

SQL Script Best Practices

How to write a better SQL script

By: Shailendra Srivastava
16.Feb.2022

Contents

- 1 Format the SQL script to improve readability
- 2 Drawbacks of using NOT IN as a subquery
- 3 Use of the Common Table Expression (CTE)
- 4 Use temporal tables
- 5 Use of Try... Catch and Error Handling
- 6 Use of transactions
- 7 Use of JSON in stored procedures
- 8 Getting accurate execution time In SQL Server
- 9 Useful SQL functions

Format the SQL script to improve readability (1/4)

Avoid “*”

Always avoid the use of “*” in a select statement. Instead, always give a specific column name.

Avoid	Prefer
select * from customers	SELECT name, age, salary FROM customers

Format the SQL script to improve readability (2/4)

Use of **uppercase** and **lowercase**

SQL keywords should be in “***uppercase***”, and table and column name should be in “***lowercase***”

Avoid

```
select id, name from customers
```

Prefer

```
SELECT id, name FROM customers
```

SQL function should come as ***GET_DATE()***, ***MIN()***, ***MAX()*** etc.

Format the SQL script to improve readability (3/4)

Format your query: use indentation and add white spaces

Indent after a keyword, and when you use a subquery or a derived table, add white spaces in the WHERE clause

Avoid	Prefer
<pre>SELECT customers.id, customers.name, customers.age, customers.gender, customers.salary, first_purchase.date FROM company.customers LEFT JOIN (SELECT customer_id, MIN(date) as date FROM company.purchases GROUP BY customer_id) AS first_purchase ON first_purchase.customer_id = customers.id WHERE customers.age<=30</pre>	<pre>SELECT customers.id, customers.name, customers.age, customers.gender, customers.salary, first_purchase.date FROM company.customers LEFT JOIN (SELECT customer_id, MIN(date) as date FROM company.purchases GROUP BY customer_id) AS first_purchase ON first_purchase.customer_id = customers.id WHERE customers.age <= 30</pre>

Format the SQL script to improve readability (4/4)

Use aliases when it improves readability

Meaningful alias **for columns** should be used with lowercase 'as', and **for tables**, with an uppercase 'AS'

Avoid	Prefer
<pre>SELECT customers.id, customers.name, customers.context_col1, nested.f0_ FROM company.customers JOIN (SELECT customer_id, MIN(date) FROM company.purchases GROUP BY customer_id) ON customer_id = customers.id</pre>	<pre>SELECT customers.id, customers.name, customers.context_col1 as ip_address, first_purchase.date as first_purchase_date FROM company.customers JOIN (SELECT customer_id, MIN(date) as date FROM company.purchases GROUP BY customer_id) AS first_purchase ON first_purchase.customer_id = customers.id</pre>

Drawbacks of using NOT IN as a subquery (1/2)

It's common to use the operator NOT IN to retrieve rows in a table (or SQL statement) that are not in another table or another SQL statement.

- ❖ NOT IN works, but as the number of records grows, NOT IN performs badly
- ❖ NOT IN doesn't always return the expected results when null values are allowed

As in regular query

```
SELECT ID FROM T1  
WHERE ID NOT IN (SELECT ID FROM T2)
```

When we may have Null

```
INSERT INTO T2 VALUES (NULL)  
  
SELECT ID FROM T1  
WHERE ID NOT IN (SELECT ID FROM T2)
```

Drawbacks of using NOT IN as a subquery (2/2)

Optional operators to avoid the use of the NOT IN

NOT EXISTS

```
SELECT ID FROM T1  
WHERE NOT EXISTS  
(SELECT ID FROM T2 WHERE T1.ID = T2.ID)
```

LEFT OUTER JOIN

```
SELECT T1.ID FROM T1  
LEFT OUTER JOIN T2 ON T1.ID = T2.ID  
WHERE T2.ID IS NULL
```

EXCEPT

```
SELECT ID FROM T1  
EXCEPT  
SELECT ID FROM T2
```


Use of the Common Table Expression (CTE) (1/2)

A CTE allows you to define and execute a query, of which the result exists temporarily and can be used within a larger query.

CTEs are available on most modern databases. It works like a derived table, with two advantages:

- ❖ Using CTE improves the readability of your query
- ❖ A CTE is defined once then can be referred to multiple times

You declare a CTE with the instruction **WITH ... AS:**

Use of the Common Table Expression (CTE) (2/2)

```
;with Create3Entries
AS
(
    SELECT fldId,StateName,CaseRequestType,CaseRequestSubType,'Mail' DeliveryMethod FROM
    UACT0b621a7da462415385a821a9caded4e7
    WHERE DeliveryMethod='Any'
    UNION
    SELECT fldId,StateName,CaseRequestType,CaseRequestSubType,'Email' DeliveryMethod FROM
    UACT0b621a7da462415385a821a9caded4e7
    WHERE DeliveryMethod='Any'
    Union
    SELECT fldId,StateName,CaseRequestType,CaseRequestSubType,'Fax' DeliveryMethod FROM
    UACT0b621a7da462415385a821a9caded4e7
    WHERE DeliveryMethod='Any'
),
Merge3Entries AS
(
    SELECT Mtbl.fldIWfld,Mtbl.fldIActId,Mtbl.fldAIId,Mtbl.fldMasterIWfld,Mtbl.CaseRequestType
    ,Create3Entries.DeliveryMethod,Mtbl.CaseRequestSubType
    FROM Create3Entries
    INNER JOIN UACT0b621a7da462415385a821a9caded4e7 Mtbl ON Mtbl.fldId=Create3Entries.fldId
)
INSERT INTO UACT0b621a7da462415385a821a9caded4e7
(fldIWfld,fldIActId,fldAIId,fldMasterIWfld,CaseRequestType,DeliveryMethod,CaseRequestSubType)
SELECT fldIWfld,fldIActId,fldAIId,fldMasterIWfld,CaseRequestType,DeliveryMethod,CaseRequestSubType
FROM Merge3Entries
```

Use temporal tables (1/4)

To maintain version/history of records of a table

- ❖ Audit
- ❖ Slowly changing dimensions
- ❖ Repair record-level corruptions

Prerequisites must be met

- A primary key must be defined
- Two columns must be defined to record the start and end date with a data type of datetime2

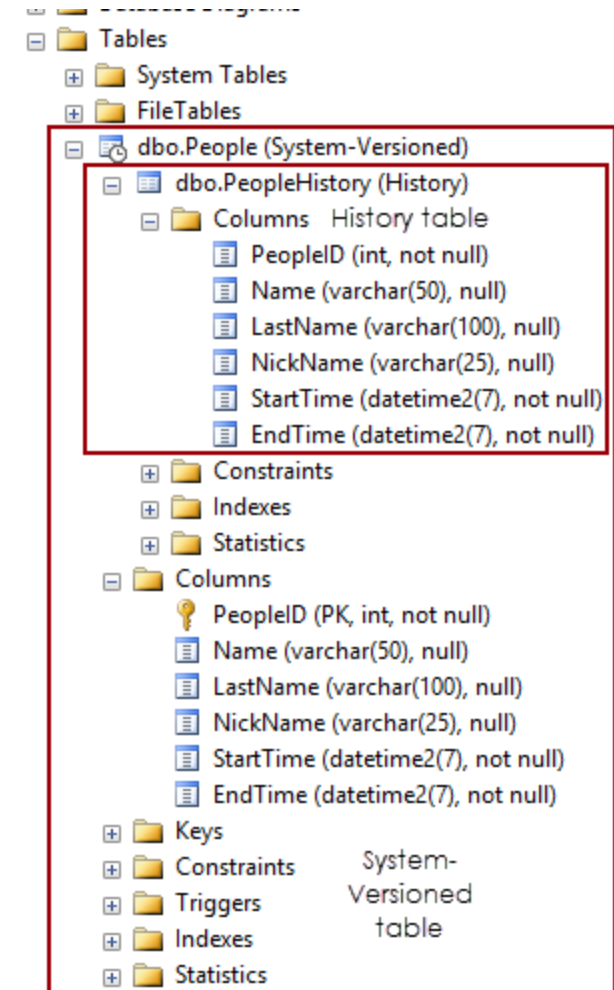
Limitation

- Temporal and history table cannot be FILETABLE
- The history table cannot have any constraints
- INSERT and UPDATE statements cannot reference the SYSTEM_TIME period columns
- Data in the history table cannot be modified

Use temporal tables (2/4)

Query to create a temporal table

```
CREATE TABLE People(  
    PeopleID int PRIMARY KEY NOT NULL, Name varchar(50) Null,  
    LastName varchar(100) NULL,  
    NickName varchar(25) NULL,  
    StartTime datetime2 (0) GENERATED ALWAYS AS ROW START NOT NULL  
    DEFAULT GETUTCDATE(),  
    EndTime datetime2(0) GENERATED ALWAYS AS ROW END NOT NULL  
    DEFAULT CONVERT(DATETIME2, '9999-12-31 23:59:59.9999999'),  
    PERIOD FOR SYSTEM_TIME (StartTime,EndTime) )  
WITH (SYSTEM_VERSIONING = ON(HISTORY_TABLE = dbo.PeopleHistory))
```



Use temporal tables (3/4)

Apply changes to records

```
INSERT INTO dbo.People VALUES(2,'James','Smith','Jam',DEFAULT, DEFAULT)
WAITFOR DELAY '00:01:00'
UPDATE dbo.People
SET dbo.People.Name = 'Thomas' WHERE dbo.People.PeopleID=2
WAITFOR DELAY '00:02:00'
INSERT INTO dbo.People VALUES(3,'Joan','Johnson','Jon',DEFAULT, DEFAULT)
WAITFOR DELAY '00:01:00'
UPDATE dbo.People
SET dbo.People.Name = 'Paul' WHERE dbo.People.PeopleID=3
WAITFOR DELAY '00:02:00'
INSERT INTO dbo.People VALUES (4,'Robert','Davis','Rob',DEFAULT, DEFAULT)
WAITFOR DELAY '00:01:00'
UPDATE dbo.People
SET dbo.People.Name = 'Nik' WHERE dbo.People.PeopleID=4
WAITFOR DELAY '00:02:00'
UPDATE dbo.People
SET dbo.People.Name = 'Brian' WHERE dbo.People.PeopleID=2
WAITFOR DELAY '00:01:00'
```

Use temporal tables (4/4)

SELECT * FROM dbo.People

	PeopleID	Name	LastName	NickName	StartTime	EndTime
1	2	Mark	Smith	Jam	2017-01-26 14:01:29	9999-12-31 23:59:59
2	3	Paul	Johnson	Jon	2017-01-26 13:55:29	9999-12-31 23:59:59
3	4	Nik	Davis	Rob	2017-01-26 13:58:29	9999-12-31 23:59:59

SELECT * FROM dbo.PeopleHistory

	PeopleID	Name	LastName	NickName	StartTime	EndTime
1	2	James	Smith	Jam	2017-01-26 13:51:29	2017-01-26 13:52:29
2	3	Joan	Johnson	Jon	2017-01-26 13:54:29	2017-01-26 13:55:29
3	4	Robert	Davis	Rob	2017-01-26 13:57:29	2017-01-26 13:58:29
4	2	Thomas	Smith	Jam	2017-01-26 13:52:29	2017-01-26 14:00:29
5	2	Brian	Smith	Jam	2017-01-26 14:00:29	2017-01-26 14:01:29

To forcefully drop a temporal table

```
ALTER TABLE [dbo].[People] SET ( SYSTEM_VERSIONING = OFF)
```

For more information, see:

<https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/sql-2016-temporal-tables-how-do-you-drop-a-temporal-table/ba-p/371177>

Use of Try... Catch and Error handling (1/2)

Error handling in the SQL Server gives us control over the Transact-SQL code.

Handling errors using TRY...CATCH

```
BEGIN TRY
    --code to try
END TRY
BEGIN CATCH
    --code to run if an error occurs
    --is generated in try
END CATCH
```

- **ERROR_NUMBER:** Returns the internal number of the error
- **ERROR_STATE:** Returns the information about the source
- **ERROR_SEVERITY:** Returns the information about anything from informational errors to errors user of DBA can fix, etc.
- **ERROR_LINE:** Returns the line number at which an error happened on
- **ERROR_PROCEDURE:** Returns the name of the stored procedure or function
- **ERROR_MESSAGE:** Returns the most essential information and that is the message text of the error

Use of Try... Catch and Error handling (2/2)

TRY...CATCH in a query with error numbers

```
1 BEGIN TRY
2     SELECT
3         CAST('Shailendra' as INT) AS Error;
4 END TRY
5 BEGIN CATCH
6     SELECT
7         ERROR_NUMBER() AS ErrorNumber,
8         ERROR_STATE() AS ErrorState,
9         ERROR_SEVERITY() AS ErrorSeverity,
10        ERROR_PROCEDURE() AS ErrorProcedure,
11        ERROR_LINE() AS ErrorLine,
12        ERROR_MESSAGE() AS ErrorMessage;
13 END CATCH;
```

%

Results Messages

Error

ErrorNumber	ErrorState	ErrorSeverity	ErrorProcedure	ErrorLine	ErrorMessage
245	1	16	NULL	2	Conversion failed when converting the varchar value 'Shailendra' to data type int.

Use of Try... Catch and Error handling with custom error message

Use of 'RAISEERROR'

```
BEGIN TRY
-- RAISERROR with severity 11-19 will cause execution to
-- jump to the CATCH block.
RAISERROR ('Error raised in TRY block.', -- Message text.
          16, -- Severity.
          1 -- State.
        );
END TRY
BEGIN CATCH
DECLARE @ErrorMessage NVARCHAR(4000);
DECLARE @ErrorSeverity INT;
DECLARE @ErrorState INT;

SELECT
    @ErrorMessage = ERROR_MESSAGE(),
    @ErrorSeverity = ERROR_SEVERITY(),
    @ErrorState = ERROR_STATE();

-- Use RAISERROR inside the CATCH block to return error
-- information about the original error that caused
-- execution to jump to the CATCH block.
RAISERROR (@ErrorMessage, -- Message text.
          @ErrorSeverity, -- Severity.
          @ErrorState -- State.
        );
END CATCH;
```

Use of 'THROW'

```
CREATE TABLE #TestRethrow
( ID INT PRIMARY KEY
);
BEGIN TRY
    INSERT #TestRethrow(ID) VALUES(1);
-- Force error 2627, Violation of PRIMARY KEY constraint to
-- be raised.
    INSERT #TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH
    DECLARE @msg VARCHAR(50)='Primery Key
error';
    THROW 60000, @msg, 1;
END CATCH;
```

For more details on Try Catch and Error handling, visit:

<https://docs.microsoft.com/en-us/sql/t-sql/language-elements/try-catch-transact-sql?view=sql-server-2017>

Use of transactions (1/2)

Transactions group a set of tasks into a single execution unit.

Transactional control commands

- ❖ **COMMIT:** to save the changes.
- ❖ **ROLLBACK:** to roll back the changes.
- ❖ **SAVEPOINT:** creates points within the groups of transactions in which to ROLLBACK.
- ❖ **SET TRANSACTION:** places a name on a transaction.

Only used with DML Commands, such as **INSERT**, **UPDATE**, and **DELETE**

Use of transactions (2/2)

How to use Transactions

```
BEGIN TRY
  BEGIN TRANSACTION SCHEDULEDELETE
  DELETE -- delete commands full SQL cut out
  DELETE -- delete commands full SQL cut out
  DELETE -- delete commands full SQL cut out
  COMMIT TRANSACTION SCHEDULEDELETE
  PRINT 'X rows deleted. Operation Successful Tara.' --calculation cut out.
END TRY

BEGIN CATCH
  IF (@@TRANCOUNT > 0)
  BEGIN
    ROLLBACK TRANSACTION SCHEDULEDELETE
    PRINT 'Error detected, all changes reversed'
  END
  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
```

Use Save Point

```
SAVEPOINT SP1;
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SAVEPOINT SP2;
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SAVEPOINT SP3;
Savepoint created.
DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

Rollback to Save Point

```
ROLLBACK TO SP2;
Rollback complete.
```

Release Save Point

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

SET TRANSACTION

```
SET TRANSACTION ISOLATION LEVEL <Isolationlevel_name>
```

<https://www.c-sharpcorner.com/blogs/using-isolation-level-in-sql-transaction2>

<https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver15>

Use of JSON in stored procedures (1/2)

Send the entire JSON text to database and parse it using the new **OPENJSON** function.

```
DECLARE @json nVARCHAR(max)='[
```

```
{ "id" : 2,"firstName": "Uday", "lastName": "Singh",  
  "age": 25, "dateOfBirth": "2007-03-25T12:00:00" },  
{ "id" : 5,"firstName": "Anurag", "lastName": "Gupta",  
  "age": 35, "dateOfBirth": "2005-11-04T12:00:00" },  
{ "id" : 7,"firstName": "Vinod", "lastName": "Mishra",  
  "age": 15, "dateOfBirth": "1983-10-28T12:00:00" },  
{ "id" : 8,"firstName": "Arvind", "lastName": "Giri",  
  "age": 12, "dateOfBirth": "1995-07-05T12:00:00" },  
{ "id" : 9,"firstName": "Rajat", "lastName": "Saxena",  
  "age": 37, "dateOfBirth": "2015-03-25T12:00:00" }  
]
```

```
SELECT *  
FROM OPENJSON(@json)  
  WITH (id int, firstName nvarchar(50), lastName  
        nvarchar(50),  
        age int, dateOfBirth datetime2)
```

	id	firstName	lastName	age	dateOfBirth
1	2	Uday	Singh	25	2007-03-25 12:00:00.0000000
2	5	Anurag	Gupta	35	2005-11-04 12:00:00.0000000
3	7	Vinod	Mishra	15	1983-10-28 12:00:00.0000000
4	8	Arvind	Giri	12	1995-07-05 12:00:00.0000000
5	9	Rajat	Saxena	37	2015-03-25 12:00:00.0000000

```
INSERT INTO Person (id, name, surname, age, dateOfBirth)  
SELECT id, firstName, lastName, age, dateOfBirth  
FROM OPENJSON(@json)  
WITH (id int,  
      firstName nvarchar(50), lastName nvarchar(50),  
      age int, dateOfBirth datetime2)
```

<https://www.codeproject.com/Articles/1087995/Inserting-JSON-Text-into-SQL-Server-Table>

Use of JSON in stored procedures (2/2)

Receive output in the form of JSON by using **FOR JSON PATH**.

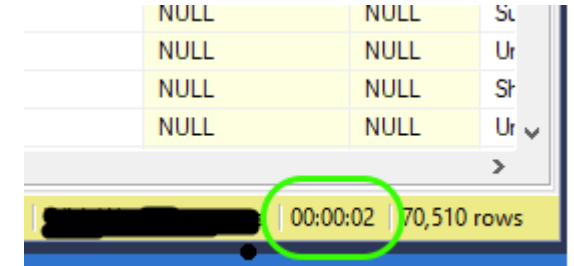
```
DECLARE @text NVARCHAR(MAX)=  
(  
  SELECT  
    fldIWfld  
    InputChannel,  
    SubmissionID,  
    UnderwriterID,  
    Underwriter,  
    Broker  
  FROM  
    VwCPR_GetALLSubmissionsDATAFromUWF  
  WITH(NOLOCK)  
  WHERE fldIWfld= 132878  
  FOR JSON PATH  
)  
SELECT @text AS
```

outputJson

Results		Messages
		output.Json
1	[{"InputChannel":132878,"SubmissionID":132878,"UnderwriterID":1135,"Underwriter":"Shailendra Srivastava","Broker":"Lockton"}]	

Getting accurate execution time in the SQL Server (1/2)

Checking the time taken to execute an SQL statement is an effective way to analyze SQL statements.

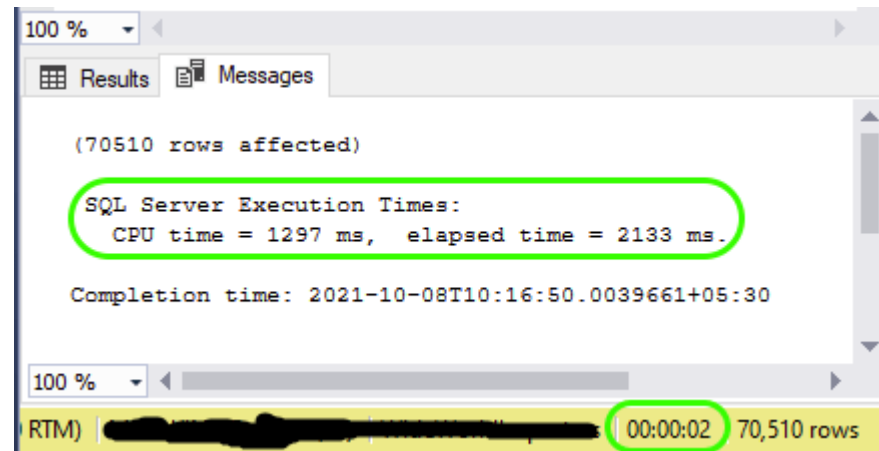


NULL	NULL	St
NULL	NULL	Ur
NULL	NULL	St
NULL	NULL	Ur

00:00:02 70,510 rows

What to do if you want to get the accurate execution time up in milliseconds?

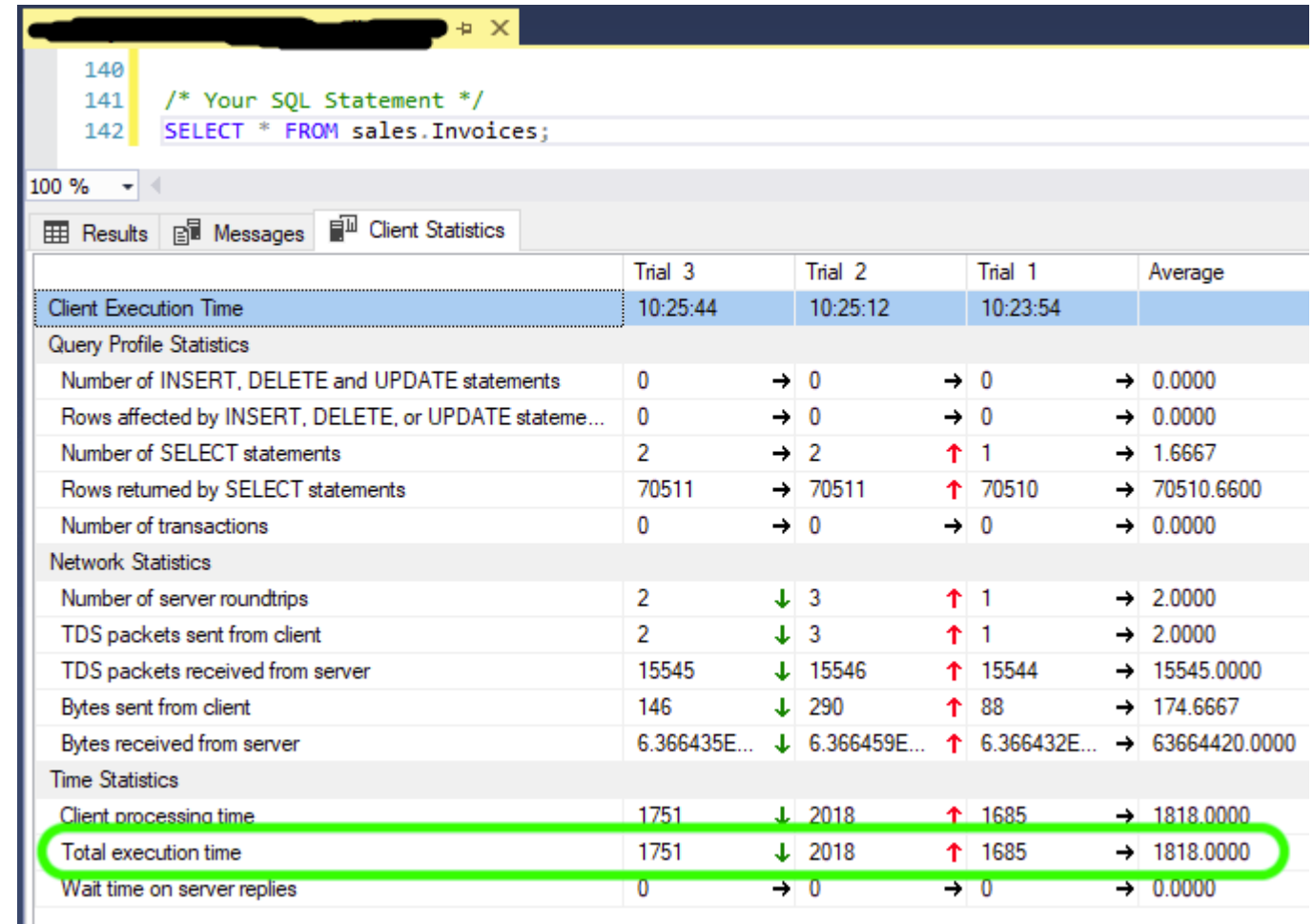
```
/* Switch on statistics time */  
SET STATISTICS TIME ON;  
/* Your SQL Statement */  
SELECT * FROM sales.Invoices;  
/* Switch off statistics time */  
SET STATISTICS TIME OFF;  
GO
```



Getting accurate execution time in the SQL Server (2/2)

Using Client Statistics

1. Go to **Menu > Query >**
Select **Include client Statistics**.
2. Execute your query.
3. In the results panel, note the new
tab **Client Statistics**.
4. On the **Client Statistics** tab, see
the execution time.



The screenshot shows the SQL Server Enterprise Manager interface. At the top, a query window displays the following SQL code:

```
140
141 /* Your SQL Statement */
142 SELECT * FROM sales.Invoices;
```

Below the query window, the 'Client Statistics' tab is selected. It displays a table with the following data:

	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	10:25:44	10:25:12	10:23:54	
Query Profile Statistics				
Number of INSERT, DELETE and UPDATE statements	0	→ 0	→ 0	→ 0.0000
Rows affected by INSERT, DELETE, or UPDATE statements	0	→ 0	→ 0	→ 0.0000
Number of SELECT statements	2	→ 2	↑ 1	→ 1.6667
Rows returned by SELECT statements	70511	→ 70511	↑ 70510	→ 70510.6600
Number of transactions	0	→ 0	→ 0	→ 0.0000
Network Statistics				
Number of server roundtrips	2	↓ 3	↑ 1	→ 2.0000
TDS packets sent from client	2	↓ 3	↑ 1	→ 2.0000
TDS packets received from server	15545	↓ 15546	↑ 15544	→ 15545.0000
Bytes sent from client	146	↓ 290	↑ 88	→ 174.6667
Bytes received from server	6.366435E...	↓ 6.366459E...	↑ 6.366432E...	→ 63664420.0000
Time Statistics				
Client processing time	1751	↓ 2018	↑ 1685	→ 1818.0000
Total execution time	1751	↓ 2018	↑ 1685	→ 1818.0000
Wait time on server replies	0	→ 0	→ 0	→ 0.0000

The 'Total execution time' row is highlighted with a green circle.

Useful SQL functions (1/2)

SQL functions	Use
TRANSLATE()	TRANSLATE(string, characters, translations) SELECT TRANSLATE('Monday', 'Monday', 'Sunday') => 'Sunday' SELECT TRANSLATE('3*[2+1]/{8-4}', '[[]]', '()()'); => 3*(2+1)/(8-4)
CONCAT()	SELECT CONCAT('Shailendra','Kumar', 'Srivastava') => ShailendraKumarSrivastava
CONCAT_WS()	Adds two or more strings together with a separator. CONCAT_WS(separator, string1, string2, ..., string_n) SELECT CONCAT_WS('-', 'SQL', ' is', ' fun!') => SQL- is- fun!
DATETIMEFROMPARTS()	SELECT DATEFROMPARTS(2018, 10, 31) AS DateFromParts => 2018-10-31
EOMONTH()	SELECT EOMONTH('2022-02-15') => 2019-02-28
CHOOSE()	SELECT CHOOSE(2, 'First', 'Second', 'Third') => 'Second'
IIF()	SELECT IIF(500<1000, 'YES', 'NO') => 'Yes'
FORMAT	FORMAT(value, format, culture) DECLARE @d DATETIME = '02.16.2022'; SELECT FORMAT (@d, 'd', 'en-US') AS 'US English Result' => 2/16/2022 SELECT FORMAT(123456789, '##-##-#####') => 12-34-56789

Useful SQL functions (2/2)

SQL functions	Use
REPLICATE	SELECT REPLICATE('Ok', 5) => OkOkOkOkOk
REVERSE	SELECT REVERSE('Shailendra') => ardneliahS

LAG() & LEAD

13 SELECT Id FROM #TempA
14

Id
1
2
3
4
5
6

```
SELECT Id,  
LAG(Id) OVER(ORDER BY Id) prev_id,  
LEAD(Id) OVER(ORDER BY Id) next_id  
FROM #TempA
```

12 SELECT Id,
13 LAG(Id) OVER(ORDER BY Id) prev_id,
14 LEAD(Id) OVER(ORDER BY Id) next_id
15 FROM #TempA

81 %

	Id	prev_id	next_id
1	1	NULL	2
2	2	1	3
3	3	2	4
4	4	3	5
5	5	4	6
6	6	5	NULL

✓ Query executed successfully.

Thank you



genpact

Transformation
Happens Here