The Flow Production-Killer Audit

20 Flow anti-patterns that work fine on one record and take down the org on 200 — each with why it breaks and the exact fix. Plus a self-audit checklist you can run before every deploy.

Governor limits | Bulkification | Fault handling | Architecture | Summer '26

THE 20 PRODUCTION-KILLERS

01	DML inside a loop	02	Get Records inside a loop
03	Multiple Update elements for the same record	04	After-save flow updating its own trigger record
05	No fault paths on DML elements	06	Unguarded recursion / re-triggering
07	One mega-flow per object	08	No entry conditions — runs on every save
09	Hardcoded IDs and values	10	Never bulk-tested with 200
11	Hand-building collections in a loop	12	Get Records pulling everything
13	Querying in a loop just to check existence	14	Large scheduled flow with no batch control
15	Using a flow where a validation rule fits	16	Too many automation tools on one object
17	No null / empty checks	18	Environment-specific template & asset references
19	Unbounded 'get all, then loop' on huge objects	20	No naming, description, or version discipline

A Flow that works in the sandbox proves nothing — it was tested on one record. Production hands it 200 at once from a data load, an integration, or a bulk update, and that's when the limits, the recursion, and the silent failures show up. Every anti-pattern below is a specific way Flows die at scale, with the fix we actually ship.

01 DML inside a loop

LIMITS

THE TRAP An Update / Create / Delete element sits inside a Loop.

WHY IT BREAKS 200 records = 200 DML statements. You hit the 150-statement limit and the whole transaction rolls back for every user.

THE FIX "No pink in loops." Get before the loop, add records to a collection inside the loop, run ONE DML after it.

```
Get (before)  →  Loop → Assign into collection  →  Update Records (after loop)
```

02 Get Records inside a loop

LIMITS

THE TRAP A Get Records element runs on every iteration.

WHY IT BREAKS Each Get is a SOQL query; 101 of them throws "Too many SOQL queries: 101".

THE FIX Query once before the loop using the In operator, then reference the collection in memory.

```
Get Records  WHERE Id In {!idCollection}   // one query, loop the result
```

03 Multiple Update elements for the same record

LIMITS

THE TRAP Two or three Update Records elements touch the same record in one run.

WHY IT BREAKS Every Update element spends a separate DML statement for no reason.

THE FIX Set every field with one Assignment, then a single Update Records.

```
Assignment (set all fields)  →  one Update Records
```

04 After-save flow updating its own trigger record

LIMITS

THE TRAP An after-save record-triggered flow updates the same record that fired it.

WHY IT BREAKS That's an extra DML + a re-entry into the save order — wasteful and a recursion risk.

THE FIX Use a before-save ("Fast Field Update") flow for same-record changes: zero DML, and 200 records update as fast as one.

```
Record-Triggered Flow → "Fast Field Update" (before-save)   // 0 DML
```

05 No fault paths on DML elements

LIMITS

THE TRAP Create / Update / Delete elements have no fault connector.

WHY IT BREAKS Failures are silent — bad data, no error, angry users, and nothing in the logs.

THE FIX Add a fault path on every DML element routing to a reusable error-handling subflow that logs the failure.

```
Every Create/Update/Delete → Fault → [Log Error subflow]
```

06 Unguarded recursion / re-triggering

LIMITS

THE TRAP A flow updates records that re-fire the same flow.

WHY IT BREAKS "Maximum trigger depth exceeded" after 16 hops — and it burns CPU, SOQL and DML on the way down.

THE FIX Gate the start with "only when a record is updated to meet the condition" + ISCHANGED, or a stamp-field guard.

```
Entry: only-meets-criteria  +  ISCHANGED({!$Record.StageName})
```

07 One mega-flow per object

ARCHITECTURE

THE TRAP A single giant flow handles validation, updates, notifications and integrations.

WHY IT BREAKS Unpredictable order of execution, impossible to debug, and one change risks everything.

THE FIX Split by intent (validation vs notification) and by event (insert vs update). Compose with subflows.

08 No entry conditions — runs on every save

ARCHITECTURE

THE TRAP The flow has no start filter, so it fires on every create/update.

WHY IT BREAKS Wasted runs, blown limits, and a prime cause of recursion.

THE FIX Set precise entry conditions so the flow only runs when it actually needs to.

```
Start → Entry Conditions → {!$Record.Status} = "Open"
```

09 Hardcoded IDs and values

MAINTAINABILITY

THE TRAP Record-type IDs, user IDs, profile IDs or queue IDs are typed into the flow.

WHY IT BREAKS They differ across sandboxes and prod — the flow silently does the wrong thing after deploy.

THE FIX Reference Custom Metadata / Custom Settings / $Setup, or look records up by a stable name.

10 Never bulk-tested with 200

LIMITS

THE TRAP The flow was tested by editing one record in the UI.

WHY IT BREAKS It works for one and crashes on a Data Loader job or a 200-row integration insert.

THE FIX Always test with 200 records before deploying. If it survives 200, it survives production.

11 Hand-building collections in a loop

LIMITS

THE TRAP A Loop + Assignment builds a new collection element by element.

WHY IT BREAKS More elements, more iterations, more governor-limit exposure than you need.

THE FIX Use the Transform element to map input → output collections declaratively.

```
Use Transform (not Loop + Assignment) for collection mapping
```

12 Get Records pulling everything

LIMITS

THE TRAP Get Records selects all fields, no filter, no row limit, no “first record only”.

WHY IT BREAKS Wastes heap and query rows; non-selective queries time out on large objects.

THE FIX Select only the fields you use, filter on indexed fields, and set “first record only” when you need one.

13 Querying in a loop just to check existence

LIMITS

THE TRAP A Get inside the loop checks “does a related record exist?”.

WHY IT BREAKS Same SOQL-in-loop trap, hidden behind a Decision element.

THE FIX Pre-query related records into a collection/map once; check membership in memory.

14 Large scheduled flow with no batch control

LIMITS

THE TRAP A scheduled- or record-triggered flow processes a huge data set in one go.

WHY IT BREAKS It blows limits mid-run and leaves data half-processed.

THE FIX Set a maximum batch size on schedule-triggered flows (Summer ’26) so volume scales safely.

```
Scheduled-Triggered Flow → Batch Size: 200   (Summer ’26)
```

15 Using a flow where a validation rule fits

ARCHITECTURE

THE TRAP A flow enforces a simple field rule that a validation rule or before-save would handle.

WHY IT BREAKS Heavier, slower, and harder to maintain than the native tool.

THE FIX Use validation rules / before-save flows for simple field checks; reserve flows for orchestration.

16 Too many automation tools on one object

ARCHITECTURE

THE TRAP Flows, Apex triggers and legacy Workflow all fire on the same object.

WHY IT BREAKS Order of execution becomes unpredictable; one tool undoes another.

THE FIX Consolidate onto one orchestration layer, document the order, and migrate legacy Workflow/Process Builder to Flow.

17 No null / empty checks

LIMITS

THE TRAP The flow loops or references a Get result without checking it returned anything.

WHY IT BREAKS Empty collections and null lookups throw runtime errors that never showed in the sandbox.

THE FIX Add a Decision to check for null / empty before looping or using a Get result.

18 Environment-specific template & asset references

MAINTAINABILITY

THE TRAP Send Email and other actions reference templates/assets by environment-specific ID.

WHY IT BREAKS The reference breaks the moment you deploy to another org.

THE FIX Summer ’26 stores the Send Email template by NAME — re-test deployments where two templates share a name.

```
Send Email → template stored by NAME (Summer ’26)
```

19 Unbounded ‘get all, then loop’ on huge objects

LIMITS

THE TRAP A flow pulls every record of a large object and loops them in one transaction.

WHY IT BREAKS Heap and CPU limits hit long before you finish.

THE FIX Filter hard at the Get, or move high-volume processing to Batch Apex.

20 No naming, description, or version discipline

MAINTAINABILITY

THE TRAP Elements are named Get1 / Decision2; no descriptions; old versions left active.

WHY IT BREAKS Nobody can safely change it later — and Spring ’26’s AI-built flows make this worse, fast.

THE FIX Name elements meaningfully, describe each, log faults, and keep a deactivate/version strategy. Treat Flow with Apex-level rigor.

★ Bonus — build the reusable error-handler subflow once

Anti-pattern #05 says every DML element needs a fault path. This is the subflow they all call — build it one time, reuse it across every flow in the org.

BUILD IT 1) New autolaunched flow **Log_Flow_Error** with text inputs: sourceFlow, recordId, faultMessage. 2) Create Records → a custom object **Flow_Error_Log__c** (Source_Flow__c, Record_Id__c, Error__c). 3) Optional: fire an email / Slack alert to the admin team.

WIRE IT On every DML element's fault path, call Log_Flow_Error with the flow name, the record Id, and the fault message — then show the user a friendly error screen instead of a stack trace.

```
Fault → Subflow Log_Flow_Error( faultMessage = {!$Flow.FaultMessage}, sourceFlow =
"Opportunity_Update", recordId = {!$Record.Id} )
```

The pre-deploy self-audit

Run every box before you activate a flow in production. Any unchecked box is a future incident.

- ☐ No Get, Update, Create or Delete element sits inside any Loop
- ☐ Every DML element has a fault path → reusable error-logging subflow
- ☐ Same-record updates use a before-save (Fast Field Update) flow
- ☐ The flow has precise entry conditions (doesn't run on every save)
- ☐ Recursion is guarded (ISCHANGED / only-meets-criteria / stamp field)
- ☐ One record is updated with a single Assignment + single Update
- ☐ No hardcoded record-type / user / profile / queue IDs
- ☐ Get Records selects only needed fields, filtered, with a row limit
- ☐ Collections are built with the Transform element, not Loop + Assignment
- ☐ Null / empty results are checked before they're used
- ☐ Logic is split by intent and event — no mega-flow per object
- ☐ Legacy Workflow Rules / Process Builder migrated to Flow
- ☐ Scheduled flows set a max batch size for large data volumes
- ☐ The flow was bulk-tested with 200 records before deploy
- ☐ Elements are named meaningfully and described
- ☐ Old flow versions are deactivated; an active-version strategy exists

Got a flow that times out or fails silently at volume?

That’s most of what we untangle. Genetrix re-architects Flow, Apex, and the whole automation layer to run at enterprise scale across Salesforce core, Marketing Cloud, Data Cloud, and Agentforce.

Book a 30-minute automation review →