

Palantir Data Engineering Certification

Volume 1 — Foundations, Architecture & Data Ingestion

How to Use This Volume

This volume establishes the **foundational mental model** required to understand Palantir Foundry and to succeed in the Data Engineering Certification exam.

It is written as **textbook-style reading material**, not notes. You should read it sequentially. Later volumes assume you fully understand the concepts defined here.

If you internalize this volume, you will understand *why* Foundry works the way it does — which is the single most important factor in passing the exam.

Chapter 1 — What Problem Foundry Was Built to Solve

1.1 Why Traditional Data Platforms Fall Short

Most data platforms evolved to solve one of two problems:

1. **Analytics** — reporting, dashboards, BI
2. **Computation** — large-scale batch or distributed processing

These platforms assume: - Data is mostly static - Errors are tolerable - Consumers are technically sophisticated

Palantir Foundry was built for a fundamentally different environment.

It was designed for organizations where: - Decisions are operational, not just analytical - Errors have legal, financial, or human consequences - Data must be explained, audited, and defended

In such environments, the question is not:

“Can we compute this?”

But rather:

“Can we explain why this result exists, months or years later?”

Foundry is optimized for **accountability**, not convenience.

1.2 Foundry as a Data Operating System

Foundry is best understood as a **data operating system**, not a tool.

Like an operating system, it: - Enforces rules by default - Prevents unsafe behavior - Makes safe behavior the path of least resistance

Foundry does not trust engineers to “do the right thing” consistently. Instead, it **bakes correctness into the architecture**.

This design philosophy explains many Foundry constraints that feel heavy to engineers accustomed to ad-hoc systems.

Chapter 2 — Foundry’s Core Design Principles (Exam-Critical)

Every Foundry feature enforces a small set of non-negotiable principles. Exam questions are almost always testing whether you understand these principles.

2.1 Immutability

In Foundry, datasets are **immutable**.

Once a dataset version is created: - It is never modified - Corrections produce a new version

This is not a technical limitation. It is a deliberate design choice.

Immutability enables: - Historical audits - Time-travel debugging - Reproducibility of past decisions

If data could be overwritten, none of these would be possible.

Exam implication: Any approach that mutates existing data is almost always incorrect.

2.2 Lineage as a First-Class Concept

Every dataset in Foundry records: - Its upstream inputs - The exact transformation logic - The version of each dependency

Lineage is not documentation. It is **executable knowledge**.

Lineage enables: - Impact analysis - Root cause investigation - Controlled change management

If you cannot trace how a value was produced, Foundry considers the system unsafe.

2.3 Explicitness Over Convenience

Foundry consistently prefers: - Explicit schemas over inferred ones - Explicit dependencies over dynamic discovery - Explicit permissions over implicit access

Implicit behavior is convenient in small systems, but dangerous at scale.

Explicit systems are: - Easier to debug - Easier to audit - Safer to evolve

Exam implication: “Auto-magic” solutions are usually wrong.

2.4 Separation of Concerns

Foundry enforces strict separation between: - Raw data - Business logic - Semantic meaning - Access control

Each concern lives in a different layer. Mixing them creates fragility and governance risk.

Chapter 3 — Foundry Architecture: The Layered Model

3.1 Thinking in Responsibility Zones

Foundry architecture should be understood as **layers of responsibility**, not tools or technologies.

```
External World (Untrusted)
↓
Ingestion Layer (Evidence Capture)
↓
Raw Data Layer (Historical Truth)
↓
Transformation Layer (Business Logic)
↓
Curated Data Layer (Business Truth)
↓
Ontology (Semantic Contract)
↓
Applications & Decisions
```

Each layer answers a fundamentally different question.

3.2 Raw Data Layer — Capturing Evidence

The raw data layer exists to answer:

“What did the source system send us, exactly as it was?”

Raw datasets: - Preserve all fields - Preserve original types - Preserve original errors

Raw data is treated like **evidence in a legal case**.

If the source system was wrong, the raw data must reflect that wrongness.

Raw data is never: - Cleaned for business use - Corrected - Rewritten

3.3 Curated Data Layer — Defining Business Truth

Curated datasets answer:

“Given the raw evidence, what does the organization believe is correct?”

This is where: - Data is cleaned - Types are enforced - Duplicates are resolved - Business rules are applied

Curated data is safe for broad consumption.

3.4 Ontology — The Human Interface

The ontology layer translates curated data into: - Real-world entities - Relationships - Actions

Ontology allows non-technical users to interact with data safely, without understanding schemas or joins.

Chapter 4 — Data Ingestion as Evidence Engineering

4.1 What Ingestion Means in Foundry

In many systems, ingestion is treated as a plumbing problem.

In Foundry, ingestion is treated as an **evidence capture problem**.

Your responsibility during ingestion is: - Faithfulness to the source - Auditability - Reprocessability

Speed is secondary to correctness.

4.2 Batch Ingestion

Batch ingestion captures: - Full snapshots, or - Periodic deltas

Advantages: - Simple mental model - Deterministic reprocessing - Easier audits

Disadvantages: - Higher latency - Higher compute cost

Batch ingestion is preferred when correctness is more important than freshness.

4.3 Incremental Ingestion

Incremental ingestion captures only new or changed records.

It requires: - Stable primary keys - Reliable change detection - Idempotent writes

Without all three, incremental ingestion introduces silent corruption.

Foundry treats incremental ingestion as **optional complexity**, not a default.

4.4 Late-Arriving Data

Late-arriving data occurs when: - Events arrive out of order - Corrections arrive days or weeks later

Correct handling requires: - Preserving original raw records - Reprocessing affected downstream windows

Overwriting history is never acceptable.

4.5 Schema Drift

Schema drift is expected in real systems.

Foundry's approach: - Allow drift in raw ingestion - Enforce schema in curated layers - Fail loudly when business logic breaks

Silently adapting schemas is dangerous.

Chapter 5 — Why Naive Designs Fail the Exam

Common incorrect instincts include: - Cleaning data during ingestion - Updating records in place - Exposing raw data to users - Collapsing layers for simplicity

These approaches violate Foundry's core principles.

The exam is designed to surface these mistakes.

End of Volume 1

Next volumes build on this foundation: - Volume 2: Transforms and incremental processing - Volume 3: Ontology and semantic modeling - Volume 4: Governance, security, and lineage - Volume 5: Debugging, operations, and exam strategy