SQL Admin Scripts: A Practical Toolkit for Managing Servers and Databases

Effective database management relies on having the right tools to streamline tasks and ensure optimal performance. Over time, I have created a set of SQL scripts designed to simplify routine administrative duties, enhance visibility into database environments, and automate routine processes. Some of these queries were developed during a comprehensive SQL Server Security Audit.

This toolkit includes everything from listing account permissions and linked servers to creating flexible backup solutions and querying server version details. These scripts are designed to save time, reduce errors, and provide quick insights into the state of your SQL Server environment. Whether you're troubleshooting, planning maintenance, or auditing access, this collection offers practical solutions for SQL administrators of all levels.

List SQL account login type

This query provides detailed information about all server principals, including their names, types, descriptions (both user-defined and system-provided), whether they are fixed roles, and their ownership relationships. This is particularly useful for auditing, analyzing, or managing server-level security and access control.

```
-- list sql account login type
select
[name],
[principal_id],
[type],
case [type]
when 'C' then 'certificate mapped login'
when 'G' then 'windows group'
when 'K' then 'asymmetric key mapped login'
when 'R' then 'server role'
when 'S' then 'sql account'
when 'U' then 'windows account'
end as [type_desc],
[type_desc],
[is_fixed_role],
[owning_principal_id]
from [master].[sys].[server_principals]
```

List SQL Database Account Access and Permissions

This query retrieves information about permissions and roles for non-system databases, focusing on database principals and their associated permissions. I use this query to gain insights into and manage permissions for user-defined databases. It helps troubleshoot access issues by providing clear visibility into the permissions and roles that could be impacting user access or privilege-related problems.

```
-- list non system databases with status
select distinct [db_prin].[principal_id],
[db_prin].[name] as [username],
[perms].[grantee_principal_id],
[perms].[grantor_principal_id],
[perms].[permission_name],
[perms].[state_desc],
[db_prin].[type_desc],
[perms].[class_desc]
from [sys].[database_permissions] as [perms]
left join [sys].[database_principals] as [db_prin] on
[perms].[grantee_principal_id] = [db_prin].[principal_id]
left join [sys].[database_role_members] as [db_role_mem] on
[db_prin].[principal_id] = [db_role_mem].[member_principal_id]
```

List Server Principal Objects and Granted Permissions for Each Principal (Filtered by Role)

The combination of these two queries helps with auditing the server security model, diagnosing role-based permission issues, and ensuring proper role memberships for users and other principals.

This query retrieves a distinct list of server-level principals, their types, descriptions, and whether they are fixed roles.

```
-- get server level principal object list
select
distinct [type],
[type_desc],
[is_fixed_role]
from [sys].[server_principals] -- contains a row for every server-level principal
```

Understanding Principal Types:

It helps you understand the various types of server-level principals in the SQL Server instance. These can be logins, roles, or certificates, among others.

Role & Permission Management:

By listing server principals, you can identify which principals exist on the server, helping you with roles and permission management. Knowing which are fixed roles (e.g., sysadmin, serveradmin) helps you differentiate between system-defined and user-defined roles.

Security Audit:

It is useful for auditing security, particularly when you want to ensure only appropriate principals (logins, roles) are configured at the server level. This query can be a starting point for reviewing who has access to the server and their role.

This query lists server principals that are roles (denoted by type = 'R').

```
-- get use granted level principal object list
select
[name],
[principal_id],
[type],
[type_desc],
[is_fixed_role],
[owning_principal_id]
from [sys].[server_principals]
where [type] = 'R' -- R is role
```

Focused on Roles:

By filtering for type = 'R', this query specifically shows only the roles that have been defined at the server level. This makes it easier to identify and analyze server roles separately from other types of principals like users or logins.

Role Membership:

This query provides a view of roles, their descriptions, and ownership information. It helps understand which roles exist on the server, who owns them, and if they are fixed or custom. This is important for troubleshooting access and privilege inheritance, as roles can group multiple principals under common permissions.

Security and Compliance:

It allows administrators to quickly check which server roles are in place, what permissions they have, and whether there is any potential misconfiguration regarding role memberships or permissions. For example, you may want to confirm that certain roles aren't mistakenly given more privileges than intended.

Why Both Queries are Useful Together:

Both queries together give you an overview of:

- All server principals (the first query) — so you can understand what types of principals exist and what their roles are (including fixed roles).
- Just the roles (the second query) — so you can dive deeper into role management, ownership, and assignment.

List All User, Groups, and Users with Their Associated Groups

The following three queries assist in managing server-level security by listing users, groups, and their memberships. The first query retrieves all users, the second lists all groups, and the third combines both to provide a comprehensive overview of users and their associated groups.

This query retrieves all server principals that are classified as users or SQL logins (type 'u' or 's'). It excludes internal logins that are created from certificates (identified by names containing %%). The query returns the login name, account type description, and the account type for each user.

```
-- list all users
select
[_usgr].[name] as [login_name],
[_usgr].[type_desc] as [account_type],
[_usgr].[type]
from [sys].[server_principals] as [_usgr]
where [_usgr].[type] in ('u', 's') -- user, sql
and [_usgr].[name] not like '%##%' -- exclude double hash internal logins created
from certificates
order by [_usgr].[name]
```

This query retrieves all server principals of type 'g', which are groups. It similarly excludes internal logins created from certificates and returns the login name, account type description, and the account type for each group.

```
-- list all groups
select
[_usgr].[name] as [login_name],
[_usgr].[type_desc] as [account_type],
[_usgr].[type]
from [sys].[server_principals] as [_usgr]
where [_usgr].[type] in ('g') -- group
and [_usgr].[name] not like '%##%' -- exclude double hash internal logins created
from certificates
order by [_usgr].[name]
```

This query is a combination of the previous two queries. It lists users and their associated groups by joining the "server_principals" table for both users and groups, through the "server_role_members" table, which links users to their respective roles (groups). It retrieves the login name, account type, and group name for each user, showing the groups they belong to. If a user is not a member of any group, the group name will appear as NULL.

```
-- list all users and their group memberships
select
[usr].[name] as [login_name],
[usr].[type_desc] as [account_type],
[usr].[type] as [account_type_code],
[grp].[name] as [group_name]
from  [sys].[server_principals] as [usr]
```

```
left join  [sys].[server_role_members] as [rolembr] on [usr].[principal_id] =
[rolembr].[member_principal_id]
left join  [sys].[server_principals] as [grp] on [rolembr].[role_principal_id] =
[grp].[principal_id]
where [usr].[type] in ('u', 's')  -- user, sql accounts
and [usr].[name] not like '%##%'  -- exclude internal logins
order by [usr].[name], [grp].[name];
```

List Linked SQL Servers and Their Logins

This query retrieves information about linked servers and their associated logins. It lists the linked server names, the principal IDs of the local logins, and the corresponding remote login names, providing a comprehensive view of the linked server configurations and their user mappings.

```
-- linked servers and their logins
select
[servers].[name],
[prin].[principal_id],
[logins].[remote_name]
from [sys].[servers] as [servers]
join [sys].[linked_logins] as [logins] on [servers].[server_id] =
[logins].[server_id]
left join [sys].[server_principals]  as [prin] on [logins].[local_principal_id] =
[prin].[principal_id]
where [servers].[is_linked] = 1
```

List of Non-System SQL Databases with Online Status

This query retrieves a list of all online databases, excluding the system databases ("master", "msdb", "model", and "tempdb"). It provides the database name and its online status, clearly marking any databases that are currently in an 'ONLINE' state.

```
-- list non system databases with status
select
[name] as [db_name],
case [state_desc]
when 'ONLINE' then 'ONLINE'
else 'NOT-ONLINE'
end as [state_desc]
from [master].[sys].[databases]
where [name] not in ('master','msdb','model','tempdb')
and [state_desc] ='online'
```

Backup and Restore Database to New Location and Name

This set of queries demonstrates how to back up a database and restore it to a new location with a new name. The first query performs a backup of the pubs database to a specified path, while the second query restores the backup to a different database name (pubs_new) and moves the data and log files to new disk locations. The restoration process also includes options to overwrite an existing database and display progress during the restoration.

```
-- backup-restore database to new location

-- backup a database
backup database pubs
  to disk = 'c:\Backups\pubs.bak'
  with stats = 10

-- and restore it to a different name
-- and a different disk drive/location
restore database pubs_new -- new database name
  from disk = 'c:\Backups\pubs.bak'
  with
    replace, -- overwrite db - if one exists
    -- norecovery, -- use if there are more files to recover (leave db in recovery
mode)
    recovery, -- use if no more files to recover (set database ready to use)
    stats = 10, -- show progress (every 10%)
  move 'pubs_data' to 'm:\new_location\pubs_new.mdf',
  move 'pubs_log' to 'l:\new_location\pubs_new.ldf'
```

Backup All Online User Databases to Network Location

This script performs backups of all online user databases (excluding system databases) to a specified network location. Each time the script is run, it creates a unique dump device name for each database by appending a timestamp, ensuring that existing dump devices are not overwritten. The script iterates over each database, creates a backup device, and backs up the database to the specified location, continuing until all eligible databases have been backed up.

```
-- database backup script with network dump device

use [master];
set nocount on;

declare @p_dbname sysname;
declare @p_command nvarchar(4000);
declare @p_dt varchar(100);
declare @p_dbdumpdevice nvarchar(4000);
declare @p_dumpdevicename varchar(100);

set @p_dt = '.' + replace(replace(convert(varchar(40), getdate(), 113), ':', '.'),
' ', '.') + '.bak';

-- start loop with first database
```

```
declare @p_rowcount int = 1;

while @p_rowcount > 0
begin
  -- get the next database name
  select top 1 @p_dbname = [name]
  from sys.databases
  where name not in ('master', 'tempdb', 'model', 'msdb')
  and [state_desc] = 'ONLINE'
  and not exists
  (
    select 1
    from sys.backup_devices
    where name = @p_dbname + '_dump_device' -- unique dump device name for each
database
  );

  -- if a database was found
  if @p_dbname is not null
  begin
    -- set the dump device name to unique database name subdirectory, device name,
and datetime
    set @p_dbdumpdevice = '\\some_network_drive\some_directory\' + @p_dbname + '\'
+ @p_dbname + @p_dt;
    set @p_dumpdevicename = @p_dbname + '_dump_device';

    -- add the dump device
    exec sp_addumpdevice 'disk', @p_dumpdevicename, @p_dbdumpdevice;

    -- backup the database to the dump device
    backup database @p_dbname
    to @p_dumpdevicename
    with noformat, init, name = N'Database Backup', skip, norewind, nounload,
compression;

    -- update the rowcount to finish the loop when no more databases are found
    set @p_rowcount =
    (
      select count(*)
      from sys.databases
      where name not in ('master', 'tempdb', 'model', 'msdb')
      and [state_desc] = 'ONLINE'
      and not exists
      (
        select 1
        from sys.backup_devices
        where name = @p_dbname + '_dump_device'
      )
    );
  end
  else
  begin
    -- exit loop when no databases are left to process
    set @p_rowcount = 0;
  end
end
```

List of Accounts with Type and Status (Enabled/Disabled)

This query retrieves a list of account names along with their account type and status (enabled or disabled). It provides detailed information on the type of account (e.g., SQL account, Windows account, certificate mapped login) and whether the account is currently enabled or disabled.

```
-- get account name with type and status (enabled/disabled)
select [member].[name] as [account],
case [member].[type]
when 'C' then 'certificate mapped login'
when 'G' then 'windows group'
when 'K' then 'asymmetric key mapped login'
when 'R' then 'server role'
when 'S' then 'sql account'
when 'U' then 'windows account'
end as [type],
case
when [member].[is_disabled] = 1 then 'true'
else 'false'
end as [account_disabled]
from [master].[sys].[server_role_members] as [rol_mem]
join [master].[sys].[server_principals] as [role] on [rol_mem].[role_principal_id]
= [role].[principal_id]
join [master].[sys].[server_principals] as [member] on
[rol_mem].member_principal_id = [member].principal_id
```

SQL Server Version Information and Details

This query retrieves SQL Server version details, including the server name, product version, edition, product level, and default collation. It also categorizes the SQL Server version into major releases (e.g., SQL 2000, SQL 2005, SQL 2019) based on the product version.

```
-- get sql server version info
select
serverproperty('servername') as [sql_server],
case
  when convert(varchar(128), serverproperty ('productversion')) like '8%' then 'sql2000'
  when convert(varchar(128), serverproperty ('productversion')) like '9%' then 'sql2005'
  when convert(varchar(128), serverproperty ('productversion')) like '10.0%' then
'sql2008'
  when convert(varchar(128), serverproperty ('productversion')) like '10.5%' then
'sql2008 r2'
  when convert(varchar(128), serverproperty ('productversion')) like '11%' then 'sql2012'
  when convert(varchar(128), serverproperty ('productversion')) like '12%' then 'sql2014'
  when convert(varchar(128), serverproperty ('productversion')) like '13%' then 'sql2016'
  when convert(varchar(128), serverproperty ('productversion')) like '14%' then 'sql2017'
  when convert(varchar(128), serverproperty ('productversion')) like '15%' then 'sql2019'
```

```
  when convert(varchar(128), serverproperty ('productversion')) like '16%' then
'sql2020+'
  else 'unknown'
  end as [majorversion],
serverproperty('productlevel') as [productlevel],
serverproperty('edition') as [edition],
serverproperty('productversion') as [productversion],
serverproperty('collation') as [default_collation]
```

Send Email Using SQL Server Database Mail

This query demonstrates how to send an email using the SQL sp_send_dbmail stored procedure. It sets up email parameters such as recipients, subject, body, and sender information, and then sends an email using a preconfigured database mail profile. Before using this, a mail account and profile must be created within SQL Server.

```
-- send sql email - sp_send_dbmail

-- you must first create a sql mail account and a database mail profile

declare @p_to varchar(250)
declare @p_cc varchar(250)
declare @p_emailbody varchar(8000)
declare @p_emailsubject varchar(400)
declare @p_fromfullname varchar(100)

set @p_to = 'recipient1@emailaddress.com, recipient2@emailaddress.com,
recipient3@emailaddress.com'
set @p_fromFullName = 'FName LName'
set @p_cc = 'cc_recipient@emailaddress.com'
set @p_emailsubject = 'the subject of the email'
set @p_emailsubject = @p_emailsubject + ' ' + @p_fromFullName
set @p_emailbody = 'the body text of the email'

exec msdb.dbo.sp_send_dbmail
@profile_name = 'TheMailProfileName', -- the database mail profile name
@recipients = @p_to,
@copy_recipients = @p_cc,
@body = @p_emailbody,
@body_format = 'HTML',
@subject = @p_emailsubject
```