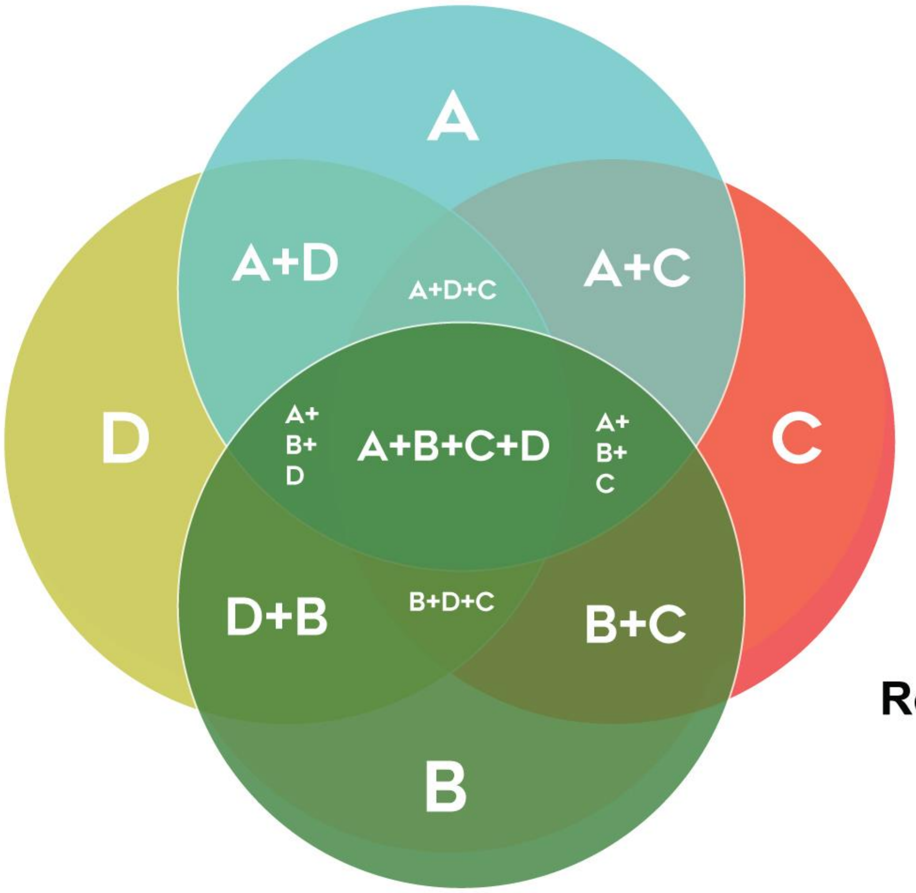
SQLServerGeeks

MARCH 2022

MAGAZINE

The Bimonthly Dossier For SQL Professionals



A Tip about Intersect

by Erland Sommarskog

SQLServerGeeks Magazine

We are thrilled to present to you, the eighth edition of the SQLServerGeeks Magazine, March 2022. You might wonder, where is the Feb 2022 edition? Well, now the SSG Magazine will be a bi-monthly feature because, in Feb 2022, we launched another bi-monthly magazine for the Data professionals, the DataPlatformGeeks Magazine. Therefore, we now have two bi-monthly magazines: SSG Mag for the SQL professionals covering all things SQL, and the DPG Mag for the Data professionals covering the larger technology landscape in the data world. So go ahead and download the first DPG Magazine.

In this edition, we bring you fresh content from Erland Sommarskog, Satya Ramesh & Josephine Bush. A massive thanks to all our authors. There's intriguing content on Intersect operation in SQL, SQL Server Ranking Functions, Azure SQL Auditing, Rolling back Truncate operations using Backups, and The Path forward for SQL Server analytics.

Another big news: Data Platform Virtual Summit 2022 dates are out and registrations have begun. The largest online learning event for data professionals is free. Book your seat today.

If you are interested in writing an article for the magazine, do let us know. Write to us at Magazine@SQLServerGeeks.com.

Help us spread the word. Ask your friends and colleagues to subscribe to the magazine.

From all of us at SQLServerGeeks, we wish you a pleasant read. Happy Learning.

Yours Sincerely
SQLServerGeeks Team

Got from a friend? Subscribe now to get your copy.

Our Social Channels

- Website
- Linkedin
- Telegram
- Youtube
- Twitter
- Facebook

COPYRIGHT STATEMENT

CORPORATE ADDRESS

Bangalore Office:

686, 6 A Cross,
3rd Block Koramangala,
Bangalore – 560034

Kolkata Offices:

Office 1:
eDominer Systems Pvt. Ltd.
The Chambers
Office Unit 206 (Second Floor)
1865 Rajdanga Main Road
(Kasba)
Kolkata 700107

Office 2:
304, PS Continental,
83/2/1, Topsia Road (South),
Kolkata 700046

TABLE OF CONTENTS

HANDLE AZURE SQL AUDITING WITH EASE

Josephine Bush | @hellosqlkitty

In this article, you will learn about the different methods for auditing in Azure SQL Database and Azure SQL Managed Instance. For a summary of what auditing is, please refer to the July SQL Server Geeks magazine.

Here's a quick summary of the different ways you can audit based on the different cloud offerings. To learn more about extended events or SQL Server Audit, please refer to the July, August, and November issues of SQL Server Geeks magazine.

Cloud solution	SQL Server Audit Available	Extended Events Available	Auditing differences
Azure SQL	No	Yes	SQL Server audit quasi equivalent via Azure portal
Azure SQL Managed Instance	Yes	Yes	Need to use cloud storage
SQL Server VM	Yes	Yes	Uses disk storage
Amazon Web Services RDS	Yes	Yes	Need to use cloud storage

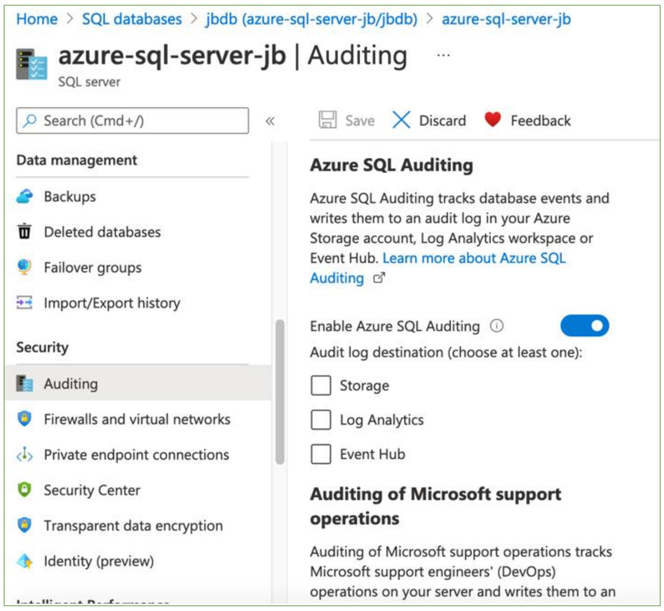
You have a few options for auditing with Azure SQL Database:

1. **Azure Portal** – The portal provides functionality that is like SQL Server Audit, but doesn't allow you to filter the auditing results. You can turn it on at the server level or the database level, but don't turn it on at both levels. If you audit at both levels, you will collect duplicate auditing records. The portal gives you three storage choices for your auditing data:
    a. **Storage** – This is blob storage in Azure. It stores your audit files in .xel format in a folder structure you can loop through to query.
    b. **Log Analytics** – This is a workspace in Azure you can use to store auditing data in a format that is queryable by the Kusto Query Language.
    c. **Event Hub** – This stores audit logs in JSON and requires a stream to consume events and write them to a target.

2. **Extended Events** – Commonly referred to as xevents. This is a lightweight and flexible option for auditing. If you've used Profiler or Trace, this will feel familiar to you. You can use the GUI or scripts to create and query the results of xevents. You will need to store this in blob storage in Azure. It has many of the same events as traditional SQL Server xevents.

Enabling Azure SQL Database Auditing

Here's a high-level explanation of how to set up Azure SQL Database auditing. Navigate to your Azure SQL Database server. I highly recommend enabling auditing at the server level so you will get all events relating to all of your databases in the audit data. The following screenshot shows you how to enable auditing at the server level:


Home > SQL databases > jbdb (azure-sql-server-jb/jbdb) > azure-sql-server-jb
azure-sql-server-jb | Auditing
SQL server
Search (Cmd+/)
Save
Discard
Feedback
Data management
Backups
Deleted databases
Failover groups
Import/Export history
Security
Auditing
Firewalls and virtual networks
Private endpoint connections
Security Center
Transparent data encryption
Identity (preview)

Azure SQL Auditing

Azure SQL Auditing tracks database events and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. Learn more about Azure SQL Auditing

Enable Azure SQL Auditing: On

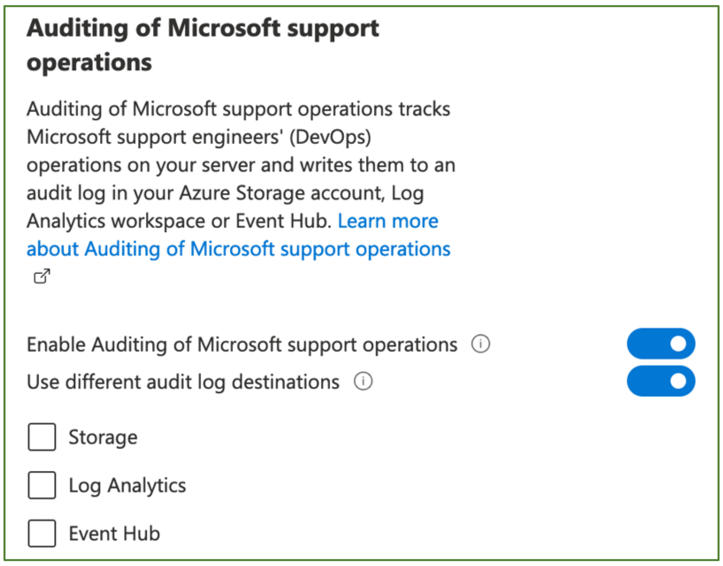
Audit log destination (choose at least one):

- [ ] Storage
- [ ] Log Analytics
- [ ] Event Hub

Auditing of Microsoft support operations

Auditing of Microsoft support operations tracks Microsoft support engineers' (DevOps) operations on your server and writes them to an

It's as easy as clicking a radio button and choosing your storage location. There is also the ability to audit Microsoft support operations events. I like to enable this, as well, because I want to see behind-the-scenes operations that might affect my databases. The following screenshot shows you how to enable the Microsoft support operations auditing:



Auditing of Microsoft support operations

Auditing of Microsoft support operations tracks Microsoft support engineers' (DevOps) operations on your server and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. Learn more about Auditing of Microsoft support operations

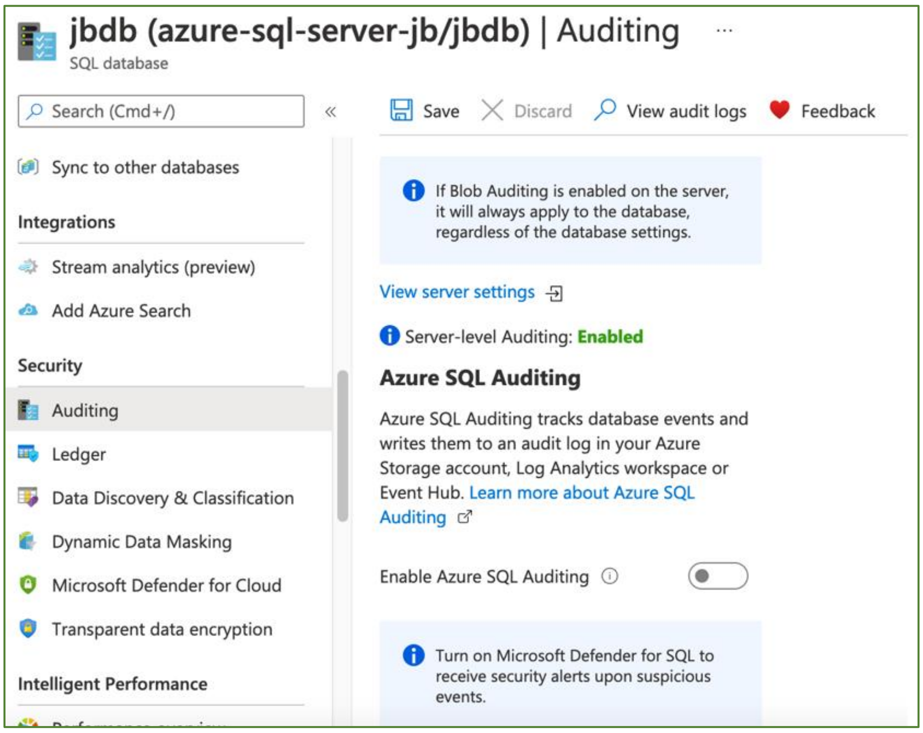
Enable Auditing of Microsoft support operations: On

Use different audit log destinations: On

- [ ] Storage
- [ ] Log Analytics
- [ ] Event Hub

You can use the same or a different audit log destination. I tend to put it all together with my other database auditing to make it easier to see all auditing data in the same place and at the same time.

If you have a use case to audit only one database or a subset of databases on a server, you wouldn't enable it at the server level. Instead, you should navigate to the specific database to enable it, like in the following screenshot:


jbdb (azure-sql-server-jb/jbdb) | Auditing
SQL database

If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.

View server settings

Server-level Auditing: Enabled

Azure SQL Auditing

Azure SQL Auditing tracks database events and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. Learn more about Azure SQL Auditing

Enable Azure SQL Auditing: Off

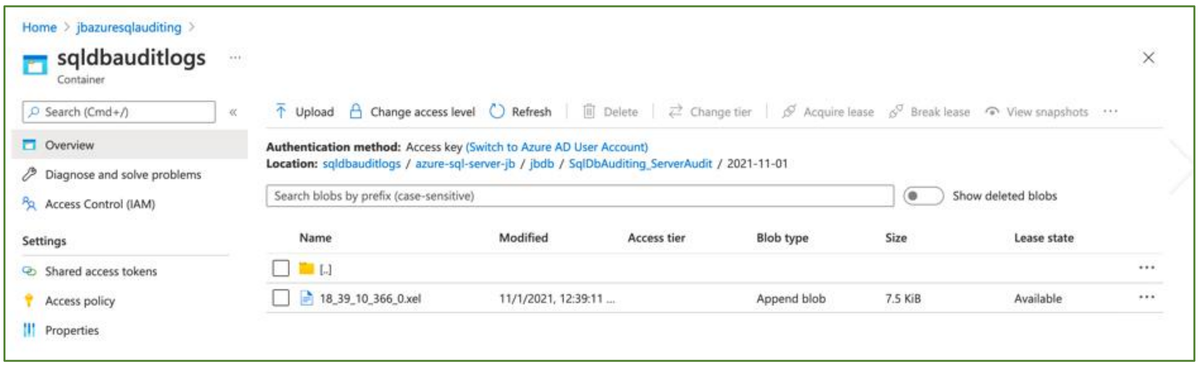
Turn on Microsoft Defender for SQL to receive security alerts upon suspicious events.

You can see in the previous screenshot that server-level auditing is enabled. You would want to disable the server auditing before you enable auditing in this one database. This prevents you from getting a lot of duplicate audit data.

Let's take a look at each of the storage options you have in Azure SQL Database auditing.

Storage

This option lets you put the .xel auditing files in a blob storage account of your choosing. The files are put in a subfolder structure that can be hard to query. You will need to loop through them with something like PowerShell to make it easier to query. The following screenshot shows you what this storage option looks like:


Home > jbazuresqlauditing >

sqldbauditlogs
Container

Authentication method: Access key (Switch to Azure AD User Account)
Location: sqldbauditlogs / azure-sql-server-jb / jbdb / SqlDbAuditing_ServerAudit / 2021-11-01

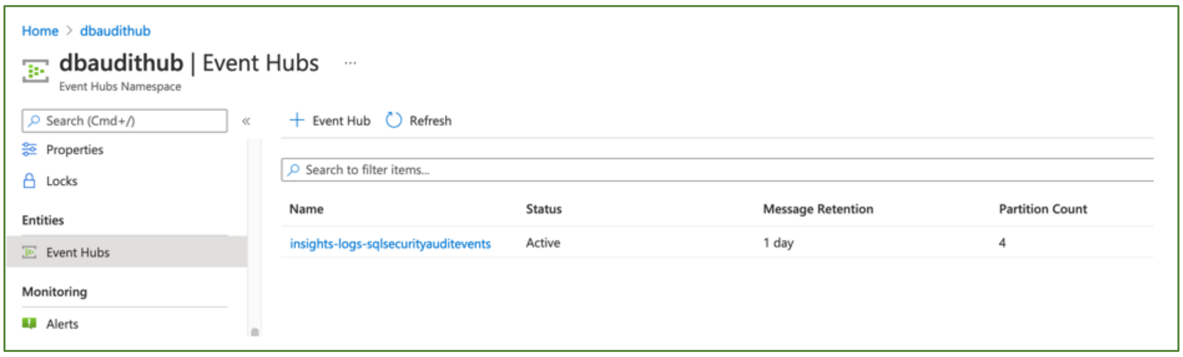
Search blobs by prefix (case-sensitive) | Show deleted blobs: Off

Name	Modified	Access tier	Blob type	Size	Lease state
[..]					
18_39_10_366_0.xel	11/1/2021, 12:39:11 ...		Append blob	7.5 KiB	Available

You can query these .xel files like you would other xevent files. The November issue of the SQL Server Geeks magazine has more information on how to query .xel files. You will need to use XQuery to parse through the XML.

Event Hub

This option lets you store your audit data in an event hub. This stores data in JSON and requires you to set up a stream to read events and write them to a target. The event hub must be in the same region as your database. The following screenshot shows you how the event hub looks after you set it up:

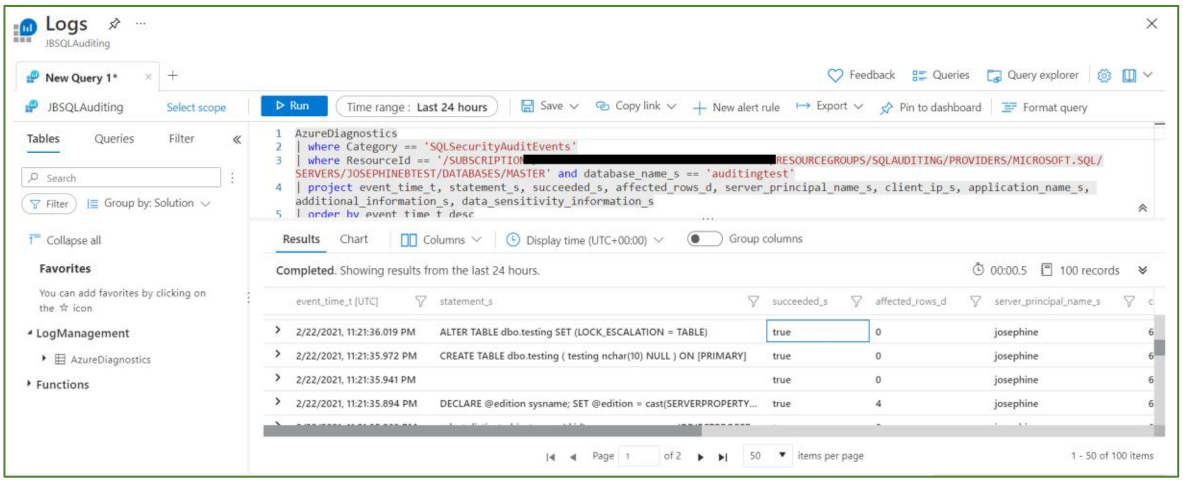


Name	Status	Message Retention	Partition Count
insights-logs-sqlsecurityauditevents	Active	1 day	4

Frankly, I haven't used this option because it's beyond the scope of activities I want to engage in. I prefer a storage account, or even better, log analytics.

Log Analytics

So far, my favorite way to store audit logs is with log analytics. I put multiple different database servers and their corresponding databases into the same log analytics for easy querying and reporting on the audit data. The following screenshot shows the log analytics audit data:



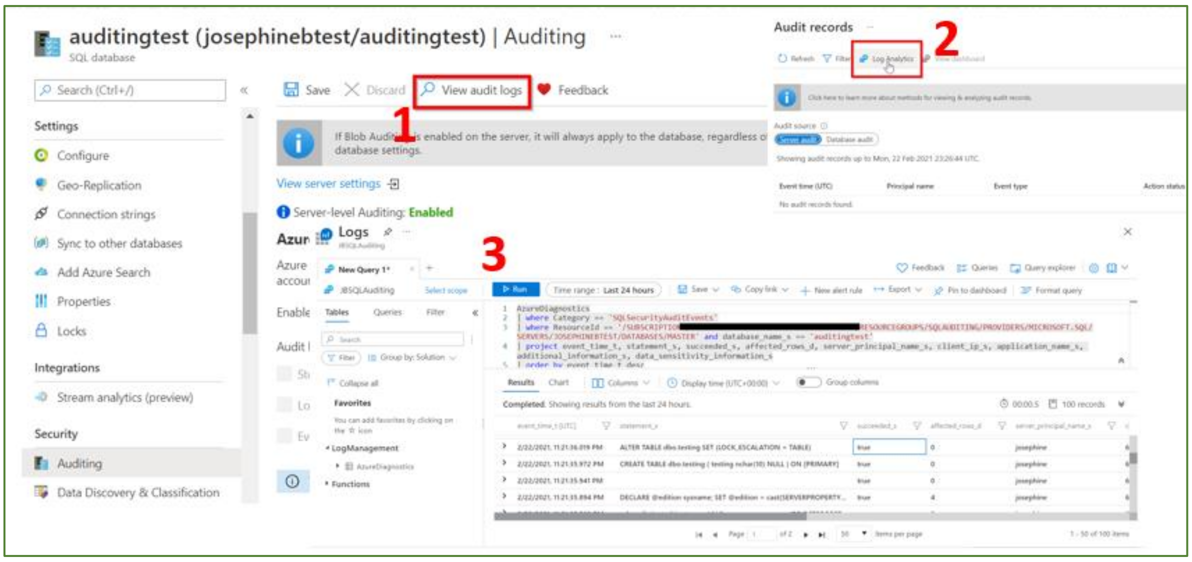
```
AzureDiagnostics
| where Category == 'SQLSecurityAuditEvents'
| where ResourceId == '/SUBSCRIPTION[redacted]RESOURCEGROUPS/SQLAUDITING/PROVIDERS/MICROSOFT.SQL/
SERVERS/JOSEPHINEBTEST/DATABASES/MASTER' and database_name_s == 'auditingtest'
| project event_time_t, statement_s, succeeded_s, affected_rows_d, server_principal_name_s, client_ip_s, application_name_s,
additional_information_s, data_sensitivity_information_s
| order by event_time_t desc
```

Completed. Showing results from the last 24 hours. 00:00.5 100 records

event_time_t [UTC]	statement_s	succeeded_s	affected_rows_d	server_principal_name_s
2/22/2021, 11:21:36.019 PM	ALTER TABLE dbo.testing SET (LOCK_ESCALATION = TABLE)	true	0	josephine
2/22/2021, 11:21:35.972 PM	CREATE TABLE dbo.testing (testing nchar(10) NULL) ON [PRIMARY]	true	0	josephine
2/22/2021, 11:21:35.941 PM		true	0	josephine
2/22/2021, 11:21:35.894 PM	DECLARE @edition sysname; SET @edition = cast(SERVERPROPERTY...	true	4	josephine

Page 1 of 2 50 items per page 1 - 50 of 100 items

You can access this data by either navigating to your log analytics workspace or into Auditing under each of the audited databases, then navigating to Log Analytics. The following screenshot shows the steps to access the auditing data from the database:



auditingtest (josephinebtest/auditingtest) | Auditing
SQL database
Save
Discard
View audit logs
Feedback
1
2
3
Audit records
Log Analytics
If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.
View server settings
Server-level Auditing: Enabled
Logs
Settings
Configure
Geo-Replication
Connection strings
Sync to other databases
Add Azure Search
Properties
Locks
Integrations
Stream analytics (preview)
Security
Auditing
Data Discovery & Classification
Run
Time range : Last 24 hours

```
AzureDiagnostics
| where Category == 'SQLSecurityAuditEvents'
| where ResourceId == '/SUBSCRIPTION[redacted]RESOURCEGROUPS/SQLAUDITING/PROVIDERS/MICROSOFT.SQL/SERVERS/JOSEPHINEBTEST/DATABASES/MASTER' and database_name_s == 'auditingtest'
| project event_time_t, statement_s, succeeded_s, affected_rows_d, server_principal_name_s, client_ip_s, application_name_s, additional_information_s, data_sensitivity_information_s
| order by event_time_t desc
```

Completed. Showing results from the last 24 hours.

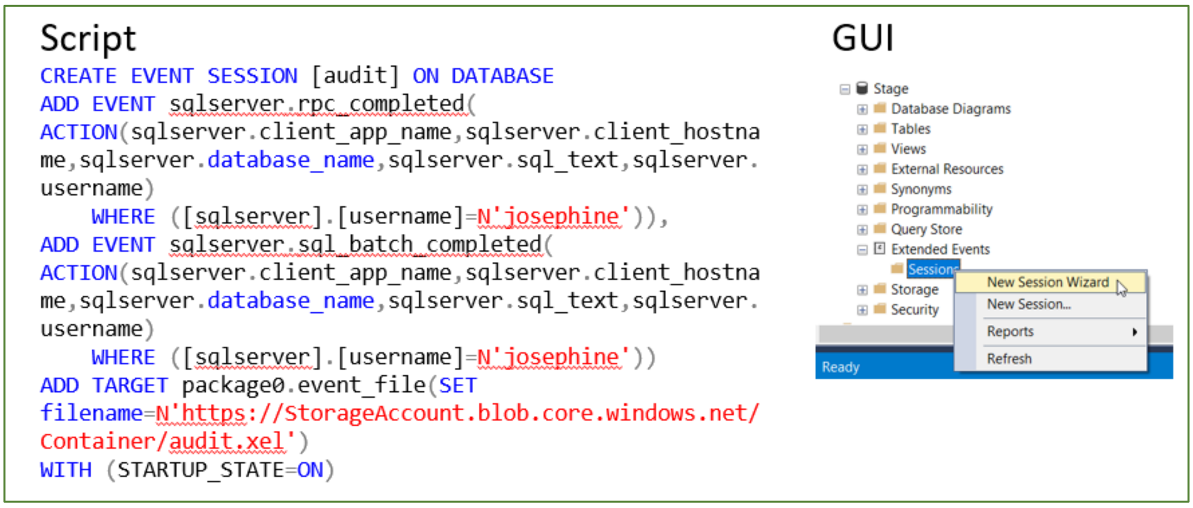
event_time_t [UTC]	statement_s	succeeded_s	affected_rows_d	server_principal_name_s
2/22/2021, 11:21:36.019 PM	ALTER TABLE dbo.testing SET (LOCK_ESCALATION = TABLE)	true	0	josephine
2/22/2021, 11:21:35.972 PM	CREATE TABLE dbo.testing (testing nchar(10) NULL) ON [PRIMARY]	true	0	josephine
2/22/2021, 11:21:35.941 PM		true	0	josephine
2/22/2021, 11:21:35.894 PM	DECLARE @edition sysname; SET @edition = cast(SERVERPROPERTY...	true	4	josephine

Even if you enable the auditing at the server level, you still have to access the audit data via the database. It's easier to go into your log analytics workspace to see all the audit data in one place instead of clicking on different databases to see the data. That's why I store all my database audit data in the same log analytics workspace. Then I can query all the database audit data at the same time.

To query the audit data in log analytics, you need to use Kusto Query Language (KQL). It's easy to figure out, especially if you know SQL. Here's a link to a SQL to Kusto cheat sheet: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/sqlcheatsheet

Extended events for Azure SQL Database

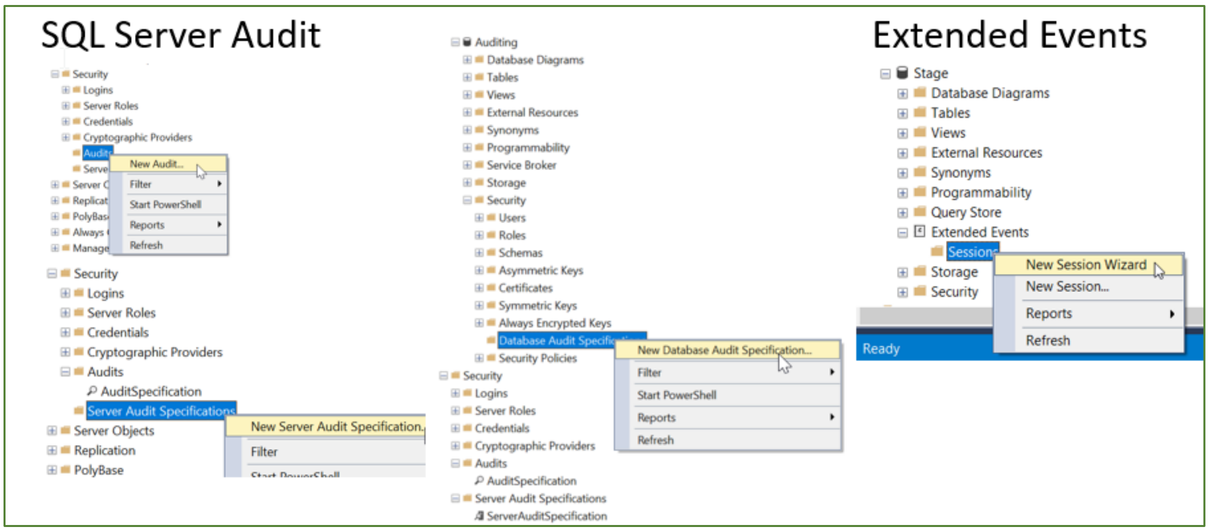
The way you set up xevents in Azure SQL Database is the same way you do it in SQL Server. The main difference is you need a storage account to store the files. The following screenshot shows you how you can set up an xevent with a query or in the GUI:



```
CREATE EVENT SESSION [audit] ON DATABASE
ADD EVENT sqlserver.rpc_completed(
ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.database_name,sqlserver.sql_text,sqlserver.username)
    WHERE ([sqlserver].[username]=N'josephine')),
ADD EVENT sqlserver.sql_batch_completed(
ACTION(sqlserver.client_app_name,sqlserver.client_hostname,sqlserver.database_name,sqlserver.sql_text,sqlserver.username)
    WHERE ([sqlserver].[username]=N'josephine'))
ADD TARGET package0.event_file(SET
filename=N'https://StorageAccount.blob.core.windows.net/Container/audit.xel')
WITH (STARTUP_STATE=ON)
```

Azure Managed Instance and Amazon Web Services Relational Database Service Auditing

These managed database options both have the same auditing options as a SQL Server running on VM, except for the fact that they need to use cloud storage. The following screenshot shows you the setup for SQL Server Audit and Extended Events for Managed Instance and AWS RDS:


SQL Server Audit
Extended Events
New Audit...
New Server Audit Specification...
New Database Audit Specification...
New Session Wizard
New Session...
Server Audit Specifications
Database Audit Specification
Sessions

You may have to take into consideration security, networking, and other settings for getting your managed instance to communicate with blob storage and/or getting your RDS instance communicating with s3

Disclaimer on Auditing

Be very careful how and what you audit. You can overload or freeze up a production server. It happened to me when I didn't even think it was possible to crash a production server on a VM with a SQL Server Audit. I just thought somehow, I could stop the audit in between things it was auditing. Sometimes it's going to be hard to stop your audit if it's auditing too much or going to be so hard to weed through all the data, it will be like trying to find a needle in a haystack with all the stuff you're auditing. I just go with the less is more method of auditing whether I'm using extended events or SQL Server audit.

This is not as important with Azure SQL Database auditing in the portal. Even though Microsoft doesn't allow you to filter your auditing data before it's collected, it's still a lightweight option for auditing. I would still use caution with auditing on a very busy production database. That's why it may be better to use xevents in that case, so you can audit more specifically only what you need.

Conclusion

When you are in charge of data you must take that responsibility with the level of seriousness that it truly requires. Guaranteeing specific levels of RTO and RPO are part of this responsibility and you need to do everything in your power to make sure you can meet them. These cases and scenarios shared here can help you cover some gaps in your general strategy so that you are never caught off-guard. And remember, even if you are in the managed Azure SQL Db cloud offering, you should still verify your configuration and test your procedures to make sure your RPO and RTOs are being met. Thanks for reading!

Additional Information

Azure SQL Audit Overview

https://docs.microsoft.com/en-us/azure/azure-sql/database/auditing-overview

Azure SQL Extended Events

https://docs.microsoft.com/en-us/azure/azure-sql/database/xevent-db-diff-from-svr

Azure SQL Managed Instance Auditing

https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/auditing-configure

AWS RDS Auditing

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.Options.Audit.html

https://aws.amazon.com/blogs/database/set-up-extended-events-in-amazon-rds-for-sql-server/

Questions? Comments? Talk to the author today. Josephine Bush on Twitter.

About Josephine Bush

Josephine Bush has over 10 years of experience as a Database Administrator.

LEARN MORE

Non-Tech World of Josephine Bush

During her free time, Josephine loves knitting, painting, and yoga.

Want to write for the magazine? Comments? Feedback? Reach out to us at magazine@sqlservergeeks.com

AB SQL SERVER NOTES

When your query is waiting on resources, a specific wait type is assigned to it by SQL Server which you can track & capture.
READ MORE…

There are many types of memory allocations in SQL Server. One such memory grant is for query execution.
READ MORE…

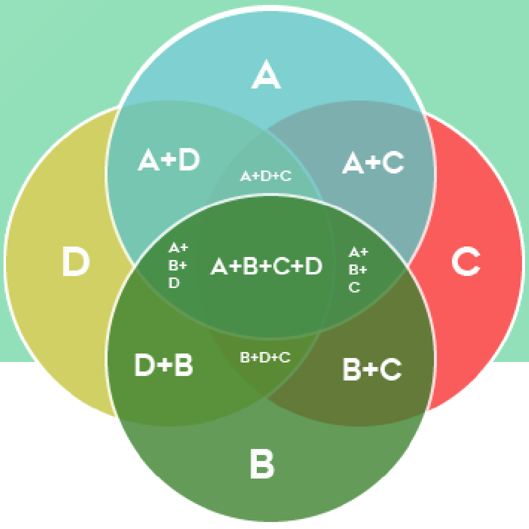
You have just moved into a new residential society with 5 huge towers. Each tower has 50 floors and each floor has 2 apartments.
READ MORE…

The SQL Optimizer is heavily dependent on Statistics for the estimates it makes and thereby a cheap execution plan.
READ MORE…

You create indexes. SQL Server creates statistics. So many of these objects get created to support your query. Stats help in estimations.
READ MORE…

LEARN MORE

A TIP ABOUT INTERSECT



Erland Sommarskog

This article is based on something I originally learnt from a blog post from my MVP colleague Paul White. I also need to thank Diethard Michaelis who was kind to point me to Paul's post.

In this month's tip you will learn a compact way to express your query when you want to handle NULL like any other value in comparisons. Let's start with setting up a demo table:

```
USE tempdb
go
DROP TABLE IF EXISTS DemoTable
CREATE TABLE DemoTable(a int NOT NULL,
                       b int NOT NULL,
                       c int NULL,
                       d int NULL,
                       CONSTRAINT pk_DemoTable PRIMARY KEY (a, b),
                       INDEX c_d_ix NONCLUSTERED (c, d)
)
go
INSERT DemoTable (a, b, c, d)
 SELECT object_id, column_id,
   IIF(max_length = -1, NULL, abs(checksum(*) % 200)),
   IIf(len(name) = 8, NULL, abs(checksum(*) % 100))
 FROM   sys.columns
```

This fills up DemoTable with a couple of hundreds of rows, exactly how many depends on your SQL Server version. You can run this SELECT to verify that some rows will have NULL in both of the columns **c** and **d**.

```
SELECT * FROM DemoTable WHERE c IS NULL AND d IS NULL
```

Here is a simple stored procedure to return data by the values in **c** and **d**:

```
CREATE OR ALTER PROCEDURE GetDataByCD @c int, @d int AS
SELECT a, b, c, d
FROM   DemoTable
WHERE  c = @c
  AND  d = @d
```

You can test the procedure with display of execution plan enabled:

```
EXEC GetDataByCD 64, 64
```

It returns a couple of rows, and you can see that the plan is an Index Seek.

Say now that the business rules for this table are such that it makes sense to also search for the rows where any of **c** or **d** has a NULL value. But if you try:

```
EXEC GetDataByCD NULL, NULL
```

You get no rows back, despite that the query above told us that there are matching rows. If we recall how NULL values work in SQL this is not surprising. NULL represents an unknown value, and when comparing a NULL with something else, even another NULL, we don't know if they are equal or not. By chance, they could have could be the same unknown value, but then again, they may not. Therefore, the result is not TRUE or FALSE, but UNKNOWN by the three-valued logic of SQL. But rows are only returned if the condition in the WHERE clause evaluates to TRUE, and whence no rows are returned.

To solve this, we need to change the procedure. A popular way to solve the issue is to use **isnull** or **coalesce**. Here is an example with the former:

```
CREATE OR ALTER PROCEDURE GetDataByCD @c int, @d int AS
SELECT a, b, c, d
FROM   DemoTable
WHERE  isnull(c, -1) = isnull(@c, -1)
   AND  isnull(d, -1) = isnull(@d, -1)
go
EXEC GetDataByCD 64, 64
EXEC GetDataByCD NULL, NULL
```

Now we get back rows with the second call as well.

There are however two problems with this solution. One is that it relies on a "magic" value, that you assume is not present in the data. It is not uncommon that such assumptions are safe to make. For instance, we can see from how this table is filled up that there can be no negative values in **c** or **d**. However, casual use of this technique can lead to incorrect results when you are assumptions are wrong. The second problem is performance. If you check the execution plan, you will find that there is an Index Scan. The optimizer in SQL Server does not have any rule to untangle **isnull** or **coalesce** to permit an Index Seek.

Here is a better way to do this:

```
CREATE OR ALTER PROCEDURE GetDataByCD @c int, @d int AS
SELECT a, b, c, d
FROM   DemoTable
WHERE  (c = @c OR c IS NULL AND @c IS NULL)
  AND  (d = @d OR d IS NULL AND @d IS NULL)
go
EXEC GetDataByCD NULL, NULL
```

Not only do we get the correct results. This time, we also get an Index Seek.

You may however find this a bit long-winded, and if you forget to place the parentheses correctly, you will get incorrect results.

The ANSI standard for SQL defines an operator that permits us to write this a little more compactly:

```
WHERE  c IS NOT DISTINCT FROM @c
  AND  d IS NOT DISTINCT FROM @d
```

This is analogous to the DISTINCT operator in SELECT, where all NULL values are handled as one and the same. This operator is implemented by for instance PostgreSQL, but it is not available in SQL Server. Maybe that is just as well, because as you may notice the operator is defined “backwards”; you have to put in a NOT when you are looking for equality. And it still is a bit verbose.

However, there is another alternative that permits for a more compact form, and that is available in SQL Server and that is the INTERSECT set operator. Before we apply it to this query, let’s first repeat the operator as such. Consider these tables:

```
CREATE TABLE #alpha(a int NULL)
INSERT #alpha(a) VALUES(1), (2), (3), (NULL)
CREATE TABLE #beta(a int NULL)
INSERT #beta(a) VALUES(3), (4), (5), (NULL)
```

We all know the UNION operator:

```
SELECT a FROM #alpha
UNION
SELECT a FROM #beta
```

This returns NULL, 1, 2, 3, 4, 5, that is, the union of the two sets. NULL and 3 that appear in both sets, occur only once in the result set, because UNION without ALL implies DISTINCT.

Note: The ANSI standard also defines INTERSECT ALL and EXCEPT ALL, analogous to UNION ALL, but they are not implemented in SQL Server

Here I had only a single column to keep the examples simple. But as with UNION, you can have any number of columns in an INTERSECT operation. Therefore, we can write our stored procedure as:

```
CREATE OR ALTER PROCEDURE GetDataByCD @c int, @d int AS
SELECT a, b, c, d
FROM   DemoTable
WHERE  EXISTS (SELECT c, d INTERSECT SELECT @c, @d)
go
EXEC GetDataByCD NULL, NULL
```

The subquery with INTERSECT is logically evaluated for every row, and if the two sets have the same values, a row is returned, else not. And as discussed above, NULL is not a special case here, but works like any other value.

The query returns the correct result, but not only that: the execution plan uses an Index Seek.

What is interesting here is that if you have plenty of columns and there is one or more where you need to handle NULL as equal to NULL, you can use the above pattern for a very compact way of expressing your WHERE clause. Or for that matter an ON condition in a join where you want to join on NULL values. At the same time, you can still expect the optimizer to use appropriate indexes.

Let’s now look at a situation where INTERSECT and EXCEPT really shine.

Here is one more demo table with similar, but not identical data. We also add one extra row to each table:

```
CREATE TABLE DemoTable2(a int NOT NULL,
                        b int NOT NULL,
                        c int NULL,
                        d int NULL,
                        CONSTRAINT pk_DemoTable2 PRIMARY KEY (a, b),
                        INDEX c_d_ix NONCLUSTERED (c, d)
)
go
INSERT DemoTable2 (a, b, c, d)
       SELECT object_id, column_id, iif(len(name) = 8, NULL, abs(checksum(*) %
       200)),
   IIf(max_length = -1, NULL, abs(checksum(*) % 100))
FROM   sys.columns

INSERT DemoTable(a, b, c, d)
       VALUES(10000, 1, 13, 13)
INSERT DemoTable2(a, b, c, d)
       VALUES(10000, 2, 13, 13)
SELECT object_id, column_id, iif(len(name) = 8, NULL, abs(checksum(*) % 200)),
   IIf(max_length = -1, NULL, abs(checksum(*) % 100))
FROM   sys.columns

INSERT DemoTable(a, b, c, d)
       VALUES(10000, 1, 13, 13)
INSERT DemoTable2(a, b, c, d)
       VALUES(10000, 2, 13, 13)
```

Say that these two tables are the result of two variations of a query or a stored procedure that you have rewritten to improve performance. You want to verify that the result is the same, because better performance is worth little if the result is wrong.

I touched at this problem in my article about full outer join last month, and I gave an outline for how to write a query to compare the tables. Applying this outline to this pair of tables we get:

```
SELECT *
FROM   DemoTable X
FULL OUTER JOIN DemoTable2 Y ON X.a = Y.a
                            AND X.b = Y.b
WHERE X.a IS NULL OR Y.a IS NULL OR
       X.c <> X.c OR
              (X.c IS NULL AND Y.c IS NOT NULL) OR
              (X.c IS NOT NULL AND Y.c IS NULL) OR
       X.d <> X.d OR
     (X.d IS NULL AND Y.d IS NOT NULL) OR
     (X.d IS NOT NULL AND Y.d IS NULL)
```

The first condition on one of the key columns is to find key values that are only in one of the tables. The remaining conditions serve to find difference in values, including differences related to NULL.

This is extremely tedious to write, and you can easily slip on one of the conditions. This is only two columns, but in a real-world case, you could easily have ten or twenty columns. It can be made shorter with **isnull** or **coalesce**, if you accept to rely on “magic” values. But even with these functions, the code will still be long-winded and error-prone.

However, we can make this a lot shorter with INTERSECT. And I mean a *lot* shorter:

```
SELECT *
FROM   DemoTable X
FULL OUTER JOIN DemoTable2 Y ON X.a = Y.a
                          AND X.b = Y.b
WHERE NOT EXISTS (SELECT X.* INTERSECT SELECT Y.*)
```

That’s not bad, is it?

Although, It is only for the special case when you want to compare each and every column that you can cut it this short with SELECT *. It is not uncommon, that you for one reason or another need to exempt a column or two from the comparison. (For instance, a datetime column holding the time when the row was created.) In that case you will need to list the columns twice, on both sides of the INTERSECT. But the query will still be a lot shorter compared to the pattern above.

Here I used NOT EXISTS + INTERSECT, but it also possible to write the query with EXISTS + EXCEPT:

```
SELECT *
FROM   DemoTable X
FULL OUTER JOIN DemoTable2 Y ON X.a = Y.a
                          AND X.b = Y.b
WHERE EXISTS (SELECT X.* EXCEPT SELECT Y.*)
```

Which form to use is a matter of taste.

When I first read Paul White’s blog post https://www.sql.kiwi/2011/06/undocumented-query-plans-equality-comparisons.html a few years back, I was amazed that I still could learn something about T-SQL that helped to enhance my technique for writing queries, and I have applied this method several times since then. I hope that you will also find use for the INTERSECT operator in your queries when you want to handle NULL values as equal.

Questions? Comments? Talk to the author today. Erland Sommarskog.

About Erland Sommarskog

Erland Sommarskog is an independent consultant based in Stockholm, working with SQL Server since 1991. He was first awarded SQL Server MVP in 2001, and has been re-awarded every year since.

LEARN MORE

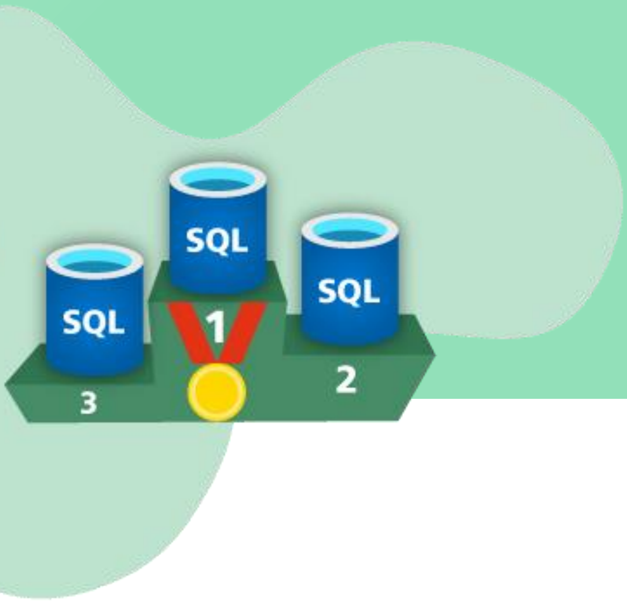
Non-Tech World of Erland Sommarskog

Erland plays bridge (when there is not a pandemic), enjoys travelling (when there is not a pandemic), and in the far too short Swedish summer he goes hiking and biking in the forests and around the lakes around Stockholm.

Want to write for the magazine? Comments? Feedback? Reach out to us at magazine@sqlservergeeks.com

SQL NUGGETS BY MICROSOFT

- Released: SCOM Management Pack for SQL Server (7.0.36.0)
- OLE DB Driver 19.0 for SQL Server Released
- JDBC Driver 10.2 for SQL Server Released
- Microsoft Drivers 5.10 for PHP for SQL Server Released
- Cumulative Update #15 for SQL Server 2019 RTM
- Released: Microsoft.Data.SqlClient 4.0.1
- Cumulative Update #28 for SQL Server 2017 RTM



SQL SERVER RANKING FUNCTIONS

SQLMaestros Hands-On-Labs | @SQLMaestros

ROW NUMBER Function

Scenario:

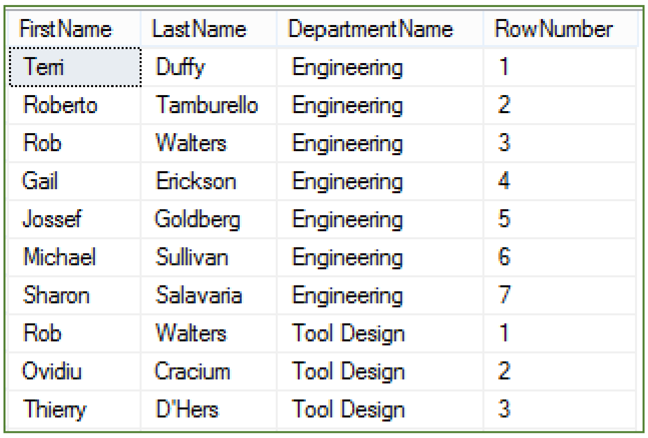
In this exercise, we will understand the concept of **ROW_NUMBER** function and how it works.

ROW_NUMBER: **A ROW_NUMBER ()** function returns a sequential numbering to the records with in the partition of a result set.

Problem Statement: "Calculate a row number to each employee in each department"

Table(s) Involved: HumanResources.EmployeeDepartmentHistory table, **Person.Person** table, **HumanResources.Department** table

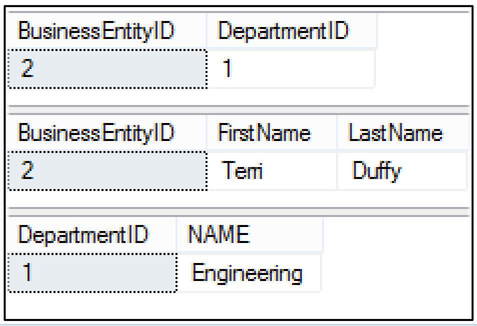
Final Output:



FirstName	LastName	DepartmentName	RowNumber
Terri	Duffy	Engineering	1
Roberto	Tamburello	Engineering	2
Rob	Walters	Engineering	3
Gail	Erickson	Engineering	4
Jossef	Goldberg	Engineering	5
Michael	Sullivan	Engineering	6
Sharon	Salavaria	Engineering	7
Rob	Walters	Tool Design	1
Ovidiu	Cracium	Tool Design	2
Thierry	D'Hers	Tool Design	3

NOTE: The above figure is a trimmed snapshot of the original result set. The actual result set contains 296 rows.

Tasks	Detailed Steps
Launch **SQL Server Management Studio**	1. Click **Start** | **All Programs** | **SQL Server 2012** | **SQL Server Management Studio** 2. In the **Connect to Server** dialog box, click **Connect**
Open **1_RowNumber.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Ranking Functions\Scritps** folder 3. Select **1_RowNumber.sql** and click **Open**

Understanding table structure(s)	Execute the following statement(s) to understand the table structure of **HumanResources.EmployeeDepartmentHistory** table, **Person.Person** table and **HumanResources.Department** table
	`-- Step 1: Execute the following statement(s)`
	`SET NOCOUNT ON;`
	`GO`
	
	`USE AdventureWorks2012`
	`GO`
	`SELECT BusinessEntityID`
	`       ,DepartmentID`
	`FROM [HumanResources].[EmployeeDepartmentHistory]`
	`WHERE BusinessEntityID = 2`
	`GO`
	
	`SELECT BusinessEntityID`
	`       ,FirstName`
	`       ,LastName`
	`FROM Person.Person`
	`WHERE BusinessEntityID = 2`
	`GO`
	`SELECT DepartmentID`
	`       ,NAME`
	
	`FROM HumanResources.Department`
	`WHERE DepartmentID = 1`
	`GO`
	
	BusinessEntityID | DepartmentID
	2 | 1
	
	BusinessEntityID | FirstName | LastName
	2 | Terri | Duffy
	
	DepartmentID | NAME
	1 | Engineering

	Explanation: HumanResources.EmployeeDepartmentHistory table contains current and historical data about employees and their departments in which they work. The **BusinessEntityID** column represents the employee id, **DepartmentID** column represents the id of the department in which the employee works or worked. **Person.Person** table contains name and id of each individual customers, employees and vendors. The **BusinessEntityID** column represents the id of each individual involved in adventure works database and **FirstName**, **LastName** column represents the name of the person**. HumanResources.Department** table contains current department's information. The **DepartmentID** column represents the id of each department and **Name** column represents the name of the department. **Observation:** From the above figure we can observe that from **HumanResources.EmployeeDepartmentHistory** table for the **BusinessEntityID '2'** the **DepartmentID** is '1'. From the **Person. Person** table, we can observe that for the **BusinessEntiityID** '2' who's **FirstName** and **LastName** is 'Terri Duffy'. From the **HumanResources. Department** table, we can observe that for the **DepartmentID** '1' the **Name** of the department is 'Engineering'. From this observation, in order to solve the problem statement, we have to retrieve **FirstName**, **LastName** and Department **Name** of each employee for this we need to correlate **EmployeeDepartmentHistory** table **BusinessEntityID** column with **Person. Person** table **BusinessEntityID** column and **EmployeeDepartmentHistory** table **DepartmentID** column with **Human Resources. Department** table **DepartmentID** column and finally we have to assign a sequential number to each employee in each and every department.

Execute a **SELECT** statement(s)	Execute the following statement(s) to calculate a row number to each employee in each department"
	
	`-- Step 2: Execute the following statement(s)`
	`SELECT P.FirstName`
	`       ,P.LastName`
	`       ,D.NAME DepartmentName`
	`       ,ROW_NUMBER() OVER (`
	`             PARTITION BY EDH.DepartmentID ORDER BY`
	`P.BusinessEntityID`
	`             ) RowNumber`
	`FROM [HumanResources].[EmployeeDepartmentHistory] EDH`
	`INNER JOIN Person.Person P ON P.BusinessEntityID =`
	`EDH.BusinessEntityID`
	`INNER JOIN [HumanResources].[Department] D ON`
	`D.DepartmentID = EDH.DepartmentID`
	`GO`
	
	Explanation: From the above query we are retrieving **FirstName, LastName** from **Person. Person** table and **Name** of the department from

	Human Resources. Department table, using **ROW_NUMBER ()** we are assigning sequential numbers to the each employee based on their **BusinessEntityID** partitioned by **DepartmentID. OVER** clause is required before ranking functions. **PARTITION BY** is not mandatory, it is going to divide the result-set into partitions and finally ranking will be given to each partition starts with '1'. By using **ORDER BY** clause in ranking functions we are sorting the order of records within a result-set and the **ORDER BY** clause is mandatory in ranking functions.

Additional example on **ROW_NUMBER ()**	**Problem Statement:** "Calculate the sum of unit prices of each sales order in every year and find maximum two sales order's in each year"
	
	Table(s) involved: Sales.SalesOrderDetail table
	
	Solution: Execute the **SELECT** statement in **step 2** to solve the above problem
	
	`-- Step 3: Additional problem`
	`SELECT SalesOrderID`
	`       ,UnitPrice`
	`       ,ModifiedYear`
	`       ,RowNumber`
	`FROM (`
	
	`         SELECT SalesOrderID`
	`              ,UnitPrice`
	`              ,ModifiedYear`
	`              ,ROW_NUMBER() OVER (`
	`                     PARTITION BY ModifiedYear ORDER BY`
	`UnitPrice DESC`
	`                     ) RowNumber`
	`       FROM (`
	`              SELECT SalesOrderID`
	`                     ,round(sum(UnitPrice), 2) UnitPrice`
	`                     ,year(ModifiedDate) ModifiedYear`
	`              FROM sales.SalesOrderDetail`
	`              GROUP BY SalesOrderID`
	`                     ,year(ModifiedDate)`
	`              ) SOD`
	`       ) SOD1`
	`WHERE RowNumber BETWEEN 1`
	`              AND 2`
	`GO`

Close all the query windows	Close all the query windows (☒) and if **SSMS** asks to save changes, click **NO**

RANK Function

Scenario

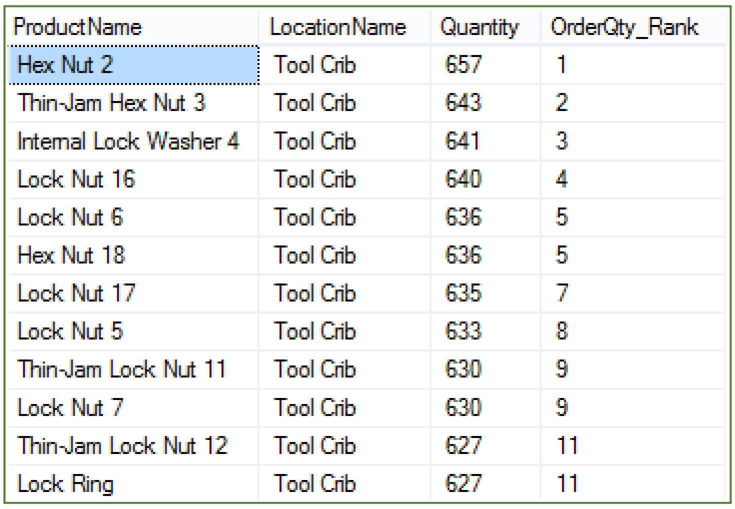
In this exercise, we will understand the concept of **RANK ()** function.

RANK: A **RANK ()** function returns rank to each record within the partition of a result set. If two or more records tie for a rank, each tied row returns same rank, after the next record it will not provide consecutive rank.

Problem Statement: "Assign rank to each product based on the amount of quantity in each location"

Table(s) involved: Production.ProductInventory table, **Production.Product** table and **Production.Location** table

Final Output:



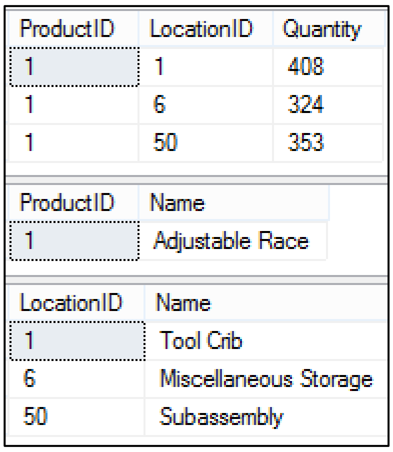
ProductName	LocationName	Quantity	OrderQty_Rank
Hex Nut 2	Tool Crib	657	1
Thin-Jam Hex Nut 3	Tool Crib	643	2
Internal Lock Washer 4	Tool Crib	641	3
Lock Nut 16	Tool Crib	640	4
Lock Nut 6	Tool Crib	636	5
Hex Nut 18	Tool Crib	636	5
Lock Nut 17	Tool Crib	635	7
Lock Nut 5	Tool Crib	633	8
Thin-Jam Lock Nut 11	Tool Crib	630	9
Lock Nut 7	Tool Crib	630	9
Thin-Jam Lock Nut 12	Tool Crib	627	11
Lock Ring	Tool Crib	627	11

NOTE: The above figure is a trimmed snapshot of the original result set. The actual result set contains 1069 rows.

Tasks	Detailed Steps
Open **2_RankFunction.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Ranking Functions\Scripts** folder 3. Select **2_RankFunction.sql** and click **Open**

Understanding table structure(s)	Execute the following statement(S) to understand the table structure of **Production.ProductInventory** table, **Production.Product** table and **Production.Location** table
	
	`-- Step 1: Execute the following statement(s)`
	`SET NOCOUNT ON;`
	`GO`
	`USE AdventureWorks2012`
	`GO`

```
SELECT ProductID ,LocationID ,Quantity FROM
Production.ProductInventory WHERE ProductID =1
GO
SELECT ProductID ,Name  FROM  Production.Product WHERE
ProductID =1
GO
SELECT LocationID, Name  FROM  Production.Location WHERE
LocationID in(1,6,50)
GO
```



ProductID	LocationID	Quantity
1	1	408
1	6	324
1	50	353

ProductID	Name
1	Adjustable Race

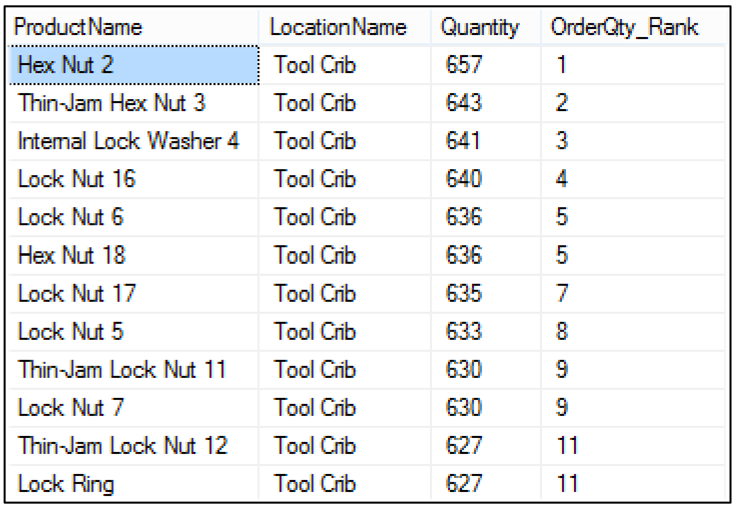
LocationID	Name
1	Tool Crib
6	Miscellaneous Storage
50	Subassembly

Explanation: Production.ProductInventory table contains the quantity of products in the inventory location. **ProductID** column represents the id of the product, **LocationID** column represents the id of the inventory location and **Quantity** represents the quantity of each products in a particular location. **Production.Product** table contains detailed information of each product. **ProductID** column represents the id of the product, **Name** column represents the name of the product. **Production.Location** table contains product inventory and manufacturing location. **LocationID** column represents the id of the location, Name column represents the description of the location name.

Observation: From the above figure we can observe that from the **Production.ProductInventory** table for the **ProductID** '1' we get the list of all **LocationID**'s and their **Quantity.** From the **Production.Product** table, we can observe that for the **ProductID** '1' the name of the product is 'Adjustable Race'. From the **Production. Location** table, we can observe that for the list of **LocationID**'s '1', '6', '50' the name of the locations is 'Tool Crib', 'Miscellaneous Storage', 'Subassembly'. From this observation, in order to solve the problem statement, we need to correlate **Production.ProductInventory** table **ProductID** column with **Production. Product** table **ProductID** column and **Production.ProductInventory** table **LocationID** column with **Production. Location** table **LocationID** column and finally we have to assign a rank to each **ProductID** based on the amount of **Quantity** partitioned by **LocationID.**

Execute the **SELECT** statement(s)	Execute the following statement(s) to assign rank to each product based on the amount of quantity in each location

```
-- Step 2: Execute the following statement(s)
 SELECT P.NAME ProductName
       ,L.NAME LocationName
       ,PIN.Quantity
       ,RANK() OVER (
             PARTITION BY PIN.LocationID ORDER BY 
PIN.Quantity DESC
             ) OrderQty_Rank
FROM Production.ProductInventory PIN
INNER JOIN Production.Product P ON P.ProductID = 
PIN.ProductID
INNER JOIN Production.Location L ON L.LocationID = 
PIN.LocationID
GO
```



ProductName	LocationName	Quantity	OrderQty_Rank
Hex Nut 2	Tool Crib	657	1
Thin-Jam Hex Nut 3	Tool Crib	643	2
Internal Lock Washer 4	Tool Crib	641	3
Lock Nut 16	Tool Crib	640	4
Lock Nut 6	Tool Crib	636	5
Hex Nut 18	Tool Crib	636	5
Lock Nut 17	Tool Crib	635	7
Lock Nut 5	Tool Crib	633	8
Thin-Jam Lock Nut 11	Tool Crib	630	9
Lock Nut 7	Tool Crib	630	9
Thin-Jam Lock Nut 12	Tool Crib	627	11
Lock Ring	Tool Crib	627	11

Explanation: From the above query we are retrieving Name of the product from **Production.Product** table, name of the location from **Production.Location** table and Quantity of each product from **Production.ProductInventory** table, using **Rank ()** we are assigning rank to each product based on the amount of quantity partitioning by **LocationID**

Observation: From the above figure we can observe that for the product's 'Thin-Jam Lock Nut 11' and 'Lock Nut 7' is having the same quantity '630' hence the rank is assigned '9' to both of the products and for the next product the rank is assigned '11'. **RANK ()** function skips the consecutive rank after the tied records.

Additional problem on **RANK ()** Function	**Problem:** "Assign rank to each employee based on the number of purchases created" **Table(s) Involved: Purchasing.PurchaseOrderHeader** table , **Person.Person** table

	Solution: Execute the **SELECT** statement in **step 3** to solve the above problem
	
	`-- Step 3: Additional problem`
	`WITH POD_CTE`
	`AS (`
	`       SELECT P.FirstName`
	`              ,P.LastName`
	`              ,count(PurchaseOrderID) PurchaseOrder_Count`
	`       FROM [Purchasing].[PurchaseOrderHeader] PO`
	`       INNER JOIN person.Person P ON P.BusinessEntityID =`
	`PO.EmployeeID`
	`       GROUP BY P.FirstName`
	`              ,P.LastName`
	`       )`
	`SELECT FirstName`
	`       ,LastName`
	`       ,PurchaseOrder_Count`
	`       ,rank() OVER (`
	`              ORDER BY PurchaseOrder_Count`
	`              ) 'Rank'`
	`FROM POD_CTE`
	`GO`
	
	Note: **WITH POD_CTE** represents common table expression, which is used to hold temporary result set.

Close all the query windows	Close all the query windows (✖) and if **SSMS** asks to save changes, click **NO**

DENSE RANK Function

Scenario

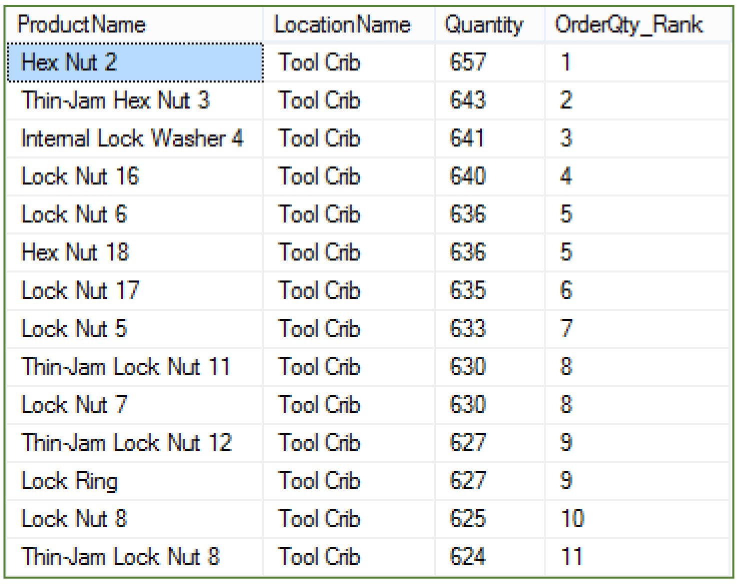
In this exercise, we will understand the concept of **DENSE_RANK ()** function.

DENSE_RANK: A **DENSE_RANK ()** function returns consecutive ranking to each record with in the partition of a result set. If two or more records tie for a rank, each tied row returns same rank, after the next record it provide consecutive rank.

Problem Statement: "Assign rank to each product based on the amount of quantity in each location"

Table(s) involved: Production.ProductInventory table, **Production.Product** table, **Production.Location** table

Final Output

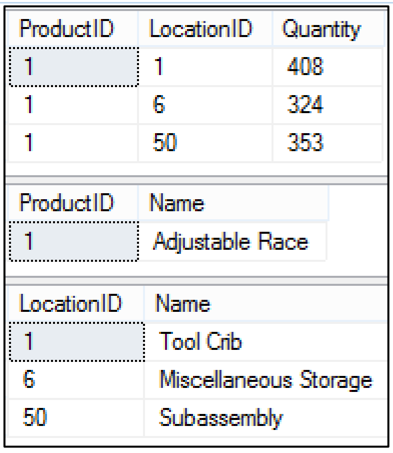


ProductName	LocationName	Quantity	OrderQty_Rank
Hex Nut 2	Tool Crib	657	1
Thin-Jam Hex Nut 3	Tool Crib	643	2
Internal Lock Washer 4	Tool Crib	641	3
Lock Nut 16	Tool Crib	640	4
Lock Nut 6	Tool Crib	636	5
Hex Nut 18	Tool Crib	636	5
Lock Nut 17	Tool Crib	635	6
Lock Nut 5	Tool Crib	633	7
Thin-Jam Lock Nut 11	Tool Crib	630	8
Lock Nut 7	Tool Crib	630	8
Thin-Jam Lock Nut 12	Tool Crib	627	9
Lock Ring	Tool Crib	627	9
Lock Nut 8	Tool Crib	625	10
Thin-Jam Lock Nut 8	Tool Crib	624	11

NOTE: The above figure is a trimmed snapshot of the original result set. The actual result set contains 1069 rows.

Tasks	Detailed Steps
Open **3_Dense_Rank.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Ranking Functions\Scripts** folder 3. Select **3_Dense_Rank.sql** and click **Open**

Understanding table structure(s)	Execute the following statement(S) to understand the table structure of **Production.ProductInventory** table, **Production.Product** table and **Production.Location** table
	
	`-- Step 1: Execute the following statement(s)`
	`USE AdventureWorks2012`
	`GO`
	`SET NOCOUNT ON;`
	`SELECT ProductID ,LocationID ,Quantity  FROM Production.ProductInventory WHERE ProductID =1`
	`GO`
	`SELECT ProductID ,Name  FROM  Production.Product WHERE ProductID =1`
	`GO`
	`SELECT LocationID, Name  FROM Production.Location WHERE LocationID in(1,6,50)`

ProductID	LocationID	Quantity
1	1	408
1	6	324
1	50	353

ProductID	Name
1	Adjustable Race

LocationID	Name
1	Tool Crib
6	Miscellaneous Storage
50	Subassembly

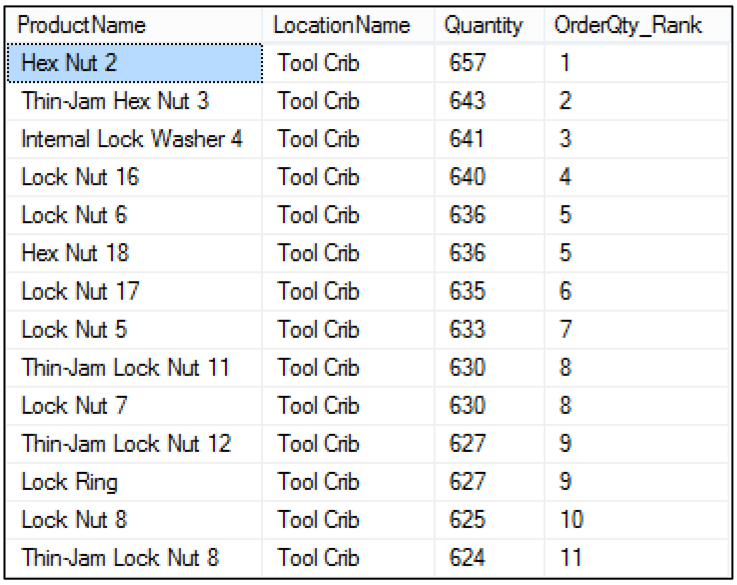
Explanation: Production.ProductInventory table contains the quantity of products in each inventory location. **ProductID** column represents the id of the product, **LocationID** column represents the id of the inventory location and **Quantity** represents the quantity of each products in a particular location. **Production.Product** table contains detailed information of each product. **ProductID** column represents the id of the product, **Name** column represents the name of the product. **Production.Location** table contains product inventory and manufacturing location. **LocationID** column represents the id of the location, **Name** column represents the description of the location name.

Observation: From the above figure we can observe that from the **Production.ProductInventory** table for the **ProductID** '1' we get the list of all **LocationID**'s and their **Quantity.** From the **Production.Product** table, we can observe that for the **ProductID** '1' the name of the product is 'Adjustable Race'. From the **Production.Location** table, we can observe that for the list of **LocationID**'s '1', '6', '50' the name of the locations is 'Tool Crib', 'Miscellaneous Storage', 'Subassembly'. From this observation, in order to solve the problem statement, we need to correlate **Production.ProductInventory** table **ProductID** column with **Production.Product** table **ProductID** column and **Production.ProductInventory** table **LocationID** column with **Production.Location** table **LocationID** column and finally we have to assign a rank to each **ProductID** based on the amount of **Quantity** partitioned by **LocationID.**

Execute a **SELECT** statement(s)

Execute the following statement(s) to assign rank to each product based on the amount of quantity in each location

```
-- Step 2: Execute the following statement(s)
  SELECT P.NAME ProductName
        ,L.NAME LocationName
        ,PIN.Quantity
        ,DENSE_RANK() OVER (
              PARTITION BY PIN.LocationID ORDER BY
PIN.Quantity DESC
```

	```
	                 ) OrderQty_Rank
	FROM Production.ProductInventory PIN
	INNER JOIN Production.Product P ON P.ProductID =
	PIN.ProductID
	INNER JOIN Production.Location L ON L.LocationID =
	PIN.LocationID
	GO
	```
	  **Explanation**: From the above query we are retrieving **Name** of the product from **Production.Product** table, **Name** of the location from **Production.Location** table and Quantity of each product from **Production.ProductInventory** table, using **DENSE_RANK ()** we are assigning rank to each product based on the amount of quantity partitioning by **LocationID**
	
	Observation: From the above figure we can observe that for the product's 'Thin-Jam Lock Nut 11' and 'Lock Nut 7' is having the same quantity '630' hence the rank is assigned '8' to both of the products and for the next product the rank is assigned '9'. **DENSE_RANK ()** function returns consecutive rank after the tied records.

Additional problem on **DENSE_RANK ()**	**Problem:** "Assign rank to each sales person based on the amount of sales year to date in each year"
	
	Table(s) involved: Sales.SalesPerson table
	```
	-- Step 3: Additional problem
	SELECT P.FirstName
	      ,P.LastName


	      ,SP.SalesYTD
	      ,year(SP.ModifiedDate) ModifiedYear
	      ,DENSE_RANK() OVER (
	```

Screenshot result:

ProductName	LocationName	Quantity	OrderQty_Rank
Hex Nut 2	Tool Crib	657	1
Thin-Jam Hex Nut 3	Tool Crib	643	2
Internal Lock Washer 4	Tool Crib	641	3
Lock Nut 16	Tool Crib	640	4
Lock Nut 6	Tool Crib	636	5
Hex Nut 18	Tool Crib	636	5
Lock Nut 17	Tool Crib	635	6
Lock Nut 5	Tool Crib	633	7
Thin-Jam Lock Nut 11	Tool Crib	630	8
Lock Nut 7	Tool Crib	630	8
Thin-Jam Lock Nut 12	Tool Crib	627	9
Lock Ring	Tool Crib	627	9
Lock Nut 8	Tool Crib	625	10
Thin-Jam Lock Nut 8	Tool Crib	624	11

	```
	              PARTITION BY year(SP.ModifiedDate) ORDER BY
	SP.SalesYTD desc
	              ) 'Rank'
	FROM sales.SalesPerson SP
	INNER JOIN person.Person P ON P.BusinessEntityID =
	SP.BusinessEntityID
	GO
	```
Close all the query windows	Close all the query windows (✖) and if **SSMS** asks to save changes, click **NO**

NTILE Function

Scenario

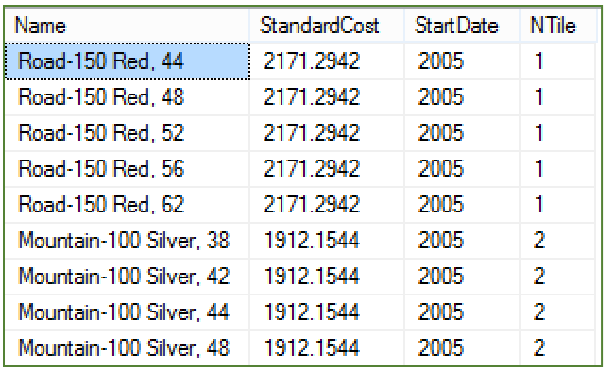
In this exercise, we will understand the concept of **NTILE ()** function.

NTILE: A **NTILE ()** function distributes rows into a specified number of groups within the partition of a result set. For example if the total number of rows is 53 and the number of groups is five, the first three groups will have 11 rows and the two remaining groups will have 10 rows each.

Problem Statement: "Calculate the maximum standard cost of each product and divide the rows into 15 groups based on standard cost for each year"

Table(s) involved: Production.ProductCostHistory table, **Production.Product** table

Final Output:



Name	StandardCost	StartDate	NTile
Road-150 Red, 44	2171.2942	2005	1
Road-150 Red, 48	2171.2942	2005	1
Road-150 Red, 52	2171.2942	2005	1
Road-150 Red, 56	2171.2942	2005	1
Road-150 Red, 62	2171.2942	2005	1
Mountain-100 Silver, 38	1912.1544	2005	2
Mountain-100 Silver, 42	1912.1544	2005	2
Mountain-100 Silver, 44	1912.1544	2005	2
Mountain-100 Silver, 48	1912.1544	2005	2

NOTE: The above figure is a trimmed snapshot of the original result set. The actual result set contains 395 rows.

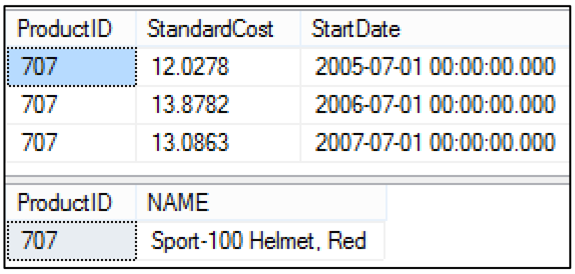
Tasks	Detailed Steps
Open **4_NTile.sql**	1. Click **File** | **Open** | **File** or press (**Ctrl + O**) 2. In **Open File** dialogue box, navigate to **SQL Server Ranking Functions\Scripts** folder 3. Select **4_NTile.sql** and click **Open**

Execute the following statement(S) to understand the table structure of **Production.ProductCostHistory** table and **Production.Product** table

```
-- Step 1: Execute the following statement(s)
SET NOCOUNT ON;
GO
USE AdventureWorks2012
GO

SELECT ProductID
       ,StandardCost
       ,StartDate
FROM Production.ProductCostHistory PCH
WHERE ProductID = 707
GO

SELECT ProductID
       ,NAME
FROM Production.Product
WHERE ProductID = 707
GO
```



ProductID	StandardCost	StartDate
707	12.0278	2005-07-01 00:00:00.000
707	13.8782	2006-07-01 00:00:00.000
707	13.0863	2007-07-01 00:00:00.000

ProductID	NAME
707	Sport-100 Helmet, Red

Explanation: Production.ProductCostHistory table contains the cost of each product. **ProductID** column represents the id of the product, **StandardCost** column represents the cost of the product and **StartDate** column represents the product cost beginning date. **Production.Product** table contains detailed information of each product. **ProductID** column represents the id of the product, **Name** column represents the name of the product**.**

Observation: From the above figure we can observe that from **Production.ProductCostHistory** table for the ProductID '707' we get the list of StandardCost and the StartDate. From **Production.Product** table, we can observe that for the **ProductID** '707' the product Name is 'Sport-100 Helmet, Red'. From this observation, in order to solve the problem statement, we need to correlate **Production. ProductCostHistory** table **ProductID** column with **Production.Product** table **ProductID** column and we have to extract the maximum **StandardCost** of each product. Finally, using **NTILE ()** function, we can divide the rows into groups.

Execute a **SELECT** statement(s)	Execute the following statement(s) to calculate the maximum standard cost of each product and divide the rows into 15 groups based on standard cost for each year

```
-- Step 2: Execute the following statement(s)
WITH PCH_CTE
AS (
       SELECT P.NAME
              ,Max(PCH.StandardCost) StandardCost
              ,year(StartDate) StartDate
       FROM production.ProductCostHistory PCH
       INNER JOIN production.Product P ON P.ProductID =
PCH.ProductID
       GROUP BY P.NAME
              ,StartDate
       )
SELECT NAME
       ,StandardCost
       ,StartDate
       ,NTile(15) OVER (
              PARTITION BY StartDate ORDER BY StandardCost DESC
              ) 'NTile'
FROM PCH_CTE
GO
```

Explanation: The above query is fetching product Name from **Production.Product** table, **StandardCost** and year of **StartDate** from **Production.ProductCostHistory** table. Using **NTile ()** function, we are dividing rows into groups based on the year of StartDate.

Additional problem on **NTILE()**

Problem: “ Calculate median for the total due amount for each customer”

Table(s) involved: Sales.SalesOrderHeader table

```
-- Step 3: Additional problem
WITH SOH_CTE
AS  (
       SELECT CustomerID, TotalDue, NTILE(2) OVER( PARTITION
BY CustomerID ORDER BY TotalDue ) AS tile
        FROM Sales.SalesOrderHeader
      )
       SELECT CustomerID, CASE WHEN cnt%2=1 THEN t1max ELSE
(t1max+t2min)/2.0 END AS MedianValue
FROM(
SELECT CustomerID, COUNT(*) AS cnt,
        MAX(CASE WHEN tile=1 THEN TotalDue END) AS t1max,
        MIN(CASE WHEN tile=2 THEN TotalDue END) AS t2min FROM
SOH_CTE GROUP BY CustomerID
      )T

GO
```

Explanation: The above query is partitioning the customerId order by totaldue and naming it as **tile** using the ntile function, this is being used in the CTE to calculate the t1max and t2min to get the median value

Close all the query windows

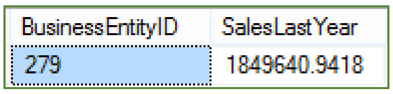
Close all the query windows (☒) and if **SSMS** asks to save changes, click **NO**

Self-Assessment

Problem Statement: "Find the seventh highest sales last year"

Table(s) involved: **Sales.SalesPerson** table

Final output:



BusinessEntityID	SalesLastYear
279	1849640.9418

Solution: Please try to solve the above problem by yourself using **DENSE_RANK ()**. If unable to solve then refer to the **5_SelfAssessment.sql** script which is kept in **SQL Server Ranking Functions\Scripts** folder.

T-SQL Challenge

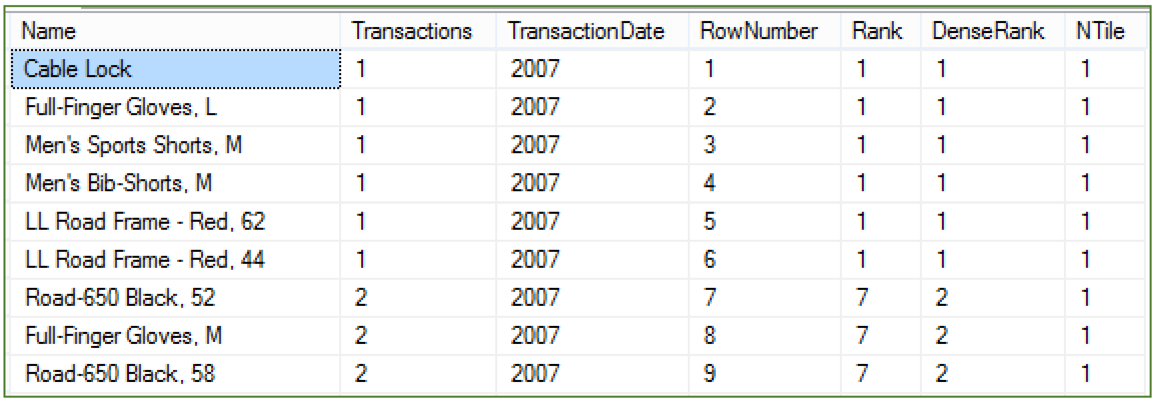
Here is a TSQL challenge for you related to subquery

Problem Statement: "calculate the number of transactions occurred for each product in every year and assign sequential numbers, **RANK ()**, **DENSE_RANK ()** and divided the rows into 10 groups"

Table(s) involved: Production.TransactionHistory table, **Production.Product** table

Final Output:

Note: Given figure is trimmed snapshot of original result. The actual result has 872 rows.



Name	Transactions	TransactionDate	RowNumber	Rank	DenseRank	NTile
Cable Lock	1	2007	1	1	1	1
Full-Finger Gloves, L	1	2007	2	1	1	1
Men's Sports Shorts, M	1	2007	3	1	1	1
Men's Bib-Shorts, M	1	2007	4	1	1	1
LL Road Frame - Red, 62	1	2007	5	1	1	1
LL Road Frame - Red, 44	1	2007	6	1	1	1
Road-650 Black, 52	2	2007	7	7	2	1
Full-Finger Gloves, M	2	2007	8	7	2	1
Road-650 Black, 58	2	2007	9	7	2	1

Solution: Please try to solve the above problem by yourself using Ranking Functions. If unable to solve then refer to the **6_T-SQL Challenge.sql** script which is kept in **SQL Server Ranking Functions\Scripts** folder.

Summary

In this article, we have learned:

- How to use **ROW_NUMBER ()** functions
- How to use **the ROW_NUMBER ()** in the filter the rows
- Use of **ROUND ()** function
- How to use **RANK ()** function

- How to use **RANK()** in the Common Table Expression
- How to use **DENSE_RANK ()** function
- How to use **NTILE ()** function

Questions? Comments? Talk to the author today. SQLMaestros on Twitter.

About SQLMaestros Hands-On-Labs

SQLMaestros Hands-On-Labs are packaged in multiple volumes based on roles (DBA, DEV & BIA). Each lab document consists of multiple exercises and each exercise consists of multiple tasks.

LEARN MORE

Want to write for the magazine? Comments? Feedback? Reach out to us at magazine@sqlservergeeks.com

www.DataPlatformVirtualSummit.com

ACCELERATING DATA-DRIVEN SUCCESS

You need the focus, the strength & the speed to accelerate data-driven success for your organization. Your are the performer.

Conference: Sep 19 to 23. Pre-Cons will be announced soon.
Registration is now open. It’s free. Go for it.

LEARN MORE

DATA PLATFORM
VIRTUAL SUMMIT 2022

Diamond Knowledge Partner Microsoft

A PATH FORWARD FOR SQL SERVER ANALYTICS

SQL Server Team | @SQLServer

NOTE: This article is an extract from ***Microsoft SQL Server Blog***.

Today, we are announcing changes to SQL Server analytics which includes:

- Customer feedback
- Retirement of SQL Server 2019 Big Data Clusters
- Retirement of PolyBase scale-out groups
- Path forward

Customer Feedback:

We continue to see increased migration to the cloud, with analytical workloads leading that charge.

Customers have indicated that analytics in the cloud best aligns to employee skillsets, deployment simplicity and manageability, and cloud flexibility and scalability.

When we first introduced cloud analytics in 2017, many were still investing in on-premises analytical workloads. Today, we offer a wealth of cloud-based services that provide users with similar functionality, including Azure Data Lake Storage (ADLS), Azure Synapse Analytics, Azure SQL, and Azure Machine Learning.

According to the Gartner® 2020 Data and Analytics survey:

- Analytics, BI, and data science are the most common use cases being accelerated to the cloud due to COVID-19. The organization needs faster delivery of analytics insights to take action. Cloud, with its fast provision and prototyping ability, is an ideal place to start analytics and data science initiatives to nimbly react to the fast pace of changes.
- In the 2020 Gartner Data and Analytics Cloud survey, 74 percent of organizations use or plan to use cloud for analytics, BI and data science.

Retirement of SQL Server Big Data Clusters:

Today, we are announcing the retirement of SQL Server 2019 Big Data Clusters. All existing users of SQL Server 2019 with Software Assurance will be **fully supported on the platform for the next three years, through February 2025**. This software will continue to be maintained through SQL Server cumulative updates until that time. In the latest version of SQL Server, we are engineering the best mix of on-premises and in-cloud relational workloads and connectivity to Azure Synapse Analytics for advanced analytics in a flexible, scalable, and integrated environment. Please see below and read our documentation on SQL Server Big Data Clusters to learn more.

Changes to PolyBase support in SQL Server

Today, we are announcing the retirement of PolyBase scale-out groups in Microsoft SQL Server. Scale-out group functionality will be removed from the product in SQL Server 2022. In-market SQL Server 2019, 2017, and 2016 will continue to support the functionality to the end of support for those products.

PolyBase data virtualization will continue to be fully supported as a scale-up feature in SQL Server.

Secondly, Cloudera (CDP) and Hortonworks (HDP) external data sources will also be retired for all in-market versions of SQL Server and will not be included in SQL Server 2022. Moving forward, support for external data sources will be limited to product versions in mainstream support by the respective vendor. You are encouraged to use the new object storage integration functionality available in SQL Server 2022. Integration with Hadoop Distributed File System (HDFS) will also be added to SQL Server 2022 using a new WebHDFS connector.

Finally, connectivity to HDFS and object storage will now use publicly documented REST APIs instead of a JAVA Hadoop client. In SQL Server 2022, users will need to configure their external data sources to use new connectors when connecting to Azure Storage. The table below summarizes the change:

External Data Source	**From**
Azure Blob Storage	wasb[s]
ADLS Gen 2	abfs[s]

The path forward

If you wish to run analytics on-premises, SQL Server 2022 also provides important new capabilities, building upon its data virtualization suite of connectors by providing object storage integration over REST APIs. We will also continue to invest in the Spark SQL connector to ensure first-class connectivity from Apache Spark to all our SQL products. Additionally, we continue to invest in expanding hybrid capabilities with Azure Arc-enabled data services.

Integrating SQL Server with cloud analytics solutions is a critical capability, which is why we are introducing Azure Synapse Link for SQL Server 2022, the latest release of SQL Server, which will be generally available to purchase later this year. This is a major investment in helping you realize cloud-scale analytics in near real-time on your operational data.

Our priority is to empower you with the tools and services that ensure SQL Server integrates seamlessly into the world of analytic workloads in the cloud by blending operational, analytical, and virtual use cases in our flagship database engine. Please contact your **Microsoft account manager** if you need assistance in exploring how Microsoft can best empower your analytical needs.

NOTE: This article is an extract from ***Microsoft SQL Server Blog***.

SQL SERVER TIPS AND TRICKS

01 Wish to enable word wrap feature in SSMS? Menu > Tools > Options > Text Editor > All Languages. Check the 'Word Wrap' box

02 Use the ISNUMERIC function to determine whether an expression is a valid numeric type or not.

03 Use sp_helpdb to retrieve the high-level overview of a specified database or all the databases.

04 Use NOT EXISTS instead of NOT IN at a WHERE condition that uses a subquery to check for data existence.

05 SQL Server cannot perform a seek operation on the index if the LIKE clause predicate begins with wild characters (%, [ ], _).

06 Update statistics with FULLSCAN to compute statistics by scanning all the rows in the table. Use wisely.

LEARN MORE

ROLLBACK THE TRUNCATE USING BACKUPS

Satya Ramesh | @satyaramesh230

In this blog we will prove the point that both the truncated and deleted data can be rolled back, if they are used in certain transactions with a proper database backup and restore mechanism.

Without any further delay let's get in to the demos. Let's create **TruncatevsDelete** database and then two tables **DeleteTest** & **TruncateTest**

```
--Step 1: Execute the following statement(s) to create a database
USE [master];
GO
IF EXISTS ( SELECT  name
      FROM    sys.databases
      WHERE   name = 'TruncatevsDelete' )
ALTER DATABASE TruncatevsDelete SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
DROP DATABASE TruncatevsDelete;
GO
CREATE DATABASE TruncatevsDelete;
GO

--Step 2: Execute the following statement(s) to create tables
USE TruncatevsDelete;
GO
--Query 1:
CREATE TABLE DeleteTest
       (
       id INT IDENTITY(1, 1) ,
       Test CHAR(20) DEFAULT 'Operation DELETE'
       );
GO
--Query 2:
CREATE TABLE TruncateTest
       (
       ID INT IDENTITY(1, 1) ,
       Test CHAR(20) DEFAULT 'Operation TRUNCATE'
       );
GO
```

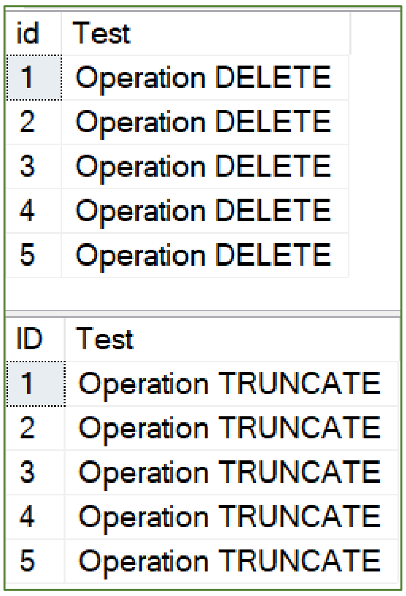
Perform a full database backup before inserting the data into the tables.

```
--Step 3: Execute the following statement(s) to take full backup of
TruncatevsDelete database.
USE [master];
GO
BACKUP DATABASE TruncatevsDelete
TO DISK = 'C:\SQLMaestros\TruncatevsDelete.BAK'; -- Backup path may vary in
your case
GO
```

Now, insert five records each into the tables.

```
--Step 4: Execute the following statement(s) to insert records into the
tables
USE TruncatevsDelete;
GO
INSERT  INTO dbo.DeleteTest
        DEFAULT VALUES;
GO 5
INSERT  INTO dbo.TruncateTest
        DEFAULT VALUES;
GO 5

--Step 5: Execute the following statement(s) to view the records in the
table
SELECT  *
FROM    dbo.DeleteTest;
SELECT  *
FROM    dbo.TruncateTest;
```



id	Test
1	Operation DELETE
2	Operation DELETE
3	Operation DELETE
4	Operation DELETE
5	Operation DELETE

ID	Test
1	Operation TRUNCATE
2	Operation TRUNCATE
3	Operation TRUNCATE
4	Operation TRUNCATE
5	Operation TRUNCATE

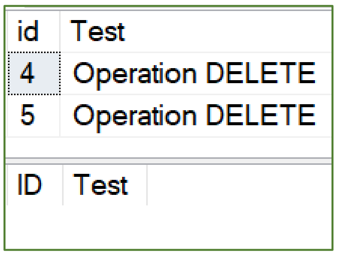
So, we have five records in **DeleteTest** table and another five in **TruncateTest** table. Let's assume that our developer, by mistake deleted some data from the table **DeleteTest** and truncate the table **TruncateTest**. So, execute the below statement(s).

```
--Step 6: Execute the following statement(s) perform delete and truncate
operations
USE TruncatevsDelete;
GO
--Query 1: Delete Operation
```

```
DELETE  FROM dbo.DeleteTest
WHERE   id <= 3;
GO
--Query 2: Truncate Operation
TRUNCATE TABLE dbo.TruncateTest;
GO
```

So, let's go ahead and view the tables status again. We've only two records in **DeleteTest** table and they have no records in **TruncateTest** table.

```
--Step 7: Execute the following statement(s) to view the records in the
table
SELECT  *
FROM    dbo.DeleteTest;
SELECT  *
FROM    dbo.TruncateTest;
```



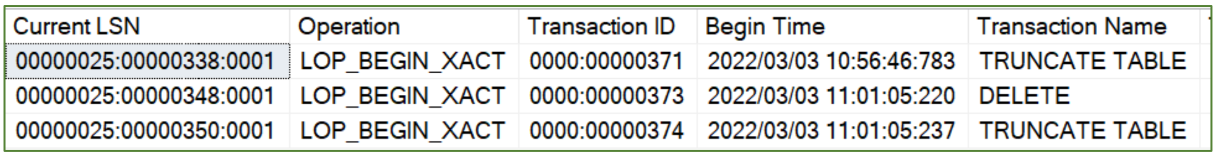
id	Test
4	Operation DELETE
5	Operation DELETE

ID	Test

So, immediately after the delete and truncate happens, we need to check when this particular transaction has been started and we need to record the log sequence numbers of these transactions. So, let's execute this particular query.

```
--Step 8: Execute the following statement(s) to read the data from
transaction log
SELECT  [Current LSN] ,
        Operation ,
        [Transaction ID] ,
        [Begin Time] ,
        [Transaction Name] ,
        [Transaction SID] ,
        AllocUnitName
FROM    fn_dblog(NULL, NULL)
WHERE   [Transaction Name] IN ( 'DELETE', 'TRUNCATE TABLE' );
```

The undocumented function **fn_dblog** accepts 2 input parameters. First parameter is starting LSN and second parameter is ending LSN. So, if we pass NULL to both the parameters, we get all the data from the log file. So, the query also filters out only the transaction whose name is DELETE or TRUNCATE. So, the query results are below.



Current LSN	Operation	Transaction ID	Begin Time	Transaction Name
00000025:00000338:0001	LOP_BEGIN_XACT	0000:00000371	2022/03/03 10:56:46:783	TRUNCATE TABLE
00000025:00000348:0001	LOP_BEGIN_XACT	0000:00000373	2022/03/03 11:01:05:220	DELETE
00000025:00000350:0001	LOP_BEGIN_XACT	0000:00000374	2022/03/03 11:01:05:237	TRUNCATE TABLE

Let's note down the CurresntLSN numbers of the last two transactions. So, first number (00000025:00000348:0001) is where our delete transaction has started and second number (00000025:00000350:0001) is where our truncate transaction has started. Then perform the log backup because SQL Server logs whenever inserts, deletes and truncate etc. happens. So, let's go ahead and perform the log backup.

```
--Step 9: Execute the following statement(s)  to log backup of
TruncatevsDelete database
USE master;
GO
BACKUP LOG TruncatevsDelete TO DISK =
'C:\SQLMaestros\TruncatevsDelete.TRN';
GO
```

Once the log back up is successful, perform the database restore using the backup file to create a new database.

```
--Step 10: Execute the following statement(s) to restore the full backup
with norecovery
RESTORE DATABASE TruncatevsDelete_DeleteCopy
    FROM DISK = 'C:\SQLMaestros\TruncatevsDelete.BAK'
WITH
    MOVE 'TruncatevsDelete' TO 'C:\SQLMaestros\TruncatevsDelete.mdf',
    MOVE 'TruncatevsDelete_Log' TO
'C:\SQLMaestros\TruncatevsDelete_Log.ldf',
    REPLACE, NORECOVERY;
    GO
```

Let's restore the log file also to complete the backup process. While appending the log data, observe the syntax, we are restoring the data from log file and stopping the process at a particular LSN number. This LSN number will vary in your case. Why are we doing this? Whenever a transaction starts, SQL Server record the log SQL number in the log file. So now we're telling SQL Server that restore the data from the log file and whenever you encounter this particular LSN number stop there. Which means this particular LSN number is where our delete transaction has started. So, before this transaction has been started our data is there in the database. Only after this transaction started our data got deleted. So, now we are restoring the file to a certain state that SQL Server delete operation has never started.

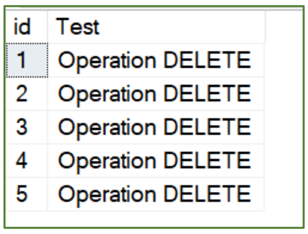
Note that, the LSN number is a hexadecimal number. We need to convert the number to decimal number and then pass the decimal value to the STOPBEFOREMARK attribute.

Note: Convert the hexadecimal LSN we got for **DELETE** operation (which we got at **step 8**) to decimal number. In this case the LSN is **00000025:00000348:0001**. Let us divide the LSN into three parts. **Part 1- 00000025**, **Part 2- 00000348** and **Part 3- 0001**. We will individually convert each part into decimal number. The decimal number of **Part 1** is **37**, **Part 2** is **840** and **Part 3** is **1** (use any online converter). Let us add all three parts together, add **Part 1** as it is (37), round of **Part 2** to 10-digit number by adding leading zeros (0000000840), round of **Part 3** to 5-digit number by adding leading zeros (00001). The final decimal number is **37000000084000001.** Replace the LSN in the following query with LSN we calculated now.

```
--Step 11: Execute the following statement(s) to restore Log backup with
STOPBEFOREMARK option to recover exact LSN
RESTORE LOG TruncatevsDelete_DeleteCopy
FROM
    DISK = 'C:\SQLMaestros\TruncatevsDelete.TRN'
WITH
    STOPBEFOREMARK = 'lsn:37000000084000001'; -- LSN value will vary in you
case
```

Let’s execute the below query. BOOM. Our deleted data is back.

```
--Step 12: Execute the following statement(s) to verify whether records are
back into table or not
USE TruncatevsDelete_DeleteCopy;
GO
SELECT  *
FROM    dbo.DeleteTest;
```



id	Test
1	Operation DELETE
2	Operation DELETE
3	Operation DELETE
4	Operation DELETE
5	Operation DELETE

Let’s DROP the database we created to rollback the deleted data

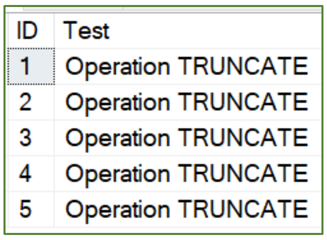
```
--Step 13: Execute the following statement(s) to drop the database
USE master
GO
DROP DATABASE TruncatevsDelete_DeleteCopy
GO
```

Now let’s repeat the same process to rollback the truncated data.

```
--Step 14: Execute the following statement(s) to restore the full backup
with norecovery
RESTORE DATABASE TruncatevsDelete_TruncateCopy
    FROM DISK = 'C:\SQLMaestros\TruncatevsDelete.BAK'
WITH
    MOVE 'TruncatevsDelete' TO 'C:\SQLMaestros\TruncatevsDelete.mdf',
    MOVE 'TruncatevsDelete_Log' TO
'C:\SQLMaestros\TruncatevsDelete_Log.ldf',
    REPLACE, NORECOVERY;
    GO

--     ALTER DATABASE TruncatevsDelete_TruncateCopy SET RECOVERY
--Step 15: Execute the following statement(s) to restore Log backup with
STOPBEFOREMARK option to recover exact LSN
RESTORE LOG TruncatevsDelete_TruncateCopy
FROM
    DISK = 'C:\SQLMaestros\TruncatevsDelete.TRN'
WITH
    STOPBEFOREMARK = 'lsn:37000000084800001';

--Step 16: Execute the following statement(s) to verify whether records are
back into table or not
USE TruncatevsDelete_TruncateCopy;
GO
SELECT  *
FROM    dbo.TruncateTest;
```



ID	Test
1	Operation TRUNCATE
2	Operation TRUNCATE
3	Operation TRUNCATE
4	Operation TRUNCATE
5	Operation TRUNCATE

WOW. Our truncated data is back.

```
	-----------------------
	--Begin: Cleanup
	-----------------------
	USE master
	GO
	DROP DATABASE TruncatevsDelete_TruncateCopy
	GO
	DROP DATABASE TruncatevsDelete
	---------------------
	--End: Cleanup
	---------------------
```

Summary: It is clear that truncate and delete both can be rolled back. And also proves that the TRUNCATE operation does write some amount of log using which we can roll back the data.

Hope you enjoyed the read. Thank you.

Questions? Comments? Talk to the author today. Satya Ramesh on Twitter.

About Satya Ramesh

Satya Ramesh has credible experience working with SQL Server. As a Senior Consultant for SQLMaestros, he has worked on countless projects involving SQL Server & Azure SQL Database.

LEARN MORE

Non-Tech World of Satya Ramesh

Satya likes to play cricket with his friends and loves to cook new dishes in his free time.

Want to write for the magazine? Comments? Feedback? Reach out to us at magazine@sqlservergeeks.com

Peopleware India (PWI) brings you affordable learning solutions from the world's best trainers. Video Courses include subjects like Power BI, SQL Server, Azure SQL, Azure Synapse, Azure Cosmos DB, Kubernetes, Dockers, Data Science, Artificial Intelligence, Big Data Clusters, Migration, Azure Data Factory, and more.

PEOPLEWARE INDIA

Learn More

www.peoplewareindia.com