THE PLAYBOOK

The Backlog

How I took a backlog of hundreds of exceptions, due by the next morning, and turned it into a few dozen decisions with owners. The repeatable method behind that: rank by reach, group by a stable key, and route each class to the one person who can fix it. Plus the honest chapter on when working row by row really is the right answer.

Paul Prado Pacardo

Executive Assistant · Project Manager · Operations me.kabuhayan.app

Contents

Hundreds of rows by morning the queue, and the quiet decision inside it

1 **A row is not a decision** why speed through rows is the losing move

2 **Rank by reach first** sort by how many places each exception touches

3 **Group by a stable key** the method: classify, key, route, prove

4 **Delete only on three yeses** the three-condition delete rule

5 **Where it breaks quietly** failure modes, and how to catch each one

6 **When not to group** the cases where grouping backfires

Decide once, apply everywhere closing

The backlog checklist and the live artifact from this story

OPENING

Hundreds of rows by morning

A back office handed me a hold queue late in the day. Hundreds of stuck records, every one under the same error label, and all of it due to be cleared or explained by the next morning.

> THE QUEUE, AS IT ARRIVED
>
> One label. Hundreds of rows. One deadline. No notes on what had already been tried, and no owner named for any of it.

A queue like this invites one reaction: start at the top and go faster. Every operator who inherits a backlog faces the same quiet fork, usually without noticing it is there, and the choice decides the night more than any amount of effort afterwards.

THE FORK IN THE ROAD

THE SAFE MOVE

Work faster through rows. Open the first record, fix it, open the next. It feels like progress from the first minute. It also spends the same effort on a row that stands alone as on a row that shares its cause with forty others.

THE MOVE I MADE

Group rows into decisions. Read the real messages, find what the rows have in common, and decide once for every row that shares a cause. Slower for the first hour, and then the queue collapses.

Figure: the choice that sets the size of the night before any row is touched.

This playbook is that right-hand branch, worked twice: once in a back office clearing a hold queue, once on a different team cleaning up a task tracker. Two moves came out of it, rank by reach and group by a stable key, plus one small triage rule that saved me from the worst mistake a cleanup can make. It ends with the honest chapter: when the row-by-row grind is the right call, because a method that claims to always apply is a slogan.

CHAPTER 1

A row is not a decision

A backlog counts rows. The work is decisions. Those two numbers are almost never the same, and the gap between them is where the night goes.

Rows overstate the work. When forty rows fail for one reason, fixing them one at a time is forty small efforts to make one decision. Speed helps, but speed is multiplying the wrong number.

Decisions are the real unit. Find the smallest key where one ruling fixes every row that shares it, and the backlog shrinks to the size of the thinking it actually needs.

> THE BET
>
> Big backlogs shrink when you group rows into decisions, not when you work faster through rows. An hour spent finding the shape saves the rest of the night.

I saw the same shape on two different teams. In the back office, grouping the hold queue collapsed hundreds of rows into a few dozen checks. On the other team, reconciling a task tracker produced about a hundred naming decisions, and a single unnamed section turned up across dozens of projects while most names appeared once.

BACK OFFICE

Hundreds of rows became a few dozen checks, about 5x fewer decisions. Roughly half cleared in one morning; the rest went to named owners.

TASK TRACKER

Ranking by reach put the one ruling that touched dozens of projects first, instead of somewhere in the middle of an alphabetical list.

Worth being precise about what this is evidence for. Two backlogs, two teams, both with causes that happened to cluster. The method travels; the 5x does not. What generalises is the trade: time spent finding the shape of a queue converts into cleared rows at a rate that working the rows never matches. A backlog with no shared causes would not shrink at all, and chapter 6 is about exactly that case.

CHAPTER 2

Rank by reach first

The task tracker taught me the first move. Working the exception list top to bottom meant the first hours cleared items that each touched a single project. The one that mattered most sat further down.

The fix is one column: **reach**, how many records, projects or people each exception touches. Sort by it, descending, and the list reorders itself by payoff.

EXCEPTION (INVENTED)	REACH	RULING
Unnamed section	Dozens of projects	Rename to the standard section, once
Two spellings of one stage name	About a dozen	Collapse to the registered spelling
Legacy status label	A handful	Map to the current status
One-off custom names	One each	The tail: decide whether to touch it at all

> **Normalise first.** Case, spacing and punctuation variants collapse before you count, or one exception shows up as three small ones.

> **Decide in batches of about ten.** One sitting, one set of rulings, then apply.

> **Write each ruling once in a crosswalk.** From value, to value, ruled by, date. The crosswalk starts empty by design: nothing is applied without a ruling.

> **Apply only from the crosswalk.** Bulk edit from the table, never scattered hand edits that drift from what was decided.

> **Stop at the tail.** When what remains touches one record each, stop and decide whether the tail is worth clearing.

> THE RULE
>
> Before you work any exception list, add a "how many rows does this affect" column and sort by it. If a ruling belongs to someone else, send it to them in a batch, never one by one.

CHAPTER 3

Group by a stable key

The hold queue taught me the second move. The label said one thing was wrong. Reading the message text row by row showed several error types, each with a different owner, some only the supervisor could fix. A label tells you what the system thinks is wrong, not what is.

Step 01 **Export with the full message**

Every row, with its real message text, not just the queue label.

Step 02 **Classify by real message**

Sort rows into types by what the message says. One label can hide four causes.

Step 03 **Name an owner per type**

You, the supervisor, the account specialist, the vendor. A type with no owner is assigned one before you move on.

Step 04 **Choose the grouping key**

Inside your own types, the smallest key where one decision fixes every row. Mine was account + person + code.

Step 05 **Build checks on stable IDs**

One row per key, each with an ID. Every lookup and summary reads that ID, never a row number.

Step 06 **Work, spot-check, re-export**

Fix each check, verify one row from it, then re-export and diff to prove the fixes landed.

Step 07 **Route the rest**

Each other owner gets one message holding only their rows.

> DECISION RULES
>
> Never edit a source plan to make a record fit. If a duplicate has no surviving original, escalate instead of deleting. Items from a prior period go to the supervisor.

> SEE IT YOURSELF
>
> The live tool ranks rows by reach, then groups them into checks with owners and a reduction ratio, on invented sample data or your own: **me.kabuhayan.app/tool-backlog-grouper**.

CHAPTER 4

Delete only on three yeses

Inside the backlog sat a class that looks easy and is not: time records with a start and no end, or an end and no start. The procedure seemed to say delete them. Deleting the wrong one erases the only evidence that someone worked that day.

So the delete has a gate. All three must hold.

ONE

Same person

The orphan and the complete record belong to the same person, by registered name.

TWO

Same date

On the same date. A complete record from the day before proves nothing.

THREE

Complete shift present

A full start-and-end record already covers the time. The orphan is a leftover, not the work.

Anything else is a contact or an escalation. Before choosing, read the whole day for that person, not only the flagged entry, and check four things:

> **A pending correction.** If the person already asked for a fix, approve it. Do not email them about it.

> **An edited original.** The orphan may be the old end of a record that was later corrected.

> **A morning and evening swap.** A clean record with the end marked in the morning is an edit, noted as a device issue, not a delete.

> **A supplied time.** If the person gave the missing time, enter it, and log exactly what you entered.

> STATED PLAINLY
>
> Delete is the one action you cannot take back. It gets the strictest rule in the method, and a hunch never meets it.

Log every action with the exact time removed or entered. When a pattern repeats across people, report it as a pattern to the owner of the rule, not as a run of one-off fixes.

CHAPTER 5

Where it breaks quietly

None of these failures announce themselves. Each one produces a clean-looking worksheet that is wrong. The catch has to be built in, because you will not notice by reading.

> THE TRAP TO AVOID
>
> My first worksheet looked up rows by row number. A simple re-sort broke every summary formula, silently, and the totals still looked plausible. I rebuilt every lookup on stable IDs. Never let a formula depend on row order.

FAILURE	WHAT IT LOOKS LIKE	THE CATCH
Row-order lookups	Totals shift after a sort	Every lookup reads a stable check ID
Printed duplicates	The export lists one row twice, so it counts twice	Dedupe the export before counting
One wording, two causes	A check that half-clears after the fix	Spot-check one row per check after fixing
Deciding twice	Two rulings for one exception	Check the crosswalk before every ruling
Drifting hand edits	Records that differ from the ruling	Apply only from the crosswalk
Orphaned types	A class nobody owns sits in limbo	No type leaves step 3 without an owner
Hours on the tail	The last stretch takes longer than the rest	A stop rule based on reach

The common thread: **prove the clear, do not assume it.** Re-export after every pass and diff against the last one. The rows you fixed should be gone, and nothing new should have appeared. If either is false, the worksheet is lying to you, and it is better to learn that before the report goes out than after.

Codes entered under the wrong person are the hardest to catch. A key built on person will group them faithfully and wrongly. The spot-check is what finds them.

CHAPTER 6

When not to group

A method that claims to always apply is a slogan. There are real cases where working the rows, or something else entirely, is the right move.

> **When the causes do not cluster.** If the reach column comes back flat, every row touching one record, there is nothing to group. Building a key costs time and returns nothing. Work the rows.

> **When the queue is small.** Twenty rows do not need a crosswalk. The method pays for itself somewhere past the point where you cannot hold the list in your head.

> **When the deadline is about this period.** On the eve of a pay run with limited hours, getting this period right comes first. Old holds can wait a day, stated plainly in the report with how long they have carried.

> **When the ruling is not yours.** If every type belongs to someone else, your job is classification and routing. Grouping your way into their decisions is overreach.

Even in these cases, half the method survives: **read the real message before you count or route anything.** Classifying by what the row says, not by its label, does not require a worksheet.

> THE TEST TO APPLY
>
> Sort by reach and look at the top ten. If they touch many rows each, group. If the top of the list already touches one row each, close the worksheet and work the queue.

CLOSING

Decide once, apply everywhere

Before the first row, one quiet decision: grind, or group. Everything in this playbook is downstream of choosing to find the shape first, ranking by reach, keying on something that does not move, and routing each class to the one person who can fix it.

The speed, the clean report, the morning deadline met: all of it follows from turning rows into decisions. That is the whole game.

The worksheet from this story did not stay a one-off. It became a free tool on this site, which is its own lesson: a method written down once can clear backlogs it never saw.

Paul Prado Pacardo

Senior Executive Assistant, Operations and Project Manager. Ten years supporting C-level leadership. Available for remote work.

me.kabuhayan.app · linkedin.com/in/paulpacardo · pacardopaul18@gmail.com

THE TOOL

The backlog checklist

Run this before you open the first row of any backlog too big to hold in your head. If the reach column comes back flat, chapter 6 applies.

BEFORE YOU TOUCH A ROW

- [ ] Full list exported with real message text and where each row occurs
- [ ] Variants normalised so duplicates collapse
- [ ] Reach column added and sorted, top ten read
- [ ] Rows classified by real message, not by queue label

BUILDING THE CHECKS

- [ ] An owner named for every type, none left in limbo
- [ ] Grouping key chosen: one decision fixes every row that shares it
- [ ] Every check has a stable ID, and every lookup reads it
- [ ] Each ruling written once in the crosswalk, applied only from it

CLEARING AND PROVING

- [ ] Deletes only on three yeses: same person, same date, complete shift present
- [ ] One row spot-checked per check after fixing
- [ ] Re-exported and diffed: cleared rows gone, nothing new appeared
- [ ] One message per other owner, holding only their rows

THE LIVE ARTIFACT

The backlog grouper from this story is free on the tools page: **me.kabuhayan.app/tools**. It ranks rows by reach, then groups them into checks with owners and a reduction ratio. The direct link is me.kabuhayan.app/tool-backlog-grouper.