

The record that exists in one system and not the other

The order is in the shop and not in the warehouse. The nightly job says it succeeded. Nobody has been told, because from the job's point of view nothing went wrong.

Author	Amit Govil, Founder and CEO, XONIK
Published	October 22, 2026
Reading time	8 min read
Online	https://xonik.com/insights/perspectives/the-record-that-exists-in-one-system-and-not-the-other/

IN BRIEF

A record goes missing between two systems when it arrived by a route the integration was never built to watch, or when the integration skipped it and still reported success. Nobody sees it, because the job, the totals and the dashboards all look normal. A check that compares identifiers at both ends, owned by a named person, finds it first.

KEY FINDINGS

1. A route the job was never built to watch leaves records behind without any fault in the job.

Hand-keyed orders, bulk imports and repair scripts sit outside the condition the integration was given. A receiving side that tightens a validation rule leaves the refused record behind in the sender, unreported.
2. A job that finished has not delivered every record.

Finished means it reached the end of its list. A record that fails validation is logged and skipped, a message in a dead-letter queue expires unless someone set an alarm, and a bookmark that moves on turns a retryable failure into a permanent absence.
3. The vendors who send events tell integrators that the feed is a convenience and the comparison is the control.

Shopify advises running reconciliation jobs that fetch the data again, and Stripe names retrieving objects through its API as the way to fill what was missed.
4. Every signal that is routinely watched sits above the single record, so the gap surfaces through a customer.

Totals look plausible and dashboards built from the destination cannot show what is not in it. By the time someone notices, logs have rotated and queues have expired, and each hand repair removes the proof.

5. A check at record level, from outside the job and in both directions, finds the gap first.

It is written from the business definition of what should exist, not from the job's own query, and its alert goes to a named person. It cannot find a record that both systems lack.

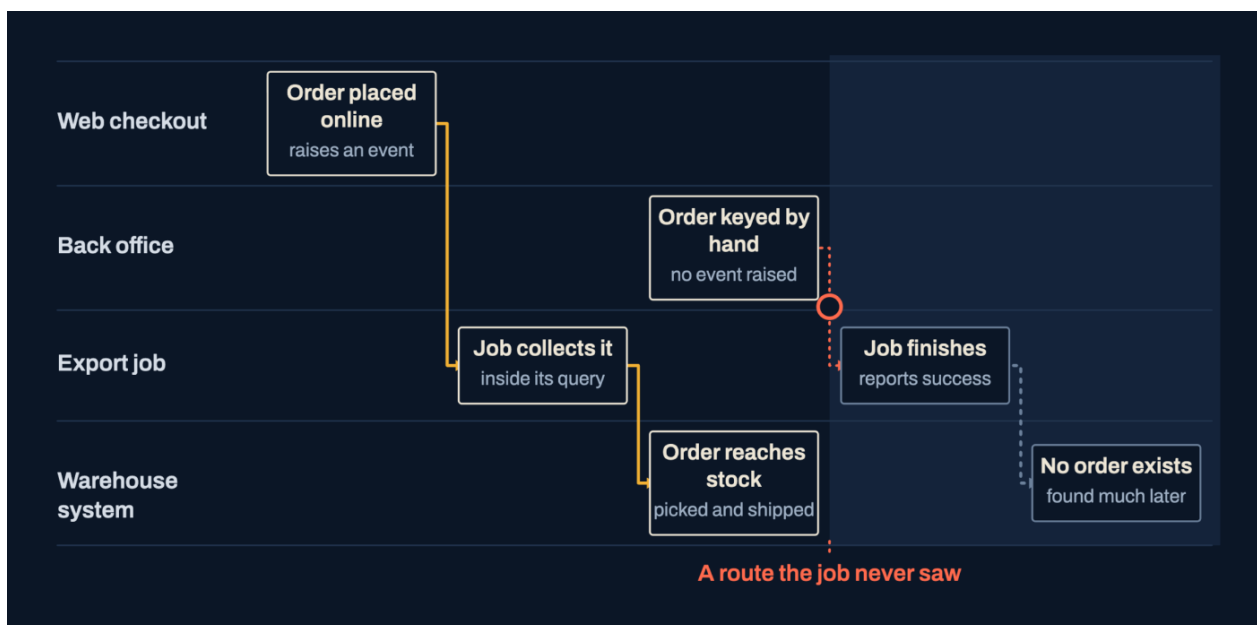
How does a record come to exist in only one system?

An integration is a rule written down once: take the records that match this condition and put them over there. The condition is drawn from the ways records were known to arrive on the day it was written. Web orders raise an event, so the job listens for the event. Later other routes appear. Staff key an order for a caller. A bulk import loads a partner's list. A repair script corrects a batch and writes straight to the table. A record is saved in a state the job's query leaves out, or edited after the job's window has moved past it. None of these routes is wrong, and none of them breaks the job. They sit outside what the job was told to look for. Exhibit 1 follows a hand-keyed order that never raises the event the job relies on.

This is why an orphan is not a failure of the integration in the ordinary sense. The integration did what it was written to do. The gap is in the definition of what should move, and the definition was correct for a business that has since added routes. The receiving side can produce the same result from the other direction: after a validation rule is tightened it begins refusing records it used to accept, and the refused record stays behind in the sender, unreported.

EXHIBIT 1

A route the job never saw



Orders take different routes. The job is correct about the one it saw and silent about the one it did not.

Why does the job say it succeeded?

A job reports on what it handled. Finished means it reached the end of its list. It does not mean the list was the right one, or that every item on it landed. A record that fails validation is logged and skipped, usually at a severity that pages nobody. Designs built on message queues add a second place for a record to go quiet.

Amazon's developer guide describes a dead-letter queue as the place where messages that were not processed successfully are set aside so the cause can be diagnosed. The messages are not kept forever: the guide advises setting a dead-letter queue's retention longer than the source queue's, and points to alarms for any message moved there (Amazon SQS Developer Guide, Using dead-letter queues). The alarm is a choice somebody has to make. Without it, the record sits in a holding area whose only regular visitor is the expiry.

Bookmarks add a third place. Some jobs ask for the records changed since the last run and then move their bookmark to the end of the batch. If a record in that batch failed and the bookmark moved on regardless, the next run never asks for it again, and a failure that could have been retried becomes a permanent absence. A job that advances its bookmark only past records it has confirmed, and keeps the rest to retry, turns the same fault into a delay.

The vendors who send events say much the same from their side. Shopify's guidance tells an app not to rely on webhooks alone, because delivery cannot always be relied on and handlers can fail or be down, and to run reconciliation jobs that fetch the data again. Stripe retries a failed delivery for a limited period, and its documentation names retrieving objects through its API as the way to fill what was missed. Both are telling the integrator that the feed is a convenience and the comparison is the control.

| The feed is a convenience and the comparison is the control.

Who finds it, and when?

Every signal that is routinely watched sits above the individual record. The job reports that it finished. The totals at each end look plausible, because one absent record is a small absence in a large number. Dashboards are built from the destination, which cannot show what is not in it. The only signal at the level of a single record is the person who needs that record: a customer chasing an order, a clerk who cannot find the matching entry, an auditor who draws a sample from one system and looks for it in the other. Exhibit 2 sets the levels out.

Direction matters. A sample drawn from the source and looked up in the destination finds records that never arrived. A sample drawn from the destination and looked up in the source finds records that arrived from nowhere. An audit that runs one way leaves the other unexamined.

Discovery is late, and the evidence has usually gone by then. Logs rotate, queues expire their messages, and the person who added the route has moved on. The record is added by hand, the customer is satisfied and the cause is never looked for. Each hand repair proves the gap exists and, in the same act, removes the proof.

EXHIBIT 2

What each level can tell you

 The job	reports that it finished	
 The totals	look plausible at each end	Nobody looks here
 The identifiers	compared record by record	
 The customer	is the one who finds the gap	

The signals people watch are the ones above the break. The check that would catch an orphan sits at the level nobody owns.

What does it look like in a business?

In retail, an order keyed by phone staff is in the shop system and absent from the warehouse feed because the feed reads only web checkouts. The customer waits, support resends the order by hand, and the stock count is wrong for the item in the meantime, since the unit was never reserved. At a dental or clinical site, a patient booked at the front desk is missing from the reminder list because the reminder service reads only online bookings. The patient does not attend, the slot is lost, and the record reads as a no-show.

Where either system is a regulator's own record, as in regulated cannabis markets or gaming, the missing record is a compliance matter and the party who finds it is an inspector, not a customer. The mechanism is the same in all three cases. Only the person who finds the gap, and what it costs by then, differ.

What finds it first?

A check that looks at both ends, at record level, from outside the job. After each run a separate process lists the identifiers that should now exist in the destination, lists those that do, and reports the extras on each side. It compares an agreed total as a cross-check on the identifiers and never as a substitute for them, and it runs in both directions. It is written from the business definition of what should exist, not from the job's own query, which is how it catches routes the job does not know about.

Start with the integration whose output leaders rely on most, and run the first comparison once by hand. It will usually find something, and what it finds shows which routes the job's definition leaves out. That list, more than any tool, is the specification for the check.

These rules keep it useful. A difference is an incident, however small. The alert goes to a named person, not a mailbox. The cause is recorded each time, including the route the record took, so the second occurrence resolves faster than the first. And any new way of creating a record gets a line in the check before it goes live. Where queues are used, an alarm on the dead-letter queue and a retention longer than the source queue's keep the evidence long enough to read.

The check has a limit. It finds records that differ between the systems. It cannot find a record that both systems lack, and it cannot say whether a record that matches is right. The question for the owner of any

integration is not whether the job runs. It is how they would learn, and how soon, that a single record had not arrived.

Sources read for this piece

Each passage cited below was read in the original on 20 October 2026. Routes and discovery are described from our delivery work, as a pattern. Nothing here is counted.

1. [Amazon Web Services, Amazon SQS Developer Guide, Using dead-letter queues: purpose, retention, alarms](#)
2. [Shopify, About webhooks: unreliable delivery, reconciliation jobs](#)
3. [Stripe, Receive Stripe events in your webhook endpoint: how long failed deliveries are retried, and fetching what was missed](#)

USE THIS

[Review your architecture with the checklist](#)

[Read why systems and data stop agreeing](#)

Pick the integration whose output your leaders depend on most, and ask who would notice, and when, if some of its records stopped arriving.

This document was made from the article at

<https://xonik.com/insights/perspectives/the-record-that-exists-in-one-system-and-not-the-other/>. The online article is the current version.