Stop Guessing. Start Designing: Order of Execution for Future Salesforce Architects

Pranav Nagrecha 7 min read · Aug 26, 2025

Ever save a record in Salesforce and feel like a dozen hidden levers just got pulled? A Flow updates something, a validation rule blocks it, then a trigger fires again… suddenly you're untangling chaos instead of delivering value. It's not random — it's the platform's Order of Execution. Without it, the system feels haunted: a checkbox pokes a Flow, which wakes a workflow, which sets off a trigger you forgot about.

The good news? The kitchen does have a recipe. Architects learn it, follow it, and stop being surprised.

Why we keep guessing (and why it hurts)

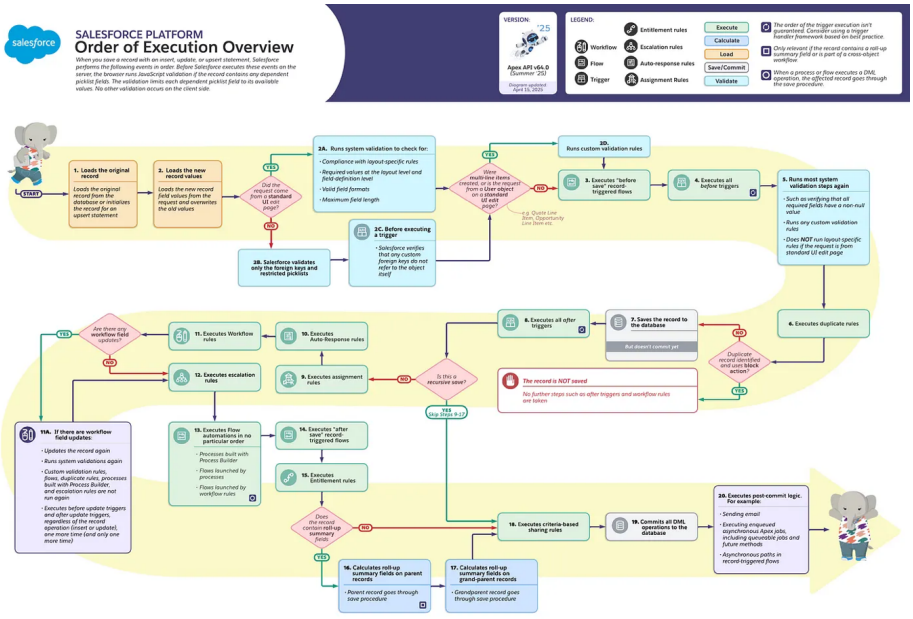
Most admins learn by building. Click, test, tweak, repeat. It's fast, it's empowering — and honestly, it's how many of us fell in love with Salesforce in the first place. You build something, refresh the page, and it just works. Until it doesn't.

The problem shows up when logic starts stacking. A validation rule here, a Flow there, maybe a trigger tucked away from a previous project. Everything runs… but not in the order you expect. Suddenly a small change snowballs into five side effects you didn't design for. That's when we end up "fixing" things by toggling them off and on like we're trying to reboot the Wi-Fi.

And here's the catch: this isn't a skills problem. It's not that admins can't write good logic. It's that the platform is running a strict sequence — and if you don't know it, you're guessing at why your automation collides. The mental model is missing.

Think about it: a developer sees errors as symptoms of a flowchart gone wrong; an admin often sees them as glitches. But Salesforce isn't glitching — it's following rules. If you can *see the sequence*, debugging shifts from trial-and-error to tracing a map. That's the mindset difference between "Why did Salesforce break my update?" and "Where in the order did this logic fire?"

The kitchen's recipe (platform Order of Execution)



Order of Execution

- Image Link — https://developer.salesforce.com/docs/platfor

m/data-models/guide/order-of-execution.html

- Documentation Link — Salesforce Developers
- Architecture basics — Salesforce Architects

Think of a record save as **one dish** moving down a kitchen line. One station, one job, in a strict sequence. When you place logic on the correct station — and only that station — behavior gets boring (in the best way).

Below is the full line. Each step has three anchors so it sticks:

- **Salesforce:** what the platform does
- **Design example:** what you design or refactor
- **Kitchen Example:** how it looks on the line, for the same dish (Spaghetti Marinara)

1) Load original record

Salesforce: Loads the current row (or initializes for upsert).
Design example: In updates, read **old** values intentionally; don't assume only new fields matter.

Kitchen Example: Pull yesterday's prep card before you start today's cook.

2) Load new values (+ initial system checks)

Salesforce: Applies incoming values; runs **UI** layout checks or **API**-style checks depending on source.
Design example: Integrations must send required and well-formed fields (e.g., `LastName`, valid foreign keys).
Kitchen Example: Lay today's ingredients next to the card; toss anything off-spec.

3) Before-save record-triggered Flows

Salesforce: Runs before the record is saved; fastest place for same-record updates.
Design example: Default and sanitize here; keep simple stamping in one Flow.
Kitchen Example: Season the crushed tomatoes before heat.

4) Apex BEFORE triggers

Salesforce: Execute after before-save Flows, still pre-save.
Design example: Do heavier transforms (bulk-safe Maps/Sets); avoid SOQL/DML in loops.

Kitchen Example: Sous-chef standardizes the dice so it cooks evenly.

5) System validations (again) + Custom Validation Rules

Salesforce: Re-verifies required, format, length; runs **custom** validation rules.
Design example: Keep rules crisp; don't depend on fields you plan to change **later** in the sequence.
Kitchen Example: Chef de cuisine checks the spec — no plate moves without basil.

6) Duplicate Rules

Salesforce: Runs matching; if **Block**, the save **stops** and nothing downstream runs.
Design example: Inside sales → **Block** exact email. Web-to-lead → **Allow/Alert**, and make downstream logic idempotent (no double tasks/emails).
Kitchen Example: Expo spots **two identical chits** for the same table *before* any pan fires — pull the extra chit (Block) or tag it "dup — watch" (Allow/Alert).

7) Save (not committed)

Salesforce: Writes the row, keeps the transaction

open.
Design example: Don't call external systems here; wait for after-layers or async.
Kitchen Example: "Fire!" garlic hits oil; sauce starts cooking.

8) Apex AFTER triggers

Salesforce: Full values + IDs available; safe for related-record work.
Design example: Update parent counts with changed-value guards; avoid recursion.
Kitchen Example: Pasta drains; call to another station for sides.

9) Assignment Rules

Salesforce: Routes Leads/Cases to the right owner/queue.
Design example: Let assignment own routing; don't tug ownership later unless designed.
Kitchen Example: Expo assigns the ticket to sauté — not grill.

10) Auto-Response Rules

Salesforce: Sends acknowledgements.
Design example: Put the "we received your case" message here *or* in Flow — never both.
Kitchen Example: Time stamp hits the chit so

the board tracks SLA.

11) Workflow Rules (legacy)

Salesforce: Emails, tasks, outbound messages, **field updates.** If a **Workflow Field Update** edits the same record, Salesforce:
• Updates the record again → **re-runs system validations** → **re-fires before & after triggers once.**
• Doesn't re-run: **custom validation, Flows, duplicate rules, Process Builder processes, escalation rules.**
Design example: Move same-record updates to **before-save Flow** to kill surprise re-saves.
Kitchen Example: An old recipe card says "add butter at the end" — chef must re-check before the plate can pass.

12) Escalation Rules (Case)

Salesforce: Timers can change priority/ownership if untouched.
Design example: Make downstream routing respect escalations.
Kitchen Example: Expo bumps a late ticket to a faster cook.

13) Processes (Process Builder) & Flows

launched by Workflow

Salesforce: These automations run here; order **not guaranteed.**
Design example: If any remain in your org, keep their side-effects small and predictable — or retire them.
Kitchen Example: Legacy side-station does extra prep before final finishing. Salesforce Developers

14) After-save record-triggered Flows

Salesforce: After the record is saved (still in-transaction).
Design example: Put related-record writes, notifications, callouts/orchestration here — or in Apex after, but not both.
Kitchen Example: Ladle sauce, basil, a sheen of oil — the finishing pass. *(In API **53.0 and earlier**, these ran after entitlements.)*

15) Entitlement Rules / Milestones (Case)

Salesforce: Evaluates entitlements; starts milestone timers.
Design example: Don't keep flipping fields that reset clocks.
Kitchen Example: "Hit the window in eight

minutes.”

16) Parent roll-up/cross-object calcs

Salesforce: Recalculates parent; that parent goes through its own save path.
Design example: Expect re-entry on the parent; add guards there.
Kitchen Example: Using two ladles of sauce updates the prep board.

17) Grandparent roll-up/cross-object calcs

Salesforce: If the parent changed, recalc the grandparent too.
Design example: Watch for “one change, three saves” cascades.
Kitchen Example: The prep board updates the prep-of-prep sheet upstream.

18) Criteria-Based Sharing evaluation

Salesforce: Re-evaluates access rules.
Design example: Batch big changes to avoid share storms and row locks.
Kitchen Example: Allergen tag flips; only specific **line cooks** can touch the pan now.

19) Commit

Salesforce: Finalizes the transaction.
Design example: Keep the path to commit lean; push “nice-to-have” work later.
Kitchen Example: Plate clears the pass.

20) Post-commit logic

Salesforce: Emails, **Queueable/Future** Apex, **asynchronous paths in record-triggered flows**, events, etc.
Design example: Put slow or fragile operations here for reliability.
Kitchen Example: Clean-down and logs — pans washed, production sheet updated.

“Who wrote last?” — collisions that bite (and how to design around them)

- **Before-save Flow vs Apex BEFORE: Apex BEFORE** wins (later in pre-save).
- **Apex AFTER vs After-save Flow: After-save Flow** wins (later in the transaction).
- **Workflow Field Update:** Triggers one **extra** pass — **system validations** and **both triggers** run **once; custom validation, Flows, duplicate rules, processes, escalation** don’t re-run.
- **Duplicate Rule (Block):** No save at all; nothing else runs. Salesforce Developers

Design move: Give each concern **one owner** at the right station.

- Same-record stamping → **before-save Flow** (or Apex before if truly complex).
- Cross-object writes/notifications → **one** place (after-save Flow *or* Apex after).
- Anything slow or external → **post-commit.** Salesforce Developers

Debug like a chef at the pass

1. **Where am I on the line?** Pre-save, after-save, or post-commit?
2. **Who touched this field last?** Walk the stations **in order** until you find the final writer.
3. **Did a Duplicate Rule block it?** If yes, stop — nothing saved.
4. **Did a Workflow Field Update re-save it?** Expect one more round of **system validations** and **triggers**.
5. **Can this move post-commit?** If it isn't urgent, move it; failures and lock contention drop fast. Salesforce Developers

Metadata vs data: the mindset shift

Admins often think in terms of data: *"I just need to update this field."* It's a task-oriented view — quick, specific, and focused on the outcome. That mindset works fine when you're building your first automation, but it breaks down once multiple processes start competing for control.

Architects, on the other hand, think in metadata. Instead of asking *what* to update, they ask *where* that logic belongs in the sequence. Should this happen before the record is saved? After it's saved but still in-transaction? Or after commit, when the database is already settled? The question shifts from *"what do I want Salesforce to do?"* to *"where's the safest, most predictable place for this logic to live?"*

Here's the metaphor I like:

- **Data is the dish.** It's the tomato sauce, the pasta, the plate that reaches the customer.
- **Metadata is the recipe.** It's the steps, the timing, the guardrails that make sure the dish comes out the same way every time.

If you only think about the dish, every save feels like trial-and-error. But when you work the recipe — placing each piece of logic in the right spot on the line — the kitchen behaves. Chaos turns into consistency.

The architect's edge (and your next habit)

The difference between an admin and an architect isn't luck. It's intent.

Admins fix symptoms. Architects place logic exactly where it belongs.

So when something breaks, don't play whack-a-mole. Step back, follow the line, and ask:

"Is this logic living where it should?"

That single habit is what turns chaos into consistency — and builders into architects.

Salesforce | Salesforce Productivity

Written by Pranav Nagrecha

Edit profile

51 followers · 43 following

Salesforce Administrator/Developer/Architect/Business Analyst/QA

No responses yet

Pranav Nagrecha he

What are your thoughts?

More from Pranav Nagrecha

Pranav Nagrecha

Boomi—How to convert Timezones i...

I use Google to the extreme, I know there are smarter...

May 17, 2022

Pranav Nagrecha

How We Built a Real-Time Integration...

Say goodbye to 15-minute polling delays—here's how...

May 14 13

Pranav Nagrecha

Roles, Profiles, and Permission Sets in...

Introduction

Aug 21 19

Pranav Nagrecha

Flow Control: Navigating...

Flows are the lifeblood of many business processes....

Sep 25, 2023 9

See all from Pranav Nagrecha

Recommended from Medium

Vaishnavi R

Database. Stateful in Salesforce

If you have ever worked with batch classes in Apex, you...

Jul 23 2

Musa Ndlala

Understanding Standard Objects a...

The Moment You Realize Salesforce Isn't Just Data...

Sep 15 5

Pedro Távora Santos

Why Your Agentforce POC Will Fail: An...

If you're reading this, you've probably sat through the...

Nov 18

Mir Sarfarajey Akram

Generating an OpenAPI Spec for...

Salesforce now provides a beta feature that can...

Aug 11 1

Sai Santhosh Kandibanda

Day 4 -Salesforce All-Star Ranger

Today, we are exploring the module called "Data...

Nov 22 1

Mani

Optimizing Long-Running Callouts in...

Jun 24 1

See more recommendations