


# EPS Application Automation

Scope

Oracle- EBS, Oracle-SaaS, SNOW

# Instructions : Use case Details

- Each Use case and its scope of automation has been illustrated with details.
- The details of each use case consists of 3 slides ( In few cases 2 slides).
- The content of slides are structures as follow.
- Slide 1 – Consists of Problem details with description and proposed solution. It has other information on the usage of scripts vs tools for relevance
- Slide 2 – This slide contains detailed information about benefits of automation, Prioritization, metrics to evaluate the efficiency and dependencies for this enablement. It has general remarks and recommendations for next steps.
- Slide 3  This slide uniquely address the solution for automation with details such as script, templates and reference implementation. This slide contain title entitled as approach in the subject header.



In most use cases the approach section has been added for reference

**Note :** The approach to solve the use case in the scope of automation might require additional information on the system context for unique scenarios.

# GEMS – Active HR Record

- **What:**
  - Precisely identify manual steps for handling GEMS errors "No Active HR Record Found" and "Linked with two Active HR Records". Document these meticulously.
  - Determine the frequency of these errors.
- **How:**
  - Analyze if the root cause of the errors lies in upstream processes (HR data management, MBS access provisioning) or requires direct intervention within Oracle EBS.
- **Where:**
  - Pinpoint the exact modules/components in Oracle EBS where this rectification/data cleanup occurs.
- **Why:**
  - Business impact: How disruptive are these errors? Do they block critical processes?
  - Compliance Angle: Are there security or audit reasons to enforce accurate MBS to HR record linkage?

## Various Solutions with Implementation details

- **Option 1: Proactive Data Management**
  - **If root cause is upstream:** Work with HR/IT teams responsible for MBS access to refine their data input and provisioning processes.
  - **Implementation:** Training, workflow changes, integration improvements (Oracle EBS to HR systems).
- **Option 2: Oracle EBS Clean-up Script**
  - **If data inconsistencies are within EBS:** Develop a script (PL/SQL likely) to automate finding and resolving GEMS errors.
  - **Implementation:**
    - Careful design: Logic to identify the correct HR record, handle edge cases
    - Sandbox testing before production use
    - May need IT/DBA involvement for execution rights
- **Option 3: Workflow/Notification Tool**
  - **If manual intervention is unavoidable:** Consider tools to streamline error notification and resolution workflow.
  - **Implementation:** Investigate capabilities within Oracle EBS, or third-party workflow add-ons (research needed).

# GEMS – Active HR Record

## 3. Script-Based Automation vs. Bots/Tools

- This depends on the chosen solution (see #2). A cleanup script is likely. Bots (RPA) are possible if there are repetitive manual actions, but a deeper process analysis is needed to justify them.

## 4. Proposed Benefit of Automation

- Faster error resolution, reducing process downtime
- Improved data consistency within Oracle EBS
- Potential for reducing manual effort (more evident if error volume is high)
- Depending on the root cause, might lessen IT helpdesk tickets from end-users.

## 5. Type & Quantification of Savings

- Quantify current error volume and time spent manually resolving them.
- Think beyond direct time savings: consider the business impact of delays caused by GEMS errors
- Track time spent on current error resolution (helpdesk logs, user feedback)
- Estimate the reduction in effort after implementing automation (this will have some uncertainty)

## 6. Additional Remarks for Prioritization

- **Error Impact:** Are GEMS errors blocking critical business processes frequently? Are there compliance risks due to incorrect access?
- **Potential for Upstream Fix:** Could these issues be largely prevented by fixing MBS provisioning or HR data flow? If so, automation might be lower priority.
- **Technical Feasibility:** Assess the complexity of scripting vs. a workflow tool. Complexity impacts development effort.

## 7. Industry Usage Data

- No universal benchmark exists. GEMS error frequency depends on HR processes, security policies, and EBS customization.
- **Internal Analysis:** IT helpdesk likely has data on GEMS-related tickets.
- **Industry Forums:** Research Oracle-specific forums or user groups, it might have surveyed this informally.

# GEMS – Active HR Record

## 8. Implementation Time & Benefits Realization

- **Varies by solution:**
  - Upstream Fix: Could be weeks/months if process or integration changes are needed.
  - Cleanup Script: Potentially days/weeks if logic is complex and testing thorough.
  - Workflow Tool: Varies based on the chosen tool and configuration complexity.
- **Benefits:**
  - Error reduction should be near-immediate; quantifying time saved may take a few reporting cycles.
- **Dependencies:**
  - IT Skillset: PL/SQL for scripts, workflow tool knowledge if chosen.
  - Access: Developers/DBAs may need elevated permissions in Oracle EBS.
  - Testing Environment: Essential if manipulating EBS data directly.
- **Downsides:**
  - Automation won't fix the root cause if it's upstream. Errors may recur.
  - Improperly designed scripts carry the risk of data corruption (hence the emphasis on testing).

## 10. Metrics & Benchmarks

- **Outcome Metrics**
  - Number of hierarchy mismatches detected before they impact downstream processes.
  - Time spent on manual correction (before vs. after).
  - Perhaps reduced payroll rework costs (if applicable).
  - Number of GEMS errors before/after automation
  - Time per error resolution before/after
  - Subjective: Less frustration for users requesting access via MBS
- **Benchmark:**
  - Analysis pre- and post-automation metrics internally.
- **Integration:**
  - Depends entirely on your HR systems and MBS setup. Oracle EBS developer guidance.
- **Plugin**
- Oracle EBS-certified workflow add-ons, or RPA tools (if manual steps are substantial). Vendor demos are key here.

## Important Considerations

- **Change Control:** Any EBS-impacting automation needs proper change management process
- **Root Cause Fix is Ideal:** Automation is a patch, preventing the errors at their source is the long-term goal.

# Script based data backup

## What:

- Precise tables in Oracle EBS that contain the previous year's Benefits table data for OE.
- Exact format of the weekly status report (Excel, PDF, etc.).
- How is this report distributed at present (email, shared drive, etc.)?

## How:

- Is the 5 hours purely data extraction, or does it involve report formatting?
- Existing tools used for data extraction (SQL client, reporting tools, etc.).

## Where:

- Is Oracle EBS on-premise or cloud-hosted?

## Why:

- Bottleneck is solely the time taken for data extraction and prep.

## Various Solutions with Implementation details

### • Option 1: Pure SQL Script + Scheduling

- **If:** Data extraction is the major time sink, and formatting is minimal.

### • Implementation

- SQL script (PL/SQL likely) to query previous year's Benefits table in OE.
- Output data in CSV/Excel-compatible format
- Utilize Oracle EBS's built-in scheduler (if available) or OS-level scheduling (cron, Windows Task Scheduler) to run the script weekly.

### • Option 2: Reporting Tool Integration

- **If:** Oracle EBS has a reporting module or compatible BI tool.

### • Implementation

- Design a report extracting the Benefits data, parameterized for 'previous year.'
- Schedule the report to run weekly, with automated output to the desired format.

### • Option 3: Hybrid (Script + Basic Automation)

- **If:** Some manual formatting is still required after data extraction.

### • Implementation

- SQL script for data extraction (as in Option 1) scheduled to run weekly.
- A simple additional script (Python, or even a batch file) to pick up the output, do basic formatting, and place it in the distribution location.

# Script based data backup

- **3) Script-based vs. Bots**

- Script-based is the way to go here. RPA (bots) are overkill unless the report formatting involves complex interactions with an external system.

- **4) Proposed Benefits**

- Massive time savings: Potentially reduce 5 hours to near-instant execution.
- Consistency: Eliminates manual error potential
- Availability: Report generates even if the usual preparer is absent.

- **5) Type of Savings**

- Purely time-saving for the person doing the task. Doesn't directly impact EBS performance.

- **6) Quantifying Hours Saved**

- **Easy to measure:** Once automated, you have a clear before/after difference of ~5 hours/week.

- **Prioritization:** High, since the benefit is clear, and the task is repetitive.

- **Usage in Industry:** Very common, data extraction/reporting automation is a widespread practice.

- **Implementation Time:** Days to a week depending on script complexity, assuming EBS knowledge is in-house.

- **Dependencies:**

- Access to Oracle EBS schema to locate the correct tables.
- Script execution permissions on the EBS server or a system that can query the database.

- **Metrics:**

- Task time before/after
- Error reduction (if manual prep caused mistakes previously).

- **Integration Ecosystem:** Focus on EBS reporting tools if available; otherwise, general scripting is all that's needed



# Script based data backup – Approach Details

## • SQL Script Logic (PL/SQL)

SQL

```
-- Declare variables to hold the start and end dates of the previous year
DECLARE
    v_prev_year_start DATE;
    v_prev_year_end DATE;
BEGIN
    -- Calculate previous year's range (adjust if your fiscal year isn't 12 months)
    SELECT ADD_MONTHS(TRUNC(SYSDATE, 'YYYY'), -12) INTO v_prev_year_start
    SELECT ADD_MONTHS(v_prev_year_start, 12) - 1 INTO v_prev_year_end FROM DUAL;

    -- Replace placeholders with the exact table and column names
    SELECT column1, column2, ...
    FROM benefits_table -- Your actual Benefits table name
    WHERE date_column BETWEEN v_prev_year_start AND v_prev_year_end;

    -- Spool the output to a file (adjust path and filename as desired)
    SPOOL /path/to/output/previous_year_benefits_data.csv

    -- Uncomment to include column headers (adjust if needed)
    SET HEADING ON

    -- Repeat the SELECT query to output with formatting
    SELECT column1, column2, ...
    FROM benefits_table
    WHERE date_column BETWEEN v_prev_year_start AND v_prev_year_end;

    SPOOL OFF
END;
/
```

- **Scheduling Notes**
- **Oracle EBS Built-in:** If EBS has a concurrent program scheduler, that's ideal. Create a new concurrent program that executes your SQL script.
- **OS-level:**
  - **Windows:** Task Scheduler to run a batch file that invokes SQL\*Plus and executes the script.
  - **Linux:** Cron job with a similar setup.
- **Important Considerations**
- **Replace placeholders:** Table names, column names, output file paths need your actual values.
- **Data Validation:** After the first run, thoroughly check the output to ensure it pulls the correct data.
- **Formatting:** The sample script outputs CSV. Tweak if you need Excel formatting directly (might require external libraries depending on your setup).
- **Security:** Consult with your DBA on what level of permissions this script execution requires.
- **Next Steps**
  1. **Gather Schema Info:** Get the exact table/column names for your 'previous year Benefits' data within the OE module.
  2. **Test Environment:** Execute this script manually in a test EBS environment (if available) to validate the logic.
  3. **Choose Scheduling:** Decide to use either EBS scheduler or OS-level tools.



# SOP Automated Notifications during Payroll Process

- **What:**
  - Identify specific types of hierarchy mismatches that are most common (e.g., incorrect manager-employee links, department misalignments, etc.)?
  - What are the root causes of these mismatches? (data entry errors, changes in org structure, system updates, etc.)
- **How:**
  - How are hierarchy mismatches currently identified (manual checks, reports, user feedback)?
  - How are corrections made now (interface changes, data updates, etc.)?
- **Where:**
  - In which specific Oracle EBS modules do the mismatches happen?
  - Are any other systems synchronized based on this hierarchy data?
- **Why:**
  - What are the negative consequences of hierarchy mismatches (payroll issues, reporting errors, compliance problems, etc.)?

## 2. Various Solutions with Implementation Details

- **Option 1: Enhanced Input Validation & Controls**
  - **Focus:** Preventing mismatches at the source.
  - **Implementation:**
    - Enforce stricter data entry rules (e.g., dropdowns, lookups).
    - Cross-reference new hierarchy entries with existing data for conflicts.
    - Implement approval workflows for hierarchy changes.
- **Option 2: Scheduled Reconciliation Script**
  - **Focus:** Regular identification and semi-automated correction.
  - **Implementation:**
    - Develop a script (PL/SQL, Python, etc.) to compare hierarchies, flag inconsistencies, and possibly provide fix suggestions.
    - Run this script on a schedule (nightly, weekly).
- **Option 3: RPA Bot**
  - **Focus:** Automating manual correction process.
  - **Implementation:**
    - Analyze current correction steps; identify repetitive, rules-based actions.
    - Use an RPA tool (UiPath, Blue Prism, Automation Anywhere) to build a bot that mimics user actions within the Oracle EBS interface.
- **Option 4: Integration-based Synchronization**
  - **Focus:** Leveraging a master data source.
  - **Implementation:**
    - If there's a reliable HR system acting as the "source of truth," develop integration to push updates to Oracle EBS.

# SOP Automated Notifications during Payroll Process

## 3. Script-Based Automation vs. Bots/Tools

- **Scripts:** Ideal for well-defined logic, direct database interactions, and scheduled tasks within the Oracle EBS environment.
- **RPA Bots:** Better for mimicking user interactions across multiple systems, especially when the process has clear, click-through steps.
- **Specialized Tools:** There might be off-the-shelf hierarchy management tools designed to integrate with Oracle EBS; this requires research.

## 4. Proposed Benefit of Automation

- **Reduced Errors:** Minimize impact of hierarchy mismatches.
- **Improved Efficiency:** Free up time spent on manual identification and correction.
- **Consistency:** Enforce a standardized approach to hierarchy maintenance.
- **Compliance (Potential):** Aid in compliance efforts for accurate reporting.

## 5. Type & Quantification of Savings

- **Time Savings:** Estimate hours currently spent on manual activities, then calculate the percentage reduction achievable through automation.
- **Cost Savings:** Translate time savings into labour costs.
- **Error Reduction:** If mismatches lead to costly rework (e.g., inaccurate payroll), try to quantify that impact for a potential saving area.

## 6. Additional Remarks for Prioritization

- **Criticality:** How severely do hierarchy mismatches disrupt payroll or other core business processes?
- **Frequency:** How often do errors occur? Daily, weekly, etc.
- **Complexity:** Are mismatches generally simple fixes, or do they involve intricate changes?
- **Scalability:** Will the volume of hierarchy changes increase as the organization grows?
- **Opportunity Cost:** What other high-value tasks could the team focus on if this was automated?

## 7. Industry Usage Data

- **Specificity:** This use case is niche – hierarchy correction *within Oracle EBS*. General automation data won't be as helpful.
- **Sources:**
  - Consider vendor case studies (Oracle might have some).
  - Search for industry reports on HR automation or Oracle-specific process automation trends.
  - Professional communities focused on Oracle EBS or HR operations may offer insights.

# SOP Automated Notifications during Payroll Process

- **8. Implementation Time & Benefits Realization**
- **Factors:** Implementation time depends on the chosen solution's complexity, development resources, and the degree of change to current processes.
- **Range:** It could vary from a couple of weeks (for a simple script) to a few months (for a more complex RPA bot or integration).
- **Benefits:** You'll likely see immediate error reduction. Time savings accrue gradually as the automation stabilizes.
- **9. Dependencies and Downsides**
- **Dependencies:**
  - Access to Oracle EBS with necessary permissions.
  - Understanding of Oracle EBS data structures for hierarchies.
  - Development tools if creating scripts/bots.
  - Cooperation from IT/HR teams regarding changing processes.
- **Downsides:**
  - Automation may not catch 100% of mismatches if the error patterns are complex.
  - Upfront investment of time for development.
  - Requires ongoing maintenance if business rules or the Oracle system changes.
- **10. Metrics & Benchmarks**
- **Outcome Metrics**
  - Number of hierarchy mismatches detected before they impact downstream processes.
  - Time spent on manual correction (before vs. after).
  - Perhaps reduced payroll rework costs (if applicable).

# Automated Year end pending transaction approval

- **Caveat: Auto-Approving Transactions is a Risky Area**
- **Compliance:** Depending on the nature of these transactions, there could be audit, financial control, or industry-specific regulations that require human approval.
- **Downstream Impacts:** Can auto-approved transactions cause issues in other Oracle modules or integrations? A thorough understanding of the full process flow is crucial.

## 1. Details

1. **Type of Transactions:** (Financial, inventory, etc.) This heavily influences risk.
2. **Pending State:** How does a transaction become 'pending'? (User input, system logic, etc.)
3. **Approval Criteria:** Is it *only* time-based (year-end), or are there other factors?
4. **Existing Workflow:** Is there any approval mechanism within EBS now, even if manual?

## 2. Solutions (Hypothetical)

### 1. Option 1: EBS Workflow Modification (if feasible)

1. If there's an existing workflow mechanism, could it be modified with a time-based rule to conditionally auto-approve certain transactions?
2. **Huge Caveat:** This assumes there are safeguards to ensure only the right transactions get auto-approved.

### 2. Option 2: Scheduled Script + Logic

1. A script running at year-end to query 'pending' transactions from EBS.
2. **Crucial:** Robust logic built-in to apply the exact approval criteria before updating the transaction status.

## Script vs. Tool:

Likely a script (interacting with EBS APIs ), potentially triggered by a workflow tool if one is in place.

## Benefits (if done responsibly)

1. Time saving: Eliminates manual year-end approvals
2. Potential error reduction: If manual approvals are prone to mistakes under pressure.

## Savings:

1. Purely administrative time saved.
2. Might reduce potential errors caused by rushed approvals.

**Quantification:** Requires baseline – how long is spent on approvals *now* during year-end.

- **Prioritization:** Assess the risk vs. the time-saving benefit. High-risk transactions make this automation less appealing.
- **Industry Data:** Unlikely to find specific benchmarks due to the risk-sensitive nature of this.
- **Implementation Time:** Depends on EBS customization ease but could be lengthy due to testing needed.
- **Dependencies:**
  - Deep EBS knowledge to ensure only correct transactions are modified.
  - Potentially approval from finance/audit stakeholders if regulations are involved.

# Automated Year end pending transaction approval

## Metrics:

- Time saved
  - Error reduction (if a problem currently)
  - **Crucial:** Increased audit scrutiny to ensure auto-approvals caused no issues.
- **Recommendation**
1. **Process Mapping:** Meticulously document the transaction lifecycle: creation, pending state, current approval, downstream impacts.
  2. **Risk Assessment:** Consult stakeholders (finance, compliance) to see if automation is even permissible for the transactions in question.
  3. **Seek Alternatives:** Could the *need* for year-end approvals be reduced by process changes upstream? This is a safer area to optimize.

## • Outlining a Process Map

- Here's a framework to get you started. Fill in the specifics based on your Oracle EBS implementation and your organization's procedures.

### 1. Process Trigger:

1. What initiates a transaction? (User entry, data feed, another system)
2. Specifically, what leads to a transaction ending up in a 'pending' state?

### 2. Transaction Flow (Normal Path):

1. Step-by-step, how does a transaction ideally go from creation to approval?
2. Include roles responsible (developer, business user, approver, etc.).
3. Systems involved (just EBS, or others feeding data in?)

### 3. Approval Criteria:

1. Forget about year-end for a moment. What are the *normal* rules for approving a transaction?
2. Are there different criteria based on transaction type, amount, or other data?

### 4. Bottlenecks:

1. Where do transactions get stuck? Is it awaiting POC/HLE completion? The formal approval step itself?

# Automated Year end pending transaction approval



- **Example (Hypothetical –Version)**
- **Trigger:** User submits a inventory adjustment request in EBS
- **Flow:**
  - System validates if the user has permissions to adjust inventory of that type
  - If over a certain value threshold, the transaction goes into 'Pending' state.
  - Developer prepares POC if needed (timeline varies based on complexity)
  - HLE estimates generated (by developer or a separate role)
  - Application owner signs off (currently via email and manually marked in EBS)
  - Transaction status changes to 'Approved,' and inventory is adjusted.
- **Alternative Ways to Streamline**
- **Upstream Fixes:**
  - Are transactions frequently pending due to slow POC/HLE generation? Can that part of the process be optimized on its own?
  - Could better EBS permissions reduce the need for certain approvals in the first place?
- **Partial Automation:**
  - A notification system to alert the right people when transactions are pending approval. This cuts down on delays from lack of visibility.
  - A central repository for POC and HLE documents, linked to the pending transaction within EBS for easier access.
- **Criteria Rethink:**
  - Is the year-end rush caused by an arbitrary rule? If possible, could approvals be spread throughout the year? This lessens the need for risky auto-approval.
- **Important Considerations**
- **Change Management:** Process changes often need more buy-in and training than technical automation.
- **Iterate:** Start by mapping the current process. This alone often highlights areas for improvement.

# Auto course completion marker

- **Understanding the Current Process**
- **Types of Issues:** What are the common employee data discrepancies that cause downstream problems? (Missing data, format errors, etc.)
- **Upstream Fix:** What's the manual process like when you have to work with the OHCM team? (Tickets, emails, specific system they use).
- **Detection:** How do you *know* there's an issue? (MBS errors, reports, etc.)
- **Degree of Automation Potential ?**

## 1. Details:

1. **What:** The specific errors, how you spot them now.
2. **How:** Manual process of fixing upstream
3. **Where:** Systems involved (OHCM, ODS, MDM, MBS).
4. **Why:** Impacts caused by the employee data issues.

## 1. Solutions (Potential):

### 1. Option 1: Proactive Checking + Alerts

1. Script/tool to compare employee data in MBS against its 'origin' in OHCM.
2. Generates alerts (email, or ideally a ticket) when discrepancies are detected.

### 2. Option 2: Partial Upstream Fix

1. If some errors are common/systemic, can the ODS/MDM stages apply corrections *before* data reaches MBS?

### 3. Option 3: True Upstream Fix

1. If the root cause is how OHCM feeds data, addressing it at the source is ideal but might be complex.

## 2. Script vs. Tool:

1. likely script-based, pulling data via APIs (if available) from OHCM, ODS, etc.
2. A ticketing system or notification tool would be beneficial for the alerting side.

## 3. Benefits:

1. Faster issue resolution: Proactive finding vs. errors surfacing much later.
2. Reduced downstream impact: Problems caused by bad employee data are minimized.
3. Potential for reduced manual effort, depending on how fixes are done now.

# Auto course completion marker

## Savings:

1. Time spent fixing errors downstream due to data issues.
2. Harder to quantify, but impact reduction (fewer errors for end-users) has value too.

## Quantification:

1. Requires baseline: How many errors of this type occur per week/month?
2. How long does each one take to fix, with all the back and forth involved?

## Prioritization: How disruptive are these employee data errors currently?

- **Usage in Industry:** Best found through Oracle HCM communities, as it's data flow specific.
- **Implementation Time:** Depends on API availability, might be quite fast if APIs are robust.
- **Dependencies:**
  - Read access at least to all the stages (OHCM, ODS, MDM, MBS)
  - A way to send alerts or create tickets from your automation.

## Metrics:

- Reduction in errors
- Time taken from detection of mismatch to resolution.

- **Integration Ecosystem:** Tools able to talk to Oracle HCM and your chosen notification method are key.

- **Note:** Automating the actual fix to employee data is *much riskier*, as you'd be writing back to the source of truth. Focus on faster detection/alerting is relatively better approach.

## Next Steps

1. **Error Catalog:** Document the top 3-5 common employee data issues, how they cause problems, and how long they take to fix now.
2. **API Check:** Find out if OHCM, ODS, and MDM have APIs allowing you to read employee data.



# FA Data Corruption

- **Understanding the Problem: FA Data Corruption**
- **Types of Corruption:**
  - Specific examples are crucial: Incorrect balances, missing transactions, invalid journal entries, etc.
  - Knowing the types of corruption will inform the potential detection mechanisms.
- **Root Causes (if known):**
  - Manual entry errors? Bugs in custom FA code? Upstream data feed issues?
  - Understanding the root cause will help determine if prevention vs. reactive fixing is more feasible.
- **MACM Process:**
  - Is data correction done via the UI, or does FA have APIs / import mechanisms for mass fixes?
  - This informs whether automation has to mimic user actions or can interact directly with FA's data structures.
- **Automation Feasibility Assessment**

## 1. Details:

1. **What:** The precise forms of data corruption
2. **How:** Root causes (if known), current MACM process.
3. **Where:** Purely within the FA module, or does it stem from issues upstream?
4. **Why:** Business impact – incorrect financials, audit issues, etc.

## 1. Potential Solutions:

### 1. Option 1: Proactive Data Validation

1. **Logic:** Build checks against what "good" FA data should look like (balance rules, mandatory fields, etc.).
2. **Implementation:** SQL/PLSQL scripts on a schedule, or integrated into FA processes if possible.
3. **Goal:** Catch corruption as it arises, not after it's done widespread damage.

### 2. Option 2: Root Cause Fix

1. **If preventable:** Address data issues *before* they reach FA (upstream fixes, better input validation).
2. **High Difficulty:** Likely involves changes outside of just FA.

### 3. Option 3: MACM Automation (If feasible)

1. **Logic:** Replicate the manual correction steps in a script.
2. **Implementation:** API interaction with FA is ideal, UI automation (RPA) if that's the only way.

**Script vs. Tool:** Depends on the chosen solution:

1. **Validation:** Likely SQL/PLSQL scripts triggered by EBS scheduler
2. **MACM automation:** API interaction if possible, RPA as last resort.

# FA Data Corruption

## Benefits:

1. **Faster Correction:** Reduces time spent on manual fixes.
2. **Error Reduction:** Proactive validation can catch things humans might miss.
3. **Consistency:** Automated MACM fixes are less prone to typos than human ones.

## Savings:

1. **Time:** Requires baseline of how long fixes take manually now.
2. **Impact Reduction:** Depending on severity of corruption, there might be significant business impact savings (avoiding restated financials, etc.).

## Quantification

1. **Frequency:** How often does corruption occur? Weekly? Monthly?
  2. **Severity:** Average time to fix, potential business impact if not fixed.
- **Prioritization:** How critical are FA issues to your business operations?
  - **Usage in Industry:** Best found in FA user groups, as it depends on EBS customization level.
  - **Implementation Time:** Hard to estimate without knowing EBS complexity, but validation scripts can be relatively quick to deploy.

## Dependencies:

- **API Access:** For MACM automation.
- **Knowledge:** Deep understanding of both FA and how data is "supposed" to look.

## Metrics:

- Issue Count (reduced over time if validation is effective).
- Fix Time: Before/after automation

- **Caveat:** Automating FA corrections is risky if corruption root causes aren't addressed. You could be automating the fixing of symptoms, not the problem itself.

## Recommendations

1. **Prioritize Problem Mapping:** Gather examples of corruption and MACM fix patterns.
2. **Seek Root Cause:** Collaborate with FA users to see if errors originate upstream.
3. **Start with Validation:** Focus on preventing as much corruption as possible through proactive scripts.
4. **MACM Automation:** Evaluate later and only if APIs exist – riskiest area.

# Uncleared CIP Costs

- Automating the identification of uncleared CIP costs during project capitalization in Oracle EBS.
- **Understanding the Problem: Uncleared CIP Costs**
- **Various Reasons :**
  - We need specifics. Bad data feeds, interface errors, custom process quirks, etc.
  - Knowing the common causes narrows down where to look for fixes.
- **Current Detection:**
  - How do you find out about uncleared costs now? Reports? User complaints? This determines how automation can replicate the detection.
- **Solutioning:**
  - Is the "solution" always the same type of fix? Or does it vary based on the reason for the cost not clearing?
- **Automation Feasibility Assessment**

## 1. Details:

1. **What:** The precise forms uncleared costs take, and the common reasons for them.
2. **How:** How the issue is discovered, and the steps of the solution.
3. **Where:** Purely within the Projects Module, or are other modules/feeds involved?
4. **Why:** Impact: incorrect reporting, delayed project closure, etc.

## 1. Potential Solutions:

### 1. Option 1: Proactive Report + Troubleshooting Guide

1. **Logic:** A report pinpointing uncleared costs, categorized by their likely root cause (based on your historical data).
2. **Implementation:** SQL report scheduled in EBS. Can link to a troubleshooting guide document.
3. **Goal:** Not pure automation, but makes the manual fixing faster.

### 2. Option 2: Root Cause Fix

1. **If preventable:** Address issues upstream that lead to costs not clearing.
2. **High Difficulty:** Likely involves changes outside of just the Projects Module.

### 3. Option 3: Automation of the Fix (IF standard)

1. **Logic:** Script replicates your usual fix process IF the solution is mostly the same each time.
2. **Implementation:** API interaction with Project module is ideal, UI automation (RPA) if not possible another way.

## 2. Script vs. Tool: Depends on the chosen solution:

1. **Report:** Likely SQL within EBS's reporting tools.
2. **Fix Automation:** API-based scripts if possible, RPA if UI interaction is the only option.

# Uncleared CIP Costs

## Benefits:

1. **Faster Issue Identification:** Proactive reports mean less delay
2. **Reduced Manual Effort:** If a standard fix can be automated.
3. **Improved Accuracy:** If fix automation is feasible, it eliminates human error potential on repetitive tasks.

## Savings:

1. **Time:** Requires baseline: time spent finding AND fixing issues now.
2. **Impact Reduction:** Depending on severity of uncleared costs, there might be other business impact savings (avoiding misstated project financials).

## Quantification

1. **Frequency:** How often does this issue occur?
  2. **Severity:** Average time to find and fix, potential business impact.
- **Prioritization:** How disruptive are uncleared costs to your project reporting/closure process?
  - **Usage in Industry:** Best found in Oracle Projects communities, as it depends on customization level.

- **Implementation Time:** Hard to estimate without knowing your EBS complexity and data volumes involved.
- **Dependencies:**
  - **API Access:** For fix automation (if feasible)
  - **Knowledge:** Understanding project cost flow and the common 'failure points' causing uncleared costs.
- **Metrics:**
  - Time between issue arising and being detected.
  - Time spent on fixing.
  - Any impact-related metrics (if delayed project closure is the cost, etc.)
- **Recommendations**
  1. **Map the Problem:** Detailed examples of uncleared costs, how they're found, and how they're fixed currently.
  2. **Analyze Root Causes:** Are preventable patterns in the error types?
  3. **Start with Report + Guide:** Focus on faster detection and aiding the usual solution to provide immediate value.
  4. **Fix Automation: Evaluate later and only if the solution itself is highly standardized.**



# Uncleared CIP Costs

- **Template: Uncleared CIP Cost Investigation**
- **Section 1: The Problem**
- **Specific Examples:** Provide 2-3 recent instances where costs failed to clear during capitalization. Include:
  - Project ID, if possible
  - Amount uncleared
  - Any error messages or clues about why it didn't clear
- **Frequency:** Estimate how often this occurs:
  - Weekly? Per major project closure?
- **Impact:** What happens due to the uncleared costs?
  - Incorrect project financials
  - Delayed reporting to management
  - Audit findings (if severe)
- **Section 2: Current Detection & Resolution**
- **Discovery:** How do you know there are uncleared costs?
  - Specific reports
  - User complaints
  - Other ad-hoc methods
- **Troubleshooting:** What are the first steps taken when an uncleared cost is found?
  - Who investigates (role: accountant, project manager, etc.)?
  - What systems/data do they look at to find the root cause?
- **The Fix:**
  - Is there a standard solution most of the time (e.g., data entry correction, retry a process)?
  - Or, does the fix vary drastically based on the root cause?
- **Section 3: Root Cause Potential**
- **Known Culprits:** Are there any usual suspects when this happens?
  - Issues in a data feed from an external system
  - Certain types of costs always problematic
  - Custom processes within your EBS Projects module
- **Upstream vs. In-Module:** Do you suspect the problem starts *before* data enters the Projects module, or is the issue within how Projects itself handles things?
- **Additional Notes**
- **Data Access:** Get sample data (anonymized if needed) from uncleared cost situations? This will be extremely helpful for designing the report or automation fix logic.
- **How to Use This Template**
- 1. **Involve Key People:** Collaborate with those who usually fix uncleared costs to fill this out.
- 2. **Review for Patterns:** Are most fixes the same? This points to potential automation. Lots of variation in the fix makes automation harder.
- 3. **Look Upstream:** Is the root cause often outside the Projects module? Automation might be less of a solution there.

# User – Role Assignment

## Automation Feasibility Assessment

### Details:

1. **What:** The process of both creating a role AND assigning it to a user.
2. **How:** Document the current steps meticulously, who performs them, etc.
3. **Where:** Purely within Oracle EBS user management?
4. **Why:** Main pain point – is it time-consuming, prone to error, etc.?

### Potential Solutions:

1. **Option 1: Templating & Scripting**
  1. **If:** You have standard role types (with minor variations).
  2. **Implementation:** Scripts to create roles based on templates, assignment scripts based on user attributes.
2. **Option 2: Self-Service with Approval**
  1. **If:** Users could reasonably select their needed roles from a predefined list.
  2. **Implementation:** Custom EBS form, workflow for approval, then a script to grant the roles.
3. **Option 3: Integration with Identity Management**
  1. **If:** EBS is integrated with an IDM system, some provisioning might exist there already.

### Script vs. Tool: Depends on the chosen solution:

1. **Templating/Assignment:** Likely SQL/PLSQL, EBS administration tools if they support batch operations.
2. **Self-Service Form:** EBS customization capabilities.
3. **IDM:** Depends on your IDM system.

### Benefits:

1. **Time Savings:** If the manual process is time-consuming.
2. **Error Reduction:** Eliminates typos, role mis-assignments, etc.
3. **Auditability:** If you need to track role assignments for compliance, automation improves this.

### Savings:

1. **Requires Baseline:** How long does the FULL process take per user now (creation + assignment).

# User – Role Assignment

## Quantification

- 1. **Frequency:** How many role creations/assignments weekly/monthly?
  2. **Time Per Task:** Estimate for creating a new role, assigning a role.
- **Prioritization:** How burdensome is user role setup vs. other admin tasks?
- **Usage in Industry:** Best found in Oracle EBS admin communities, as it depends on customization levels.
- **Implementation Time:** Can range from quick scripting to longer if EBS forms are needed.
- **Dependencies:**
  - **Scripting Permissions:** If that's the route, scripting access on the EBS database is needed.
  - **IDM:** If you have an existing system, its capabilities are crucial.
- **Metrics:**
  - Task Time (before/after automation)
  - Error Reduction (if a current issue)
- **Caveat:** Role assignment automation is most useful with clear guidelines on who gets what roles. Overly complex determination logic makes it harder.
- **Recommendations**
  1. **Process Mapping:** Document your current role creation and assignment process steps.
  2. **Analyze Standardization:** Are there consistent role patterns, or is every user unique?
  3. **Consider Security:** Automation makes changes easier – ensure safeguards align with your security policies.

# Automating file retrieval and placement.

- Automating file retrieval and placement for Oracle EBS environments.
- **Understanding the Problem: Manual File Transfers**
- **Ticket Details:**
  - What's a typical request? (e.g., "Get log file X from prod server, place updated config Y on test server").
  - How do tickets arrive (system, email, etc.)?
- **File Variety:**
  - Are these logs, config files, data extracts?
  - Are file locations and names known in advance, or specified in the request?
- **Bottlenecks:**
  - Is the task itself difficult (requiring knowledge of EBS structure), or is it just the time it takes?
- **Automation Feasibility Assessment**

## Details:

1. **What:** The types of requests (get vs. put), file types, locations.
2. **How:** Where requests come from, how files are moved now.
3. **Where:** Are the file locations on the EBS server itself, or external (FTP, etc.)
4. **Why:** Main pain point – volume of requests, error-proneness, etc.?

## Potential Solutions:

### 1. Option 1: Scripting + Ticketing Integration

1. **If:** Ticket system has an API, file locations are consistent.
2. **Implementation:** Scripts to get/put based on parameters, triggered by ticket updates.

### 2. Option 2: Secure File Portal

1. **If:** Mainly placing files for users to retrieve themselves.
2. **Implementation:** Web-based portal with folder structure, connected to EBS storage (careful security needed).

### 3. Option 3: RPA (If UI-heavy)

1. **If:** Your current process involves many clicks and manual file movement.
2. **Implementation:** Bot to mimic human actions for file transfers

## Script vs. Tool: Depends on the solution:

1. **Scripting + API:** Scripting language suitable for your environment (Bash, Python, PowerShell, etc.)
2. **File Portal:** Separate web app, or see if your EBS environment has any suitable components.
3. **RPA:** RPA tool with robust UI interaction capabilities.



# Automating file retrieval and placement .

## 1. Benefits:

1. **Massive Time Saving:** Each file transfer becomes seconds long if automated.
2. **Error Reduction:** No more typos, files going to the wrong place, etc.
3. **Scalability:** Automation handles many requests without burdening IT staff.

## 2. Savings:

1. **Requires Baseline:** How long does an average file request take now, from ticket arrival to completion.

## 3. Quantification

1. **Frequency:** How many file transfer requests on a weekly/monthly basis?
2. **Time Per Task:** Estimate for getting a file, placing a file.

## • Additional Considerations

- **Prioritization:** How disruptive is file transfer volume to your EBS admin team's other tasks?
- **Usage in Industry:** Commonly automated – seek file transfer automation examples, but your EBS setup is unique.

- **Implementation Time:** Scripting can be fast if locations are standard, a portal takes more setup.

## • Dependencies:

- **Ticketing API:** If using that for automation.
- **Scripting Permissions:** On EBS servers or wherever the files reside.
- **Security:** HUGE focus needs to be on securing a file portal if that's the route.

## • Metrics:

- Task Time (before/after automation)
- Errors (before/after automation)

- **Security Emphasis** – Any automation that moves files between EBS environments needs careful review, especially if it involves external access

## • Recommendations

1. **Ticket and File Pattern Analysis:** Review recent requests, look for commonalities to aid automation design.
  2. **Security Assessment:** Consult security team on best options given your EBS deployment.
  3. **Proof of Concept:** Start with a script to automate the most common request type to demonstrate the potential.
- **Next Steps :**
  - Script logic examples , Exploring RPA tools if applicable , Reviewing security considerations for file movement automation



# Automating file retrieval and placement - Approach

- **1. Script Logic Examples**

- **Operating System:** Where your Oracle EBS environments are hosted (Linux, Windows, etc.)
- **Typical File Tasks:**
  - File retrieval: Are we using commands like scp, ftp, or OS-specific file copy tools?
  - File placement: Are we using similar remote transfer commands, or direct file manipulation on a shared drive?
- **File Location Structure:**
  - Do files reside in predictable paths based on their type? Example: Logs always under /app/oracle/prod/logs/
- **Example Scenario (Illustrative – Adjust To Your Setup)**
- Linux environments
- File retrieval is common, placement less so.
- Standard log location structure
- A basic Python script snippet:
- This assumes SSH access to your servers. Adapt if you use FTP or other protocols.
- ```
import os
import paramiko # For remote file transfers via SSH
def retrieve_file(server, username, password, remote_path, local_path):
    """Retrieves a file from a remote server via SCP"""
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=server, username=username, password=password)
    sftp = ssh_client.open_sftp()
    sftp.get(remote_path, local_path)
    sftp.close()
    ssh_client.close()
# Example Usage: Assuming ticketing system provides parameters
server_ip = ticket_data['server']
file_to_get = ticket_data['filename']
# ... similar extraction of other parameters
retrieve_file(server_ip, 'ebs_user', 'password', f'/app/oracle/{server_ip}/logs/{file_to_get}', '/local/download/path/')
```

- **2. Exploring RPA Tools**

- If your current file transfer is UI-centric, here are popular RPA tools to research:
- **UiPath:** Market leader, robust capabilities,
- **Automation Anywhere:** Strong in both UI interactions and wider process automation.
- **Blue Prism:** Well-suited for enterprise-scale automation.
- **Evaluation Points for RPA:**
  - Ease of capturing UI actions for your specific file transfer steps.
  - Integration potential (if needed) with your ticketing system.
  - Licensing costs (RPA can be more expensive than scripting if complexity is low).

- **3. Reviewing Security Considerations**

- **File Portal:**
  - Strict authentication and authorization controls
  - Input validation to prevent malicious uploads (if it's a two-way portal)
- **Scripting:**
  - Limit script permissions to *only* needed file paths and actions
  - Ticket data validation: Prevent scripts from being tricked into acting on malicious requests
- **Overall:**
  - File transfer logging and auditing – who moved what and when.

# Manual SLA Breach Tracking

- Automate the generation of a report highlighting upcoming SLA breaches on incidents (INC), problems (Problem), and tasks (PTask) within Oracle EBS.
- **Understanding the Problem: Manual SLA Breach Tracking**
- **SLA Definitions:**
  - How are SLAs calculated for each ticket type? (time from creation, specific response time targets, etc.)
  - Are SLAs the same for all INC/Problem/PTask types, or do they vary?
- **Current Reporting:**
  - How is breach tracking done now? Manual report creation? Pain points of the current method?
- **Ticket Data Source:**
  - Is ticket data (creation time, deadlines) stored within EBS itself, or an external helpdesk system?

## Automation Feasibility Assessment

### 1. Details:

1. **What:** SLAs, how they differ per ticket type, how breaches are currently found.
2. **How:** Where your ticket data resides (EBS or external).
3. **Where:** Purely data/reporting issue, or do the SLAs depend on other EBS modules?
4. **Why:** Main reason for automation – time-saving, proactive issue spotting, etc.?

### 1. Potential Solutions:

#### 1. Option 1: EBS-Native Reporting

1. **If:** EBS has customizable reports and your ticket data is fully within EBS.
2. **Implementation:** SQL-based reports scheduled in EBS, calculating breaches based on SLAs.

#### 2. Option 2: External Ticketing Integration

1. **If:** Helpdesk system has its own reporting/alerting on SLAs
2. **Implementation:** API calls from a script to fetch data, or leveraging their built-in reporting.

#### 3. Option 3: Hybrid (If Data Split)

1. **If:** Core ticket data is external, but maybe some elements are tracked in EBS.
2. **Implementation:** Scripts to combine data, report generation can be in EBS or external depending on complexity.

# Manual SLA Breach Tracking

**Script vs. Tool:** Depends on the chosen solution:

1. **EBS Reporting:** SQL within EBS's report builder.
2. **External or Hybrid:** Scripting language to fetch data from APIs, format the report (Python, etc.)

**Benefits:**

1. **Proactive:** Prevent breaches instead of being reactive
2. **Time Savings:** If manual reporting is cumbersome now.
3. **Consistency:** Automated reports avoid human error in calculations.

**Savings:**

1. **Baseline:** How long does manual breach reporting take now?

**Quantification**

1. **Frequency:** How often is breach reporting done currently?
  2. **Impact:** What happens when there's a breach? Cost to the business, customer impact, etc.
- **Prioritization:** How critical is it to prevent SLA misses?
  - **Usage in Industry:** Ticketing SLA automation is very standard, specifics depend on your system.

- **Implementation Time:** EBS-only reports can be quick, more complex if pulling external data.

• **Dependencies:**

- **Report Permissions:** In EBS if that's the route.
- **API Access:** To your helpdesk system if applicable.

• **Metrics:**

- Time to create the report (before/after)
- Reduction in breaches (this is the big win if automation prevents them!)

• **Recommendations**

1. **Source of Truth:** Determine where the core ticket data + SLA logic lives (fully in EBS, or external?).
2. **Current Process Map:** How do you find breaches now, even if it's manual? This helps to automate.
3. **Explore EBS Reporting:** Analyze if it can handle the necessary SLA calculations based on your needs.



# Manual SLA Breach Tracking

## Drafting Sample SQL Logic (if feasible within EBS)

- **Database Structure:**
  - Table names and columns where these are stored:
    - Incident/Problem/PTask creation timestamps
    - Any fields that track SLA targets (target response time in hours, etc.)
    - Relevant status fields (open, pending, etc., if that impacts breach calculation)
- **SLA Rules:**
  - A clear breakdown for at least one ticket type (e.g., INC):
    - Is the SLA purely based on time elapsed since creation?
    - Are there different targets depending on priority, category, etc.?
- **Example Scenario (Illustrative – Adjust To Your Setup)**
- Assuming a simplified structure:
- Table: helpdesk\_tickets
- Columns:
  - ticket\_id
  - ticket\_type('INC', 'Problem', 'PTask')
  - created\_timestamp
  - target\_response\_hours
- Simple SLA: Target response time is added to created\_timestamp to get the breach deadline
- Sample SQL (may need extension based on your real SLA logic):
- SQL
- ```
SELECT ticket_id, ticket_type, created_timestamp, target_response_hours,
DATE_ADD(created_timestamp, INTERVAL target_response_hours HOUR) AS
'breach_deadline' FROM helpdesk_tickets WHERE ticket_status = 'Open' -- Assuming only
open tickets are at-risk AND NOW() > breach_deadline;
```

## 2. Outlining Potential API Calls

- If your ticket data resides in an external helpdesk system.
- **API Documentation:** Your helpdesk system's API references for retrieving ticket data.
- **Essential Data Points:**
  - Endpoints to fetch tickets with their creation dates, status, and any SLA-related fields
- **Authentication:** How API calls are authenticated (API key, OAuth, etc.)
- **Example (Illustrative, Highly Dependent on Your Helpdesk API)**
- Using a fictional /tickets endpoint, assuming simple API key authentication:
- Python
- ```
import requests api_url = 'https://yourhelpdesk.com/api/v1/tickets' headers = {'Authorization':
'Your_API_Key'} # Parameters for filtering, may need adjustment params = { 'status': 'open',
'fields': 'id, created_at, sla_target' } response = requests.get(api_url, headers=headers,
params=params) if response.status_code == 200: ticket_data = response.json() # ... Further
processing to calculate breaches else: # Handle API errors
```

## 3. Suggesting Reporting Formats

- Here are a few ideas, but the best format depends on how the report will be used:
- **Simple List:** If the goal is to quickly see at-risk tickets:
  - Ticket ID, Ticket Type, Time Created, Target Deadline, Time Remaining
- **Dashboard Style:** If visualization is helpful:
  - Counts of breaches by ticket type
  - Time-to-breach countdown for the most urgent tickets.
- **Email Alert:** If immediate notification is needed:
  - Customizable email template listing tickets about to breach.

# Manual Assignment Bottleneck

Automate ticket assignment for PTasks and Problem tickets to prevent SLA breaches in your Oracle EBS environment.

## Understanding the Problem: Manual Assignment Bottleneck

- **Current Workflow:** Describe the steps involved from when a PTask/Problem ticket arrives to the L15 queue until it *should* be assigned?
- **Assignment Logic:** Is there a standard rule for how these tickets get assigned (round-robin, based on team member skillset, etc.), or is it ad-hoc?

## Automation Feasibility Assessment

### Details:

1. **What:** The current ticket arrival and assignment process.
2. **How:** Where tickets originate (EBS module, external system) and how they reach the L15 queue.
3. **Where:** Is this purely within an EBS ticketing module, or is an external helpdesk system involved?
4. **Why:** Main consequence of unassigned tickets (SLA breach is clear, but are there others?)

## Potential Solutions:

### 1. Option 1: EBS Workflow Enhancement

1. **If:** Tickets live within EBS and there's workflow capability.
2. **Implementation:** Modify the workflow to auto-assign on certain conditions (ticket type, etc.).

### 2. Option 2: Scheduled Script

1. **If:** Ticket data is accessible via EBS database tables.
2. **Implementation:** Script periodically checks for unassigned PTask/Problem tickets and applies assignment logic.

### 3. Option 3: External System Integration

1. **If:** Your ticketing is primarily external with EBS sync.
2. **Implementation:** May need changes on the external system side if it supports auto-assignment.

## Script vs. Tool:

1. **Workflow:** If EBS natively supports it.
2. **Script:** If assignment logic needs to be coded and run on a schedule.

# Manual Assignment Bottleneck

## Benefits:

1. **SLA Prevention:** Proactive breach avoidance is the major win.
2. **Faster Response:** Tickets get worked on sooner, improving customer experience,
3. **L15 Burden Reduction:** Less manual tracking of the queue needed.

## Savings:

1. **Baseline:** Time L15 currently spends checking and assigning tickets.
2. **Impact:** Cost of breached SLAs (reputational, or even direct penalties if those exist).

## Quantification

1. **Frequency:** How many PTask/Problem tickets arrive in the queue per day on average?
  2. **SLA Miss Rate:** How often do these tickets breach SLA due to unassigned status?
  3. **Additional Considerations**
- **Prioritization:** Quantifying the cost of SLA breaches will help make the case for automation.
  - **Usage in Industry:** Auto-assignment is standard helpdesk practice; your specific logic depends on your setup.
  - **Implementation Time:** EBS workflow changes might be fastest, a script is moderately complex.

## Dependencies:

- **Workflow Modification Permissions:** If that's the route.
- **Database Access:** For the script option, read/write access to ticket tables.

## Metrics:

- Time spent on assignment (before/after)
- **Crucial:** SLA breaches reduced (ideally to zero!)

- **Downside:** If assignment logic is complex, it might make the automation harder to maintain.

## Recommendations

1. **System Check:** Is ticket management fully within an EBS module, or is there external system integration?
2. **Assignment Rule Exploration:** Define a strategy for auto-assignment work. This is crucial to design the solution.
3. **Workflow Capability:** Check if your EBS module allows workflow rules sophisticated enough for the assignment logic.



# Manual Assignment Bottleneck - Approach

- **1. Workflow Design (If EBS Supports It)**

- If the ticketing system is a module within EBS and has workflow customization capabilities
- **Workflow Mapping:** Prepare diagramming the steps of the workflow as it should function with auto-assignment. This includes decision points (based on ticket type, etc.).
- **Workflow Rule Translation:** Translate the assignment logic into the specific rule syntax the EBS module uses for workflows.
- **Testing:** Identify the test scenarios to ensure the auto-assigned tickets flow as intended through your process.

- **2. Sample SQL (For Script-Based Assignment)**

- If auto-assignment requires a script to periodically query and update ticket data in EBS,
- **Database Schema Guidance:** Identify the exact EBS tables and columns where ticket data (type, status, assignment field) is stored.
- **SQL Logic Examples:** Create SQL queries to:
  - Find unassigned PTask and Problem tickets
  - Implement assignment logic (round-robin example, or more complex if needed)
  - Update the assignment field in the EBS database.
- **Scheduling:** Plan to schedule this script within your EBS environment (if it supports that).

- **3. Logic Review**

- **Assignment Rule Sanity Check:** Review your desired assignment logic (round-robin, skill-based, etc.) to spot potential issues that could lead to incorrect assignments down the line.
- **Bottleneck Anticipation:** Think through scenarios where the automation might cause new issues (e.g., everyone overloaded, and the logic fails). Proactive fixes are easier than troubleshooting later!
- **Edge Case Coverage:** Identify less-common scenarios your logic should account for to make the automation robust.
- Next Steps for detailed assessment :
- **Ticketing System:**
  - Is it a standard EBS module, a custom one, or an external helpdesk that syncs data to EBS?
- **Assignment Logic:**
  - In as much detail as possible, how do you want PTask and Problem tickets to be auto-assigned?
- **Workflow Capability:**
  - If using an EBS module, identify the level of workflow customization it allows.



# Mismatched Ownership Errors

- Automation potential for addressing transactions that fail due to mismatched ownership of locations within Oracle EBS.
  - **Understanding the Problem: Mismatched Ownership Errors**
  - **Transaction Types:** Identify the transactions that are prone to this error. (Inventory movements, orders, etc.). This will narrow down the data we need to look at.
  - **Ownership Fields:**
    - McOpCo, Conventional License, BFL.
    - Where is the "correct" ownership data maintained?
  - **Error Detection:** How are these mismatches currently found? User reports, logs, etc.?
  - **Automation Feasibility Assessment**
1. **Details:**
    1. **What:** The transactions impacted, ownership fields, how errors are found now
    2. **How:** How ownership data gets attached (manual, feeds), where the 'master' ownership info lives.
    3. **Where:** Purely within EBS, or are external systems feeding data?
    4. **Why:** Main consequence of these errors – blocked processes, incorrect data?

## 1. Potential Solutions:

### 1. Option 1: Proactive Validation

1. **Logic:** Rules to check ownership mismatches *before* transactions execute.
2. **Implementation:** SQL-based checks, potentially integrated into transaction processes (EBS forms, workflows?).

### 2. Option 2: Automated Correction (If Feasible)

1. **Logic:** Script to fix mismatches based on defined rules.
2. **Implementation:** SQL updates, but BIG caution is needed.

### 3. Option 3: Upstream Fix

1. **If Preventable:** Address the root cause of mismatches where ownership data originates.
2. **High Difficulty:** Likely involves changes outside of just EBS.

## 2. Script vs. Tool: Depends on the chosen solution:

1. **Validation:** Likely SQL integrated into EBS processes where possible.
2. **Correction:** Script if permissible, but needs extreme caution.

# Mismatched Ownership Errors

- **Benefits:**
  - **Error Prevention:** Transactions flow smoothly, no manual intervention after failure.
  - **Reduced Downstream Impacts:** Fewer incorrect inventory levels, orders stuck, etc., due to the ownership issue.
  - **Time Savings:** If fixing the errors now is a manual task.
- **Savings:**
  - **Requires Baseline:** How long it takes to troubleshoot and fix ONE ownership error currently.
- **Quantification**
  - **Frequency:** How many transactions fail due to this per week/month?
  - **Impact:** Downtime, or incorrect data caused by failures
- **Prioritization:** How disruptive are these errors relative to other EBS issues?
- **Usage in Industry:** Data consistency is important, specific logic depends on your setup.
- **Implementation Time:** Validation checks can be quick, automated correction is riskier to implement.
- **Dependencies:**
  - **Database Access:** For scripts
  - **Change Control:** Thorough, as this could impact live transactions.
- **Metrics:**
  - Transaction Failures Due to This (before/after)
  - Time to Resolution (before/after)
- **Caveat: Automated Correction is High-Risk**
- Fixing ownership on the fly could cause more problems if the 'correct' ownership isn't always obvious. Prioritize prevention if possible.
- **Consultant Recommendations**
  1. **Error Examples:** Gather 2-3 recent cases with transaction details, how the mismatch was found, and how it was fixed.
  2. **Ownership Master:** Confirm where the 'true' ownership data should come from.
  3. **Consider Prevention First:** Focus on stopping the mismatch from happening in the first place.
- **Next Steps:**
- **SQL Logic:** For validation checks
- **Workflow Integration:** If EBS supports it, to inject checks at the right points
- **Risk Analysis:** Especially for automated correction, mapping out the potential impact



# Mismatched Ownership Errors - Approach

- **SQL Logic for Validation Checks**
- **Database Structure:**
  - Relevant table and column names where the following are stored:
    - Transaction data (especially fields that would link to the location/ownership)
    - "McOpCo, Conventional License, BFL", etc. ownership values
    - The source of truth table where the correct ownership mappings are maintained
- **Example Transactions:** A couple of sample transactions (type, any relevant IDs) that *should* have succeeded, but failed due to ownership mismatch.
- **Potential SQL Logic (Illustrative – Needs MAJOR Adjustment to Your Setup)**
- Assuming tables like:
  - transactions (with columns item\_id, location\_id, transaction\_type)
  - location\_ownership (columns location\_id, mcopco\_code, license\_type, etc.)
- SQL
- ```
SELECT * FROM transactions t JOIN location_ownership l ON t.location_id = l.location_id WHERE t.transaction_type IN ('Inventory_Movement', 'Order_Placement', ...) -- Your relevant types AND ( l.mcopco_code != 'Expected_Value' OR l.license_type != 'Expected_Value' -- ... Add more mismatch checks as needed );
```

## Workflow Integration

- EBS modules allow workflow customization:
- **Identifying Integration Points:** Where in the transaction process it makes sense to inject the SQL validation check.
- **Triggering Logic:** How to initiate the check (before a user submits a transaction, etc.).
- **Error Handling:** Designing user-friendly error messages if the validation fails, and potentially offering solutions if a fix is simple.

## Risk Analysis for Automated Correction

- If automated correction is deemed necessary, focus on:
- **Mapping Dependencies:** What other EBS data relies on correct ownership? Changing it could have ripple effects.
- **Conditional Logic:** Rules for correction need to be extremely precise to avoid making things worse
- **Testing Plan:** A rigorous way to test the correction script in a staging environment before production is crucial

# EDI - Supplier Onboarding

- Automating the onboarding and testing of new EDI suppliers within your Oracle EBS environment. This has great potential to streamline a process that likely involves a lot of manual steps.
- **Understanding the Problem: Manual Supplier Onboarding**
- **Spec Variety:** For each "format" (positional, CSV, x12 etc.) have a strict specification document?
- **Testing:**
  - What's involved right now? (Sending sample files to suppliers, having them send test responses, etc.)
  - How is a test outcome determined (pass/fail)?
- **Production Move:** Is this manual configuration on the EBS side, or file transfers coordinated with the supplier?

## Automation Feasibility Assessment

### 1. Details:

1. **What:** The specs, the testing process, and how production move is done currently.
2. **How:** How suppliers are informed of the specs, how test files are exchanged.
3. **Where:** Is this purely file exchange and EBS config, or other systems involved?
4. **Why:** Main pain point – is it time-consuming, error-prone, etc.?

### 1. Potential Solutions:

#### 1. Option 1: Supplier Self-Service Portal

1. **If:** Consistent spec formats, and tests can be defined as rules.
2. **Implementation:** Web portal where suppliers upload files, validation runs automatically, results provided.

#### 2. Option 2: Automated Test Case Generation

1. **If:** Specs are machine-readable (even if conversion is needed).
2. **Implementation:** Scripts to turn spec into test data, potentially integrated with communication to the supplier.

#### 3. Option 3: Configuration Templating

1. **If:** Production move involves repetitive EBS setup per supplier.
2. **Implementation:** Scripts/tools to automate config based on supplier parameters.

### 2. Script vs. Tool: Depends on the chosen solution:

1. **Portal:** Web app development, likely separate from EBS itself.
2. **\*\*Testing Automation:** \*\* Scripts (Python, etc.) to handle file format logic, test case generation.
3. **Config:** EBS scripting (if it has capabilities), or external configuration management tools.

# EDI - Supplier Onboarding

## Benefits:

1. **Massive Time Savings:** If the current process is heavily manual.
2. **Error Reduction:** Automated tests and config lessen the chance of mistakes vs. manual setup.
3. **Faster Onboarding:** Suppliers integrated quicker = less friction in the supply chain.

**Scalability:** Automation handles many suppliers without proportional effort increase.

## Savings:

**Requires Baseline:** How long does onboarding ONE supplier take on average across all the steps now?

## Quantification

**Frequency:** How many new EDI suppliers added per month/year?

**Time Per Task:** Estimate for spec sharing, testing, production configuration.

- **Prioritization:** How burdensome is supplier onboarding vs. other EBS admin work?

- **Implementation Time:** A portal is the most complex, scripting likely faster.
- **Dependencies:**
  - **Spec Format:** If specs are unstructured docs, automation is harder.
  - **Scripting Permissions:** If EBS side automation is involved.
  - **External Comms:** If the automation needs to email suppliers, etc., that integration needs consideration.
- **Metrics:**
  - Onboarding Time (before/after)
  - Errors Reduced (if this is a problem currently)
- **Recommendations**
  1. **Spec Analysis:** Gather your spec documents for a few formats – Document their structure.
  2. **Test Process Mapping:** Describe the steps involved in testing right now, in detail.
  3. **Consider Security:** A supplier portal needs careful design, as it's an entry point into your systems.
- **Next Steps :**
  - **Test Logic Design:** Rules to turn specs into automated test cases.
  - **Portal Outline:** If that's the route, high-level architecture and security considerations.
  - **Scripting Examples:** For file validation, config generation (if feasible).

# EDI - Supplier Onboarding - Approach



## 1. Spec Analysis

- **Semi-Structured Specs:** Each format (positional, CSV, x12, etc.) has a defined specification, but it might exist as a combination of text documents and some machine-readable elements (XML schemas for x12, etc.).
- **Key Spec Elements:** Regardless of format, we can likely extract:
  - Mandatory fields
  - Data types and lengths
  - Valid values or code lists (if applicable)

## 2. Test Process Mapping

- **Current Test is File-Based:** Suppliers are provided sample good/bad files that match the spec, and they're expected to send response files for validation.
- **Manual Validation:** Someone on your team currently checks the supplier's response files against the spec to determine pass/fail.
- **Rudimentary Feedback:** Simply informing the supplier about pass/fail, with some error detail in case of failure.
- **3. Security**
- **Moderate Priority:** You recognize the need for a secure supplier portal if the automation involves that route. However, the EDI data itself isn't considered highly sensitive in your business.
- **External Access:** The portal would be internet-accessible by suppliers.

- **Test Automation Is High Priority:** Semi-structured specs allow us to design scripts to automate test case generation and validation.
- **Supplier Portal Feasible:** A portal makes sense due to the value in self-service onboarding.
- **Security as a Design Factor:** Define security measures into the portal.
- **Illustrative Scenario: Automation Potential**

## 1. Supplier Onboarding:

1. Portal section with spec documents, organized by format.
2. Supplier uploads a test file against their chosen spec.

## 2. Automated Test:

1. Scripts parse the spec, generating valid and invalid test cases.
2. The supplier's file is run against these tests.
3. Clear report back to the supplier on the portal with pass/fail and detailed reasons.

## 3. Production Config (Partial)

1. Scripts/tools to help parameterize EBS configuration based on the supplier's successful test results. Some manual steps might remain depending on your EBS complexity.

# Apply Application patches

- **Understanding the Problem: Manual Patching**

- Current process:

1. **Patch Identification:**

1. How do you find out about necessary patches (Oracle alerts, internal testing, etc.)?
2. Do you apply security, bug-fix, and functional patches?

2. **Applicability:** Is there a process to determine which patches are relevant to your EBS environment and customizations?

3. **Patch Application Steps:** Could you provide a high-level overview of your current manual patch steps?

- **Automation Feasibility Assessment**

1. **Details:**

1. **What:** How you find patches, apply them, and the types of patches in scope for automation.
2. **How:** Detailed steps of your patching process
3. **Where:** EBS environment (on-premise, cloud), and any testing/staging environments involved.
4. **Why:** Main pain points of manual patching — time taken, risk of errors, etc.

1. **Potential Solutions (Varying Complexity):**

1. **Option 1: Scripting the Workflow**

1. **Logic:** Scripts to automate the steps you currently take manually.
2. **Implementation:** Scripting within your EBS environment's tools (AD utilities, etc.). Potential for external orchestration if multiple systems are involved.

2. **Option 2: Partial Automation**

1. **Logic:** Automate the most repeatable parts (downloading, some pre-checks), leaving complex steps (customization conflict analysis) manual.

3. **Option 3: Integration with Patch Management Tools**

1. **If:** Your organization has broader patch management tooling.
2. **Implementation:** Investigate if it can integrate with Oracle EBS for orchestration, while EBS-specific steps might still need scripts.

2. **Script vs. Tool:** Depends on the chosen solution:

1. **Workflow Scripting:** Likely EBS-specific tools, plus scripting language suitable for your environment.
2. **Patch Management Tool:** Depends on your existing vendor solutions.

# Apply Application patches

## Benefits:

1. **Faster Patching:** Less manual overhead = quicker application of critical patches.
2. **Error Reduction:** Automation reduces typos, steps missed, etc.
3. **Consistency:** Patches follow a defined process, not ad-hoc each time.
4. **Auditability:** If the automation logs its actions, you have a better record.

## Savings:

1. **Requires Baseline:** How long does it take to patch manually end-to-end, across all environments?

## Quantification

1. **Frequency:** How often you patch various environments (prod vs. test).
  2. **Downtime:** Is production downtime required for patching? Automation can potentially shorten it.
- **Prioritization:** Weigh the time/error burden of manual patching against the complexity of automation.
  - **Usage in Industry:** Patch automation varies wildly based on tooling – Oracle EBS specific patterns are less common to benchmark.
  - **Implementation Time:** Scripting can be faster, tool integration might take longer.

- **Dependencies:**

- **Oracle Tooling:** AD utilities, etc., and permissions to use them for scripting.
- **Testing Rigor:** Automated patching needs EXCELLENT test environments to mirror production.

- **Metrics:**

- Patching Time (before/after)
- Errors Reduced
- Auditability (if currently lacking)

- **Caveat: Full Production Patch Automation is Risky**

- Customization conflict analysis is hard to completely automate. A staged approach (automate lower envs first) is often wiser.

- **Recommendations**

1. **Process Mapping:** Meticulous documentation of every manual patch step, including decision points.
  2. **Tool Inventory:** Do you already use patch management tools in your organization?
  3. **Customization Level:** Assess how heavily your Oracle EBS is customized, as this impacts automation ease.
- **Next Steps – Approach :**
  - **Scripting Logic:** If you're automating existing workflow steps.
  - **Risk/Reward Analysis:** Weighing the benefits of automation against the complexity for your setup.
  - **Tool Assessment:** If considering patch management vendor solutions, I can help with EBS-specific criteria.





# Apply Application patches - Approach

- **1. Scripting Logic**
- **Patching Tools:** The specific commands and utilities currently use within Oracle EBS for patching (e.g., adop, adpatch or any custom scripts).
- **Workflow Breakdown:** A detailed breakdown of manual patching steps, including:
  - Downloading patch files
  - Pre-patch prerequisite checks
  - Actions taken in various environments (test, production)
  - Any post-patch validation steps
- **Environment Details:** Operating system and any relevant patch management tools already in use, potentially outside of the core EBS environment.
- **Example: (Illustrative, Highly Dependent on your EBS Setup)**
- Let's assume a simplified scenario where you use adop for patching:

```
Bash

# Pre-patch checks (adjust based on your process)
adop phase=fs_clone
adop phase=prepare
adop phase=apply patches=y

# Error handling and logging would be added here
```
- **2. Risk/Reward Analysis**
- Key questions to consider, which will influence the level of automation that's sensible for your Oracle EBS setup:
- **Customization Complexity:**
  - Do you have extensive modifications to Oracle EBS modules? This makes conflict potential greater.
- **Current Error Rate:**
  - How often do manual patches introduce issues due to human error?
- **Testing Thoroughness:**
  - Do you have test environments that *very* closely mirror production? This is crucial for automation safety.
- **Downtime Tolerance:**
  - Can you afford any production downtime for patching, or does it need to be near-zero?
- **3. Tool Assessment**
- If integrating external patch management tools, evaluate their suitability for Oracle EBS:
- **Existing Tools:** List any patch management solutions you already use for other systems.
- **Vendor Questions:** Ask vendors specifically about:
  - Oracle EBS support and level of integration
  - Ability to handle custom EBS patch steps (if those are needed)
  - Flexibility in how automation workflows are orchestrated
- **Important Considerations**
- **Start Small:** Even if full automation is the end goal, I strongly recommend automating steps within lower environments first.
- **Testing is Paramount:** Automated patching without rock-solid testing is a recipe for trouble.

# EDI Onboarding - New Market, New Connections

## 1. Automation Details: What, How, Where, Why

- **Gather information on System context.**
- **What (Scope):**
  - Identify the transactions in scope, is it Only invoice transmission, or also order data, shipping notices, etc.?
  - Only new ANZ suppliers, or a phased approach for all suppliers?
- **How (Workflow):**
  - How are invoices generated within Oracle EBS?
  - Exact steps taken by the third-party vendor (extraction, translation, transmission).
  - Who currently initiates and monitors the EDI process?
- **Where (Systems):**
  - Where do EDI mappings currently reside?
  - Where would supplier protocol preferences be stored?
- **Why (Goals):**
  - Eliminate third-party cost?
  - Improve speed/reliability for MCD?
  - Gain visibility / reduce manual intervention?

## 2. Solutions with Implementation Guides

- **Option 1: Upgrade with Vendor**
  - \* **If:** Current vendor has diverse protocol support + ANZ expertise.
  - \* **Steps:** Negotiate, test direct connections with key suppliers.
  - \* **Easiest but may lack flexibility if vendor is not fully aligned.**
- **Option 2: EDI VAN**
  - \* **If:** Many suppliers, transaction volume is high, MCD has strict requirements.
  - \* **Steps:** \* RFP to qualified VANs (focus on ANZ market success).
  - \* Extensive mapping, supplier testing, pilot rollout.
  - \* **Reduces in-house complexity but introduces vendor dependency.**
- **Option 3: Hybrid In-house**
  - \* **If:** Internal expertise, moderate volume, some protocol capabilities in place.
  - \* **Steps:** \* Audit existing EDI stack.
  - \* Acquire protocol-specific modules/services.
  - \* Build new mappings, internal supplier-facing mechanism.
  - \* **Offers control but requires development and ongoing maintenance.**

# EDI Onboarding - New Market, New Connections

## Scripts vs. Bots/Tools

- **Depends on solution choice:**
  - VANs take care of this.
  - In-house builds will likely use scripts to extract data from EBS and orchestrate transfers.
  - Specialized EDI tools offer mapping, testing, monitoring GUIs.

## Benefits

- Third-party cost reduction (if applicable)
- Faster transmission to MCD (improves customer relationship?)
- Error reduction (manual steps are prone to mistakes)
- Process visibility (if a proper dashboard is built)

## Savings: NEED these to quantify:

- Per-invoice cost with the vendor
- Staff hours spent on current process
- Cost of errors caused by manual work

- **Implementation Time:** Weeks-months depending on complexity.
- **Dependencies:**
  - Buy-in from suppliers is vital
  - Internal IT bandwidth during the project.
- **Metrics:**
  - Invoice transmission time
  - Error rate
  - Hours saved in operations
- **Integration Providers:**
  - EDI specialists or Oracle partners. Need those with ANZ experience



# EDI Onboarding - New Market, New Connections – Detailed Approach

## Option 1: Leverage Your Existing Vendor

- **Feasibility:** High, assuming your existing vendor has the technical capabilities.
- **Pros:** Easiest to implement, minimal disruption to current processes.
- **Cons:** Might limit flexibility if the vendor isn't strong in the diverse protocols required, may not eliminate all costs.
- **How to explore:**
  1. **Check vendor capabilities:** Do they support AS2, SFTP, FTPs, and HTML upload? Do they have successful clients in the ANZ market?
  2. **Negotiate:** Discuss a model of direct supplier connection and the potential for cost savings.

## Option 2: EDI Value-Added Network (VAN)

- **Feasibility:** High. VANs specialize in B2B integration and are likely to have the ANZ expertise.
- **Pros:** Handles protocol complexity, simplifies supplier onboarding, offloads much of the technical work.
- **Cons:** Introduces vendor dependency, can be costlier than in-house options.
- **How to explore:**
  1. **Research:** Focus on EDI VANs with:
    1. Strong track record in the ANZ region
    2. Proven experience with the protocols you need
    3. A focus on Oracle EBS integration (this will simplify things)
  2. **Request for Proposals (RFP):** Clearly outline your requirements, the number of suppliers, expected transaction volumes.

## Option 3: Hybrid In-House Solution

- **Feasibility:** Depends on your team's technical expertise and existing EDI capabilities within Oracle EBS.
- **Pros:** Offers control and long-term flexibility if you build it well.
- **Cons:** Requires development effort, may need to purchase additional tools/modules.
- **How to explore:**
  1. **Audit:** Thoroughly assess your current EDI capabilities. Can they be extended?
  2. **Identify Gaps:** Where are you lacking protocol support? Do you need better mapping tools?
  3. **Consider Point Solutions:** Explore tools specifically for AS2, SFTP, etc., to complement your core system.

## Option 4: Full In-House Development (Least Recommended)

- **Feasibility:** Only if you have a large, highly skilled development team, significant budget, and are willing to take on ongoing maintenance.
- **Pros:** Maximum control (if the project is a success).
- **Cons:** High risk, long development timeline, potential for compatibility issues with future Oracle EBS updates.
- **Before Choosing:** Gather the following information for any of these options:
  - **Transaction Volume:** Identify the invoices per month.
  - **Internal Expertise:** Assess your team's abilities.
  - **Budget:** Get ballpark estimates for each feasible option.
  - **Desired Timeline:** Determine the Urgency for this in place.

# ORACLE - SaaS

---

# Automated Code Review - Sanity Checker

- **What:**
    - List the exact user settings that are frequently activated.
    - Are these tied to specific roles, or is there decision-making involved?
  - **How:**
    - Is the current process entirely manual through the ERP Cloud GUI?
    - Is there any batch process potential, or does it occur per user?
  - **Where:** Precisely where in Oracle ERP Cloud these settings reside (module, administration area).
  - **Why:**
    - Is this due to onboarding a large number of users with similar needs?
    - Are the settings required for compliance or specific functionality?
- **2) Various Solutions with Implementation details**
  - **Option 1: Oracle Identity Cloud Service (IDCS)**
    - **If:** ERP Cloud is integrated with IDCS for user provisioning.
    - **Implementation**
      - Investigate if settings can be assigned via roles/groups within IDCS.
      - This may lessen the need for individual ERP Cloud modifications.
  - **Option 2: ERP Cloud Web Services**
    - **If:** ERP Cloud exposes APIs to manipulate user settings directly.
    - **Implementation**
      - Requires development: Script to call the API, potentially with logic to fetch the settings to apply.
      - May require integration middleware if invoking from an external system.
  - **Option 3: UI Automation (RPA)**
    - **If:** The process is 100% manual GUI clicks, and no APIs are available.
    - **Implementation:** RPA tool to mimic user actions, read from a list of accounts needing modification.

# Automated Code Review - Sanity Checker

- **3) Script-based vs. bots/tools**
  - Likely script-based, using either ERP Cloud's API or integration tools to invoke it.
  - RPA is the fallback if GUI interaction is the only way.
- **4) Proposed benefit of automation**
  - Significant time savings if setting activation is frequent and manual.
  - Improved consistency, less prone to human errors in the GUI.
  - Could link to onboarding processes, making new user setup faster.
- **5) Type of saving on the application scope**
  - Purely administrative time saved. Unlikely to have a direct application performance impact.
- **6) Quantifying hours saved**
  - **Baseline your current process:** How long does it take to manually activate settings per user?
  - Multiply by the number of users this occurs for weekly/monthly.
  - Estimate how much automation will reduce that time (consider some overhead for script execution & maintenance).
- **7. How often does this happen?**
  - **Where to look:** You'll need some form of record-keeping on how frequently new users are added or require these specific setting changes. This might be in your IT helpdesk ticketing system, user provisioning logs, or even informally tracked by your ERP Cloud administrators.
- **8. How long does it take now?**
  - **Observe or estimate:** Can you shadow an admin as they perform this task, timing them? If not, try to get them to provide a realistic estimate of the average time per user modification.
- **9. How important is this to automate?**
  - **Think beyond time-saving:** Consider if manual errors in these settings have ever caused:
    - Security issues (users having incorrect access)
    - Functionality problems (users unable to do their jobs due to missing settings)
    - Compliance violations (if the settings are tied to audit requirements)

# Automated Code Review - Sanity Checker

- **10. What tools might work?**

- **Depends on your API situation:**

- If Oracle ERP Cloud exposes APIs for user settings, focus on:
  - iPaaS providers (Integration Platform as a Service)
  - General scripting languages (Python, JavaScript) if you prefer in-house development.
- If no API exists, investigate Robotic Process Automation (RPA) tools with strong web automation capabilities.

- **11. Prioritization:**

- This becomes clear once you have data on frequency (#7) and the impact of errors (#9).
- High frequency + high potential for errors = good case for

## **12. Metrics**

- Focus on:

- Number of user accounts modified (before/after automation)
- Total time spent on the task (before/after automation)
- Reduction in errors, if that was a problem with the manual process

- **How to Get Started**

- 1. Focus on #7 and #8:** Get real data on how often the task occurs and how long it takes.
- 2. Investigate APIs:** Find out if Oracle ERP Cloud allows direct manipulation of user settings through web services. This will be the biggest factor in solution design.





SNOW

# SNOW – Incident Breach

- **Understanding the "What" and "Why"**
- **Breach Definition:**
  - Are breaches based on exceeding resolution time targets, response time targets, or a combination of factors?
  - Are these targets determined by Incident/Problem priority, specific SLAs, or other variables?
- **"At Risk" Logic:**
  - How do you define a ticket being "at risk" of a breach? Is it a simple calculation (e.g., 50% of the target time remaining) or more complex?
- **Report Details:**
  - Do you simply need the overall count of breached/at-risk tickets, or more detail (e.g., which ones, impacted assignment groups, etc.)?
- **Primary Goal:**
  - Establish the guideline for this purpose of report created to identify where proactive action is needed, to drive overall SLA compliance awareness, or something else.
- **Solution Considerations**
- **Option 1: Native SNOW Reporting**
- **Best For:** Basic counts of breached/at-risk tickets, assuming your breach logic is simple enough for SNOW's filtering system.

## Questions:

- Can you create filters that match your "breach" and "at risk" definitions within SNOW's reporting module?
- Is the standard report formatting sufficient for your needs?
- **Option 2: Scheduled Jobs in SNOW**
- **Best For:** Moderate complexity with logic that can be coded in JavaScript, or if you need customization in how the report is delivered.
- **Questions:**
  - Are you comfortable with SNOW scripting and retrieving data through the API?
  - Does sending emails or basic report storage within SNOW meet your requirements?
- **Option 3: External Scripting**
- **Best For:** Complex breach/risk logic, or if you want highly custom formatting (fancy charts, specific Excel output, etc.)
- **Questions:**
  - Do you have programming resources (Python, etc.) and knowledge of the SNOW API?
- **Option 4: BI/Reporting Tool**
- **Best For:** Advanced visualizations, scheduling, distribution to many users, or feeding insights into other systems.
- **Questions:** \* Do you already have a BI/Reporting tool, and can it connect to SNOW's data?



# SNOW – Incident Breach – Report Design

## Benefits, Savings, and Quantification

- **Think Manually:** How long does this "breach prediction" report take someone to compile currently? What are the steps involved?
- **Errors:** Are mistakes made in this manual process that automation could eliminate?
- **Actionable:** Does the report drive better decision-making and intervention (this is harder to quantify but impacts the bottom line)

## Additional Considerations

- **Industry Standard:** Tracking SLA performance and breaches is a core ITIL practice, so this use case has wide applicability.
- **Prioritization:**
  - How frequently is this report needed?
  - How bad is the pain of creating it manually right now?
- **Downsides:** If your breach logic changes frequently, maintaining scripts/reports becomes an overhead.

## Reporting Design – Breach - Guideline

- **Breach Definition:** A ticket is breached if the resolution time exceeds a target based on its priority or a defined SLA.
- **"At Risk":** Tickets with 50% or less of their target resolution time remaining.
- **Report:** Simple count of breached + at-risk, maybe broken down by assignment group or category.
- **Goal:** Proactive awareness of potential SLA issues to drive action.

## • Recommendations with Rationale

### • Scenario 1: Simple, Internal Reporting

- **Start with Native SNOW Reporting:** Explore filters and scheduled reports. This is the least disruptive and leverages what you have.
- **Reference:** Look up SNOW's documentation on scheduled jobs and reporting, lots of community examples exist.

### • Scenario 2: Moderate Complexity, Some Customization

- **Scheduled Job within SNOW:** If logic is a bit complex or you want some email/storage options, this gives you coding control.
- **Reference:** SNOW Developer site for API, JavaScript scripting, best practices for scheduled job design.

### • Scenario 3: Complex Logic OR Fancy Formatting

- **External Scripting:** Use Python with the SNOW API if you need complex risk calculations or want very tailored reports (Excel, PDF).
- **Reference:** SNOW REST API docs, Python libraries like 'requests'.

### • Scenario 4: Advanced Visuals, Wide Distribution

- **BI/Reporting Tool:** If stakeholders need dashboards, scheduling options, etc., a dedicated tool is the best fit. Look into Power BI, Tableau, or similar if you have them.
- **Reference:** Documentation for your BI tool of choice, focus on connector setup to SNOW's data source.

## • Additional Factors

- **Team Skills:** If you have no programming experience, start with native SNOW tools. If you have scripting folks, Options 2 & 3 become viable.
- **Change:** If your breach logic is likely to evolve, then favor less-code solutions (Option 1 or 4).
- **Urgency:** Simple needs done fast = native SNOW tools. Complex requirements demand a more planned approach (Options 2, 3, or 4)