

Change Data Capture at Billion-Record Scale: Architecture Patterns for Incremental ETL in High-Volume Healthcare Claims Environments

Saikrishna Ala

Independent Researcher

Abstract: Healthcare claims adjudication systems accumulate transactional histories that reach billions of records, and moving that data into an analytical environment without disrupting the operational system that produces it is a harder engineering problem than general data warehouse literature usually acknowledges. This article examines change data capture (CDC), the practice of identifying and extracting only the records that changed since the previous processing cycle, rather than re-extracting an entire dataset, as the architectural answer to that problem at a billion-record scale. A tiered maturity framework organizes the discussion around three approaches: query-time delta detection using high-watermark timestamps, database-level transaction log mining, and event-driven real-time capture built on message-queue decoupling. Each tier is evaluated against five dimensions drawn from practitioner engagements in pharmacy benefits management and claims adjudication environments: extraction impact on the operational source system, change detection completeness, change detection accuracy, pipeline latency, and implementation complexity. The article gives particular attention to a constraint that moderate-volume Extract, Transform, Load (ETL) literature rarely has to confront directly: adjudication databases are latency-sensitive operational systems bound by regulatory data-handling obligations, which means an extraction architecture that would be a reasonable default at a smaller scale can become an operational risk once record counts cross into the billions. Data quality validation for incremental delta record sets receives separate treatment, since a change data capture cycle that correctly identifies every changed record can still load a delta set that is duplicated, out of order, or incomplete. The framework's engineering conclusions extend beyond healthcare to any high-volume transactional environment, financial services, telecommunications billing, retail transaction history, and government administrative systems that face the same underlying tension between operational latency sensitivity and analytical data currency.

Keywords: Change data capture; incremental ETL; healthcare claims data; transaction log mining; event-driven data integration; data warehouse architecture

1. INTRODUCTION

Pharmacy benefits management organizations sit between insurance plan sponsors, pharmacies, and plan members, adjudicating prescription drug, dental, and extended health claims in real time at the point of dispensing. The largest of these organizations process well over a billion claims transactions annually [1], and the databases that support this adjudication accumulate a complete historical record of prescription utilization, benefit application, and payment authorization for millions of plan members. That same database is asked to do two jobs at once: serve as the latency-sensitive operational system behind live claims processing, and feed the population-health analytics, drug-utilization trend analysis, and regulatory reporting that plan sponsors expect as a matter of course.

Those two jobs have conflicting requirements. The operational side needs the adjudication database to respond to a pharmacy's claim submission in well under a second. The analytical side needs a complete, current, and auditable copy of that same data in a separate environment built for reporting rather than transaction processing. Extracting the data to satisfy the second requirement without degrading the first is where most of the engineering difficulty in this article actually lives.



The obvious approach, extract, transform, and loading the entire adjudication dataset on every cycle, works acceptably at moderate volume. It stops working once record counts move into the billions. A full-load extraction against a billion-record source table is not merely slower than an incremental one; past a certain scale, it competes directly with the operational workload for the same database resources, and a poorly timed full extraction can measurably degrade claims-processing response times during the exact hours when pharmacy transaction volume is highest. Change data capture, identifying and moving only the records that changed since the previous cycle, is the standard answer to this problem in general data warehouse literature, but that literature was mostly written with moderate-volume environments in mind, where a full-load fallback remains available if the incremental approach runs into trouble and where the source database exposes transaction-log access without much friction [2], [3].

Billion-record healthcare claims environments do not provide the same options for fallback. The adjudication database may not expose a standard change-capture mechanism at all, depending on the platform and how it was originally implemented. Regulatory obligations under the Health Insurance Portability and Accountability Act (HIPAA) in the United States [4] and the Personal Information Protection and Electronic Documents Act (PIPEDA) in Canada [5] impose audit-trail, encryption, and access-control requirements that the extraction pipeline has to satisfy end to end, not just at the point of final storage. And the adjudication data model itself frequently encodes claim state through combinations of status flags and related records rather than through a simple timestamp or monotonically increasing sequence number, which complicates even the basic question of how to detect that a record has changed.

This article works through that gap directly, building a tiered architecture framework for change data capture at a billion-record scale, grounded specifically in the engineering constraints of pharmacy benefits management and claims adjudication systems. The framework's three tiers, query-time delta detection, database-level transaction log mining, and event-driven real-time capture, are not presented as a strict maturity ladder where every organization eventually reaches Tier 3. Each tier is a legitimate architectural choice depending on what the source platform supports and what the analytical use case actually requires, and Section 4 develops selection criteria alongside the description of each tier rather than treating tier progression as an unconditional goal.

A concrete version of the failure mode is worth naming before moving into architecture. Consider a pharmacy benefits management organization running full-load extraction against a two-billion-record adjudication table on a nightly cycle. As claim volume grows year over year, the extraction window creeps later into the morning until it begins overlapping with the earliest wave of pharmacy claim submissions. The extraction job and the live adjudication queries are now both competing for the same database resources during a window when the business assumed the system was idle. Claim response times degrade gradually enough that the connection to the nightly extraction job is not obvious from the symptom alone; pharmacy staff and plan members simply experience a system that has become slower over the past several months, with no single incident to point to. By the time the extraction team is asked to investigate, disentangling the accumulated effect of full-load extraction growth from other possible causes takes considerably longer than it would have taken to catch the trend early through simple extraction-duration monitoring. This is the specific, gradual failure mode that a change data capture migration exists to prevent rather than to fix after the fact.

The scale of the problem is worth stating plainly before the architecture discussion begins. Healthcare data volumes are growing at roughly 36 percent annually, the fastest compound annual growth rate of any industry sector tracked, a rate that outpaces most organizations' ability to integrate and analyze it [6]. IBM Research estimates that poor data quality costs the United States healthcare system more than 314 billion dollars annually, with a meaningful share of that figure attributable to integration failures between transactional and analytical systems rather than errors at the point of original data entry [7]. A 2023 industry survey found that two-thirds of health plans cite data integration latency as a primary obstacle to real-time population health analytics [8]. Those figures frame why the architecture choices in the sections that follow are not a purely academic exercise in database engineering; they determine whether population-health analytics programs actually run on current data or on data that is stale by the time it becomes actionable.

2. RELATED WORK: FOUNDATIONS AND CURRENT PRACTICE

The theoretical basis for changing data capture predates the term itself. Widom and Finkelstein's work on active database systems, published in the early 1990s, established the trigger-based mechanisms that let a database system detect and react to state changes without requiring an external process to poll for them [3]. That line of research answered a narrower question than this article addresses how a single database system can react to its own changes but it supplies the mechanical foundation that transaction-log mining, described in Section 4.2, still depends on today.

The commercial application of change-tracking to data warehousing followed a different path. Kimball and Ross's dimensional-modeling methodology introduced slowly changing dimensions as a structured way to maintain historical analytical data without refreshing an entire dimension table on every load cycle [2]. Inmon's treatment of data warehouse architecture, developed around the same period, established the broader case for separating operational and analytical systems in the first place—the organizational and technical justification for why a change-capture pipeline needs to exist at all [9]. Both remain foundational, specifically because the underlying engineering problem they describe, incrementally maintaining an analytical copy of operational data, has not changed even as the tooling built to solve it has moved several generations forward.

Contemporary practitioner literature on change data capture has three main threads. The first treats transaction-log mining as the preferred mechanism wherever the source database platform supports it, on the reasoning that reading a database's own recovery log imposes less load on the operational system than repeated delta queries do. Debezium has become something close to a reference implementation for this approach, providing log-based capture across Oracle, Microsoft SQL Server, PostgreSQL, and MySQL through a single open-source platform [10]. The second thread treats change data capture as an ETL automation problem in its own right. Ravi's recent treatment of ETL automation patterns catalogs the operational discipline (scheduling, monitoring, failure recovery) that a production incremental pipeline needs regardless of which detection mechanism it uses [11]. The third thread, more recent still, folds machine learning into the pipeline itself: Saraswat, Varshney, and Vashist describe an ETL pipeline purpose-built for electronic health record data that feeds directly into downstream prediction models [12], while Sökkula combines blockchain-based audit logging with a neural anomaly-detection layer to harden ETL pipeline security [13].

What this literature does not treat with much depth is the specific combination this article is built around: change data capture against a source system that is simultaneously massive in scale, operationally latency-sensitive in a way that directly affects patient care delivery, and subject to healthcare-specific regulatory constraints that most general ETL security literature discusses only at the storage layer rather than across the full extraction pipeline. Singu's analysis of data security and compliance in ETL processes for healthcare and financial services comes closest, arguing that compliance requirements must be built into the pipeline architecture rather than added later [14], but it does not address the scale-specific engineering problems of index design under billion-record volumes, log retention sizing, and delta-set completeness validation that dominate Sections 4 and 5 of this article. That combination of extreme scale, operational sensitivity, and regulatory constraint is the specific gap this article's framework addresses.

A related but distinct thread in this literature treats real-time streaming architecture as a general-purpose pattern rather than one tailored to a specific source-system constraint. Kreps, Narkhede, and Rao's original description of log-structured distributed messaging [15], the conceptual basis for platforms such as Apache Kafka, discussed further in Section 4.3, was written to solve a general problem of decoupling event producers from consumers at scale, not a healthcare-specific one. That generality is precisely why it transfers cleanly into the event-driven tier of this article's framework, but it also means the messaging layer literature by itself offers no guidance on the healthcare-specific questions this article treats as central: how the events entering that stream should be detected from an adjudication system that may not natively expose a change feed and how those events satisfy audit trail obligations once they leave the source environment. The messaging pattern and the change-detection mechanism that feeds it are frequently discussed as though they were the same problem; this article treats them as two separate engineering decisions that happen to compose well together in a mature Tier 3 implementation.

3. METHODOLOGY: PRACTITIONER-DERIVED EVALUATION FRAMEWORK

The framework presented here comes from synthesizing integration architecture work across high-volume claims-adjudication and healthcare data-warehouse engagements, and is cross-checked against the published change data capture and data-quality research cited throughout this article. It is a practitioner-derived synthesis rather than a controlled experiment: the tier descriptions and the five evaluation dimensions reflect engineering decisions and trade-offs that recurred across engagements, not a single instrumented study with a fixed experimental protocol. Concretely, this synthesis draws on more than a dozen high-volume claims-adjudication and data-warehouse integration engagements undertaken over approximately six years, primarily in pharmacy benefits management and healthcare claims contexts; a given architectural trade-off was treated as a general pattern in this framework once it recurred across at least three independent engagements rather than being generalized from a single client's experience.

This methodological choice carries the same limitation any practitioner-derived synthesis does: it reflects the engagements it was drawn from. Those engagements concentrated on adjudication systems built on established relational database platforms, predominantly Oracle and Microsoft SQL Server, that operated at or above the billion-record threshold motivating this article's scope. Organizations running substantially smaller adjudication databases, or built on database platforms without mature transaction-log access, may find the three tiers still apply conceptually but require different specifics than the ones described here.

Judging where a given implementation sits within this framework requires evaluation dimensions specific to the extraction problem rather than to database performance in the abstract. Extraction impact measures how much additional load the change-capture mechanism places on the operational source system the dimension that matters most for adjudication databases specifically, since operational degradation here has a direct patient-care consequence rather than merely an inconvenience. Change detection completeness is the probability that a given extraction cycle actually captures every record that changed, rather than merely most of them; a completeness gap that goes unnoticed produces an analytical dataset that looks complete while silently missing records. Change detection accuracy is the converse concern the probability that a record flagged as changed genuinely did change, since a mechanism prone to false positives wastes downstream processing capacity independent of how reliably it captures genuine changes. Pipeline latency is the delay between a change occurring in the source system and its availability in the target analytical environment. Implementation complexity captures both the initial engineering effort and the ongoing operational burden of keeping the mechanism running correctly.

These five dimensions are scored qualitatively High, Medium, or Low against each tier in Section 4, because the underlying engagements differed enough in transaction volume, database platform, and regulatory environment that a single quantitative instrument applied uniformly across them would imply a precision the evidence does not actually support. A qualitative comparison communicates relative standing without overstating what a practitioner-derived synthesis can responsibly claim to measure.

The five dimensions are not independent, and a tier-selection decision that optimizes one without regard for the others tends to produce disappointing results in practice. Pushing pipeline latency toward Tier 3's near-real-time delivery, for instance, increases the coordination burden described in Section 4.3 and raises implementation complexity in a way that a team evaluating tiers purely on latency requirements can easily underestimate going in. Similarly, minimizing extraction impact by moving fully to transaction-log mining does not automatically improve change detection completeness on its own a log-mining implementation with an undersized retention window can still lose changes during a pipeline outage, just through a different mechanism than the timestamp-column gap that affects Tier 1. None of this means the five dimensions trade off against each other in every case; Table 1 shows Tier 2 and Tier 3 both scoring High on completeness and accuracy simultaneously. It does mean that a team selecting a tier for one dimension's sake should expect to spend deliberate design effort on the others rather than assuming gains in one dimension carry over automatically.

4. TIERED ARCHITECTURE FRAMEWORK

4.1 Tier 1 Query-Time Delta Detection

The foundational tier identifies changed records through direct database queries that compare current state against a reference baseline, most commonly using high-watermark timestamp comparison: each extraction cycle selects records whose last-modified timestamp exceeds the timestamp recorded at the end of the previous cycle. This is the lowest-complexity tier to implement, and it remains the only option available in a meaningful share of engagements not every legacy adjudication platform exposes richer change-tracking mechanisms, and timestamp comparison against an existing column can often be implemented without any change to the source system's schema.

At a billion-record scale, that simplicity comes with a sharp edge. A delta query without a covering index on the timestamp column, one that includes the timestamp predicate and every column the extraction needs to read, degrades to a full table scan, which costs the same as the full-load extraction the incremental approach exists to avoid. Index design at this tier has to balance extraction efficiency against the index-maintenance overhead that the index itself places on a high-throughput transactional table, since every additional index on a heavily written table adds write-path cost that the operational claims-processing workload absorbs directly.

Timestamp-based detection also depends entirely on the source system reliably updating the last-modified column on every write that should count as a change. Any write path that bypasses that column—most commonly a direct database correction applied during an incident or an administrative batch update that touches the underlying tables without going through the application layer that normally maintains the timestamp—creates a change that the delta query will not detect. In an adjudication environment where administrative corrections to claim records are not rare, this scenario is not a theoretical edge case; it is a completeness gap that a Tier 1 implementation has to actively guard against, typically through periodic reconciliation queries that compare aggregate counts between source and target rather than relying on the delta mechanism alone to catch every case.

A reconciliation query designed for this purpose is deliberately simpler than the delta query it is checking. Rather than attempting to identify which specific records the delta mechanism might have missed, an exercise that risks reproducing the same blind spot, an effective reconciliation check compares a small set of aggregate figures between source and target on a recurring schedule: total record counts within a defined date range, sum totals on key numeric fields such as claim payment amounts, and a count of distinct claim identifiers. A mismatch in any of these aggregates does not by itself identify the missing record, but it does reliably signal that one exists, which is enough to trigger a targeted investigation before the gap accumulates across multiple extraction cycles. Running this reconciliation on a schedule independent of the delta extraction itself, rather than treating a clean delta-query execution as sufficient evidence that nothing was missed, is what actually closes the blind spot that timestamp-based detection cannot close on its own.

4.2 Tier 2 Database-Level Transaction Log Mining

Transaction log mining reads the database's own transaction log, the sequential record every relational database management system maintains for crash recovery and replication, to capture changes with substantially less operational impact than repeated delta queries. Because the log is read sequentially rather than queried against the live tables, this approach does not compete with operational transactions for the same locks and index structures that a tier 1 delta query does, which is precisely why it is the preferred tier for adjudication systems where extraction impact on the operational workload is the dominant concern.

TABLE 1: Tier Comparison Across the Five Evaluation Dimensions

Tier	Extraction Impact	Change Detection Completeness	Change Detection Accuracy	Pipeline Latency	Implementation Complexity
-------------	--------------------------	--------------------------------------	----------------------------------	-------------------------	----------------------------------

Tier 1: Query-Time Delta Detection	Medium-High	Medium	Medium	Minutes to hours	Low
Tier 2: Transaction Log Mining	Low	High	High	Seconds to minutes	Medium
Tier 3: Event-Driven Real-Time Capture	Low	High	High	Seconds	High

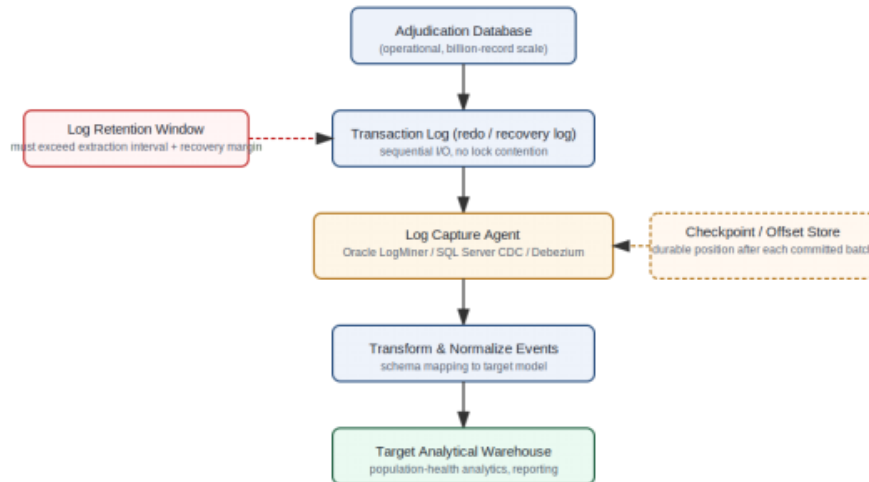


Figure 1. Tier 2 Transaction-Log-Mining Pipeline.

For Oracle-based adjudication systems, Oracle LogMiner provides programmatic access to the redo log stream without requiring changes to the source schema [16]. For Microsoft SQL Server environments, the platform's own Change Data Capture feature maintains change tables automatically once enabled, which shifts some of the operational burden from the extraction pipeline to the database engine itself [17]. Debezium sits a layer above both, offering a single open-source platform that wraps log-mining mechanisms across several database engines behind a common event-streaming interface, which is part of why it has become a common default choice when an organization's source platform already supports log-based capture [10]. Figure 1 traces the resulting pipeline: the capture agent reads the source database's transaction log, transforms committed changes into a normalized event format, and delivers them to the target warehouse, a path that does not touch the claims-processing workload's dependent operational tables directly.

This tier's principal operational cost is log retention management. The transaction log has to be retained for long enough to cover the interval between successful extraction cycles plus a safety margin for failure recovery, and retaining an adjudication system's full transaction log for an extended window carries its storage and, in a regulated environment, audit-scope cost. Sizing that retention window correctly long enough to survive a realistic extraction-pipeline outage, but not so long that log storage becomes its own unmanaged liability, is a design decision every Tier 2 implementation has to make explicitly rather than defaulting to a vendor-recommended retention period without checking whether it actually matches the pipeline's own recovery-time requirements.

Failure recovery deserves the same explicit design attention as the steady-state pipeline. When a Tier 2 extraction pipeline goes down for a period longer than expected, causing a downstream target-warehouse outage, a

capture-agent failure, or a network partition between the source and target environments, the recovery path depends on the transaction log still covering the entire outage window. A pipeline that resumes cleanly from exactly where it left off, replaying every committed change from the log in its original order, is straightforward to reason about. A pipeline that has to reconstruct its position in the log after a failure, without a reliable checkpoint of the last successfully processed transaction, risks either replaying changes that were already applied or skipping changes that were not. Establishing a durable checkpoint after each successfully committed batch, rather than only at the end of a full extraction cycle, is a comparatively small implementation cost that materially reduces the risk of either failure mode during an actual outage.

4.3 Tier 3 Event-Driven Real-Time Capture

The highest tier propagates changes to the analytical environment as discrete events within seconds of their occurrence in the operational system, using an Enterprise Service Bus (ESB) or message-queue architecture to decouple change detection from the downstream consumers that act on it. This tier enables use cases that the batch-oriented cycles of Tiers 1 and 2 cannot support at all, such as live dashboard reporting, intraday drug-utilization monitoring, and real-time fraud detection that depends on seeing a suspicious claim pattern within minutes rather than discovering it the next morning.

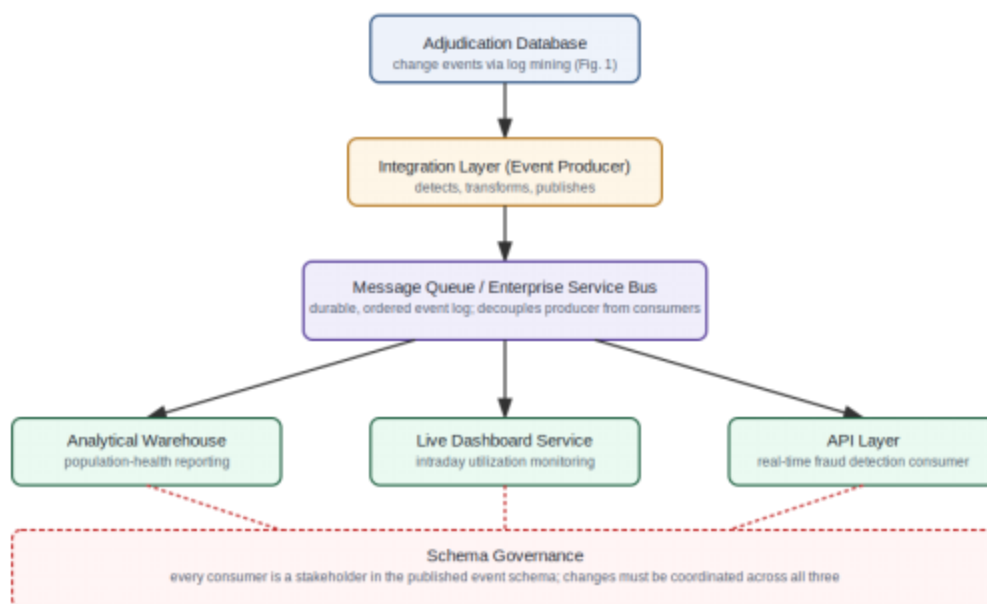


Figure 2. Tier 3 Event-Driven Real-Time Capture with Message-Queue Decoupling.

Figure 2 shows the resulting architecture: the integration layer subscribes to change events at the source typically produced by the same transaction-log-mining mechanism described in Section 4.2, rather than a separate detection method, and publishes transformed events to a message queue for consumption by any number of downstream analytical systems, reporting services, and Application Programming Interface (API) layers, each consuming independently and at its own pace. Kreps, Narkhede, and Rao's original description of the log-structured messaging pattern underlying platforms such as Apache Kafka [15] remains the conceptual basis for this decoupling: the message queue durably records every event exactly once, and downstream consumers can be added, removed, or restarted without the producing side needing to know anything about who is currently listening.

The cost of this tier is coordination, not raw engineering effort. Every downstream consumer who subscribes to the event stream becomes a stakeholder in the schema of the events being published, which means a schema change on the source side now has to be communicated to and absorbed by every consuming system rather than contained within a single extraction job. Organizations adopting Tier 3 for the first time consistently underestimate this coordination cost relative to the technical build itself. The message queue infrastructure is comparatively well-documented and available as a managed service in most cloud environments, but the organizational discipline of schema governance across an expanding set of independent consumers has to be built deliberately rather than assumed to emerge on its own.

Validating an event-driven capture implementation before it carries production traffic requires exercising failure paths that a functional demonstration of the happy path will not surface. A useful pre-go-live test suite for this tier includes deliberately stopping and restarting a downstream consumer mid-stream to confirm it resumes from the correct offset rather than either reprocessing already-consumed events or silently skipping ahead; publishing a malformed or unexpected event shape to confirm consuming systems reject or quarantine it rather than failing in a way that halts the entire stream; and a sustained-load test confirming that message-queue throughput holds up during a claims-volume spike rather than only under the moderate load typically used for functional testing. Skipping this category of testing in favor of validating only straightforward single-consumer scenarios is a common shortcut under delivery-timeline pressure, and it is also the shortcut most likely to produce a defect that surfaces only once a second or third downstream consumer is added in production.

5. DATA QUALITY VALIDATION FOR DELTA RECORD SETS

Whichever tier produces the delta record set, that set requires its own validation before loading a step distinct from, and in addition to, whatever transformation logic maps the delta into the target schema. A change data capture cycle can correctly identify every record that changed and still produce a delta set that is duplicated, out of sequence, or missing required fields due to an upstream extraction failure, which means the cycle has technically succeeded at change detection while still handing the target warehouse a wrong dataset. Ballou and Tayi's treatment of data quality in warehouse environments, though written before change data capture at this scale was a mainstream architectural pattern, still frames the underlying issue correctly: a data quality defect that enters at the extraction stage propagates into every downstream analytical use of that data, and the cost of catching it early is consistently lower than the cost of catching it after an analyst has already built a report on top of it [18].

Two categories of check are necessary companions to delta detection rather than optional refinements. Completeness checks compare the expected change volume derived from source-system transaction counts over the extraction window against the actual delta record count the pipeline produced, catching the case where a detection mechanism silently under-captures without raising any error. Ordering checks confirm that updates apply to the target system in the sequence they occurred in the source, which matters specifically because an out-of-order delta application can overwrite a newer value with an older one and leave the target dataset silently wrong in a way that a simple row-count check will not catch.

Both categories of check also intersect with the broader data governance discipline that established data-management practice treats as a baseline expectation for any organization maintaining an analytical warehouse documented data lineage, defined stewardship for data-quality exceptions, and a consistent standard for what "complete" and "accurate" mean for a given dataset, rather than an informal understanding that varies by whoever built the original pipeline [19]. Framing the completeness and ordering checks as an application of that broader governance discipline, rather than as pipeline-specific engineering choices invented from scratch, makes it easier to justify the validation investment to stakeholders who are not themselves reviewing the extraction code.

In a regulated healthcare context, these checks do double duty. The same completeness and ordering validation that protects analytical accuracy also satisfies the audit-trail obligations that HIPAA and PIPEDA impose on data handling more broadly. A pipeline that can demonstrate, after the fact, that every changed record in a given period was captured, in the order it occurred, with an unbroken chain of custody from the adjudication system to the analytical

warehouse answers both the engineering question and the regulatory one with the same evidence. Building these checks as a compliance artifact from the outset, rather than as an engineering nice-to-have that happens to also satisfy an auditor later, tends to produce a more defensible pipeline than treating the two requirements as separate work.

6. CROSS-CONTEXT APPLICABILITY

The engineering constraints this article develops around healthcare claims data are not, in their essentials, specific to healthcare. Any organization running a high-volume operational database that also has to feed an analytical environment faces some version of the same tension between extraction impact and analytical currency the specifics of what makes the source system latency-sensitive differ by industry, but the architectural response looks similar across contexts. Table 2 summarizes the dominant tier and primary constraint driver across five representative industries.

TABLE 2: Cross-Context Applicability Matrix

Industry	Dominant Tier	Primary Constraint Driver
Healthcare claims adjudication	Tier 2 (Transaction Log Mining)	Regulatory audit-trail obligations combined with sub-second operational latency requirements
Financial services (trading, banking transactions)	Tier 3 (Event-Driven Real-Time Capture)	Real-time fraud detection and regulatory transaction reporting windows
Telecommunications (billing, call detail records)	Tier 2 (Transaction Log Mining)	Extreme per-day record volume with moderate latency tolerance
Retail (point-of-sale transaction history)	Tier 1 to Tier 2 (scale-dependent)	Seasonal volume spikes; latency tolerance varies by use case
Government (tax and benefits administration)	Tier 1 to Tier 2 (scale-dependent)	Legacy platform constraints; batch-oriented reporting cycles remain acceptable

Financial services trading and banking-transaction systems tend to gravitate toward Tier 3 specifically because fraud detection and regulatory transaction-reporting windows create the same real-time pressure that drives adjudication systems toward event-driven capture, but for a different underlying reason: the cost of a missed fraud signal rather than the cost of a delayed population-health insight. Telecommunications billing and call-detail-record systems generate per-day volumes that can rival or exceed healthcare claims volume, but typically tolerate longer pipeline latency, which keeps Tier 2 an adequate and lower-complexity choice in many telecommunications contexts even at extreme scale. Retail point-of-sale history and government tax and benefits administration systems both show more variation; the right tier depends heavily on the specific platform's scale and how tolerant the organization's reporting cycle actually is of latency, which is consistent with this framework's own premise that tier selection is a fit decision rather than a race to the highest tier available.

The three tiers also differ in what they demand of the team that maintains them once the initial build is complete, which matters as much for staffing a data engineering program as the technical maturity scores in Table 1 do. Tier 1 concentrates its ongoing burden on index and query maintenance. The team needs to monitor delta-query execution plans as data volume grows and adjust indexing before a query silently degrades to a full scan, but the mechanism itself requires little day-to-day intervention once tuned correctly. Tier 2 shifts the burden toward log-retention and failure-recovery discipline. The team maintaining a transaction-log-mining pipeline needs enough platform-specific expertise to configure and monitor log retention correctly and needs to periodically rehearse the failure-recovery path described in Section 4.2 rather than assuming it works because it has not yet been tested under a real outage. Tier 3 concentrates its burden on schema governance across an expanding consumer base, which is as much an organizational coordination skill as a technical one, and tends to be the tier where a team's readiness matters more than the readiness

of the technology itself, since the message-queue infrastructure underlying most Tier 3 implementations is mature and well-documented in a way that the surrounding governance discipline usually is not.

7. DISCUSSION: BROADER IMPLICATIONS

Change data capture's computational efficiency has a direct environmental dimension. A full-load extraction reprocesses an entire billion-record dataset regardless of how much of it actually changed, which means its computational cost stays roughly constant even as the change rate, the genuinely useful fraction of the work, shrinks over the life of a mature system. Gartner's benchmark finding that organizations with mature change data capture implementations cut data-integration processing time by 70 to 90 percent compared to full-load alternatives [20] is, in effect, a measure of how much computation a mature incremental pipeline avoids performing at all; that avoided computation corresponds directly to avoided energy consumption and infrastructure footprint, though the exact conversion from processing-time reduction to energy savings varies by cloud provider pricing model and by how aggressively an organization right-sizes its extraction infrastructure once the incremental approach is in place.

The population-health analytics case for getting this architecture right is not abstract. Chronic disease management programs, medication-adherence monitoring, and opioid-utilization surveillance all depend on claims data reaching the analytical environment while it is still current enough to support a clinical intervention rather than merely a retrospective report. A change data capture pipeline that adds days of latency to that data flow not only produces a less impressive dashboard but also risks identifying a care gap or an adherence lapse too late for the intended intervention to be useful. Getting the underlying integration architecture right is, in that sense, an operational precondition for the analytics program's stated purpose, not merely infrastructure that sits underneath it.

Machine learning is beginning to be layered onto change data capture pipelines themselves, most visibly for anomaly detection flagging when a given extraction cycle's delta volume, change-rate distribution, or completeness signal deviates from historical patterns in a way that suggests a pipeline failure rather than a genuine shift in claims activity. Sokkula's combination of blockchain-based audit logging with a neural anomaly-detection layer is an early example of this direction applied specifically to ETL pipeline security [13]. These capabilities are still maturing, and this article does not treat them as a substitute for the completeness and ordering checks described in Section 5 an anomaly-detection layer that flags an unusual delta pattern is a useful complement to explicit validation logic, not a replacement for it, since a validation check that fails deterministically is more trustworthy than a statistical model's judgment that something looks unusual [21].

Organizations operating full-load extraction against adjudication or similarly high-volume databases exceeding roughly a hundred million records should treat query-time delta detection as an immediate efficiency intervention worth prioritizing, with a roadmap toward transaction log mining once the operational impact of even a well-indexed delta query becomes a documented constraint rather than a hypothetical concern [22]. Waiting until a full-load extraction visibly degrades operational claims-processing performance before beginning that transition tends to leave less runway for a careful migration than starting the tier-selection exercise while the constraint is still a projection rather than an incident [23].

The workforce dimension of this shift deserves direct mention, since it is frequently underweighted relative to the architecture decisions that receive most of the planning attention. A data engineering team that has spent years operating scheduled full-load extraction jobs brings deep expertise in batch scheduling, job-dependency management, and batch failure recovery expertise that transfers only partially to a transaction-log-mining or event-driven environment, where the operational concerns shift toward log-retention sizing, consumer-offset management, and schema-change coordination across independent downstream systems [24]. Programs that budget for the technical migration but not for this retraining curve tend to discover the gap at the least convenient moment: during the cutover window, when the team most needs to interpret an unfamiliar failure mode correctly and is instead encountering it for the first time under delivery pressure. Building retraining time into the migration schedule itself, rather than assuming the existing team absorbs the new operational model incidentally while also delivering the migration, is a planning discipline that this article's engagements suggest is applied unevenly at best [25].

8. CONCLUSION

Change data capture is usually presented as a general-purpose data warehouse optimization, and at a moderate scale it is one. At billion-record scale in a regulated, operationally latency-sensitive environment such as healthcare claims adjudication, it is closer to a structural requirement than an optimization—the point past which a full-load extraction stops being merely inefficient and starts actively competing with the operational workload it sits alongside. The three-tier framework and five-dimension evaluation developed in this article are meant as a structured way to make that tier-selection decision deliberately, informed by what a specific source platform actually supports and what a specific analytical use case actually requires, rather than by default or by whichever tier happened to be easiest to build first.

None of the three tiers is a new invention; query-time delta detection, transaction-log mining, and event-driven capture all appear, in some form, throughout the change data capture and data-integration literature this article cites. What this article adds is the specific framing of these three tiers against the engineering constraints of billion-record healthcare claims environments, extraction impact on a patient-care-critical operational system, regulatory audit-trail obligations that extend across the full pipeline rather than only final storage, and adjudication data models that resist simple change-indicator logic and the observation, developed in Section 6, that those same constraints recur in recognizable form across financial services, telecommunications, retail, and government contexts even where the specific regulatory and latency drivers differ. Practitioners evaluating their change data capture maturity are better served by working through this framework's five dimensions explicitly, against their own source system's actual constraints, than by assuming the highest-maturity tier is automatically the right target for every environment.

9. ETHICS STATEMENT

Conflict of Interest: The author declares no conflict of interest.

Statement of Human and Animal Rights: This study did not involve human participants, animal subjects, or identifiable personal data requiring ethics board review. No claims-level or patient-level data are presented; all volume and cost figures are drawn from cited published sources.

Informed Consent: Not applicable.

10. FUNDING STATEMENT

This research received no external funding.

11. BIOGRAPHICAL SKETCH

Saikrishna Ala is a data integration and ETL architecture consultant with more than ten years of experience designing and delivering enterprise data integration programs, including full and incremental claims-data load architectures for healthcare claims adjudication and pharmacy benefits management clients. His work spans Talend Data Fabric, Hadoop-based data lake platforms, and cloud data warehouse environments, with a recurring focus on the change-data-capture and incremental-loading engineering that lets high-volume operational systems feed analytical environments without disrupting the operational workload itself.

12. CREDIT AUTHOR CONTRIBUTIONS

Saikrishna Ala: Conceptualization; Methodology; Investigation; Writing - Original Draft; Writing - Review and Editing; Visualization.

13. DATA AVAILABILITY STATEMENT

No new datasets were generated or analyzed in this study. All cited figures and findings are drawn from the referenced published sources.

14. ACKNOWLEDGMENTS

The author used Claude (Anthropic, claude-sonnet model) to assist with manuscript drafting and language refinement. The author takes full responsibility for the accuracy, originality, and integrity of all content presented in this manuscript and confirms that all data, analysis, and conclusions reflect independent work and verified sources.

REFERENCES

1. Drug Channels Institute, "The Top Pharmacy Benefit Managers of 2025," Drug Channels, 2026. [Online]. Available: <https://www.drugchannels.net>
2. R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed. Wiley, 2013.
3. J. Widom and S. Finkelstein, "Set-oriented production rules in relational database systems," in *Proc. ACM SIGMOD International Conference on Management of Data*, 1990.
4. U.S. Department of Health and Human Services, "HIPAA Administrative Simplification," HHS, 2023. [Online]. Available: <https://www.hhs.gov/hipaa>
5. Government of Canada, "Personal Information Protection and Electronic Documents Act (PIPEDA)," Department of Justice Canada, 2023.
6. D. Reinsel, J. Gantz, and J. Rydning, "The Digitization of the World: From Edge to Core," IDC White Paper (Data Age 2025), sponsored by Seagate, 2018.
7. IBM Research, "The Cost of Poor Health Data Quality," IBM Institute for Business Value, 2023.
8. KPMG, "Healthcare Analytics Survey: Real-Time Data Integration," KPMG Healthcare Practice, 2023.
9. W. H. Inmon, *Building the Data Warehouse*, 4th ed. Wiley, 2005.
10. Debezium Contributors, "Debezium Documentation," Red Hat / Debezium Open Source, 2024. [Online]. Available: <https://debezium.io/documentation/>
11. C. Ravi, "ETL (Extract, Transform & Load) Automation," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 6, no. 1, pp. 52-55, 2025.
12. B. K. Saraswat, N. Varshney, and P. C. Vashist, "Development of ETL pipeline for Electronic Health Record to support Machine Learning based Approaches for Security and Prediction," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 19s, pp. 168-189, 2024.
13. M. R. Sokkula, "Integrating Blockchain and AI for Data Encryption and Secure ETL Pipelines," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 13, no. 1, pp. 395-406, 2025.
14. S. K. Singu, "Ensuring Data Security and Compliance in ETL Processes for Healthcare and Financial Services," *International Journal of Computing and Engineering*, vol. 6, no. 7, pp. 1-11, 2024.
15. J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB Workshop*, 2011.
16. Oracle Corporation, "Oracle LogMiner Administrator's Guide," Oracle Database 19c Documentation, 2024. [Online]. Available: <https://docs.oracle.com>
17. Microsoft, "About Change Data Capture (SQL Server)," Microsoft SQL Server Documentation, 2024. [Online]. Available: <https://docs.microsoft.com/sql/relational-databases/track-changes>
18. D. P. Ballou and G. K. Tayi, "Enhancing data quality in data warehouse environments," *Communications of the ACM*, vol. 42, no. 1, pp. 73-78, 1999.
19. DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed. Technics Publications, 2017.
20. Gartner, "Best Practices for Implementing Change Data Capture," Gartner Research, 2023.
21. [21] H. Gaggar, "PromptDLP: Deterministic Pre-Inference Egress Control for Preventing Secret and PII Leakage in LLM Agents," *Journal of Information Systems Engineering and Management*, vol. 9, no. 1, 2024. [Online]. Available: https://www.jisem-journal.com/download/59_JISEM_9_1.pdf
22. [22] C. T. Bobba, "Advancements in Cosmetic Dentistry," *Innovation*, vol. 1, p. 2.
23. A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 12, no. 2, 2026. [Online]. Available: <https://doi.org/10.22399/ijcesen.5207>
24. [24] C. T. Bobba, R. Choksi, V. S. Grewal, H. K. Basra, and S. Momin, "Evaluation of Effects of Deproteinization and Intermediate Bonding on the Retention of Pit and Fissure Sealants-An In-Vivo Study," *SVOA Dentistry*, vol. 2, no. 4, pp. 139-142, 2021.
25. [25] H. Gaggar, "Two Planes of Truth: Execution Fidelity and BehavioralCorrectness in Tool-Using LLM Agents," *International Journal of Computational and Experimental Science and Engineering*, vol. 10, no. 3, 2024. [Online]. Available: <https://doi.org/10.22399/ijcesen.5498>