Comprehensive possible methods to transfer SQL Server logins from one SQL Server instance to another, including **use cases, steps, pros, cons, and limitations**.
The coverage applies to SQL Server 2012 through 2025 unless otherwise stated.

1. Using Microsoft-Provided Stored Procedures (Best-Practice Method)

Method: sp_help_revlogin

This is the **most widely accepted and recommended** method because it preserves **SID and password hashes**, preventing orphaned users.

What It Transfers

- SQL logins
- Password hash
- SID
- Default database
- Password policy settings

What It Does NOT Transfer

- Server role memberships (sysadmin, securityadmin, etc.)
- Permissions at server level (must be reapplied separately)

Steps

1. Run Microsoft's sp_help_revlogin script on the **source server**
2. Script generates CREATE LOGIN statements with:
    - Same SID
    - Encrypted password
3. Execute generated script on the **target server**
4. Reassign server roles manually

Pros

- Prevents orphaned users
- Works across versions
- Secure (no plaintext passwords)

Cons

- SQL logins only (not Windows logins)
- Requires manual role reassignment

Best Use Case

- Database migrations
- Side-by-side server upgrades

2. SQL Server Management Studio (SSMS) – Script Login As

Method

Right-click Login → **Script Login As → CREATE To**

What It Transfers

- Login name
- Login type
- Default database

What It Does NOT Transfer

- Password (must be reset)
- SID
- Permissions

- Server roles

Pros

- Simple and GUI-based
- No scripting knowledge required

Cons

- Causes orphaned users
- Password reset required
- Not suitable for production migrations

Best Use Case

- Small environments
- Non-production systems

3. Backup and Restore System Databases (Advanced / Risky)

Method

- Backup **master** (and optionally msdb)
- Restore on target server

What It Transfers

- All logins
- Passwords
- SIDs
- Server roles
- Jobs (if msdb restored)

Pros

- Complete server-level migration
- No scripting required

Cons

- Server version must match exactly
- Overwrites target configuration
- High risk
- Not supported for cross-version restores

Best Use Case

- Disaster recovery
- Full server replacement

4. Using SQL Server Integration Services (SSIS)

Method

- Transfer Logins Task
- Transfer SQL Server Objects Task

What It Transfers

- SQL logins
- Windows logins
- Optional: passwords and SIDs

Pros

- Automated
- Supports bulk migration

Cons

- Complex setup
- Less transparent
- Deprecated usage in modern deployments

Best Use Case

- Enterprise automation
- Repeated migrations

5. Using T-SQL Scripts (Manual Scripting)

Method

- Query sys.server_principals
- Generate custom CREATE LOGIN scripts

Example (Simplified)

```
SELECT
 'CREATE LOGIN [' + name + '] WITH PASSWORD = 0x' +
 CONVERT(VARCHAR(MAX), password_hash, 2) + ' HASHED, SID = 0x' +
 CONVERT(VARCHAR(MAX), sid, 2)
FROM sys.sql_logins;
```

Pros

- Fully customizable
- Automatable

Cons

- Error-prone
- Requires deep knowledge
- Password policies not always preserved

Best Use Case

- Custom migration tooling
- DevOps pipelines

6. Using Database Backup and Restore (Indirect Method)

Method

- Restore databases to target server
- Fix orphaned users afterward

What Happens

- Database users are restored
- Logins are NOT transferred

Follow-Up Step

Run:

ALTER USER username WITH LOGIN = loginname;

Pros

- Simple
- Common approach

Cons

- Manual user-login mapping
- Time-consuming
- Risk of security mismatches

Best Use Case

- When login transfer was missed
- Small databases

7. Windows Logins (Active Directory)

Method

- Recreate logins on target server:

CREATE LOGIN [DOMAIN\User] FROM WINDOWS;

Notes

- SID is managed by Active Directory
- Passwords not stored in SQL Server

Pros

- No password handling
- Seamless if domain is same

Cons

- Requires domain access
- Cannot migrate across domains without trust

Best Use Case

- Domain-based environments

8. Using Third-Party Tools

Common Tools

- Redgate SQL Compare / SQL Data Compare
- ApexSQL Diff
- Idera SQL Admin Toolset

Pros

- GUI-based
- Auditable
- Safer for large environments

Cons

- Licensing cost
- Vendor dependency

Best Use Case

- Enterprise migrations
- Compliance-driven environments

9. Azure-Related Migrations (Special Case)

Azure SQL Managed Instance

- Logins can be migrated
- Supports SQL logins and Windows logins (with limitations)

Azure SQL Database

- No server-level logins
- Use database users instead

Recommended Approach by Scenario

Scenario	Recommended Method
Production migration	sp_help_revlogin
Full server replacement	Master DB restore
Small environment	SSMS scripting
Automated pipeline	Custom T-SQL / SSIS
Domain-based security	Recreate Windows logins
Cloud migration	Azure-specific tooling

Key Best Practices

- Always preserve **SID** to avoid orphaned users
- Script **server role memberships separately**
- Test on non-production first
- Document security mappings
- Avoid deprecated procedures in new deployments

Login-migration differences between on-premises SQL Server and Azure platforms, with **architecture, supported objects, limitations, and recommended approaches**.
This reflects real-world migration scenarios and Microsoft-supported behavior.

1. Fundamental Architectural Difference

SQL Server (On-Prem / IaaS VM)

- **Server-level security model**
- Logins are stored in:
  - master database
  - sys.server_principals
- Databases map users to logins using **SID**
- Supports:
  - SQL Logins
  - Windows (AD) Logins
  - Server roles and permissions

Azure SQL (PaaS)

- **Database-level security model**
- No access to master for login creation (except MI)
- Authentication handled at:
  - Database scope
  - Azure AD (Entra ID)

Key Impact:

Traditional login migration methods (e.g., sp_help_revlogin) **do not work in most Azure SQL offerings**.

2. Platform-by-Platform Comparison

A. SQL Server → SQL Server (On-Prem or Azure VM)

Aspect	Behavior
Logins	Fully supported
SIDs	Preserved
Password hashes	Preserved
Server roles	Supported
Migration method	sp_help_revlogin, SSIS, scripts
Orphaned users	Avoidable

This behaves exactly like traditional SQL Server.

B. SQL Server → Azure SQL Managed Instance (MI)

Azure SQL Managed Instance is **closest to on-prem SQL Server**.

Supported

- SQL Logins
- Azure AD logins
- SIDs preserved
- Password hashes preserved
- Server roles (with some limitations)
- sp_help_revlogin works

Not Supported / Limited

- Some server-level permissions
- Certain system logins (e.g., sa)

Migration Methods

- sp_help_revlogin
- Azure Database Migration Service (DMS)
- Native backup/restore

Verdict

Best Azure target if login fidelity is required.

C. SQL Server → Azure SQL Database (Single / Elastic Pool)

Azure SQL Database **does NOT support server-level logins**.

Not Supported

- SQL Server logins
- CREATE LOGIN
- sys.server_principals
- SIDs
- Password hashes
- sp_help_revlogin

Supported

- Contained database users
- Azure AD authentication
- Database-scoped permissions

Required Migration Approach

- Convert logins to:
  - **Contained database users**
  - **Azure AD users/groups**

Example Conversion

```
CREATE USER AppUser WITH PASSWORD = 'StrongPassword!';
```

Impact

- Passwords must be reset
- Application connection strings must change
- Each database manages its own users

D. SQL Server → Azure SQL Database with Azure AD

This is the **recommended modern security model**.

Characteristics

- Centralized identity (Entra ID)
- No passwords stored in SQL
- MFA supported
- Group-based access

Example

```
CREATE USER [AAD_Group] FROM EXTERNAL PROVIDER;
```

Benefits

- Strong security
- No login migration complexity

- Enterprise-ready

Limitation

- Requires Azure AD integration
- Application must support AAD auth

3. Login Type Compatibility Matrix

Login Type	SQL Server	Azure MI	Azure SQL DB
SQL Login	Yes	Yes	No
Windows AD Login	Yes	Limited	No
Azure AD Login	No	Yes	Yes
Password Hash	Yes	Yes	No
SID Preservation	Yes	Yes	No
Server Roles	Yes	Partial	No

4. Orphaned User Behavior

SQL Server / Azure VM / MI

- Orphaned users can occur
- Fixable using:

ALTER USER user_name WITH LOGIN = login_name;

Azure SQL Database

- No orphaned users concept
- Users are self-contained
- Login-to-user mapping does not exist

5. Migration Tool Behavior Differences

Azure Database Migration Service (DMS)

Target	Login Migration
SQL Server / Azure VM	Fully supported
Azure SQL Managed Instance	Supported
Azure SQL Database	Not supported

6. Common Migration Mistakes

- Trying to use sp_help_revlogin with Azure SQL Database
- Expecting SIDs to exist in Azure SQL Database
- Migrating SQL logins instead of redesigning security
- Not updating application connection strings
- Ignoring Azure AD integration

7. Recommended Strategy by Target

Target Platform	Recommended Security Model
SQL Server (VM/IaaS)	Traditional logins
Azure SQL Managed Instance	Traditional + Azure AD
Azure SQL Database	Azure AD users / Contained users

8. Decision Guidance

If you must preserve logins, passwords, and SIDs: → Choose **Azure SQL Managed Instance**

If you want cloud-native security: → Choose **Azure SQL Database + Azure AD**

If lift-and-shift is required: → Use **Azure VM with SQL Server**

9. Executive Summary

- SQL Server uses **server-level logins**
- Azure SQL Database uses **database-level users**
- Azure SQL Managed Instance bridges the gap
- Login migration is **not a direct process** in Azure SQL DB
- Security redesign is often required, not migration

Login-migration differences between on-premises SQL Server and Azure data platforms, written from an architectural and operational perspective. This is especially relevant when moving from **SQL Server (2019–2025)** to **Azure SQL Database**, **Azure SQL Managed Instance**, or **SQL Server on Azure Virtual Machines**.

1. Fundamental Architecture Difference

On-Premises SQL Server

- Uses **server-level logins**
- Logins are stored in **master**
- Database users are mapped to logins via **SID**
- Supports:
  - SQL logins
  - Windows (AD) logins
  - Server roles (sysadmin, securityadmin, etc.)

Azure

Azure security depends on the deployment model:

- **Azure SQL Database (Single / Elastic Pool)** → *No server-level logins*
- **Azure SQL Managed Instance (MI)** → *Full login support*
- **SQL Server on Azure VM** → *Same as on-prem*

This is the most important conceptual difference.

2. Platform-by-Platform Login Support

A. SQL Server → Azure SQL Database

Login Support

- ❌ Server-level logins **not supported**
- ✅ Only **database users**
- Authentication methods:
  - Azure AD users
  - Azure AD groups
  - Contained SQL users

Migration Impact

- CREATE LOGIN scripts **do not work**
- sp_help_revlogin **cannot be used**
- SIDs are irrelevant

Required Changes

- Convert logins to **contained users**
- Re-create users inside each database

Example

```
CREATE USER AppUser WITH PASSWORD = 'StrongPassword';
```

Key Limitation

- No sysadmin
- No server roles
- No cross-database login reuse

Best Use Case

- Cloud-native applications
- SaaS or PaaS-oriented workloads

B. SQL Server → Azure SQL Managed Instance (MI)

Login Support

- ✅ Fully supports server-level logins
- ✅ Supports:
  - SQL logins
  - Azure AD logins
  - Azure AD groups
- ❌ Windows (on-prem AD) logins not supported directly

Migration Compatibility

- sp_help_revlogin **works**
- SIDs are preserved
- Server roles are supported

Differences from On-Prem

- No local Windows accounts
- Uses **Azure AD** instead of traditional AD
- Some server permissions restricted

Recommended Migration Path

- Use sp_help_revlogin
- Replace Windows logins with Azure AD identities

Best Use Case

- Lift-and-shift migrations
- Minimal application change

C. SQL Server → SQL Server on Azure VM

Login Support

- ✅ Identical to on-prem SQL Server
- Supports:
  - SQL logins
  - Windows logins
  - AD authentication
  - Server roles

Migration Impact

- No changes required
- All existing login migration methods apply

Recommended Methods

- sp_help_revlogin
- Master database restore
- SSIS Transfer Logins Task

Best Use Case

- Legacy workloads
- Full OS control required

3. Authentication Type Differences

Authentication Type	On-Prem SQL	Azure SQL DB	Azure SQL MI	Azure VM
SQL Login	Yes	Contained only	Yes	Yes
Windows AD Login	Yes	No	No	Yes
Azure AD User	No	Yes	Yes	Yes
Azure AD Group	No	Yes	Yes	Yes
Server Roles	Yes	No	Yes	Yes

4. SID Handling Differences

On-Prem SQL Server

- SID is critical
- Mismatched SID → orphaned users
- sp_help_revlogin preserves SID

Azure SQL Database

- No SID concept
- Users are self-contained
- Orphaned users do not exist

Azure SQL Managed Instance

- SID behavior same as on-prem
- Preserving SID still essential

5. Server Roles and Permissions

On-Prem SQL Server

- Full access to:
  - sysadmin
  - securityadmin
  - server-level permissions

Azure SQL Database

- ❌ No server roles
- Limited administrative permissions
- db_owner is highest role

Azure SQL Managed Instance

- Supports server roles
- Some permissions restricted for platform security

6. Password Handling Differences

Aspect	On-Prem	Azure SQL DB	Azure SQL MI
Password Hash Migration	Yes	No	Yes
Password Reset Required	No	Yes	No
Enforced Policies	Optional	Mandatory	Optional

7. Migration Tool Compatibility

Tool / Method	On-Prem → On-Prem	To Azure SQL DB	To Azure SQL MI
sp_help_revlogin	Yes	No	Yes
SSMS Script Login	Partial	No	Partial
Master DB Restore	Yes	No	No
SSIS Transfer Logins	Yes	No	Limited
Azure DMS	Yes	User-level only	Yes

8. Common Migration Pitfalls

Moving to Azure SQL Database

- Expecting logins to migrate
- Relying on server roles
- Cross-database access assumptions

Moving to Azure SQL Managed Instance

- Using Windows logins instead of Azure AD
- Forgetting to migrate server role memberships

9. Recommended Migration Strategy by Target

Azure SQL Database

- Convert logins → contained users
- Use Azure AD authentication
- Redesign security model

Azure SQL Managed Instance

- Use sp_help_revlogin
- Replace Windows logins with Azure AD
- Minimal application changes

Azure VM

- Treat as on-prem SQL Server
- Use existing login migration methods

10. Executive Summary

- **Azure SQL Database** requires a **security redesign**
- **Azure SQL Managed Instance** supports **traditional login migration**
- **Azure VM** offers **full compatibility**
- Choice of Azure platform directly determines:
    - Login strategy
    - SID relevance
    - Authentication method
    - Administrative capabilities

1. Step-by-Step Login Conversion Guide

On-Prem SQL Server → Azure SQL Database
Target: **Azure SQL Database (Single / Elastic Pool)**
Key concept: **Server logins do not exist** → convert to **contained database users**

Step 1: Inventory Existing Logins (On-Prem)
Identify all logins that access the database.

```
SELECT
  sp.name AS LoginName,
  sp.type_desc AS LoginType,
  dp.name AS DatabaseUser
FROM sys.server_principals sp
LEFT JOIN sys.database_principals dp
  ON sp.sid = dp.sid
WHERE sp.type IN ('S','U','G');
```

Classify logins into:

- SQL Logins
- Windows Users
- Windows Groups
- Application/service accounts

Step 2: Identify Authentication Strategy
Choose one per login:

On-Prem Login Type	Azure SQL DB Replacement
SQL Login	Contained SQL User
AD User	Azure AD User
AD Group	Azure AD Group
Service Account	Contained SQL User or Azure AD App

Best Practice: Prefer **Azure AD groups** for human users.

Step 3: Enable Azure AD Admin (Once per Server)
In Azure Portal:

- Set **Azure AD Admin** for the SQL logical server
- Required to create Azure AD users/groups

Step 4: Create Users in Azure SQL Database
A. Convert SQL Login → Contained SQL User

```
CREATE USER AppUser
WITH PASSWORD = 'StrongPassword!2025';
```

B. Convert AD User → Azure AD User

```
CREATE USER [user@company.com]
FROM EXTERNAL PROVIDER;
```

C. Convert AD Group → Azure AD Group

```
CREATE USER [DB_Readers_Group]
FROM EXTERNAL PROVIDER;
```

Step 5: Reapply Database Permissions

Extract permissions from on-prem:

```
SELECT * FROM sys.database_permissions;
```

Apply in Azure:

```
ALTER ROLE db_datareader ADD MEMBER AppUser;
ALTER ROLE db_datawriter ADD MEMBER AppUser;
```

There is **no sysadmin** or server-level permission in Azure SQL DB.

Step 6: Validate Access

Test:

- Application connectivity
- Cross-schema access
- Stored procedure execution

Step 7: Decommission Legacy Assumptions

Remove or refactor:

- Cross-database queries
- USE master
- Server-level permissions
- SQL Agent–based security logic

Key Outcome

✓ No orphaned users

✓ No SID dependency

✓ Cloud-native security model

2. Comparison Diagrams (Text-Based, Presentation-Ready)

Diagram 1: On-Prem SQL Server Security Model

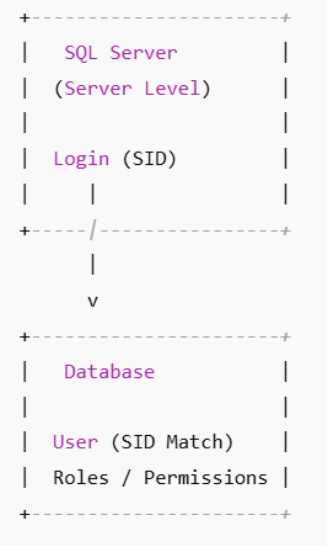

Diagram 2: Azure SQL Database Security Model

```
+----------------------+
| Azure SQL Database   |
|  (No Server Logins)  |
|                      |
|  User (Contained)    |
|  Azure AD Identity   |
|  Roles Only          |
+----------------------+
```

Diagram 3: Migration Conceptual Shift

```
ON-PREM                    AZURE SQL DB
---------                  -------------
Login (Server)      ❌ --->  User (Database)
SID Dependency      ❌ --->  Contained Identity
Windows AD          ❌ --->  Azure AD
Sysadmin            ❌ --->  db_owner
```

3. Migration Decision Matrix (Management / Executive View)

Platform Selection Matrix

Requirement	Azure SQL DB	Azure SQL MI	Azure VM
Minimal code change	❌	✅	✅
Server-level logins	❌	✅	✅
Azure-native PaaS	✅	⚠️	❌
Windows authentication	❌	❌	✅
Lowest admin overhead	✅	⚠️	❌
Legacy app compatibility	❌	✅	✅

Login Migration Complexity Matrix

Factor	Azure SQL DB	Azure SQL MI	Azure VM
Login migration effort	High	Low	Very Low
Password migration	No	Yes	Yes
SID preservation	N/A	Required	Required
Security redesign	Mandatory	Minimal	None

Risk vs Flexibility Matrix

Platform	Risk	Flexibility	Cost Efficiency
Azure SQL DB	Medium (Redesign)	Low	High
Azure SQL MI	Low	Medium	Medium
Azure VM	Low	High	Low

Executive Recommendation Logic

If application can change → Azure SQL DB

If lift-and-shift needed → Azure SQL MI

If OS / AD dependency → Azure VM

Final Summary for Stakeholders

- **Azure SQL Database** requires a **security redesign**, not a migration
- Logins become **contained users**
- Azure AD replaces Windows authentication
- Server-level permissions are eliminated
- Long-term benefits: lower cost, higher security, less admin overhead