Your Data is Your Foundation. Are There Cracks in Yours?

Poor data quality isn’t just an inconvenience; it’s a liability. Inconsistent entries, missing information, and broken business rules lead to inaccurate reports, failed data imports, and flawed decision-making. Every time a user leaves a critical field blank or enters the wrong code, a crack forms in your foundation.

Become the Data Quality Gatekeeper

Ostendo provides a powerful, built-in tool to protect the integrity of your database: **User Defined Validations**. This feature allows you to act as a gatekeeper, defining and enforcing your unique business rules directly within the system. Ensure only clean, correct, and complete data ever makes it into your records.

More Than Just a "Required Field"

User Defined Validations go far beyond simple checks for blank fields. They allow you to build detailed, conditional logic that runs automatically whenever a record is inserted or updated. This avoids the need for complex Required Fields, Custom Trigger Data Scripts for many common validation tasks.

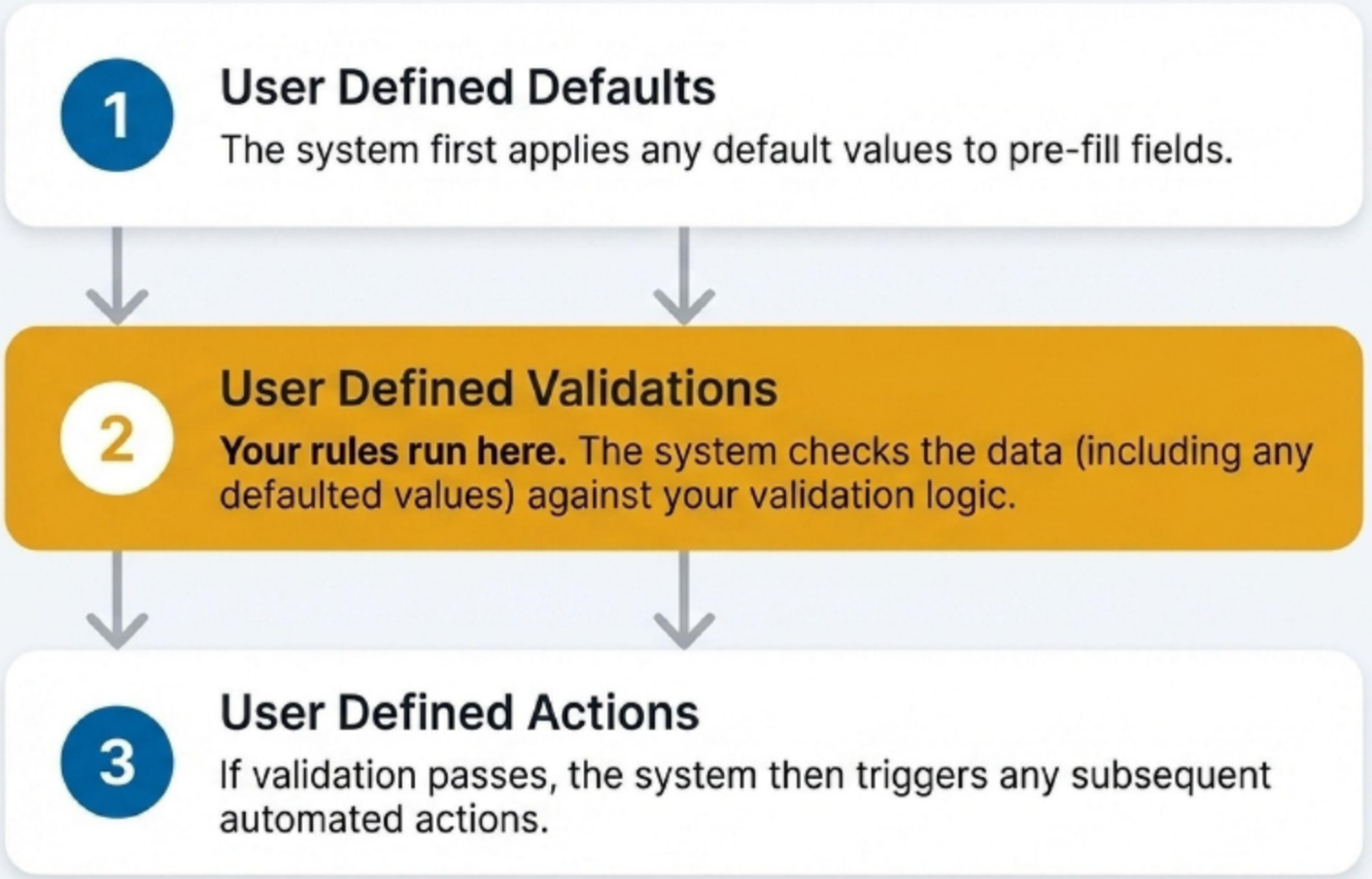
Example Box

****Scenario****
If an item's Category is 'Accessories' or 'Fasteners', then the Analysis Group for that item must not be left blank.

****Power****
This level of conditional logic is impossible with a standard 'Required Field' setting.

Understanding Ostendo’s Automation Sequence

Validations don’t operate in a vacuum. Ostendo processes automation rules in a specific, logical order. Understanding this sequence is key to building reliable workflows.



Anatomy of a Validation Rule

Let's break down the User Defined Validations screen (found under`File -> Customisation Configuration -> User Defined Validations`). Each field plays a critical role in defining your rule's behavior.

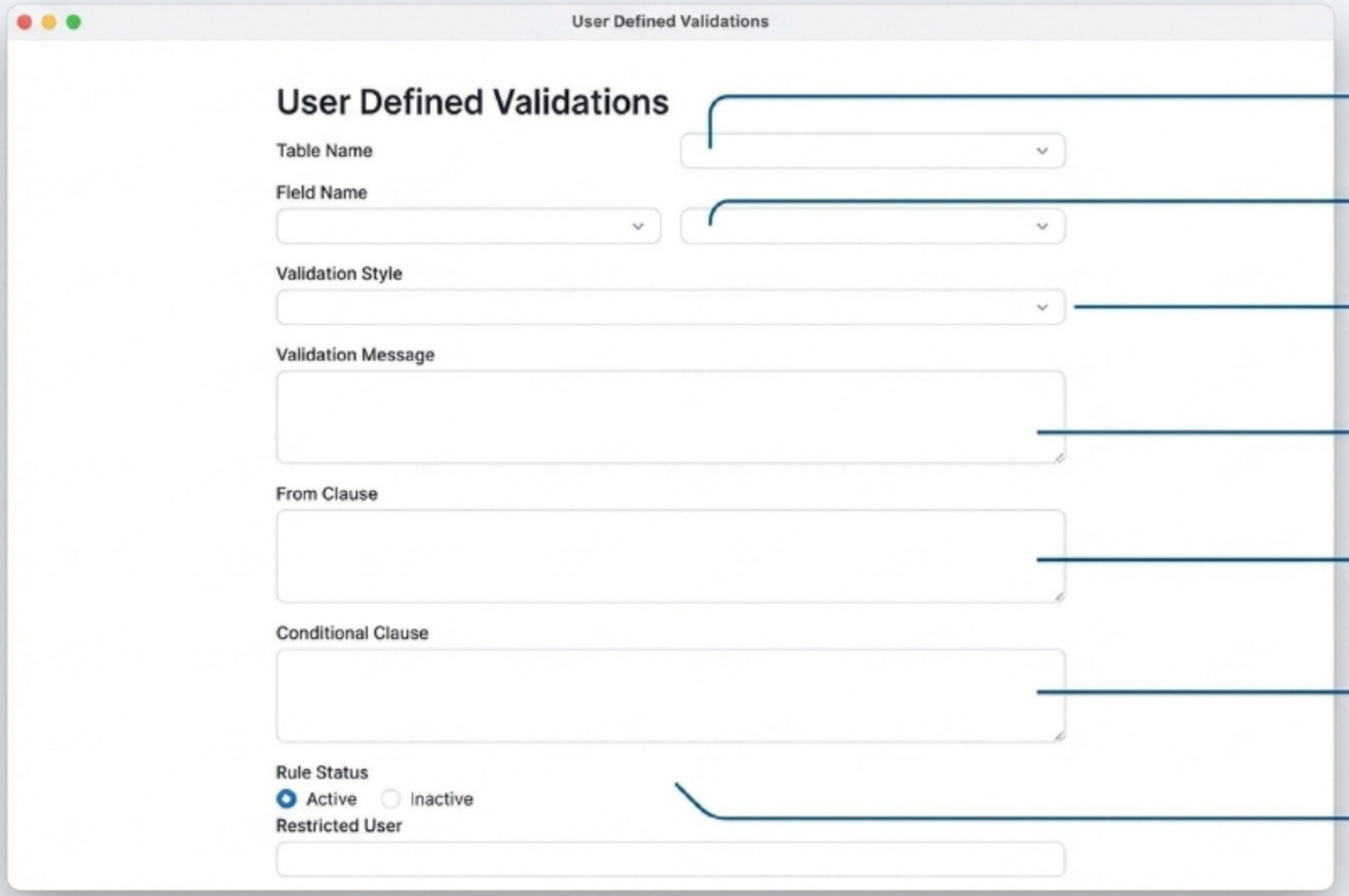


Table Name: The database table where the rule applies.

Field Name: The specific field being checked.

Validation Style: *When* the rule triggers (on insert, update, etc.).

Validation Message: The custom error message the user will see.

From Clause: The table(s) your logic will query.

Conditional Clause: The heart of the rule—the SQL condition that must be met.

Rule Status & Restricted User: Controls for testing and deploying the rule.

Crafting Your Logic: The `From` and `Conditional` Clauses

These two fields are where you define the precise logic for your validation.

`From Clause`

Purpose

Specifies the table name(s) to be used in your condition.

Best Practice

While it can sometimes be left blank if you are only using the main `Table Name', it is good practice to always define the table name here for clarity.

`Conditional Clause`

Purpose

This is the 'brain' of your rule. It is the syntax that follows the `WHERE` in a standard SQL statement.

The validation triggers based on whether this condition evaluates to true or false.

Pro-Tip Box: When your clause involves more than one table, you *must* prefix each field with its table name. Example: `CustomerMaster.SalesPerson`.

Precisely Controlling When Your Rule Fires

The ‘Validation Style’ determines the exact moment your rule is checked. The options give you granular control over the user workflow.

On Insert Only

Checks the rule only when a new record is created.

- `Raise Error if Condition is met`
- `Raise Error if Condition is not met`

On Update Only

Checks the rule only when an existing record is saved.

- `Raise Error if Condition is met`
- `Raise Error if Condition is not met`

On Insert or Update

Checks the rule in both scenarios.

- `Raise Error if Condition is met`
- `Raise Error if Condition is not met`

For most data integrity rules, you will use `` `Raise Error if Condition is met` `` on either ‘Insert Only’ or ‘Insert or Update’.

The Pro’s Toolkit: Writing Bulletproof Syntax

Test Before You Deploy

Before implementing a rule, always test your `Conditional Clause` syntax in the Data Spreadsheet (`General -> Data Spreadsheet`). This ensures your logic is correct before it goes live.

```
Select ITEMMASTER.ANALYSISGROUP
from ITEMMASTER
where ITEMMASTER.ITEMCATEGORY IN ('Accessories','Fasteners')
```

Master the Art of Checking for Blanks

To reliably check for a blank entry (which could be a `NULL` value or an empty string), use the `coalesce` function. This function handles both cases gracefully.

```
(coalesce(CUSTOMERMASTER.CUSTOMEREMAIL, '') = '')
```

The Pro's Toolkit: Deploying with Confidence

Avoid disrupting users with faulty rules. Use the built-in status controls to test and deploy your validations safely.

1. Test (`Restricted to User`)

Set the rule to this status and specify a `Restricted User`. Only that user will experience the validation, allowing for safe, isolated testing.

2. Deploy (`Active For All Users`)

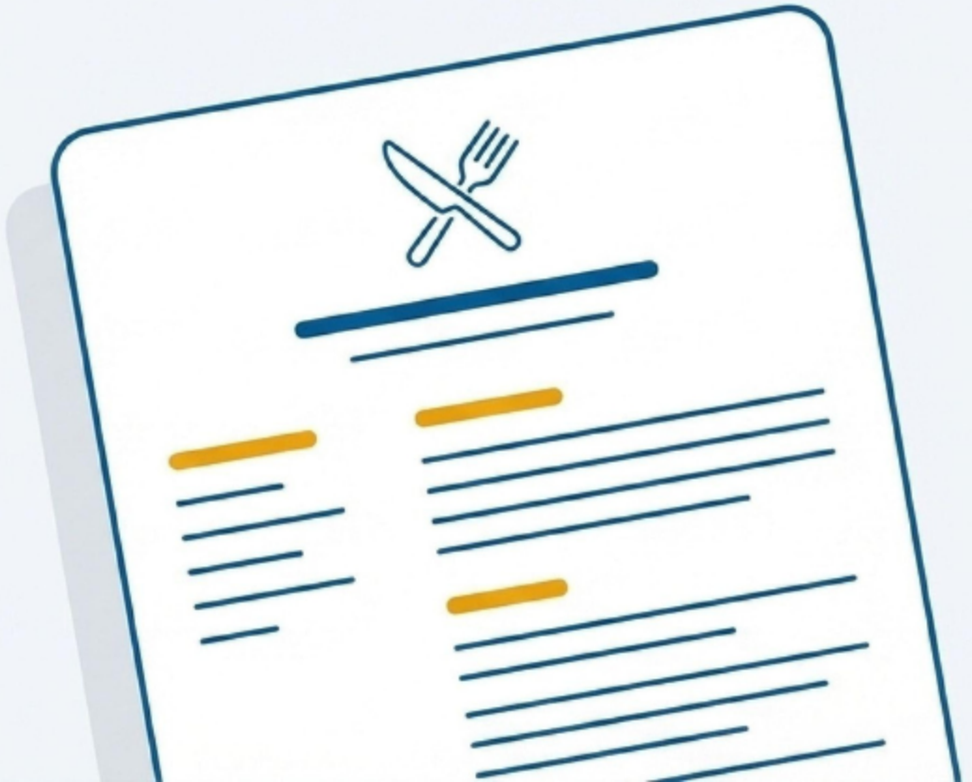
Once testing is complete, switch the status to make the rule live for everyone in the database.

3. Disable (`Not Active`)

Temporarily turn off a rule without deleting it. The rule remains in the system but will not be triggered.

The Masterclass: Real-World Validation Recipes

The best way to learn is by doing. The following slides present common validation scenarios in a simple 'recipe' format. Each recipe provides the goal, the required settings ('ingredients'), and the logical clause ('method') you need to implement it.



Recipe: Require a Primary Supplier for 'Fasteners'

Goal

Force the user to enter a Primary Supplier, but *only* when creating a new Item with a Category of 'Fasteners'.

Ingredients (The Settings)

- Table Name: `ITEMMASTER`
- Field Name: `PRIMARYSUPPLIER`
- Validation Style: Raise Error if Condition is met on Insert Only
- Validation Message: "A Supplier must be specified for Items categorised as Fasteners."

Method (The `Conditional Clause`)

```
(coalesce(ITEMMASTER.PRIMARYSUPPLIER, '') = '') AND ITEMMASTER.ITEMCATEGORY = 'Fasteners'
```

Logic Explained

This condition is true (and raises an error) only if *both* parts are true: the supplier field is blank AND the item category is 'Fasteners'.

Recipe: Mandate Customer Email on Creation

Goal

Ensure an email address is always entered when a new customer record is created.

Ingredients (The Settings)

- Table Name: `CUSTOMERMASTER`
- Field Name: `CUSTOMEREMAIL`
- Validation Style: Raise Error if Condition is met on Insert Only
- Validation Message: "Email Address must be entered."

Method (The `Conditional Clause`)

```
(coalesce(CUSTOMERMASTER.CUSTOMEREMAIL, '') = '')
```

Logic Explained
The condition is met simply if the customer email field is blank upon creation of the record.

Upgrading Your Recipe: Mandate Email on Create *and* Update

Goal

Expand the previous rule to ensure an email address is present when creating a customer *or* when an existing customer record is updated.

The Only Change

To apply this rule to updates as well as inserts, you only need to change one setting. The core logic remains the same.

Updated Ingredient

- **Validation Style:** Raise Error if Condition is met on **Insert or Update**

Method (The `Conditional Clause`)

```
(coalesce(CUSTOMERMASTER.CUSTOMEREMAIL, '') = '')
```

Logic Explained

The logic is identical. We are still raising an error if the email field is blank, but we are now checking at two different times: insert and update.

Advanced Playbook: Defaults + Validations for Clean Imports

The Scenario

You import Job Orders from a web source that doesn't provide a Job Category. You need to allow the import but force staff to categorize the job later.

Step 1: The Default (Allow the Import)

Create a **User Defined Default**.

When a Job Order of type 'Web Order' is inserted, automatically set its `Job Category` to 'Undefined'. This allows the record to be created without error.

Step 2: The Validation (Enforce the Fix)

Create a **User Defined Validation**.

Set `Validation Style` to `Raise Error if Condition is met on Update Only`.
Condition: `JOBMASTER.JOBCATEGORY = 'Undefined'`

Message: "Please select a valid Job Category before saving."

The Result

Imports run smoothly. The first time a user tries to edit an imported job, they are forced to correct the missing data, ensuring long-term data quality without disrupting automation.

Your Blueprint for Data Integrity

Use these core principles as your guide to building robust and effective validations in Ostendo.

Start with 'Why': Clearly define the business rule you need to enforce before you start building.

Test in the Spreadsheet: Always validate your SQL syntax in the Data Spreadsheet first.

Use `coalesce`: Make this your go-to function for reliably checking blank/null text fields.

Deploy Safely: Always use the `Restricted to User` status to test new rules before activating for everyone.

Combine and Conquer: Remember that Validations, Defaults, and Actions work together to create powerful automated workflows.

Data Quality
Gatekeeper