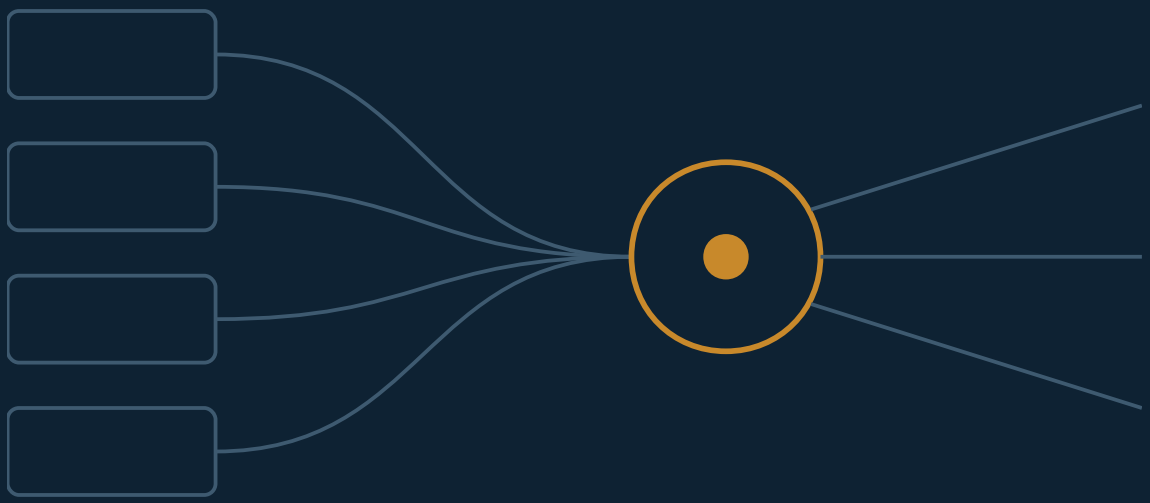
REAL-WORLD ARCHITECTURES

VOLUME 1

Single View

One Governed View of Customers and Citizens



MARIO NOIOSO

FIELD GUIDES ON DATA, SYSTEMS AND INSTITUTIONS

REAL-WORLD ARCHITECTURES · VOLUME 1

Single View

One Governed View of Customers and Citizens

Mario Noioso

Software architect and author · Rome

Free edition · not for sale · CC BY-NC-ND 4.0

Real-World Architectures, Volume 1
Single View: One Governed View of Customers and Citizens
First edition, 2026. Version 1.1.
Distributed free of charge. Not for sale.

Author
Mario Noioso · ORCID 0009-0000-9052-944X · ISNI 0000 0004 2128 1497
https://marionoioso.com

Licence
© 2026 Mario Noioso. This work is licensed under the Creative Commons Attribution, NonCommercial, NoDerivatives 4.0 International licence (CC BY-NC-ND 4.0). You may share it freely, in unmodified form and with attribution, for non-commercial purposes.

Disclaimer
The views expressed in this guide are the author's own and do not represent those of his employer or of any organization he works or has worked with. This guide is an independent, non-commercial publication, distributed free of charge, and is not sponsored, reviewed or endorsed by MongoDB, Inc. MongoDB is a trademark of MongoDB, Inc.; other product names are trademarks of their respective owners and are used for identification only. Industry scenarios and the worked example are composites built from common architectural patterns and do not describe any specific organization. This guide is provided for information and does not constitute legal, regulatory or professional advice for any specific system.

CONTENTS

About this guide 5

1 Executive summary 6

2 The fragmentation problem 7

3 What a Single View is, and what it is not 9

4 Why the document model fits 11

5 Reference architecture 13

6 The data model 17

7 Identity resolution 22

8 Governance, privacy and sovereignty 25

9 Industry patterns 28

10 A worked example 30

11 Anti-patterns 32

12 Adoption roadmap and checklist 34

13 References 36

14 About the author 37

Free edition · not for sale · CC BY-NC-ND 4.0

PREFACE

About this guide

Audience

Enterprise and solution architects, data platform leads, technical decision makers in banks, insurers, telecom operators, retailers, healthcare organizations and public administrations, and the delivery teams who build Single View platforms for them.

Prerequisites

Familiarity with enterprise integration (change data capture, event streaming, APIs), with basic data modelling, and with the general principles of data protection regulation. Prior experience with document databases helps but is not required.

Scope and limits

The guide describes an architectural pattern and the decisions that make it work in production. It is not a product manual, a sizing guide or a legal opinion. Regulatory references are illustrative: obligations depend on jurisdiction, sector and the specific processing, and must be assessed with the organization's own legal and data protection functions.

Environment and verification status

The reference implementation uses MongoDB 8.0 syntax. Search examples use MongoDB Search (Atlas Search) syntax. Unless stated otherwise, capabilities are described as available in MongoDB Atlas; availability in self-managed deployments depends on edition and version, summarized in Chapter 4. Code listings in this edition have been reviewed statically for syntax and consistency; they have not yet been executed against a declared reference environment. Readers should test them in their own environment before relying on them, and corrections are welcome.

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 1

Executive summary

Large organizations already hold much of the information they need about the people they serve. That information is fragmented, incomplete and sometimes contradictory.

A bank holds a customer's identity in one system, her accounts in another, her mortgage in a third and her complaints in a fourth. A public administration holds a citizen's residence in a population registry, his tax position in a revenue system, his benefits in a welfare platform and his health records in a separate regional network. Each system describes its own slice, with its own gaps and errors. None of them describes the person as a whole.

A Single View is the architectural answer to that gap: a continuously updated, governed, queryable representation of each customer or citizen, assembled from the systems that already hold the data, and served to the channels, processes and applications that need it. It does not replace the existing authoritative systems, which keep ownership of their data and of the transactions that change it. It can, however, take ownership of attributes explicitly assigned to it, typically new engagement data that no legacy system holds. It is neither a data warehouse nor a cache: it is an operational layer between the systems that own the data and the systems that use it.

This guide explains how to build one that survives production. It covers the problem, the reference architecture, the data model, the hardest part (deciding which records describe the same person), the governance obligations that come with concentrating personal data, a worked example, and the mistakes that sink many programmes.

> THE ARGUMENT IN FIVE POINTS
>
> 1. Fragmentation is the natural result of decades of product-by-product investment. It is rarely fixed by replacing the sources.
> 2. The Single View works when it is built as an operational data layer: fed continuously by change data capture and events, read by many consumers, owned by a clear team, and honest about freshness.
> 3. The document model fits because an operational profile maps naturally to a document, grouping the attributes that are commonly accessed together.
> 4. Identity resolution and lineage, not storage, decide whether the Single View is trusted. Values must be traceable to their source, and merges must be reversible.
> 5. Concentrating personal data concentrates obligations. Privacy, consent, residency and erasure must be designed into the data model and into every access path from the first release.

Organizations that get this right gain more than a dashboard. They gain a shared, governed memory of the people they serve: consistent answers across channels, services designed around life events rather than internal departments, and a controlled foundation on which analytics and AI can operate.

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 2

The fragmentation problem

Silos are not built by bad architects. They are built by good projects, one at a time.

Fragmentation accumulates through reasonable decisions. A new product line arrives with its own platform. An acquisition brings a second core system. A regulatory deadline is met with a dedicated application. A digital channel is launched quickly on its own database. Twenty years later, the organization runs dozens of systems that each describe the same people, in different formats, with different identifiers and different update cycles.

How fragmentation shows up

- The contact centre agent opens several screens to answer one question, and the customer repeats the same information on every call.
- Digital channels are slower or less complete than the branch, because they query systems that were not designed for interactive traffic.
- New initiatives build their own local copy of customer data, their own synchronization jobs and their own cache. The number of copies can grow faster than the number of products.
- Legacy systems of record, often mainframes, carry an increasing load of read-only queries that consume capacity, licences and change windows.
- Compliance questions such as "what do we hold about this person, and on what legal basis" take a long time to answer and may be answered differently by different departments.
- In the public sector, citizens are asked repeatedly for information the administration already holds, against the spirit of the once-only principle.

Why replacing the sources is rarely the answer

The instinctive response is to consolidate the systems of record themselves. In practice this is seldom feasible within a reasonable horizon. Core systems carry decades of business rules, certifications and integrations; replacing them is a multi-year programme with high execution risk, and the business usually cannot wait for it to finish before offering better service.

The Single View takes a different path. It leaves the systems of record in place as owners of their data, and builds a layer that assembles, reconciles and serves that data. It can deliver value early, reduce load on the sources, and create the conditions for their gradual modernization rather than demanding it upfront.

Free edition · not for sale · CC BY-NC-ND 4.0

The cost of doing nothing

The cost of fragmentation rarely appears as a single budget line, which is why it persists. It is spread across duplicated integration work, oversized legacy capacity, longer time to market, inconsistent answers to regulators and customers, and a reduced ability to use data for analytics and AI. Data quality is not the only factor in the quality of a model, but it is a decisive one: fragmented and contradictory inputs limit what any model can reliably do.

> A USEFUL DIAGNOSTIC
>
> Count how many teams maintain their own copy of customer or citizen data, and how many synchronization jobs keep those copies alive. In the author's experience, once there are multiple independently maintained copies, the organization is already paying for a Single View. It is simply paying for several partial ones instead of one governed one.

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 3

What a Single View is, and what it is not

Many programmes that struggle did not fail at building a Single View. They failed at agreeing on what it was.

A working definition

A Single View is an operational data layer that maintains a current, consolidated representation of each real-world party, customer or citizen, assembled from multiple source systems, continuously synchronized with them, traceable back to them, and served through governed interfaces to the applications and processes that need it.

Four properties distinguish it from adjacent concepts:

- **Operational**: it serves live channels and processes with interactive latency, not overnight reports.
- **Consolidated**: it resolves identities across sources. The goal is that, within a defined domain and governance perimeter, one person corresponds to one entry. In practice the platform must also handle provisional identities, duplicates not yet resolved and cases under review, and make their status visible.
- **Continuous**: it is updated as source data changes, typically within seconds, through change data capture and events, with a stated freshness contract.
- **Traceable**: values carry their provenance, so that consumers and auditors know where they came from and when.

Neighbouring patterns and their role in this architecture

The table describes the prevalent purpose of each pattern and the role it plays in the architecture proposed here. It does not describe the universal limits of any technology: many products in each category have broadened their scope over time.

Pattern	Prevalent purpose	Role in the proposed architecture
System of record	Owns data and executes the transactions that change it.	Remains the owner of its attributes. The Single View reads from it and, where agreed, routes changes back to it.
Master data management	Governs master data through stewardship workflows and golden-record rules.	Complementary. Matching and survivorship logic may live in an MDM tool or in the Single View; the Single View provides the operational serving layer either way.
Data warehouse	Historical and analytical reporting.	Different latency and access pattern. The warehouse may consume resolved identities and change events from the Single View where useful.
Data lake or lakehouse	Large-scale storage and processing for analytics and ML.	Complementary. Typically not the place for per-person, interactive operational reads.
Cache	Speed up reads of an underlying system.	The Single View may relieve load like a cache, but it adds identity resolution, lineage and governance, and has its own availability and freshness commitments.
Operational data layer	Serve operational reads from a consolidated copy of source data.	The Single View is an operational data layer specialized on parties: customers, citizens, patients, suppliers.

Read view, write-through, or system of engagement

Single Views evolve along a maturity path. Decide explicitly where each one sits, because ownership and consistency rules change at every step.

- **Read view.** Consumers only read. All changes happen in the sources and flow in through capture. This is the safest starting point and delivers much of the early value.
- **Write-through.** Selected changes, such as contact preferences, are accepted by the Single View's API and committed to the owning source through a defined contract. The source remains the owner; the Single View reflects the change once capture confirms it (Chapter 5).
- **System of engagement.** For attributes explicitly assigned to it, such as digital interactions, preferences or derived insights that no legacy system holds, the Single View becomes the system of record. The boundary of ownership is documented field by field.

> RULE OF THUMB
>
> Never let a field have two owners. For every attribute it must be possible to answer one question: which system is allowed to change it?

CHAPTER 4

Why the document model fits

An operational profile maps naturally to a document, grouping the attributes commonly accessed together.

A Single View can be built on many technologies. This guide uses a document database, MongoDB, as its reference implementation, because the shape of the problem maps closely onto the shape of the model. The patterns remain valid on other platforms; the effort required to implement them changes.

Four properties of the problem

THE DATA IS HIERARCHICAL

A customer has several identifiers, names, addresses, contact points, products, relationships, consents and preferences. In a normalized relational model these typically become separate tables, and reading the full profile requires joins across them, or denormalized views and caches maintained alongside. In a document model they can become nested fields and arrays inside one record, and the common request for the operational profile becomes a single indexed read.

THE DATA IS POLYMORPHIC

A retail customer and a corporate client, a resident and a non-resident, a current account and a mortgage carry different attributes. Documents of different shapes can coexist in the same collection, each carrying only the fields that apply, while sharing a common core that is indexed and validated.

THE DATA KEEPS CHANGING

New products, regulations and channels add attributes continuously. Additive changes can be absorbed without a blocking schema migration. Not all change is free: application contracts, validators, indexes and backfills of existing documents still need to be planned, versioned and deployed.

THE DATA IS READ FAR MORE THAN IT IS WRITTEN

Single Views typically serve many reads for every update. The model should be optimized for the dominant read paths: the work of assembling the profile is done once, when sources change, rather than at every query. Different consumers still receive different projections of the same document, according to their purpose (Chapter 8).

Free edition · not for sale · CC BY-NC-ND 4.0

Capabilities used in this guide and where they are available

Capability	Atlas	Self-managed	Notes and limits
Documents, indexes, JSON Schema validation, multi-document transactions	Yes	All editions	Transactions require a replica set or sharded cluster.
Change streams, with pre- and post-images	Yes	All editions	Pre- and post-images must be enabled per collection. Resumability is bounded by the oplog window.
Zone sharding for residency	Yes (Global Clusters)	All editions	Requires a shard key that includes the zone field. Backups and other components must be configured separately (Chapter 8).
MongoDB Search (full-text, fuzzy) and Vector Search	Yes	Depends on edition and version	Search indexes are eventually consistent with the collection. Fields encrypted client-side cannot be searched, and Search cannot be used on a collection that uses Queryable Encryption.
Client-side field-level encryption and Queryable Encryption	Yes	Automatic encryption: Enterprise. Explicit encryption: all editions	Queryable Encryption supports equality and, from 8.0, range queries on encrypted fields. A collection using it cannot be indexed for Search: identifiers protected this way need a separate collection.
Database auditing	Yes, dedicated tiers	Enterprise	Auditing successful reads requires explicit configuration and has a performance cost.
Kafka source and sink connectors	Yes	Separate component	Deployed in the organization's Kafka Connect infrastructure.

Availability changes over time. Verify current documentation for the edition and version you deploy.

Consistency of reads

Reads from the primary see the latest acknowledged writes. Reads from secondaries or analytics nodes scale read capacity but may be slightly behind; where a consumer needs to read its own writes in the Single View, use causally consistent sessions with majority read and write concerns, or read from the primary. This is distinct from the freshness of the Single View relative to its sources, discussed in Chapter 5.

None of these capabilities is unique in isolation. What matters in a Single View programme is having them in one platform and one data model, so that search, events, encryption and residency are properties of the same record rather than separate systems to keep in sync. Where they conflict, notably search and client-side encryption, the trade-off must be designed explicitly.

CHAPTER 5

Reference architecture

A Single View is not a database. It is a pipeline with a database at its centre.

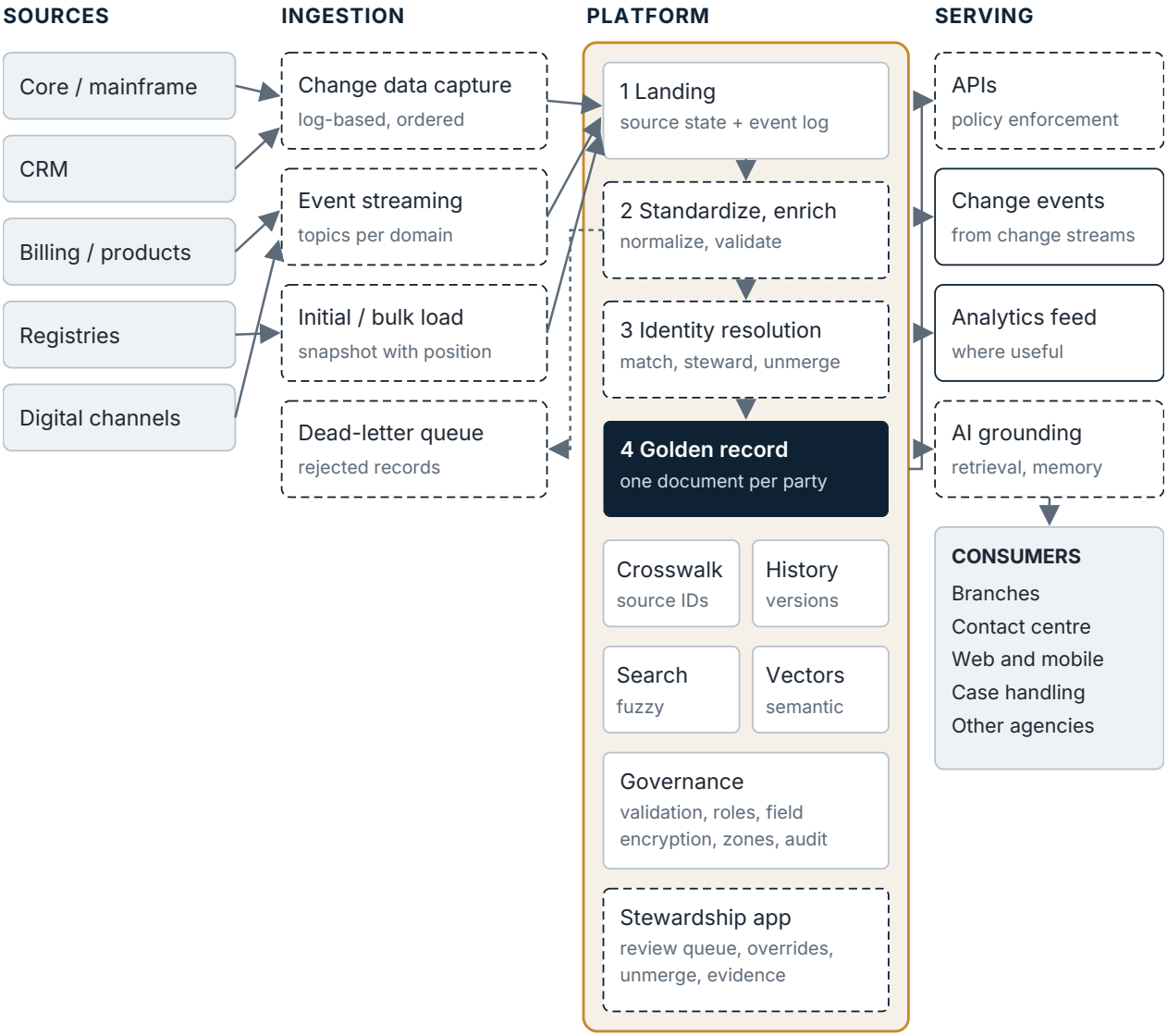


Figure 1. Reference architecture. Solid outlines are platform capabilities; dashed outlines are components the programme must build or integrate (pipelines, standardization, matching engine, stewardship application, APIs). Rejected records flow to the dead-letter queue.

The flow, step by step

Data enters through ingestion and is written to the landing layer (1) exactly as the source expressed it, with source key and source position. Standardization (2) normalizes formats, validates content and enriches it, for example by geocoding addresses. Identity resolution (3) decides which standardized source records describe the same party, records the evidence in the

crosswalk and routes uncertain cases to stewardship. The golden record (4) is then rebuilt from the linked source records according to survivorship rules, and every change to it is published as an event.

The ingestion contract

The Single View must reflect the sources continuously. The preferred mechanism is log-based change data capture, which reads the source's transaction log and emits inserts, updates and deletes in commit order, without adding load to its query path. Where capture is not possible, sources publish domain events, or the Single View uses incremental extracts. The contract with each source must cover six situations explicitly:

- **Initial load to capture handover.** Take a consistent snapshot together with the log position it corresponds to, load it, then start capture from that position. Without this, changes made during the load are lost or applied twice.
- **Ordering and staleness.** The source key identifies the record; a source position or version (log sequence number, commit timestamp, row version) orders its changes. Every write to the landing layer is conditional: it applies only if the incoming position is newer than the stored one, so that a late or replayed event never overwrites a newer state.
- **Duplicates.** Writes are idempotent: applying the same event twice produces the same result.
- **Deletes.** A delete in the source becomes a tombstone in the landing layer, not a physical removal, so that identity resolution and downstream consumers can react to it.
- **Rejected records.** Records that fail validation go to a dead-letter collection with the reason, are monitored and are reprocessed after correction.
- **Reconciliation.** Periodic comparison of counts and checksums between each source and the landing layer detects silent drift, and triggers a targeted reload when needed.

```
// Conditional, idempotent upsert into the landing layer.
// An event older than the stored position is ignored.
// sourcePosition must be stored in a form that compares correctly,
// for example a number or a zero-padded string.
db.landing_crm.updateOne(
  { _id: { system: "CRM", key: evt.key },
    $or: [ { sourcePosition: { $lt: evt.position } },
           { sourcePosition: { $exists: false } } ] },
  { $set: { payload: evt.after, deleted: evt.op === "d",
            sourcePosition: evt.position, capturedAt: new Date() } },
  { upsert: true }
)
// If a newer state is already stored, the filter does not match and the
// upsert attempts an insert with an existing _id: the resulting
// duplicate-key error must be caught and treated as "stale event, skip".
```

Listing 1. Positional guard on the landing layer.

What the landing layer keeps

Decide explicitly whether the landing layer keeps only the current state of each source entity, or also its history. The current state alone lets the platform rebuild golden records when rules change, but not reconstruct what a source said at an earlier date. Keeping an immutable log of source events, in a separate collection with a retention period, adds that ability and supports audit and dispute resolution, at a storage cost. The reference design keeps both: a current-state document per source entity, and an append-only event log retained for the period defined by audit requirements.

Serving

Consumers should not query the collections directly. They access the Single View through interfaces designed for them: APIs that return the projection each consumer is entitled to and enforce policy, change events that let downstream systems react without polling, and governed feeds to analytical platforms where useful. The change stream of the golden record collection is one of the most valuable assets of the architecture: every change to a party becomes an ordered event that can update an index, notify a channel, trigger a rule or refresh an AI agent's context.

Offloading the systems of record

A frequent early benefit is diverting read traffic away from legacy systems of record. When the Single View holds a current copy of the data channels read most, those reads need not reach the mainframe or the core, while writes continue to go to the owner and flow back through capture. The share of traffic that can be moved depends on the workloads selected and must be measured, not assumed; for organizations that pay for legacy capacity by consumption, it can carry much of the business case.

Freshness and read-your-own-writes

State the freshness guarantee explicitly, for example "changes committed in the source are visible in the Single View within five seconds at the 99th percentile under normal operation", monitor it, and expose it. Each API response should carry, per source, the time of the last applied change, so that consumers can decide whether the data is fresh enough for their purpose.

Write-through does not make two independent systems atomic. A sound flow is: the Single View API forwards the change to the owning source and waits for its acknowledgement, which includes the source position of the commit; the API reports success to the caller only after that acknowledgement; the Single View marks the attribute as pending until capture delivers a change at or beyond that position, and API responses expose the pending state. If the source rejects the change, nothing is written. If the Single View fails after the source has committed, capture will still deliver the change. A consumer that must read its own write immediately either reads from the owner, or waits for the pending state to clear.

Free edition · not for sale · CC BY-NC-ND 4.0

WHEN THINGS GO WRONG

- **Source unavailable.** The Single View keeps serving the last known state; the per-source freshness timestamp ages and consumers see it. Writes for that source's attributes are rejected or queued, never silently accepted.
- **Pipeline lag.** Lag beyond the freshness target raises an alert. Consumers with strict requirements can switch to a degraded mode, for example showing a notice or routing specific reads to the owner.
- **Lost resumability.** If a capture or change-stream consumer is down longer than its log retention, it cannot resume. The recovery procedure is a fresh snapshot of the affected source with handover, followed by reconciliation.
- **Replay.** Because landing writes are positional and idempotent, any interval of events can be replayed safely.
- **Targets.** Define recovery point and recovery time objectives for the platform and for each pipeline, and rehearse them.

A federated variant

In the public sector, and in groups with strong legal separation between entities, a single central platform may not be acceptable. A federated variant keeps one Single View per entity, each authoritative for its own data, and connects them through authorized, purpose-bound queries rather than bulk replication.

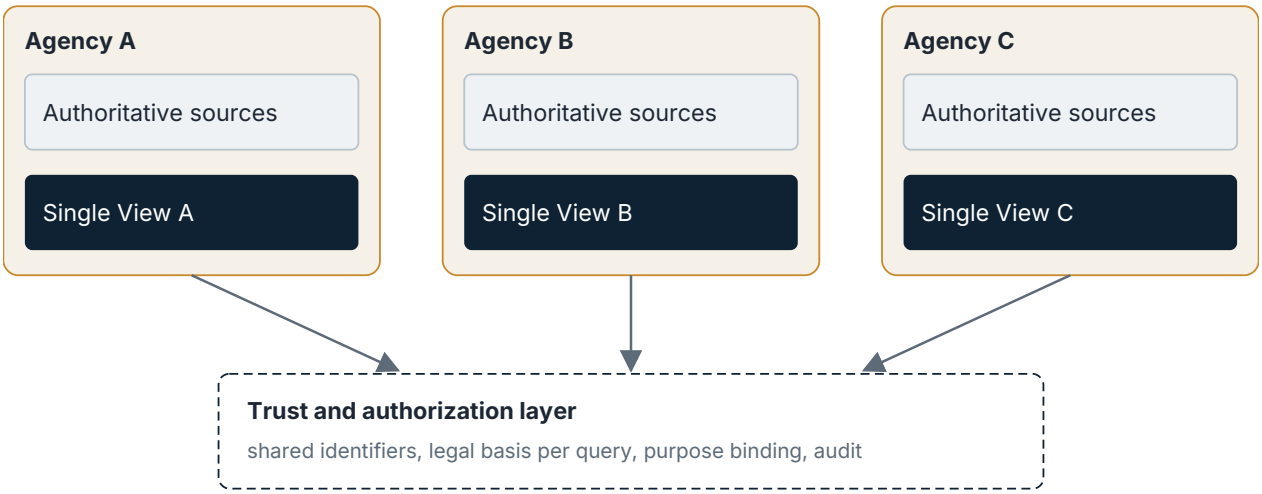


Figure 2. Federated variant. Each entity keeps authority over its data; cross-entity reads are individual, authorized and audited queries.

CHAPTER 6

The data model

Model for the question asked ten thousand times a day, not for the one asked once a quarter.

Design principles

- One golden document per party, containing what consumers commonly read together. Unbounded or high-volume data, such as transactions or interaction logs, lives in separate collections referenced by the party identifier.
- Attributes that can come from more than one source carry provenance: source system, source identifier, timestamp and the survivorship rule, with its version, that selected the value.
- The nature of the entity and its roles are kept apart: `partyType` says what the party is (individual, organization); `roles` say how the organization relates to it (customer, citizen, patient, guarantor).
- Repeated attributes of variable nature, such as identifiers and contact points, are arrays of typed elements, so that new types need no schema change and one index covers all of them.
- Indicators distinguish a verified negative from missing information: a flag is not simply true or false, it has a status.
- The document carries a schema version and a record version, to support evolution and optimistic concurrency.

The listings below show the principles in full for the attributes displayed. Some repetitive metadata is omitted for brevity where marked.

Free edition · not for sale · CC BY-NC-ND 4.0

A customer golden record

```
{
  "_id": "P-0004821733",
  "partyType": "individual",
  "roles": ["customer"],
  "schemaVersion": 4,
  "recordVersion": 57,
  "resolution": { "status": "resolved" },
  "name": { "given": "Laura", "family": "Bianchi", "full": "Laura Bianchi",
    "_src": { "system": "CORE", "id": "C77120", "at": "2026-09-14T08:12:03Z",
              "rule": "authoritative-source", "ruleVersion": 3 } },
  "birthDate": "1981-03-22",
  "_prov": { "birthDate": { "system": "CORE", "id": "C77120", "rule": "authoritative-
source", "ruleVersion": 3 } },
  "identifiers": [
    { "type": "taxId", "issuer": "IT-REVENUE", "country": "IT",
      "value": "XXXXXX81C62H501X", "verified": true },
    { "type": "customerNo", "issuer": "CORE", "value": "C77120" },
    { "type": "crmId", "issuer": "CRM", "value": "0016g00000AbCdE" }
  ],
  "contactPoints": [
    { "type": "email", "value": "laura.bianchi@example.org", "purpose": ["service"],
"preferred": true,
      "_src": { "system": "DIGITAL", "at": "2026-08-02T17:40:00Z", "rule": "most-recent-
verified", "ruleVersion": 2 } },
    { "type": "mobile", "value": "+39 333 0000000", "purpose": ["service", "otp"],
      "_src": { "system": "CORE", "at": "2025-11-19T10:05:44Z", "rule": "authoritative-
source", "ruleVersion": 3 } }
  ],
  "addresses": [
    { "use": "residence", "line": "Via Esempio 10", "city": "Roma", "postcode": "00100",
"country": "IT",
      "validFrom": "2019-06-01",
      "_src": { "system": "CORE", "at": "2019-06-03T09:00:00Z", "rule": "authoritative-
source", "ruleVersion": 3 } }
  ],
  "products": { "count": 4, "recent": [
      { "type": "currentAccount", "ref": "ACC-552910", "status": "open", "since":
"2012-04-10" },
      { "type": "mortgage", "ref": "MTG-118204", "status": "open", "since":
"2020-09-30" } ] },
  "relationships": [
    { "type": "jointHolder", "party": "P-0004821734", "name": "Marco Rossi", "on":
"ACC-552910" } ],
  "consents": [
    { "purpose": "marketing", "granted": false, "at": "2026-01-12T09:00:00Z",
      "channel": "web", "noticeVersion": "PN-2025.2" },
    { "purpose": "profiling", "granted": true, "at": "2026-01-12T09:00:00Z",
      "channel": "web", "noticeVersion": "PN-2025.2" } ],
  "indicators": {
    "deceased":   { "status": "verified", "value": false, "source": "CORE", "at":
"2026-09-01T00:00:00Z" },
    "vulnerable": { "status": "unknown" } },
  "residency": { "zone": "EU-IT" },
  "freshness": { "CORE": "2026-09-14T08:12:03Z", "CRM": "2026-09-14T07:58:41Z",
                 "DIGITAL": "2026-09-14T08:10:12Z" },
  "updatedAt": "2026-09-14T08:12:05Z"
}
```

Listing 2. A golden record for an individual customer. Values are fictitious. Structured attributes carry provenance in `_src`; scalar attributes such as `birthDate` keep it in the `_prov` map, so that the value itself stays simple to index and query. Provenance on products and relationships is omitted for brevity.

The `identifiers` array follows the attribute pattern. An identifier is meaningful only within its namespace, so the issuer and, where relevant, the country are part of its identity. One compound multikey index serves lookups by any identifier, and the query uses `$elemMatch` so that type, issuer and value are matched on the same array element:

```
db.parties.createIndex({ "identifiers.type": 1, "identifiers.issuer": 1,
"identifiers.value": 1 })

db.parties.find({ identifiers: { $elemMatch: {
  type: "taxId", issuer: "IT-REVENUE", value: "XXXXXX81C62H501X" } } })
```

Listing 3. Identifier lookup within its namespace.

The `products.recent` array follows the subset pattern: the most relevant items are embedded, the full list lives in its own collection. The `relationships` array follows the extended reference pattern: the related party's identifier plus the few fields needed for display.

A citizen golden record

In the public sector the structure is the same; the content stresses authoritative registries, legal basis and entitlements.

```
{
  "_id": "CIT-IT-58210993",
  "partyType": "individual",
  "roles": ["citizen"],
  "schemaVersion": 2,
  "recordVersion": 12,
  "resolution": { "status": "resolved" },
  "name": { "given": "Giovanni", "family": "Esposito",
    "_src": { "system": "POPULATION_REGISTRY", "authoritative": true,
              "at": "2004-05-12T00:00:00Z", "rule": "authoritative-source", "ruleVersion":
1 } },
  "identifiers": [
    { "type": "taxCode", "issuer": "IT-REVENUE", "country": "IT", "value":
"XXXXXX74A01F839X" },
    { "type": "eID", "issuer": "IT-EID", "value": "eid-7c1f0a", "assuranceLevel":
"high" } ],
  "residence": { "municipality": "Napoli", "since": "2004-05-12",
    "_src": { "system": "POPULATION_REGISTRY", "authoritative": true, "at":
"2004-05-14T00:00:00Z" } },
  "household": { "ref": "HH-20981", "members": 3,
    "_src": { "system": "POPULATION_REGISTRY", "authoritative": true, "at":
"2023-01-10T00:00:00Z" } },
  "entitlements": [
    { "scheme": "childBenefit", "status": "active", "from": "2023-02-01",
      "_src": { "system": "WELFARE", "authoritative": true } },
    { "scheme": "energyBonus", "status": "eligible", "computedAt": "2026-09-01",
      "_src": { "system": "SINGLE_VIEW", "rule": "eligibility-energy", "ruleVersion":
4 } } ],
  "openCases": [
    { "case": "CASE-77310", "domain": "permits", "status": "awaitingDocuments",
      "_src": { "system": "MUNICIPALITY" } } ],
  "lifeEvents": [ { "type": "birthInHousehold", "at": "2023-01-08", "triggered":
["childBenefit"] } ],
  "dataSharing": [
    { "purpose": "onceOnlyPrefill", "legalBasis": "publicTask",
      "legalReference": "to be specified per processing", "recipients": ["MUNICIPALITY",
"WELFARE"] } ],
  "residency": { "zone": "IT-SOVEREIGN" },
  "updatedAt": "2026-09-02T06:30:11Z"
}
```

Listing 4. A golden record for a citizen. Values are fictitious.

Attributes reference the registry that is authoritative for them, which matters for trust and for legal accountability. The record also models life events and the entitlements they trigger: this is what enables proactive services, informing a family of a benefit it is entitled to instead of waiting for an application. Note that a derived attribute such as eligibility has the Single View itself as source, with the rule that computed it.

Source records and the crosswalk

Each landing document keeps the source payload, source key, source position and the party it is resolved to, with the evidence of the match. The crosswalk must support two efficient indexed lookups: from a source identifier to its party, served by the `_id`, and from a party to all its source records, served by a secondary index on `party`. The second lookup returns as many documents as the party has source records.

```
{
  "_id": { "system": "CRM", "key": "0016g00000AbCdE" },
  "party": "P-0004821733",
  "match": { "method": "deterministic", "rule": "taxId-namespace-exact", "ruleVersion": 2,
             "score": 1.0, "at": "2024-03-03T11:20:00Z", "by": "auto" },
  "sourcePosition": "lsn:0000A3F1/0B2C4D10",
  "capturedAt": "2026-09-14T08:12:01Z",
  "deleted": false,
  "payload": { "...": "original CRM record, unmodified" }
}
// db.landing_crm.createIndex({ party: 1 })
```

Listing 5. A landing document that also acts as crosswalk entry.

Validation without rigidity

A JSON Schema validator on the golden record collection enforces what every consumer depends on: party identifier, party type, roles, schema version, and the structure of identifiers, consents and indicators. Type-specific sub-documents can be validated more loosely. Validation can start in warning mode during migration and move to error mode once data quality is confirmed.

History and versioning

Consumers usually want the current state; auditors and investigators want the state at a given moment. The reference design keeps current versions in the golden record collection and previous versions in a history collection. The most robust way to populate history is in the same operation that changes the record: the process that rebuilds a golden record writes the previous version to history and the new version to the golden collection in one multi-document transaction, guarded by the record version.

Populating history asynchronously from a change stream is possible, but requires care. The consumer must capture the correct state: `fullDocument: "updateLookup"` returns the document as it is when the event is processed, which may already include later changes, so pre- and post-images must be enabled on the collection and used instead. Writes to history must be idempotent, keyed by party and record version. The resume token must be persisted durably after each processed event, and if the consumer falls behind the change stream's retention it must fall back to a reconciliation procedure.

CHAPTER 7

Identity resolution

Storing data about people is easy. Knowing which data belongs to which person is the actual product.

Identity resolution decides which source records describe the same real-world party. Wrong in one direction, it merges two people into one record: a serious privacy and service failure. Wrong in the other, it leaves one person split across several records, defeating the purpose of the Single View. Both errors happen at some rate; the goal is to keep them low, measure them, detect them and make them reversible.

A layered matching strategy

- **Deterministic matching on strong identifiers.** Records sharing a strong identifier are candidates for automatic merge, under three conditions: the identifiers are compared within the same namespace (type, issuer, country); the identifier passes validity checks such as format and checksum; and no contradictory evidence exists, such as a different birth date or a different verified name. Contradictory evidence blocks the automatic merge and sends the case to review.
- **Probabilistic matching on weaker attributes.** Names, birth dates, addresses, phone numbers and emails are compared with similarity functions and combined into a score. Above a high threshold the merge is automatic; below a low threshold the records stay apart; in between, a steward decides. Thresholds are calibrated on a labelled sample.
- **Semantic matching as an aid.** Embeddings of names and addresses can surface candidates that string similarity misses, such as transliterations or reordered components. They help generate candidates; the decision should rest on explainable rules.

Blocking, then scoring

Comparing every record with every other is quadratic and infeasible at scale. Matching therefore first retrieves a small set of plausible candidates (blocking), then compares each in detail (scoring). The example below implements one blocking strategy with MongoDB Search. It is reproducible with the index definition, the minimal data and the expected result shown.

```
// Search index "party_match" on collection "parties"
{ "mappings": { "dynamic": false, "fields": {
    "birthDate": { "type": "token" },
    "name": { "type": "document", "fields": { "full": { "type": "string", "analyzer":
"lucene.standard" } } },
    "addresses": { "type": "document", "fields": { "postcode": { "type": "token" } } },
    "contactPoints": { "type": "document", "fields": { "value": { "type": "token" } } }
} } }
```

```
// Minimal data: Listing 2 inserted into "parties".
// Incoming record to resolve: name "Laura Bianci" (misspelt), birthDate "1981-03-22",
// postcode "00100", email "laura.bianchi@example.org".
db.parties.aggregate([
  { $search: { index: "party_match", compound: {
      filter: [ { equals: { path: "birthDate", value: "1981-03-22" } } ],
      should: [
        { text:   { path: "name.full", query: "Laura Bianci",
                    fuzzy: { maxEdits: 2, prefixLength: 1 } } },
        { equals: { path: "addresses.postcode", value: "00100" } },
        { equals: { path: "contactPoints.value", value: "laura.bianchi@example.org" } }
      ],
      minimumShouldMatch: 1 } } },
  { $limit: 20 },
  { $project: { name: 1, birthDate: 1, identifiers: 1, score: { $meta: "searchScore" } } }
])
// Expected result: one candidate, _id "P-0004821733". The token "bianci" is one
// edit from "bianchi", within maxEdits 2; birthDate matches as a token string.
```

Listing 6. Candidate retrieval with fuzzy search: index, data, query and expected result. Birth date is stored and queried as a civil-date string indexed as token.

The search score is a ranking signal, not a match probability. The final decision combines attribute-level comparisons into a calibrated score, with rules that can be explained to a steward, an auditor or a data subject.

The limits of a blocking strategy

Every blocking strategy trades recall for speed. The example filters on birth date, so it will miss candidates whose birth date is missing or wrong in one source, and the limit of twenty results can exclude valid candidates for common names. In practice several blocking passes run in parallel, for example birth date plus name, postcode plus phonetic surname, exact email or phone, and their union is scored. Measure blocking coverage on a labelled sample: the share of true matching pairs that appear in at least one candidate set.

Survivorship

Once records are linked, the golden record chooses for each attribute which value survives. Typical rules: prefer the authoritative source for the attribute; otherwise the most recently verified value; otherwise the most recent; otherwise the most frequent. Rules are configuration, identified and versioned, and the provenance of each attribute records the rule and version that selected it.

Stewardship, overrides and reversibility

A stewardship application presents uncertain matches with the evidence behind the score and records each decision. A steward's decision is stored as evidence or as an override for that specific case. It is not automatically generalized: changing a matching rule because of patterns in steward decisions goes through rule review, testing on the labelled sample and versioned release.

Every automated merge must be reversible. Because the landing layer keeps source records and the crosswalk keeps the evidence of each link, an unmerge relinks source records and rebuilds the affected golden records. Identifiers need a lifecycle: on a merge, the surviving party keeps its identifier and the retired identifier is kept as an alias that redirects to it; on an unmerge, the original identifier stays with the party that best matches its history and the separated records receive a new identifier. Both operations publish events, for example `party.merged` and `party.split`, that downstream systems must handle to update their references.

> MEASURE WHAT MATTERS
>
> Operational indicators: automatic match rate, stewardship queue size and age, unmerges per month, complaints traceable to wrong merges. Quality indicators, measured on a regularly refreshed labelled sample: precision, recall, false-merge rate, missed-match rate and blocking coverage. Operational numbers say whether the process runs; quality numbers say whether it is right.

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 8

Governance, privacy and sovereignty

A Single View concentrates personal data by design. It must concentrate protection with the same determination.

Bringing the facts about a person into one record creates value and an attractive target. It also creates a single place where data protection obligations can be met consistently, instead of unevenly across dozens of systems. Treated properly, the Single View is a privacy asset.

Governance data is not enforcement

Consents, purposes and legal bases stored in the golden record, as in Listings 2 and 4, are information that policies use. They do not enforce anything by themselves. Enforcement happens at defined control points, each of which must evaluate the relevant governance data:

- **APIs**: each consumer is registered with its purposes, and receives only the projection those purposes allow. A database role and a read-only view restricted to the same fields act as a second line of defence.
- **Events**: change events are filtered or projected per subscriber, so that a downstream system does not receive fields it is not entitled to.
- **Exports and analytical feeds**: extracts apply the same rules, and are recorded with their purpose and recipient.
- **AI retrieval and agent memory**: retrieval for assistants and memory for agents are consumers like any other, with purposes, projections and audit; content retrieved for one purpose is not retained for another.

Revocations must propagate. When a consent is withdrawn or a purpose ends, the change is published as an event, and every control point and every downstream copy that depends on it must act on it within a defined time.

Encryption, and its trade-off with search

Encryption at rest and in transit is a baseline. For the most sensitive attributes, client-side field-level encryption ensures values are encrypted before leaving the application and remain unreadable to database administrators, backups and anyone without the keys. Queryable Encryption extends this with equality and range queries on encrypted fields.

Free edition · not for sale · CC BY-NC-ND 4.0

These protections cannot be combined freely with search. Fields encrypted client-side cannot be indexed or queried by MongoDB Search, so they cannot take part in fuzzy or semantic matching. The design must therefore decide, attribute by attribute, which data participates in matching and how it is protected:

- Attributes used for fuzzy blocking and scoring, typically name, birth date, postcode and contact points, remain searchable. They are protected by encryption at rest, network isolation, restricted roles and views, and audit.
- Strong identifiers used for deterministic matching can be protected with Queryable Encryption, since deterministic matching needs only equality. Because MongoDB Search cannot be used on a collection that uses Queryable Encryption, the protected identifiers are held in a separate collection, an identifier vault keyed by party and queried by equality, while the golden record collection indexed for Search keeps only the party reference and non-identifying attributes. The listings in Chapter 6 show identifiers inline for readability; with Queryable Encryption they move to the vault.
- Special-category and high-risk attributes that are not needed for matching, such as health or vulnerability details, are encrypted client-side.

The right to erasure

Erasure is difficult in a system built to keep history and lineage. Within the Single View, the crosswalk, the history collection and the landing event log identify where a party's data is held. They do not, by themselves, list caches, exports, search and vector indexes, or downstream systems that received the data: that inventory must be maintained separately, for example as a registry of subscribers and extracts, so that erasure requests can be propagated and confirmed.

Crypto-shredding can help: sensitive fields of each party are encrypted with a key specific to that party, and destroying the key renders those fields unreadable wherever the encrypted values are stored. It is not a complete answer on its own. Copies of the key must be controlled and destroyed too; plaintext may persist in caches, indexes or exports; and backups restored after the erasure must be followed by re-application of the erasure list. Erasure must also respect legal retention obligations, which may require keeping some data.

Residency and sovereignty

Zone sharding can pin records to specific regions or facilities. A field such as `residency.zone` achieves this only together with a shard key that includes it, zone ranges defined on that key, and zones assigned to the right shards. Replicas, backups, snapshots, logs, search nodes and any downstream copy must be located consistently; residency of the primary data alone is not sufficient.

The deployment model shapes what is possible. In Atlas, Global Clusters and region selection provide zone placement, with backup locations configured accordingly. In self-managed deployments the organization defines shards, zones and backup locations on its own infrastructure. In isolated or air-gapped environments the platform runs with no dependency on external services, and keys, updates and monitoring are handled inside the perimeter.

Sovereignty goes further than residency. It concerns who operates the platform, who holds the keys, which jurisdiction's law applies to the operator, and whether the system can run disconnected. These choices should be made explicitly and documented, not inherited from defaults.

Audit and accountability

Every change to the golden record should be attributable, and reads by human operators should be traceable. Database auditing can record operations, but logging successful reads requires explicit configuration and appropriate filters, and has a performance cost. On its own, a database log shows that a technical identity, usually an application, read a document; it does not show what a person actually saw. The API layer must record the operator, the purpose, the case and the fields returned, with a correlation identifier that links its records to the database audit trail.

> A PRACTICAL DRILL
>
> Before going live, run a mock subject access request and a mock erasure request end to end, including downstream copies. As an internal exercise target, the author suggests completing both within one working day. It is not a regulatory deadline, and meeting it is a sign of maturity, not proof that governance is complete.

CHAPTER 9

Industry patterns

The pattern is the same everywhere. What changes is which source owns the truth, and which failure costs the most.

The scenarios below are composites. They combine architectural situations that recur across organizations in each sector and do not describe any specific institution. Benefits are indicative and must be measured on the workloads actually selected.

Retail and commercial banking

Customer data lives in a core system, often on a mainframe, alongside separate platforms for cards, lending, wealth and digital channels. Digital adoption multiplies read traffic on the core. The Single View is fed by change data capture from the core and events from the other platforms, and serves customer information, product holdings and contact data to channels and the contact centre. It can absorb a significant share of read traffic for the selected workloads, a benefit to be measured. Writes remain on the core.

Insurance

Separate policy administration systems per line of business and per acquisition, plus claims and distribution platforms. Agents and claims handlers need the whole relationship: policies across lines, open claims, payments and interactions. Relationship modelling links policyholders, insured parties and beneficiaries, who are often different people, and gives channels the same view of the household.

Telecommunications

Subscriber data spread across billing, CRM, provisioning and self-care. Churn prevention and service assurance need commercial and technical signals per customer in near real time. The golden record holds subscriber, contracts and recent service indicators; high-volume network events stay in time series collections referenced by subscriber, with computed indicators stored on the record.

Retail and consumer brands

Customers interact through stores, e-commerce, marketplaces, loyalty and service, often with different identifiers in each. Identity resolution is largely probabilistic, on emails, phone numbers, payment tokens and loyalty cards. The Single View supports recognition across channels and preference management, with consent checked at every activation point.

Public administration

Citizens' data is held by population registries, tax authorities, social security, municipalities and sector agencies. Policy goals include the once-only principle, which in the European Union is supported at cross-border level by the once-only technical system established under the Single Digital Gateway Regulation, proactive services triggered by life events, and benefit fraud prevention. The Single View here is often federated (Figure 2). Authoritative sources per attribute and a documented legal basis for every data flow are legal obligations in most jurisdictions; sovereign deployment is frequently a policy requirement and in any case a strong architectural recommendation.

Healthcare

A patient's history is distributed across hospitals, primary care, laboratories, pharmacies and regional networks. The HL7 FHIR standard defines resources such as Patient, Encounter and Observation, which can be exchanged in JSON as well as XML and RDF. A JSON exchange format makes a document platform a convenient operational store for a longitudinal patient view, but the persistent model remains a design choice: it need not mirror FHIR resources one to one. Identity resolution relies on national health identifiers where available. Under the GDPR, health data is a special category, and Article 32 requires security measures appropriate to the risk; it does not prescribe a specific technique. For special-category data, this guide recommends field-level encryption, purpose-bound access and complete audit as the architectural default.

Sector	Typical authoritative source	Main driver	Hardest problem
Banking	Core banking system	Legacy offload, service quality	Freshness guarantees against the core
Insurance	Policy administration systems	Household view for agents and claims	Party roles across policies
Telecom	Billing and CRM	Churn and service assurance	Joining commercial and network signals
Retail	Loyalty and e-commerce	Recognition and personalization	Probabilistic identity at scale
Public sector	Population and tax registries	Once-only, proactive services	Legal basis and federation across agencies
Healthcare	Clinical systems, national health ID	Continuity of care	Protection of special-category data

CHAPTER 10

A worked example

One person, three sources, one error and its correction, end to end.

A composite retail bank runs a core system (CORE), a CRM and a digital banking platform (DIGITAL). The example follows Laura Bianchi, the party of Listing 2, from discordant sources to a golden record, through an update, an error and its correction.

1. Discordant sources

CORE holds customer C77120: Laura Bianchi, born 1981-03-22, tax identifier verified, residence in Rome, mobile number. CRM holds contact 0016g00000AbCdE: "Laura Bianci", same birth date, same tax identifier, an older email. DIGITAL holds user account U-88412: "Laura Bianchi", same birth date, a newer verified email, no tax identifier.

2. Initial load and resolution

Each source is snapshotted with its log position and loaded into its landing collection; capture starts from those positions. Deterministic matching links CORE and CRM: same tax identifier in the same namespace, valid checksum, no contradictory evidence, since the surname difference is within tolerance and the birth date agrees. DIGITAL has no strong identifier; probabilistic matching scores it against the party on name, birth date and email similarity, above the automatic threshold. Party P-0004821733 is created with three linked source records.

3. Golden record

Survivorship selects the name and birth date from CORE (authoritative-source, version 3) and the email from DIGITAL (most-recent-verified, version 2). Each value records its source and rule. The record is published as a `party.created` event.

4. An update

Laura moves house and updates her address at the branch, in CORE. Capture delivers the change with its log position; the landing write is applied because the position is newer; the golden record is rebuilt in a transaction that also writes version 56 to history; version 57 is published as `party.updated`. The contact centre sees the new address within the freshness target, with the CORE freshness timestamp updated.

5. An error

A second DIGITAL account, U-91207, belongs to a different Laura Bianchi, born the same day in another city, with an email that differs by one character. The probabilistic score lands just above the automatic threshold and the account is linked to P-0004821733. Two weeks later the first Laura reports an unknown device in her digital profile.

6. Correction

A steward reviews the evidence: different residence, different device history, the email similarity was the decisive factor. The steward unmerges U-91207. Its crosswalk entry is relinked to a new party, P-0005110298, and both golden records are rebuilt from their source records. A `party.split` event lists the moved source record; the digital platform and the fraud system update their references. The steward's decision is stored as an override for this pair. Separately, the matching team reviews whether a single-character email difference should weigh less, tests a revised rule on the labelled sample, and releases it as version 3 of the probabilistic rule set. The false merge is recorded in the quality metrics.

> **WHAT MADE THE CORRECTION CHEAP**
>
> Source records were retained, the evidence of each link was stored, identifiers had a lifecycle, and downstream systems consumed split events. Without any one of these, the same error would have required manual data repair across several systems.

CHAPTER 11

Anti-patterns

Most Single View failures are not technical. They are decisions that looked reasonable at the start.

The big-bang programme

Integrating every source and serving every consumer in the first release means a long programme with little visible value until the end. Start with one domain, a handful of sources and one or two consumers with a measurable problem.

The relational copy

Migrating each source table into its own collection and reassembling the person with joins at query time reproduces the original problem on a new platform. The document model pays off when documents are shaped for the dominant read paths.

The unbounded document

Embedding every transaction, interaction or event in the party document leads to documents that grow without limit, slow every read and eventually reach the maximum document size. Embed summaries and recent items; keep high-volume history in its own collection.

The copy without lineage

A Single View whose values cannot be traced to a source and a time will not be trusted, and an untrusted Single View is quietly bypassed. Provenance belongs in the first data model.

The irreversible merge

Merging destructively, without keeping source records and the evidence of their linkage, makes matching errors permanent. Keep source records, keep the crosswalk, make unmerge routine.

The ownerless platform

A Single View built by a project and handed to operations without a product owner decays: sources are not added, rules are not maintained, consumers build workarounds. It needs a permanent team accountable for data quality, source contracts and roadmap.

Free edition · not for sale · CC BY-NC-ND 4.0

The silent write-back

Letting consumers change data owned by another system without a contract for propagation and confirmation creates divergence that no one notices until a customer does. Write-back is explicit, field by field, with acknowledgement and reconciliation.

Governance as a later phase

Adding consent enforcement, access control and erasure after go-live means retrofitting them into every consumer. They belong in the first data model and the first API contract.

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 12

Adoption roadmap and checklist

Deliver value in months, not years, and let each phase fund the next.

Durations are indicative and depend on the number and quality of sources, the regulatory context and the team.

Phase 0 · Discovery (indicatively 4 to 6 weeks)
Identify the first consumers and their most frequent questions. Inventory the sources that answer them, their owners, identifiers and change mechanisms. Define the freshness contract, the recovery objectives and the success metrics.

Phase 1 · Read view for one domain (indicatively 3 to 4 months)
Ingestion with snapshot handover from two or three key sources; landing with positional, idempotent writes; deterministic matching with namespace and conflict checks; basic survivorship as versioned configuration; retention of match evidence; a working unmerge procedure; the first golden record shape; APIs with purpose-bound projections for one or two consumers; validation, roles, auditing and field encryption from the start.

Phase 2 · Probabilistic matching and stewardship
Probabilistic matching calibrated on a labelled sample, multiple blocking passes, the stewardship application, identifier lifecycle and merge and split events, quality metrics.

Phase 3 · Expansion
Further sources and consumers, domain by domain. Residency zones if the scope crosses jurisdictions. Feeds to analytical platforms where they add value.

Phase 4 · Engagement and write-through
The Single View becomes system of record for explicitly assigned engagement attributes; write-through for agreed fields with acknowledgement, pending state and reconciliation.

Phase 5 · Intelligence
The governed golden record as grounding for AI: retrieval for assistants, memory for agents, features for models, each registered as a consumer with purposes, projections, audit and revocation handling.

Readiness checklist

Each criterion has an expected evidence and an accountable owner; the result column is filled during review.

Criterion	Expected evidence	Owner	Result
First consumers and questions defined	Documented use cases with measurable targets	Product owner	
One owner per attribute	Field-level ownership register	Data architect	
Freshness contract	Target, dashboard, per-source timestamps in API responses	Platform lead	
Ingestion contract	Tests for handover, ordering, duplicates, deletes, dead letters	Integration lead	
Replay	Successful replay of a past interval with identical result	Integration lead	
Reconciliation	Scheduled count and checksum comparison per source	Integration lead	
Reversible merges	Unmerge test with split event consumed downstream	Matching lead	
Versioned rules	Rules in configuration, tested on labelled sample	Matching lead	
Matching quality	Precision, recall, false-merge rate, missed-match rate and blocking coverage reported	Matching lead	
Provenance	Source, time and rule on multi-source attributes	Data architect	
Purpose-bound access	Consumer register; API and database projections match	Security lead	
Revocation propagation	Consent withdrawal test across APIs, events and copies	Data protection officer	
Encryption and keys	Field classification; key management documented	Security lead	
Access revocation	Removal of a consumer's and an operator's access tested	Security lead	
Subject access and erasure	End-to-end drill including downstream inventory	Data protection officer	
Residency	Zones, shard key, backups and replicas mapped	Platform lead	
Backup and restore	Restore test, with erasure list re-applied afterwards	Platform lead	
Recovery objectives	Rehearsed failover and pipeline recovery within targets	Platform lead	
Ownership	Permanent team with product owner and roadmap	Sponsor	

Free edition · not for sale · CC BY-NC-ND 4.0

CHAPTER 13

References

Standards and regulation

Regulation (EU) 2016/679 (General Data Protection Regulation), in particular Articles 5, 9, 17, 25 and 32. eur-lex.europa.eu

Regulation (EU) 2018/1724 establishing a single digital gateway, including the once-only technical system.

HL7 FHIR, Resource formats. fhir.hl7.org/fhir/resource-formats.html

Record linkage

Fellegi, I. P. and Sunter, A. B. (1969). A Theory for Record Linkage. Journal of the American Statistical Association, 64(328), 1183 to 1210.

Christen, P. (2012). Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer.

Reference implementation documentation

Change Streams. mongodb.com/docs/manual/changestreams

Zone sharding. mongodb.com/docs/manual/core/zone-sharding

Multikey index bounds. mongodb.com/docs/manual/core/indexes/index-types/index-multikey/multikey-index-bounds

MongoDB Search, equals operator. mongodb.com/docs/search/query/operators-collectors/equals

MongoDB Search, FAQ. mongodb.com/docs/search/faq

Client-side field-level encryption, right to erasure. mongodb.com/docs/manual/core/csfle/tutorials/right-to-erasure

Atlas database auditing. mongodb.com/docs/atlas/database-auditing

Web references were checked at the time of writing; documentation changes over time.

CHAPTER 14

About the author

Mario Noioso is a software architect and author based in Rome, with more than thirty years of experience designing and delivering information systems for large enterprises and public institutions. His career spans IBM development laboratories, Microsoft and ServiceNow; he currently works as an Advisory Solutions Architect for the EMEA public sector at MongoDB, focusing on data architecture, artificial intelligence and digital sovereignty.

He is named as an inventor on United States patents in software distribution and real-time analytics, and is the author of *Who Owns Memory Owns the Future: Power, Knowledge, and Imagination from Fire to Mars* (2026), an essay on the relationship between memory, power and technology. He writes on data, AI and technology culture at marionoioso.com.

ORCID 0009-0000-9052-944X · ISNI 0000 0004 2128 1497 · marionoioso.com

About this series

Real-World Architectures is a series of short field guides on data and systems architecture, written from the perspective of practice: what works in production, in regulated industries and in public institutions, and what does not. Each volume addresses one architectural pattern, with its problem, reference design, data model, governance and pitfalls.

Feedback and corrections are welcome through marionoioso.com.

◆ ◆ ◆

Free edition · not for sale · CC BY-NC-ND 4.0