https://www.sqldbachamps.com Praveen Madupu
https://github.com/PMSQLDBA/PraveenMadupu Mb: +91 98661 30093
Telegram Group→SQLDBACloud-DevOps https://t.me/+r2OYsT2qRZJkYWVl Database Administrator

In **SQL Server**, a **Credential** is a *database object that stores authentication information (usually a Windows username and password or a token)* so that SQL Server can access external resources **securely** on behalf of a user or process.

✅ **Key Purpose**

Credentials are used when SQL Server needs to access resources **outside of SQL Server** — for example:

- Windows file system (folders, backup locations)
- Azure storage
- Linked servers
- SSIS packages
- External scripts or jobs

They store a **login + secret** (like a password) so SQL Server can authenticate to that external system.

✅ **Real-Time Scenario 1: SQL Server Backup to a Network Share**

Suppose your SQL Server needs to back up database files to a network folder:
\\NetworkServer\BackupFolder
This folder requires a Windows login, such as DOMAIN\BackupUser.

🔷 Steps:

```
-- Create credential with Windows account
        CREATE CREDENTIAL NetworkBackupCredential
        WITH IDENTITY = 'DOMAIN\BackupUser',
        SECRET = 'Password123';

-- Map this credential to a SQL Server login
        CREATE LOGIN BackupLogin WITH PASSWORD = 'SqlLogin@123';
        ALTER LOGIN BackupLogin WITH CREDENTIAL = NetworkBackupCredential;
```

Now, when this SQL Login runs a backup, SQL Server uses the credential to access the shared folder.

✅ **Real-Time Scenario 2: Accessing Azure Blob Storage**

If SQL Server wants to back up directly to Azure storage, you store the **Storage Access Key** as a credential.

```
        CREATE CREDENTIAL [https://myazureblob.blob.core.windows.net/backups]
        WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
        SECRET = 'AzureBlobKeyHere==';
```

SQL Server will now authenticate to Azure using this credential.

✅ **Real-Time Scenario 3: SQL Server Agent Job**

A job step (PowerShell, SSIS, CmdExec, etc.) may need access to a file system or network resource. The Agent job can run using a **Proxy**, which internally uses a **Credential**.
This allows you to:

- Run SQL Agent job steps with **least privilege**
- Avoid giving unnecessary permissions to the SQL Server service account

✅ **Key Points to Remember**

Feature	Description
Object Type	Security object
Stores	External login credentials (not SQL logins)
Used For	External resource access
Linked To	SQL Logins or Proxies

📝 **Conclusion**

A **Credential in SQL Server** is like a *secure password locker*. SQL Server uses it when it needs to authenticate to **external systems**, without exposing the password directly in code or jobs.

Real-time demo step-by-step. The scenario we will implement:

✅ **REAL-TIME USE CASE**

A SQL Server Agent job will **back up a database to a network shared folder**: \\FileServer\SQLBackups
This shared folder is protected and only accessible using a Windows domain account, for example:

DOMAIN\BackupUser
Password: Backup@123

SQL Server itself **does not have permission** to access this network path. So we will:

1. **Create a Credential** (stores Windows username/password)
2. **Create a SQL Login** and attach it to the Credential
3. **Create a SQL Server Agent Proxy** (using the Credential)
4. **Create a Job Step that uses the Proxy to perform backup**

✅ **STEP 1 — Create the Credential**

```
CREATE CREDENTIAL NetworkBackupCredential
WITH IDENTITY = 'DOMAIN\BackupUser',
SECRET = 'Backup@123';
```

This stores the Windows login inside SQL Server securely.

✅ **STEP 2 — Create Login and Attach Credential**

```
CREATE LOGIN BackupLogin WITH PASSWORD = 'SqlLogin@123';
ALTER LOGIN BackupLogin WITH CREDENTIAL = NetworkBackupCredential;
```

Now, any actions done by **BackupLogin** outside SQL Server will be authenticated using the Credential.

✅ **STEP 3 — Create a Proxy for SQL Server Agent**

```
USE msdb;
EXEC sp_add_proxy
  @proxy_name = 'NetworkBackupProxy',
  @credential_name = 'NetworkBackupCredential';
```

Give the proxy permission to run backup-related job steps (PowerShell / CmdExec / SSIS — depending on your job).
Example (CmdExec):

```
EXEC sp_grant_proxy_to_subsystem
  @proxy_name = 'NetworkBackupProxy',
  @subsystem_name = 'CmdExec';
```

✅ **STEP 4 — Create SQL Agent Job**

```
USE msdb;
EXEC sp_add_job @job_name = 'Network Backup Job';

EXEC sp_add_jobstep
  @job_name = 'Network Backup Job',
  @step_name = 'Backup Step',
  @subsystem = 'TSQL',
  @command = 'BACKUP DATABASE TestDB TO DISK = ''\\FileServer\SQLBackups\TestDB.bak'' WITH INIT',
  @proxy_name = 'NetworkBackupProxy';

EXEC sp_add_jobserver @job_name = 'Network Backup Job';
```

Now the job will use the proxy → which uses the credential → which uses the Windows authentication → to access the network folder.

✅ **RESULT**

- No password exposed in backup scripts
- Least-privilege access
- Secure external resource authentication
- Fully automated network backups

📌 **IMPORTANT NOTES**

Object	Why We Used It
Credential	Stores Windows login + password
SQL Login	Used by SQL Agent to map credential
Proxy	Allows job to run with credential privileges
Job Step	Uses proxy to run backup securely

Scenario 1 → Scenario 2 → Scenario 3 in order, each with:
Short description, architecture/flow (simple diagram), step-by-step commands (T-SQL / PowerShell where appropriate), SQL Agent job setup, and security notes.

Scenario 1 — Backup to a Network Share (Domain account)
Goal: SQL Agent runs a job that backs up TestDB to \\FileServer\SQLBackups using a domain account (DOMAIN\BackupUser) without embedding the password in scripts.
Architecture / Flow (simple)

```
[SQL Agent Job Step (PowerShell/CmdExec)]
        |
      uses
        v
  [SQL Agent Proxy]  <--->  [Credential w/ DOMAIN\BackupUser]
        |
        v
  Executes backup command (sqlcmd / PowerShell)
  which writes to \\FileServer\SQLBackups
```

Why use a Credential + Proxy here

- A **Credential** stores the domain account and secret securely inside SQL Server.
- A **Proxy** lets a non-TSQL job step (PowerShell or CmdExec) run under those credentials.
- T-SQL job steps **cannot** use a proxy to impersonate a Windows account for external file access — so use PowerShell/CmdExec.

Step-by-step (T-SQL + PowerShell approach)
1) Create the Credential

```
-- Run as sysadmin in master
CREATE CREDENTIAL [NetworkBackupCredential]
WITH IDENTITY = 'DOMAIN\BackupUser',
    SECRET = 'Backup@123';
GO
```

2) Create a SQL login (optional but common) — *not strictly required for proxy*

```
CREATE LOGIN [BackupLogin] WITH PASSWORD = 'SqlLogin@123';
GO
```

Note: The proxy links directly to the Credential (not necessarily to the SQL login). The SQL login is useful if you want interactive logins or map permissions.
3) Create the Proxy in msdb and grant it to a subsystem

```
USE msdb;
GO

EXEC msdb.dbo.sp_add_proxy
   @proxy_name = N'NetworkBackupProxy',
   @credential_name = N'NetworkBackupCredential',
   @enabled = 1;
GO

-- Grant proxy rights to the CmdExec and PowerShell subsystems (pick one you'll use)
EXEC msdb.dbo.sp_grant_proxy_to_subsystem
   @proxy_name = N'NetworkBackupProxy',
   @subsystem_name = N'PowerShell';
GO
```

```
EXEC msdb.dbo.sp_grant_proxy_to_subsystem
   @proxy_name = N'NetworkBackupProxy',
   @subsystem_name = N'CmdExec';
GO

-- Grant a specific database principal or role permission to use the proxy (optional)
EXEC msdb.dbo.sp_grant_login_to_proxy
   @proxy_name = N'NetworkBackupProxy',
   @principal_name = N'DBU';  -- or a specific login/user
GO
```

4) Create the SQL Agent Job that uses the Proxy (PowerShell step recommended)

PowerShell Step (runs under proxy account and calls BACKUP via Invoke-Sqlcmd):

```
USE msdb;
GO

EXEC msdb.dbo.sp_add_job @job_name = N'Network Backup Job', @enabled = 1;
GO

EXEC msdb.dbo.sp_add_jobstep
  @job_name = N'Network Backup Job',
  @step_name = N'PowerShell Backup Step',
  @subsystem = N'PowerShell',
  @command = N'Invoke-Sqlcmd -Query "BACKUP DATABASE [TestDB] TO DISK = ''\\\\FileServer\\SQLBackups\\TestDB.bak'' WITH
INIT, FORMAT" -ServerInstance "localhost"',
  @proxy_name = N'NetworkBackupProxy',
  @on_success_action = 1;
GO

EXEC msdb.dbo.sp_add_jobserver @job_name = N'Network Backup Job';
GO
```

Alternative (CmdExec) — using sqlcmd:

```
EXEC msdb.dbo.sp_add_jobstep
  @job_name = N'Network Backup Job',
  @step_name = N'CmdExec Backup Step',
  @subsystem = N'CmdExec',
  @command = N'sqlcmd -S localhost -Q "BACKUP DATABASE [TestDB] TO DISK = ''\\\FileServer\SQLBackups\TestDB.bak'' WITH INIT,
FORMAT"',
  @proxy_name = N'NetworkBackupProxy';
GO
```

5) Test the job and verify the file appears on \\FileServer\SQLBackups

Security & Operational Notes

- **Least privilege:** DOMAIN\BackupUser should have **only** write permission to the backup folder, not elevated server rights.
- **Rotate secrets** periodically, and update the Credential with ALTER CREDENTIAL … WITH SECRET = 'newPass'.
- **Avoid** embedding passwords in scripts. The Credential keeps the secret safe in the instance.
- **Network/service considerations:** The file server must allow NETBIOS/SMB access and not block the SQL Server host; if using clustered SQL Server, ensure the account is accessible from all nodes.
- **Audit:** Monitor sys.credentials and msdb proxy tables for changes.

Scenario 2 — Backup to Azure Blob Storage (SQL Server -> Azure)

Goal: Back up database to Azure Blob Storage (e.g., container backups in mystorageaccount) using a secure credential. Two common auth options: **Shared Key** (legacy) or **SAS token / Managed Identity** (recommended). Here I'll show **SAS / Shared Key** credential approach and a short note about Managed Identity.

Architecture / Flow

[SQL Server BACKUP DATABASE ... TO URL = 'https://<acct>.blob.core.windows.net/<container>/...']
|
uses
v
[SQL Server Credential] (stores SAS token or storage key)
|
v
Authenticates to Azure Blob Storage

Prereqs

- SQL Server 2012+ supports BACKUP TO URL (best in recent versions).
- Storage account + container created in Azure.
- SAS token or storage account key with write permission to the container.

Step-by-step (Using Storage Account Key or SAS token)

A) Using Storage Account Key (simple credential)

```
-- Create credential named exactly as the URL or any name you choose; recommended to use the storage URL as name
CREATE CREDENTIAL [https://mystorageaccount.blob.core.windows.net/backups]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',  -- or 'storageaccountname' for some approaches
   SECRET = 'your_storage_account_key_or_sas_token_here';
GO
```

B) Backup command (T-SQL)

```
BACKUP DATABASE [TestDB]
TO URL = 'https://mystorageaccount.blob.core.windows.net/backups/TestDB.bak'
WITH CREDENTIAL = 'https://mystorageaccount.blob.core.windows.net/backups',
   COMPRESSION, STATS = 10;
GO
```

Notes:

- The CREDENTIAL parameter points to the credential name you created.
- If using an **SAS token**, the IDENTITY may be SHARED ACCESS SIGNATURE and SECRET is the SAS token (starting with ?sv=...). When using SAS, you often include the token in the credential secret, not in the URL.

C) Automate with SQL Agent Job

- Create a T-SQL job step that runs the BACKUP ... TO URL statement. T-SQL steps can use the credential directly for Azure URLs — no proxy needed.

Recommended — Use Managed Identity (if SQL Server is in Azure)

If SQL Server runs in Azure VM or Azure SQL Managed Instance, prefer **Azure AD Managed Identity** to avoid storing keys. Managed Identity + RBAC grant to the storage account/container is more secure (no secret to rotate). Implementation differs by deployment type (Azure SQL MI, SQL Server on VM), so follow Azure docs for Azure VM system-assigned managed identity + MSI and grant Storage Blob Data Contributor to the identity.

Security & Operational Notes

- **Prefer SAS or RBAC** over long-lived account keys. Use least-scope SAS tokens (narrow expiry & permissions).
- **Encrypt** backups if required by policy.
- **Monitor** access via Azure Storage logs and rotate keys/SAS tokens.
- When using a credential name that is the same as the URL, SQL Server maps correctly — but any unique name is OK if used in the WITH CREDENTIAL = 'name'.

Scenario 3 — Linked Server Authentication to an External Server

Goal: Configure SQL Server to connect to a remote server (or data source) using credentials stored in SQL Server rather than embedding passwords in connection strings.

Typical use cases

- Stored procedures query a remote SQL Server instance.
- Jobs or linked server queries need a specific Windows or SQL login to authenticate.

Architecture / Flow

[Local SQL Server] --linked server--> [Remote Server]
|
uses
v
Credential (if using Windows account for remote access) OR
Linked Server security mapping (SQL Auth)

Two common patterns:

Pattern A — Linked Server using SQL Authentication

If remote server accepts SQL auth, you can store the SQL username/password in the linked server configuration (no need for a SQL credential object).

Example (T-SQL):

```
EXEC sp_addlinkedserver
  @server = N'RemoteSQL',
  @srvproduct = N'SQLServer';

-- Add security mapping: local login to remote login
EXEC sp_addlinkedsrvlogin
  @rmtsrvname = N'RemoteSQL',
  @useself = N'false',
  @locallogin = NULL,
  @rmtuser = N'remote_sql_user',
  @rmtpassword = N'remote_password';
GO
```

Security note: The remote password will be stored inside server metadata; treat linked server configs sensitively (limit who can read server metadata).

Pattern B — Linked Server using Windows Authentication via Credential & Proxy (for external non-SQL servers or when using CmdExec)

If you need Windows credentials for external resources, you can store a Credential and use it for Agent job steps or proxies — but linked server connections from the SQL Engine to another SQL Engine using Windows Auth require Kerberos delegation / constrained delegation and service account configuration (this is complex). A common solution:

- Use **SQL Authentication** for linked server to avoid double hop/Kerberos issues.
- Or configure **Kerberos delegation** (set SPNs, allow delegation for the SQL Server service account, Active Directory config).

Example: Use Credential with EXECUTE AS + OPENROWSET to access external resource (conceptual)

Credentials are not directly used by sp_addlinkedserver. But you can use credentials for job steps that run queries against the linked server, or for CmdExec steps that call sqlcmd -S RemoteSQL -U ... -P ... (less ideal).

If you need T-SQL code running as a Windows account to access remote resources, you can:

1. Create a Credential for the Windows account.
2. Create a stored procedure signed or use EXECUTE AS LOGIN for impersonation (requires careful security setup).
3. For cross-server Windows auth, configure Kerberos so the impersonated account can be delegated to the remote server.

Because linked server/Windows auth is the trickiest (Kerberos), many teams choose SQL authentication on the linked server and tightly control the remote SQL account.

Security & Operational Notes (Linked Servers)

- **Prefer least privilege** for remote accounts.
- **Avoid** storing high-privileged credentials in linked servers.
- **Be cautious**: linked server configuration and credentials are visible to server admins—treat as sensitive.
- **Kerberos:** if using Windows auth across servers, ensure SPNs and delegation are correctly configured (coordinate with AD admins).
- **Test** performance and security — linked queries can impact both servers.

Final checklist & best practices (applies to all scenarios)

- Use **least privilege** for any domain/storage account user.
- **Rotate** secrets regularly; prefer SAS with short expiry or Managed Identities where possible.
- Limit which **principals** can create/alter Credentials and Proxies (server role: securityadmin / sysadmin).
- **Audit** changes to sys.credentials, msdb.dbo.sysproxies, and jobs.
- Keep **network paths** reachable from the SQL Server host(s); for clustered instances, validate configuration across nodes.
- Document where credentials are used and who owns them.

Single, clean, ready-to-run SQL script you can paste and execute, with placeholders for:

✅ **Scenario 1:** Network Share Backup (Credential + Proxy + Job)
✅ **Scenario 2:** Azure Blob Backup (Credential + BACKUP TO URL)
✅ **Scenario 3:** Linked Server (SQL Authentication – most secure & simplest)

You may fill in your values where indicated.

🚀 **FULL READY-TO-RUN SCRIPT (copy/paste)**
🖍️ Replace placeholders like: <DOMAIN>, <USERNAME>, <PASSWORD>, <FILESERVER>, <STORAGEACCOUNT>, <CONTAINER>, etc.

✅ **SCENARIO 1 — Backup to Network Share using Credential + Proxy + SQL Agent Job**

```
/* 1. Variables */
USE master;
GO
DECLARE @DomainUser NVARCHAR(200) = N'<DOMAIN>\<BackupUser>';
DECLARE @DomainPassword NVARCHAR(200) = N'<WindowsPassword123>';
DECLARE @NetworkPath NVARCHAR(MAX) = N'\\<FILESERVER>\SQLBackups';
DECLARE @DBName NVARCHAR(200) = N'TestDB';
DECLARE @JobName NVARCHAR(200) = N'Network Backup Job';
DECLARE @CredentialName NVARCHAR(200) = N'NetworkBackupCredential';
DECLARE @ProxyName NVARCHAR(200) = N'NetworkBackupProxy';
GO

/* 2. Create Credential */
CREATE CREDENTIAL [NetworkBackupCredential]
WITH IDENTITY = @DomainUser,
    SECRET = @DomainPassword;
GO

/* 3. Create Proxy */
USE msdb;
GO
EXEC sp_add_proxy
   @proxy_name = @ProxyName,
   @credential_name = @CredentialName,
   @enabled = 1;
GO

/* 4. Grant Proxy to PowerShell subsystem */
EXEC sp_grant_proxy_to_subsystem
   @proxy_name = @ProxyName,
   @subsystem_name = 'PowerShell';
GO

/* 5. Create Backup Job (PowerShell step) */
EXEC sp_add_job @job_name = @JobName, @enabled = 1;
GO
```

```
EXEC sp_add_jobstep
   @job_name = @JobName,  @step_name = 'Backup To Network',
   @subsystem = 'PowerShell', @proxy_name = @ProxyName,
   @command = N'Invoke-Sqlcmd -Query "BACKUP DATABASE [' + @DBName + '] TO DISK = ''' + @NetworkPath + '\' + @DBName +
'.bak'' WITH INIT, FORMAT"';
GO
EXEC sp_add_jobserver @job_name = @JobName;
GO
```

✅ *Result:* Job will back up TestDB to \\FileServer\SQLBackups\TestDB.bak

✅ **SCENARIO 2 — Backup to Azure Blob Storage Using Credential + BACKUP TO URL**

```
/* Azure Settings */
DECLARE @AzureCredentialName NVARCHAR(200) = N'https://<STORAGEACCOUNT>.blob.core.windows.net/<CONTAINER>';
DECLARE @AzureSasToken NVARCHAR(MAX) = N'<Paste_SAS_Token_Without_QuestionMark>';
DECLARE @AzureBackupURL NVARCHAR(MAX) = N'https://<STORAGEACCOUNT>.blob.core.windows.net/<CONTAINER>/TestDB.bak';
DECLARE @DBName2 NVARCHAR(200) = N'TestDB';

/* 1. Create Azure Credential */
CREATE CREDENTIAL [https://<STORAGEACCOUNT>.blob.core.windows.net/<CONTAINER>]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
SECRET = @AzureSasToken;
GO
/* 2. Run Backup */
BACKUP DATABASE @DBName2
TO URL = @AzureBackupURL
WITH CREDENTIAL = @AzureCredentialName,
    COMPRESSION, STATS = 10;
GO
```

✅ *Result:* Backup uploads to Azure Storage container <CONTAINER>

✅ **SCENARIO 3 — Linked Server to another SQL Server (SQL Authentication)**

```
/* Linked Server Settings */
DECLARE @LinkedServerName NVARCHAR(200) = N'RemoteSQL';
DECLARE @RemoteUser NVARCHAR(200) = N'<RemoteSqlUser>';
DECLARE @RemotePassword NVARCHAR(200) = N'<RemotePassword>';

/* 1. Create Linked Server */
EXEC sp_addlinkedserver
    @server = @LinkedServerName,
    @srvproduct = N'SQL Server';

/* 2. Security Mapping */
EXEC sp_addlinkedsrvlogin
    @rmtsrvname = @LinkedServerName,
    @useself = 'false',
    @locallogin = NULL,
    @rmtuser = @RemoteUser,
    @rmtpassword = @RemotePassword;
GO
```

```
/* 3. Test */
EXEC ('SELECT name FROM sys.databases') AT RemoteSQL;
GO
```

✅ *Result:* Local SQL Server can query remote SQL Server

✅ **What this one script gives you**

Scenario	Authentication Mechanism	Output
Network Share	Credential + Proxy	Backup .bak to shared folder
Azure Blob	Credential (SAS)	Cloud backup
Linked Server	SQL Login stored securely	Remote query access