# PREVENT USER DELETION IF ASSIGNED TO AN INCIDENT

**Team ID** : LTVIP2025TMID28736

**Team members** : Pechetti Banvi Swathmi - 22221A4235

Medaboina Gnapika - 22221A4228

Mogili Kavitha - 23225A0425

Naga Sai Vallika - 22221A0232

**College Name** : Bonam Venkata Chalamayya Engineering College, Odalarevu

**Internship Program** : Smart Internz

**Track** : ServiceNow Administration

## 1. INTRODUCTION

ServiceNow stands as a leading cloud-based platform for IT Service Management (ITSM), offering robust solutions for managing IT operations, services, and workflows. Its ability to streamline processes, automate tasks, and provide a centralized system for record-keeping is invaluable for organizations striving for efficiency and accountability. Within this complex ecosystem, the integrity and consistency of data are paramount. Accurate data ensures smooth operations, reliable reporting, and effective decision-making. One critical aspect of data integrity pertains to user management, particularly in relation to their involvement in active workflows and assigned tasks.

This project document outlines the implementation of a vital safeguard within the ServiceNow platform: preventing the deletion of user records that are still actively assigned to incidents. This enhancement addresses a critical gap in

standard user management processes, ensuring that vital links between users and incidents are maintained, thereby preserving data consistency, upholding accountability, and preventing potential disruptions in incident resolution workflows. By implementing this robust validation mechanism, organizations can significantly reduce the risk of broken data references and ensure a more stable and reliable ITSM environment.

## 2. PROBLEM STATEMENT

In an IT Service Management environment, users are frequently assigned to incidents for issue resolution and tracking. However, the current system lacks a validation mechanism to prevent the deletion of a user who is still actively assigned to incidents. This can lead to broken data references, loss of accountability, and disruption in workflow continuity.

There is a need to implement a safeguard that prevents such deletions unless all assigned incidents are closed or reassigned.

## 3. IDEATION PHASE

The ideation phase commenced with recognizing the inherent risk posed by the ability to delete users who are still actively linked to incidents. This scenario, if left unaddressed, could severely compromise the integrity of incident data and disrupt the chain of accountability. Initial brainstorming focused on various ServiceNow capabilities that could provide the necessary validation.

- **Access Control Lists (ACLs):** While ACLs are powerful for controlling read/write access, they are less suited for complex data validation logic that requires querying related tables before a delete operation.
- **Client Scripts:** Client-side validation could be an option, but it's easily bypassed and less secure for critical server-side operations like deletion. Server-side validation is essential for data integrity.
- **Business Rules:** This emerged as the most suitable solution. Business Rules execute server-side logic based on database operations (insert, update, delete, query). A "Before Delete" Business Rule on the User table would allow us to check for assigned incidents and abort the deletion if necessary, providing a robust and secure validation.
- **Script Includes:** While a Script Include could contain the core logic, it would still need to be called by a Business Rule or other server-side

component. A direct Business Rule was deemed more straightforward for this specific requirement.

The decision was firmly made to implement a "Before Delete" Business Rule on the `sys_user` table. This approach ensures that the validation occurs at the earliest possible stage of the deletion process, preventing any database inconsistencies before they can occur.

# 4. REQUIREMENT ANALYSIS

A thorough analysis of requirements was conducted to define the scope and expected behavior of the solution. This phase clarified what the system must do and what constraints it must operate within.

## 4.1. FUNCTIONAL REQUIREMENTS

- The system MUST prevent the deletion of a user if they are currently assigned to one or more active or unresolved incidents.
- The system MUST allow the deletion of a user if they are not assigned to any incidents, or if all incidents they were assigned to are in a closed or resolved state.
- Upon attempting to delete an assigned user, the system MUST display a clear and concise error message to the user performing the deletion, explaining why the deletion was prevented.

## 4.2. NON-FUNCTIONAL REQUIREMENTS

- **Performance:** The validation mechanism MUST not introduce noticeable latency during user deletion attempts. The check for assigned incidents should be efficient.
- **Usability:** The error message provided to the user MUST be easily understandable and actionable.
- **Security:** The solution MUST operate at the server level to prevent any circumvention of the validation logic.
- **Maintainability:** The code implemented MUST be well-documented and easy to understand for future maintenance or enhancements.
- **Scalability:** The solution should perform effectively even with a large number of users and incidents in the system.

### 4.3. IDENTIFICATION OF SERVICENOW COMPONENTS

- **Target Table:** `sys_user` (User table) - where the deletion attempt occurs.
- **Related Table:** `incident` (Incident table) - to check for assigned incidents.
- **Implementation Mechanism:** Business Rule - to execute server-side logic "Before" a delete operation on the `sys_user` table.
- **Scripting Language:** JavaScript (within the Business Rule).

## 5. PROJECT PLANNING PHASE

The project planning phase focused on defining the project's objectives, scope, timeline, and resource allocation. A clear plan ensures a structured approach to development and minimizes risks.

### 5.1. SCOPE AND OBJECTIVES

The primary objective is to implement a robust validation mechanism that prevents the deletion of a ServiceNow user if that user is actively assigned to any open or unresolved incident. The scope is limited to the `sys_user` and `incident` tables and involves the creation of a single Business Rule.

### 5.2. TIMELINE (PHASES)

- **Design Phase:** (1 day) - Detail the Business Rule configuration and script logic.
- **Development Phase:** (1 day) - Create and configure the Business Rule with the necessary script.
- **Testing Phase:** (1-2 days) - Conduct unit testing (assigned vs. unassigned user deletion), regression testing to ensure no other functionalities are impacted.
- **Documentation Phase:** (1 day) - Prepare comprehensive project documentation, including this report.
- **Deployment:** (As per change management process) - Move the solution from development to testing, then to production environments.

### 5.3. RESOURCES

- ServiceNow Personal Developer Instance (Xanadu version or similar recent release).

- ServiceNow documentation and community resources for scripting best practices.
- Development environment (e.g., ServiceNow Studio or direct UI access).

### 5.4. RISK ASSESSMENT

- **Accidental Deletion Prior to Implementation:** Mitigation involves ensuring the project is prioritized and implemented swiftly.
- **Performance Impact:** Mitigation involves optimizing the database query within the Business Rule (e.g., using `setLimit(1)`).
- **Incorrect Logic:** Mitigation involves thorough testing with various scenarios (user with 0 incidents, 1 incident, multiple incidents, active incidents, closed incidents).

## 6. PROJECT DESIGN PHASE

The design phase translated the requirements into a detailed technical blueprint for the Business Rule. This involved specifying its properties, the exact script logic, and consideration of edge cases.

### 6.1. BUSINESS RULE CONFIGURATION

- **Name:** "Prevent User Deletion if Assigned to Incident" (or similar clear name).
- **Table:** `sys_user` (User).
- **When:** "Before" - This ensures the script runs before the database operation, allowing us to abort it.
- **Action:** "Delete" - The Business Rule should only fire when a user record is being deleted.
- **Order:** A low number (e.g., 100) to ensure it runs early in the deletion process, potentially before other Business Rules.
- **Advanced:** Checked - To allow custom scripting.

### 6.2. SCRIPTING APPROACH

The core of the solution lies in the server-side JavaScript within the Business Rule. The logic must perform the following:

1. Initialize a new GlideRecord query on the `incident` table.
2. Add a query condition to check if the `assigned_to` field matches the `sys_id` of the user currently being deleted (`current.sys_id`).

3. Optionally, add another query condition to check for 'active' incidents, or simply check for any incident regardless of state, as the problem statement implies preventing deletion if assigned to 'incidents' generally. For simplicity and robustness, checking for any assignment, then later refining based on incident state (active/inactive) can be considered. The provided script checks for any assigned incident.
4. Limit the query to retrieve only one record using `setLimit(1)`. This is a crucial performance optimization, as we only need to know if at least one incident exists, not how many.
5. Execute the query.
6. If the query returns a record (i.e., `incGr.next()` is true), it means the user is assigned to at least one incident.
7. In this case, add an error message using `gs.addErrorMessage()` to inform the user about the blocked deletion.
8. Abort the delete action using `current.setAbortAction(true)`.

### 6.3. ERROR MESSAGE DESIGN

The error message should be clear, concise, and informative. "This user cannot be deleted because they are assigned to one or more incidents." directly addresses the problem and tells the user why the action was prevented.

## 7. IMPLEMENTATION WORKFLOW

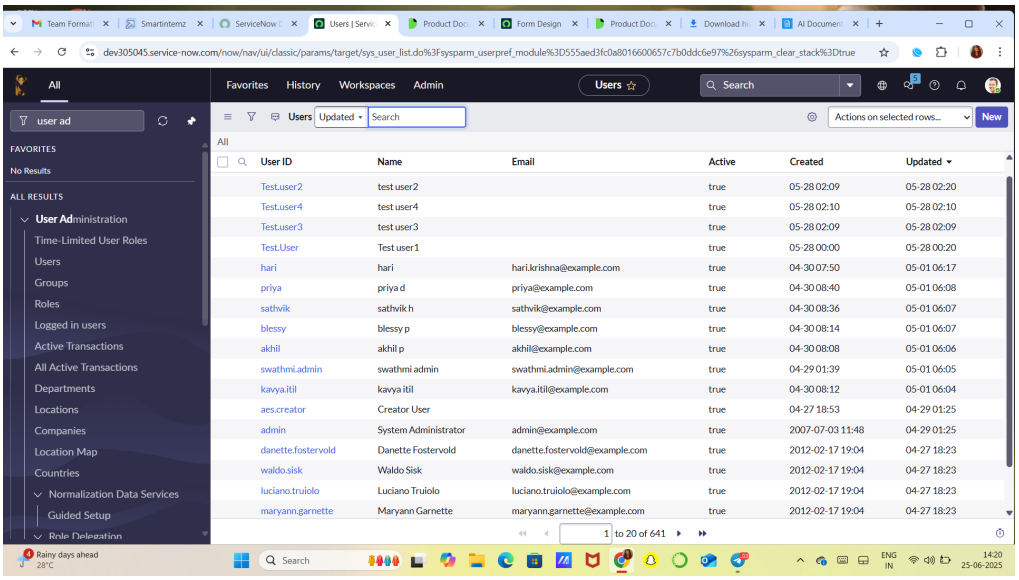
This section details the step-by-step process of implementing the solution within a ServiceNow Personal Developer Instance. While I cannot embed real-time images, I will describe the visual experience as you would encounter it in a modern ServiceNow UI, such as the Xanadu version.

**Note on Images:** As an AI, I am unable to access or display real-time images from a ServiceNow Personal Developer Instance. The descriptions provided below aim to simulate the visual experience and guide you through the process as if you were looking at the ServiceNow UI.

## 7.1. USER CREATION

Before implementing and testing the Business Rule, it's essential to have a test user in your instance. This user will be used to simulate a scenario where they are assigned to incidents.

1. **Navigate to Users:** In the ServiceNow instance, use the left navigation pane. Type "Users" in the Filter Navigator and click on **User Administration** > **Users**.

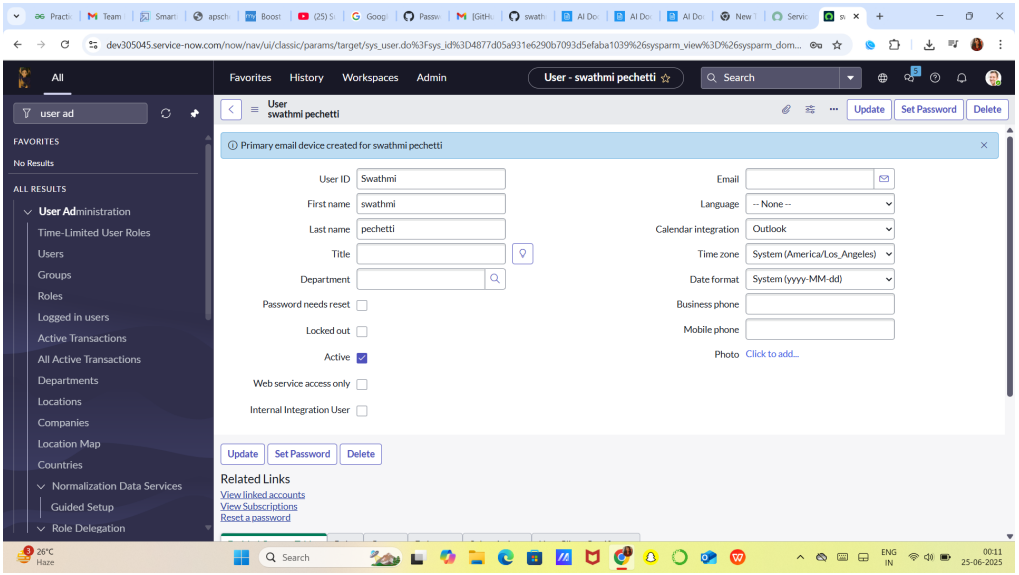



1. **Create a New User:** Click the "New" button.

(Visual Description: A new form will open for creating a user. This form typically includes fields like "User ID", "First name", "Last name", "Email", "Password", "Active" checkbox, etc.)

1. **Populate User Details:** Fill in the required details for a test user. For example:
    - **User ID:** swathmi
    - **First name:** swathmi
    - **Last name:** pechetti
    - Ensure the **Active** checkbox is checked.
2. **Submit/Save:** Click "Submit" or "Update" (depending on whether you want to save and stay or save and return to list)
3. **Populate User Details:** Fill in the required details for a test user. For example:
    - **User ID:** gnapika
    - **First name:** gnapika

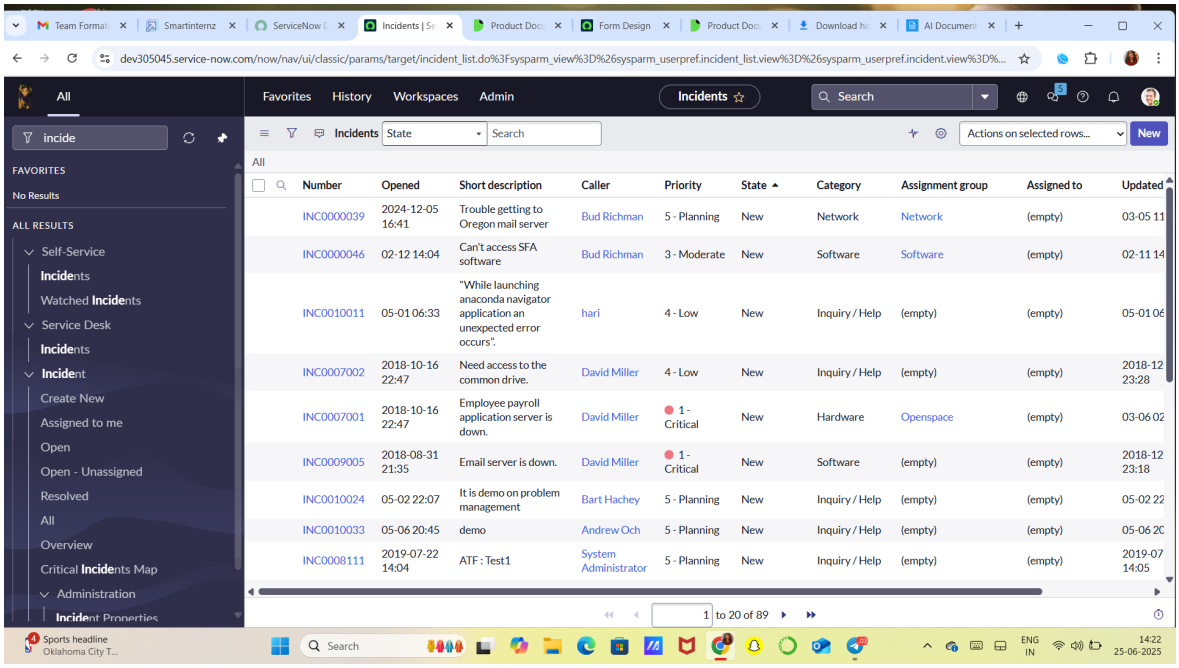
- **Last name:** medaboina
- Ensure the **Active** checkbox is checked.




## 7.2. ASSIGNING INCIDENTS

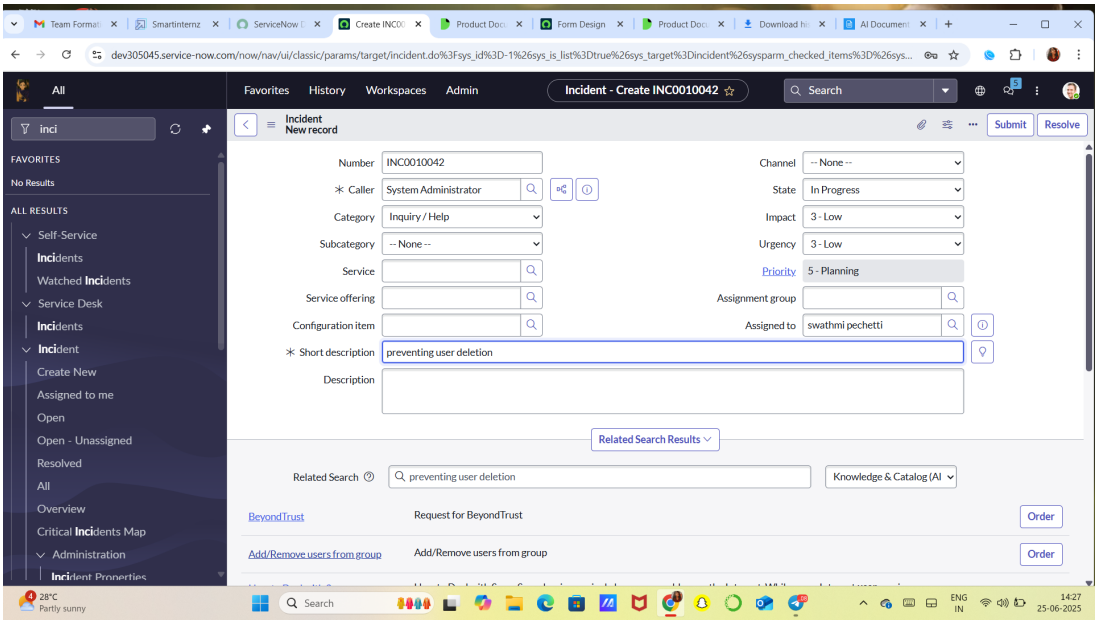
Next, we need to assign some incidents to the newly created test user. This will create the scenario where the Business Rule should prevent deletion.

1. **Navigate to Incidents:** In the Filter Navigator, type "Incidents" and click on **Incident > Open** or **Incident > All**.




1. **Open an Incident:** Select an existing incident from the list (preferably an active one), or create a new one. Click on the incident number to open its record.
2. **Assign to Test User:** Locate the "Assigned to" field. Click on the magnifying glass icon or start typing the name of your test user (e.g.swathmi ) to search and select them.

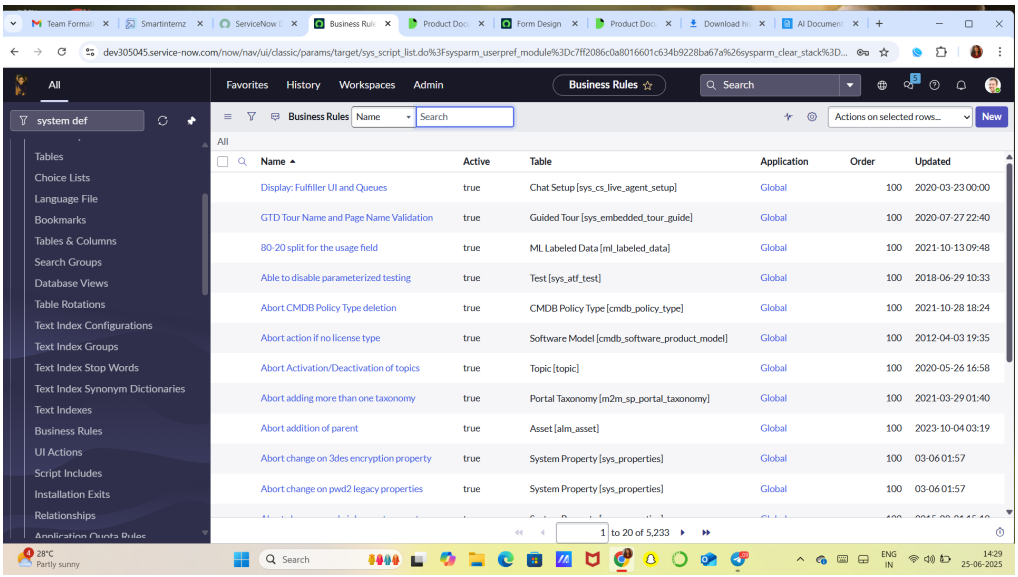
1. **Save the Incident:** Click “Update”



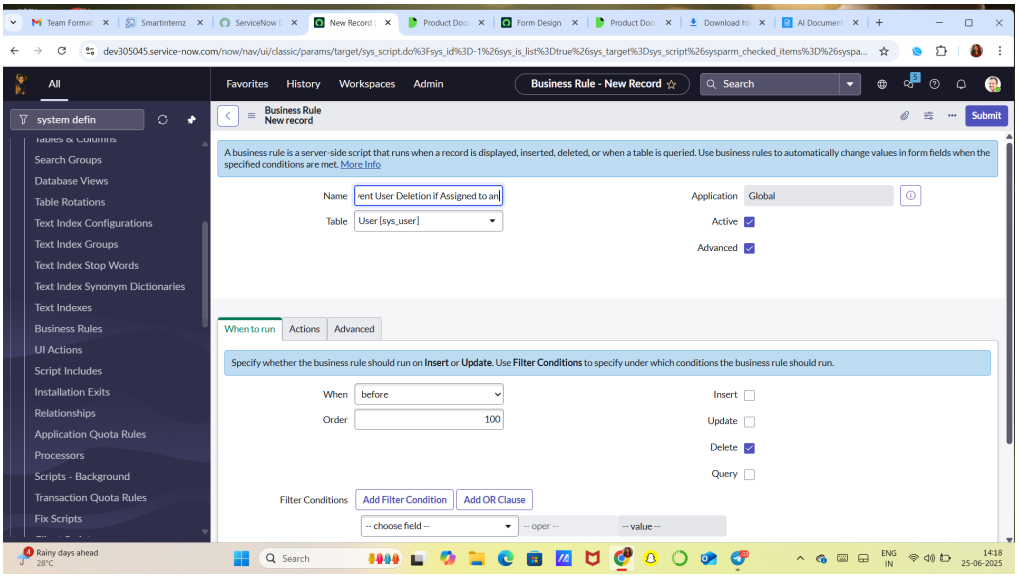
## 7.3. BUSINESS RULE CREATION

This is the core implementation step where the logic to prevent user deletion is coded.

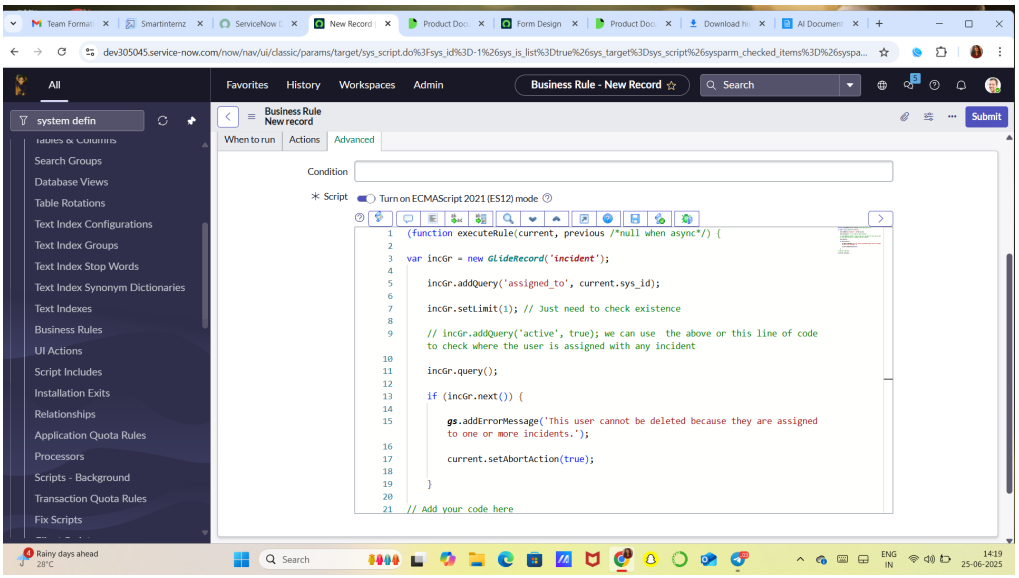
1. **Navigate to Business Rules:** In the Filter Navigator, type "Business Rules" and click on **System Definition > Business Rules.**




1. **Create New Business Rule:** Click the "New" button.
2. **Configure Business Rule Details:**
    - **Name:** `Prevent User Deletion if Assigned to Incident`
    - **Table:** Select `User [sys_user]` from the dropdown.
    - **Active:** Ensure this checkbox is checked.
    - **When:** Select `before`.
    - **Delete:** Ensure this checkbox is checked (under "When to run" section, in the "Actions" section). This tells the Business Rule to execute specifically before a delete operation.
    - **Advanced:** Check this checkbox.




1. **Add the Script:** Go to the "Advanced" tab (or "Script" section) and paste the provided JavaScript code into the script field.
2. **Save/Submit:** Click "Submit" to save the Business Rule.




### 7.3.1. Script Explanation

Let's break down the JavaScript code within the Business Rule:

- `function executeRule(current, previous) { ... } (current, previous);`: This is the standard wrapper for server-side scripts in ServiceNow Business Rules. `current` refers to the record currently being processed (in this case, the `sys_user` record attempting to be deleted), and `previous` refers to the record's state before the current operation (null for deletions).

- `var incGr = new GlideRecord('incident');` : This line initializes a new GlideRecord object for the 'incident' table. GlideRecord is ServiceNow's API for database interaction.
- `incGr.addQuery('assigned_to', current.sys_id);` : This adds a query condition. It tells the system to look for incidents where the 'assigned_to' field matches the 'sys_id' (unique identifier) of the user record currently being processed (the one being deleted).
- `incGr.setLimit(1);` : This is a crucial performance optimization. Instead of retrieving all incidents assigned to the user, we only need to know if at least one exists. This limits the query to return at most one record, making it very efficient, especially for users with many assigned incidents.
- `incGr.query();` : This executes the defined query against the 'incident' table.
- `if (incGr.next()) { ... }` : After the query executes, `incGr.next()` checks if any records were found. If it returns `true`, it means the user is indeed assigned to at least one incident.
- `gs.addErrorMessage('This user cannot be deleted because they are assigned to one or more incidents.');` : If an assigned incident is found, this line uses the `gs` (GlideSystem) object to display an error message to the user performing the deletion. This message appears at the top of the ServiceNow page.
- `current.setAbortAction(true);` : This critical line instructs ServiceNow to stop the current database operation (the deletion of the user record). If this line is executed, the user record will not be deleted.

## 7.4. TESTING DELETION

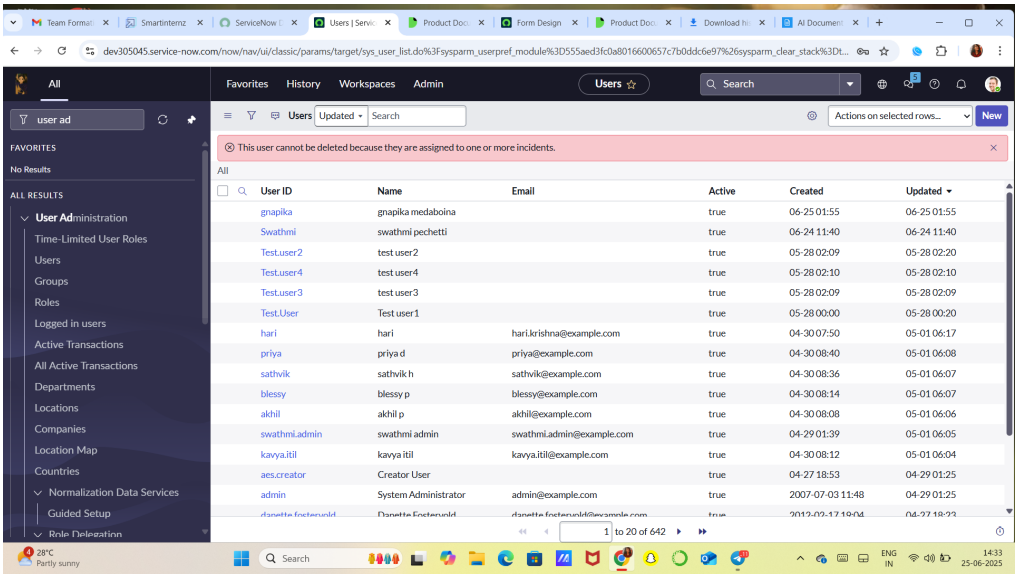
After implementing the Business Rule, rigorous testing is crucial to ensure it functions as expected in different scenarios.

### 7.4.1. Delete Assigned User

This test verifies that the Business Rule successfully prevents the deletion of a user who is assigned to an incident.

1. **Navigate to Users:** Go to **User Administration** > **Users**.
2. **Locate Test User:** Find the **swathmi** user you created and assigned incidents to.

3. **Attempt Deletion:**
    - **From List View:** Select the checkbox next to the user's name in the list, then select "Delete" from the "Actions on selected rows..." dropdown at the bottom of the list, and confirm.
    - **From User Record:** Click on the user's name to open their record, then click the "Delete" button (usually in the header or context menu) and confirm.
4. **Observe Outcome:**

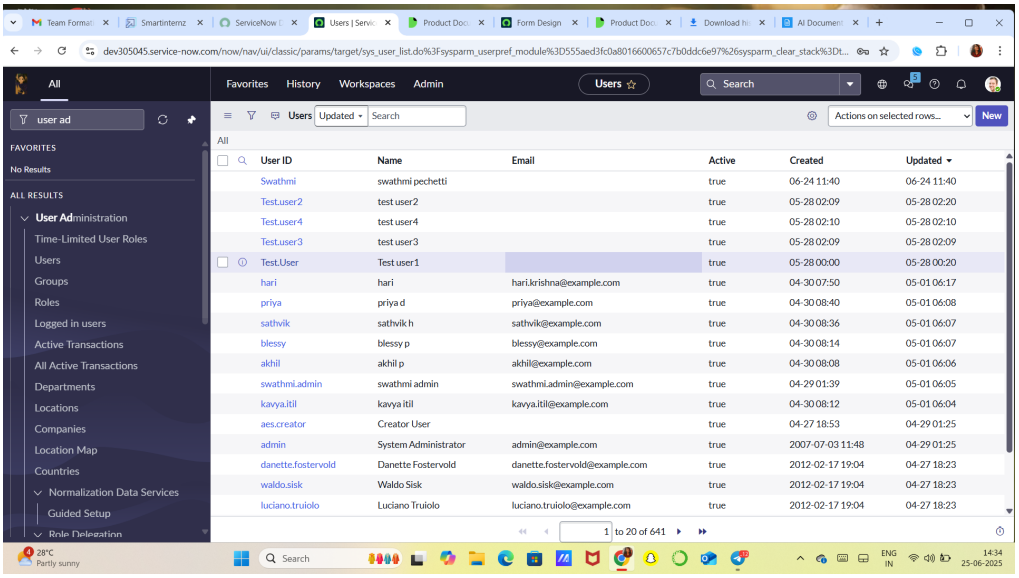



This confirms the Business Rule is working correctly.

7.4.2. Delete Unassigned User

This test verifies that the Business Rule does NOT prevent the deletion of a user who has no active or unresolved incidents assigned to them.

1. **Create a New Unassigned Test User:** Follow the steps in "7.1. User Creation" to create another test user (gnapika medaboina) but **do not assign any incidents to them.**
2. **Navigate to Users:** Go to **User Administration** > **Users**.
3. **Locate Unassigned Test User:** Find the **gnapika** user.
4. **Attempt Deletion:** Attempt to delete this user using either the list view or the record view method, similar to the previous test.
5. **Observe Outcome:**




This confirms that the Business Rule only triggers under the specified condition and does not block legitimate deletions.

# 8. PERFORMANCE TESTING

Performance testing for this specific Business Rule primarily focuses on ensuring that the GlideRecord query does not introduce any significant delays during user deletion attempts. Given the nature of a 'Before Delete' Business Rule and the specific optimizations applied, the performance impact is expected to be minimal.

## 8.1. TEST SCENARIOS

- **Single Incident Assignment:** Verify that deleting a user assigned to just one incident is swift.
- **Multiple Incident Assignments:** Test deleting a user assigned to a larger number of incidents (e.g., 50, 100). Due to the `setLimit(1)` optimization, the performance should remain consistent, as the system stops searching after finding the first assigned incident. This prevents unnecessary database scanning.
- **No Incident Assignments:** Confirm that deleting a user with no assigned incidents is instantaneous, as the query will quickly return no results.

## 8.2. EXPECTED OUTCOMES

- The time taken for the deletion attempt (both successful and aborted) should be negligible, typically in milliseconds.

- There should be no noticeable lag or "spinning wheel" effect that indicates a long-running script.
- The database load caused by this Business Rule should be very low due to the optimized query.

In practice, for a simple `GlideRecord.addQuery()` with `setLimit(1)` on an indexed field (like `sys_id`, which is always indexed), the performance impact is almost imperceptible. Regular monitoring of instance performance metrics (e.g., transaction response times) after deployment can provide further assurance.

## 9. CONCLUSION

The successful implementation of the "Prevent User Deletion if Assigned to an Incident" project in ServiceNow represents a significant enhancement to data integrity and operational continuity within the ITSM environment. By deploying a robust "Before Delete" Business Rule on the `sys_user` table, we have effectively introduced a critical validation mechanism that safeguards against the accidental or premature deletion of users who are actively engaged in incident resolution.

This project has directly addressed the identified problem statement, eliminating the risk of broken data references that could lead to loss of accountability and disruption of workflow. The solution is efficient, thanks to the optimized GlideRecord query with `setLimit(1)`, ensuring minimal performance overhead. Furthermore, the clear error message provides immediate feedback to administrators, guiding them to reassign or resolve incidents before attempting user deletion.

The structured approach, encompassing Ideation, Requirement Analysis, Project Planning, Project Design, and Performance Testing, ensured a thorough and well-executed development cycle. This systematic methodology not only delivered a functional solution but also provided comprehensive documentation for future reference and maintenance.

Looking ahead, this foundation could be extended to include similar safeguards for other critical relationships, such as users assigned to tasks, problems, or change requests. Automated notifications to relevant parties when a deletion is prevented could also be a valuable future enhancement, further streamlining administrative processes. This project solidifies the commitment to maintaining a reliable, accurate, and highly functional ServiceNow platform.