Script Finds Users in WS Account Table that do not Exist in AspNetUsers

07/16/2025 10:10 am EDT

Overall Goal

This script identifies and resolves email collisions or misalignments between:

- WS_Account — the SedonaWeb Legacy 1.0 account table
- AspNetUsers — the newer ASP.NET Identity table used for SedonaWeb 2.0 logins

These conflicts often result in:

- “Account already exists” errors (because an email is reused in the wrong place)
- “Account doesn’t exist” (because the expected linkage isn't present)

What This Script Does

Identify invalid WS_Account records
Look for emails that do not match in the SedonaWeb 2.0 login system
Fix email clashes in WS_Account
Adds 11 prefix to prevent conflicts
Backup for safety
Creates WS_Account_Bak
Undo support
Allows WS_Account rollback using the backup
Find orphaned AspNetUsers
Finds web users not tied to a WS_Account
Fix email collisions in AspNetUsers
Renames them to allow the reuse of the actual email

When to Run This

When customers can't register on the portal because:
The system says their account doesn't exist
Their email is already in use by an old or conflicting record
After migrations or bulk imports
As a one-time or manual maintenance operation

Step-by-Step Breakdown

Step 1 – Backup

-- Creates a full backup of the WS_Account table, just in case something goes wrong.
-- This is a safe practice before modifying data.

```
Select * into WS_Account_Bak from WS_Account
```

Step 2 – Identify WS_Accounts without a match in SedonaCloud

-- Finds all records in WS_Account that do not have a matching Email in AspNetUsers.
-- These are candidates for correction, because they reference users that don't exist (yet) in the web system.

```
SELECT asp.Id, asp.BaseCompanyId, ws.*

FROM WS_Account AS ws

LEFT OUTER JOIN SedonaCloud.dbo.AspNetUsers AS asp ON ws.Account_Name = asp.Email

WHERE asp.Email IS NULL
```

Step 3 – Fix WS_Account email collisions

-- Modifies WS_Account records by prefixing their Account_Name field with 11
-- This effectively prevents future email collisions by ensuring those records don't clash with AspNetUsers.Email.

-- This helps resolve errors where the WS_Account table has stale or invalid email addresses.

```
Update WS_Account

set Account_Name = '11'+Account_Name

...

WHERE asp.Email IS NULL
```

Step 4 – Restore if needed

-- Rolls back the change made in step 3, if needed
-- This restores the Account_Name field from the backup.

-- Useful for undoing any unintended impact.

```
UPDATE WS_Account

SET WS_Account.Account_Name = BAK.Account_Name

FROM WS_Account WS

INNER JOIN WS_Account_Bak bak ON ws.Account_Id = bak.Account_Id
```

Step 5 – Identify unused AspNetUsers emails
-- Finds AspNetUsers entries that:
-- Have no corresponding WS_Account
-- Are internal users (e.g., not tied to an employee)
-- Belong to BaseCompanyId = 1 (a specific customer or system)
-- These may be orphaned or incorrectly provisioned user records.

```
SELECT ws.Account_Name, asp.*

FROM SedonaCloud.dbo.AspNetUsers asp

LEFT OUTER JOIN WS_Account ws ON ws.Account_Name = asp.Email

WHERE (ws.Account_Name IS NULL) AND BaseCompanyId = 1 AND Ext_EmployeeId IS NULL
```

Step 6 – Fix AspNetUsers conflicts

-- Modifies conflicting or legacy AspNetUsers records by prefixing their usernames/emails with 11.
-- This allows new valid users to reuse those email addresses when registering.
-- Essentially, this is a "soft delete" or email disassociation of invalid users.

```
UPDATE asp

SET

  Email = '11' + Email,

  NormalizedEmail = '11' + NormalizedEmail,

  NormalizedUserName = '11' + NormalizedUserName,

  UserName = '11' + UserName

...

WHERE (ws.Account_Name IS NULL) AND BaseCompanyId = 1 AND Ext_EmployeeId IS NULL
```