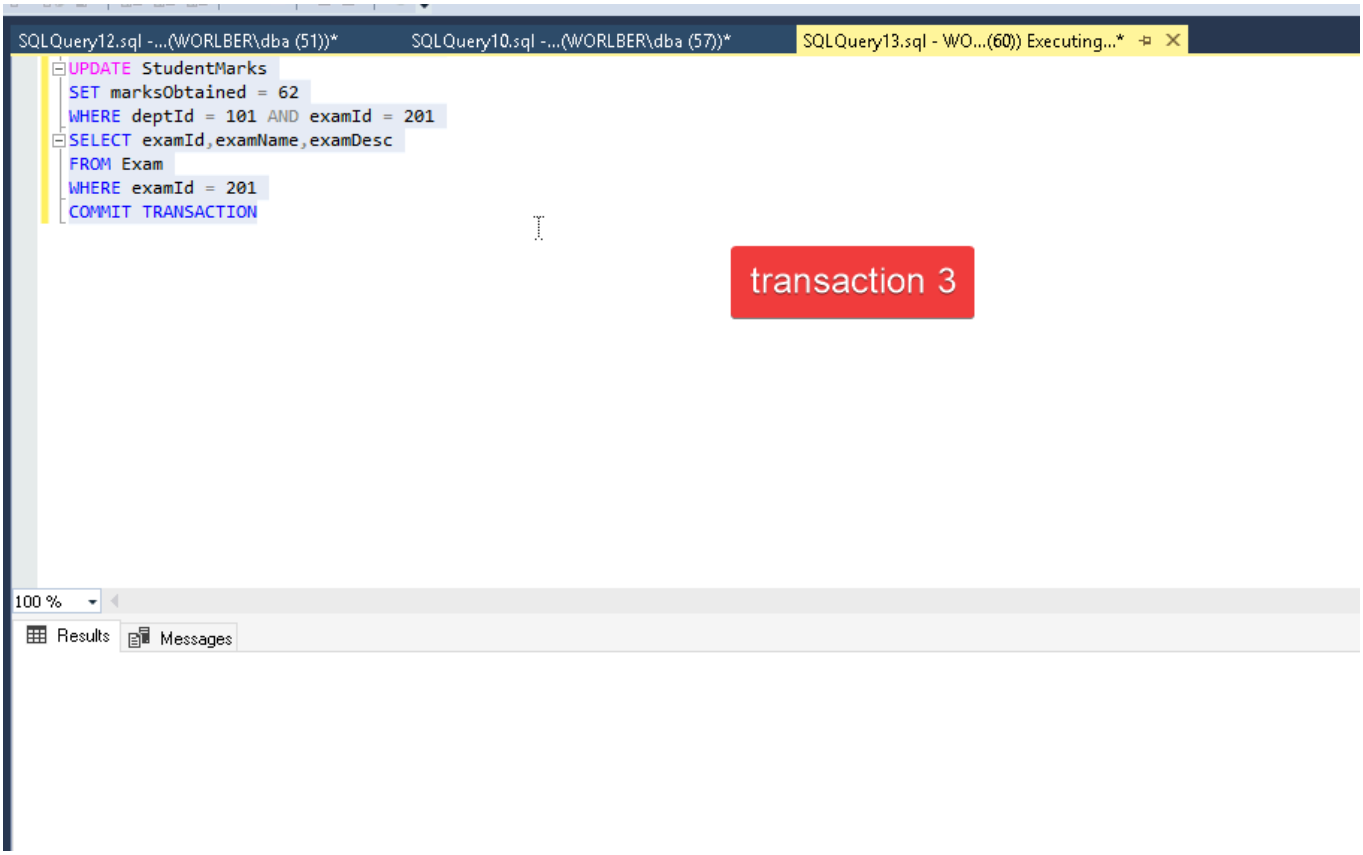



```
 Transaction 1 (Query3.sql) is started.
 BEGIN TRANSACTION
SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
 transaction 2

 BEGIN TRANSACTION
Update Exam
```

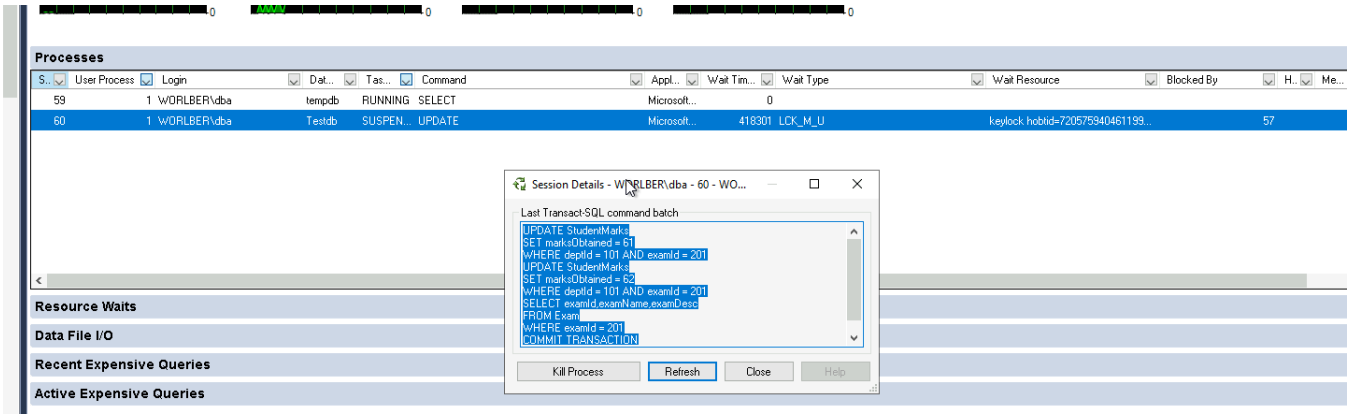
```
SET examDesc = 'Corrected: Theory Paper and Lab Assignment in Data
Structure'
WHERE examId = 201
COMMIT TRANSACTION

transaction 3

UPDATE StudentMarks
SET marksObtained = 61
WHERE deptId = 101 AND examId = 201
UPDATE StudentMarks
SET marksObtained = 62
WHERE deptId = 101 AND examId = 201
SELECT examId,examName,examDesc
FROM Exam
WHERE examId = 201
COMMIT TRANSACTION
```

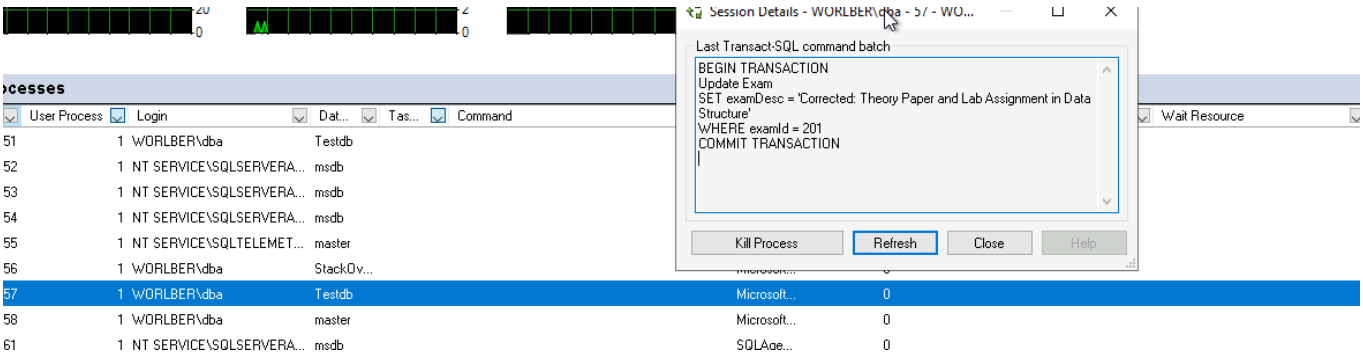



SQLQuery12.sql -...(WORLBER\dba (51))*
SQLQuery10.sql -...(WORLBER\dba (57))*
SQLQuery13.sql - WO...(60)) Executing...*
BEGIN TRANSACTION
Update Exam
SET examDesc = 'Corrected: Theory Paper and Lab Assignment in Data Structure'
WHERE examId = 201
COMMIT TRANSACTION
transaction 2
.00 %
Messages
(1 row affected)
Completion time: 2024-06-20T12:30:02.1618172-07:00



You can see also that session is blocked by update commit transaction not the select uncommitted transaction





Serializable

- **Prevent Uncommitted Read**: A transaction cannot read data that has been changed but not yet saved by other transactions.
- **No Modification by Others**: Once a transaction reads data, no other transactions can change that data until the first transaction is finished.
- **Prevent New Inserts**: Other transactions cannot add new rows with key values that would fit into the range of keys read by the current transaction until it finishes.
- **Highest Isolation Level**: This isolation level ensures the highest degree of isolation by making transactions appear as if they are executed one after the other, rather than concurrently.

These rules help prevent issues such as dirty reads, non-repeatable reads, and phantom reads, ensuring data consistency and integrity during transactions.

Snapshot Isolation

- **Consistent Data View**: Any data read by a transaction is the version of the data that existed at the start of the transaction, ensuring consistency.
- **Isolation from Other Transactions**: Changes made by other transactions after the current transaction starts are not visible to the current transaction.
- **No Read Locks**: Snapshot transactions do not lock data when reading it.
- **No Read-Write Blocking**: Snapshot transactions reading data do not prevent other transactions from writing data.
- **No Write-Read Blocking**: Transactions that write data do not prevent Snapshot transactions from reading data.

These rules help maintain data consistency without the need for locking, allowing for higher concurrency.

8.Optimizing TempDB Performance

${toc}
TempDB provides a workspace for temporary data storage during query execution and plays a crucial role in the performance and stability of SQL Server databases. T

at is TempDB?

- **Default System Database**: TempDB is one of the four system databases that come with SQL Server by default.
- **Non-Durable Storage**: It is used for temporary storage, meaning transactions are minimally logged.
- **High Usage**: TempDB is probably the busiest database on most SQL Server instances because it supports many internal functions.
- **Immutable**: TempDB cannot be:
  - Dropped
  - Detached
  - Taken offline
  - Renamed
  - Restored
    Attempting any of these operations will result in an error.
- **Regeneration on Restart**: TempDB is regenerated every time the SQL Server instance starts. Any objects created in TempDB during a previous session will not persist after:
  - A server restart
  - An instance update management operation
  - A failover
- **High Write Activity**: Due to its workload, TempDB handles a large number of writes, requiring low-latency, high-throughput storage.
- **Proportional Fill Mechanism**: SQL Server uses a proportional fill mechanism where larger files receive proportionally more writes. Therefore, all TempDB data files should be equally sized.

Configure TempDB correctly

- The initial size of TempDB should be set based on the expected workload of the server,

- the number of data files should be equal to the number of logical processors help minimize contention and improve parallelism.
- The auto growth settings should be set to a fixed value rather than a percentage
- Place TempDB on Separate Disks
-