**Project Design Phase**

**Solution Architecture**

|  |  |
| --- | --- |
| Date | 15 February 2026 |
| Team ID | LTVIP2026TMIDS52445 |
| Project Name | Prevent User Deletion if Assigned to an Incident in ServiceNow |
| Maximum Marks | 4 Marks |

# Solution Architecture Overview

The proposed solution follows a **three-layer cloud architecture** within the ServiceNow platform to ensure secure and reliable validation before user deletion.

The architecture bridges the business problem (loss of incident ownership) with a technology solution (server-side validation using Business Rules).

# Architectural Layers

## 1️⃣ Presentation Layer (User Interface)

**Actor:** ServiceNow Administrator  
**Component:** ServiceNow Web UI

* Admin initiates user deletion from the user form
* Sends delete request to the ServiceNow application layer

## 2️⃣ Application Layer (Business Logic)

**Components:**

* Before Delete Business Rule
* GlideRecord Validation Logic
* Error Message Handler

**Process Flow:**

1. Delete request is triggered by admin
2. Business Rule executes on sys\_user table
3. GlideRecord queries incident table where:

assigned\_to = current user

1. Decision logic:
   * If incident exists → Abort deletion and show error
   * If no incident → Allow deletion

This ensures real-time validation and prevents data inconsistency.

## 3️⃣ Data Layer (ServiceNow Cloud Database)

**Data Stores:**

* sys\_user table → stores user records
* incident table → stores incident assignments

The Business Rule reads from the incident table and controls operations on the sys\_user table.

# Data Flow Summary

Admin → Delete User → Business Rule Trigger →  
Incident Table Validation → Decision →  
❌ Error Message (if assigned)  
✅ User Deleted (if not assigned)

# Key Architectural Characteristics

* **Server-side validation** prevents client-side bypass
* **Cloud-based deployment** on ServiceNow SaaS
* **High availability** and reliability
* **Scalable design** to support multiple ITSM tables
* **Low latency** using optimized GlideRecord queries

# Security Considerations

* Role-based access control (only admins can delete users)
* Server-side execution of validation logic
* Prevents unauthorized data manipulation

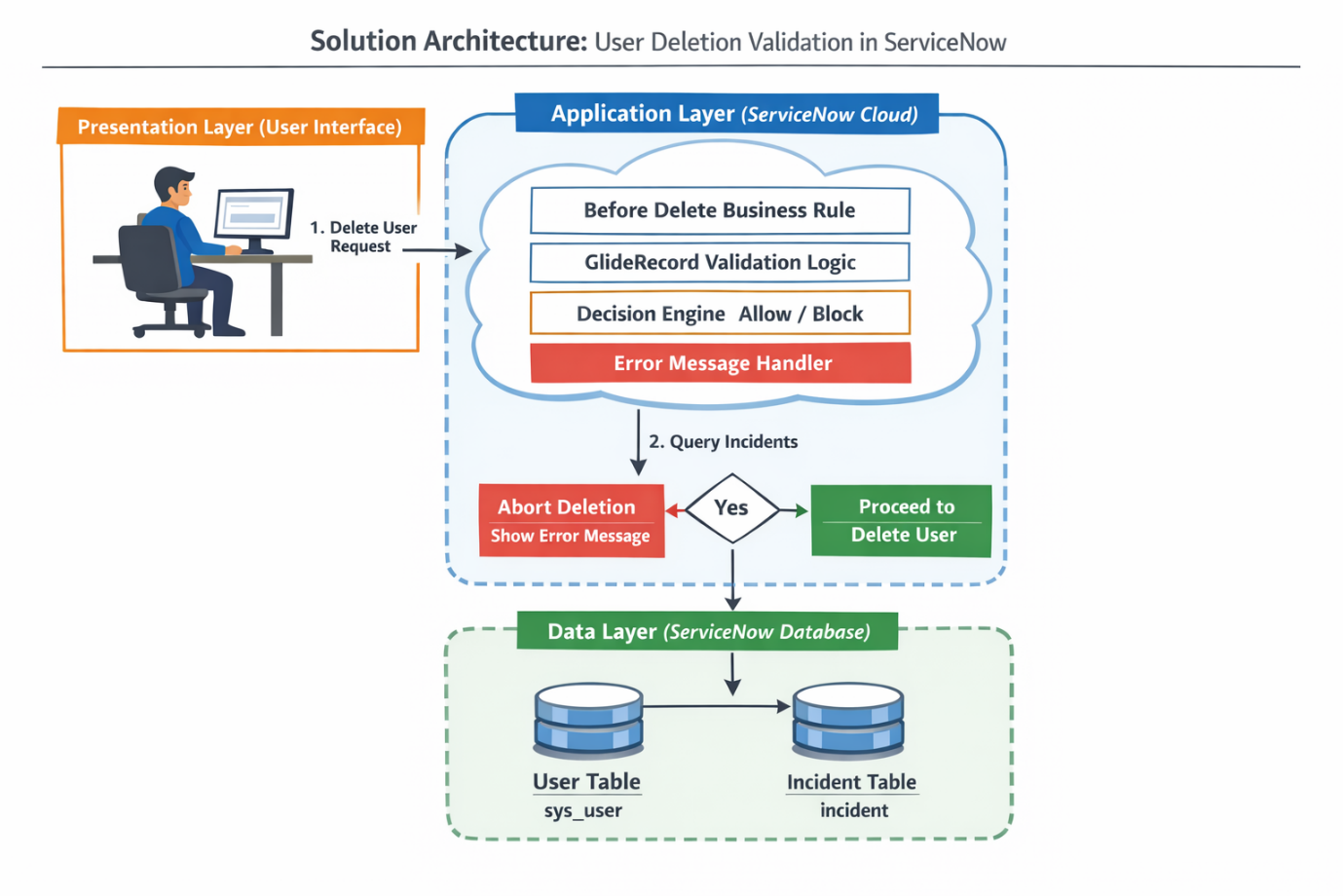
# Scalability

The architecture can be extended to:

* Problem table
* Change Request table
* Task table

It can also support:

* Role-based override
* Dependency dashboard

****