Palantir Data Engineering Certification

Volume 2 — Transforms, Incremental Processing & Business Logic

---

How to Use This Volume

This volume is written as **print-ready textbook material**. You should be able to: - Read it linearly - Annotate margins - Refer back during revision

This volume focuses on the **core skill Palantir evaluates most aggressively**: your ability to design **correct, reproducible, production-safe transformations** in Foundry.

If you understand this volume deeply, you will be able to reason through most scenario-based exam questions.

Chapter 5 — Transforms: The Heart of Foundry

5.1 What a Transform *Is* (Precise Definition)

In Foundry, a transform is **not** merely a script or query.

A transform is: - A **pure, deterministic function** - That consumes one or more *versioned datasets* - And produces a *new, immutable dataset version*

Formally:

> Output Dataset Version = f(Input Dataset Versions, Transform Code, Configuration)

This definition has three critical implications: 1. The same inputs always produce the same output 2. Outputs are reproducible at any point in the future 3. Transforms can be reasoned about independently

If any solution breaks determinism, it breaks Foundry's model.

5.2 Why Foundry Forces Explicit Transforms

In many data platforms, engineers: - Run ad-hoc queries - Overwrite tables - Debug live data

Foundry **explicitly forbids this** because it makes: - Auditing impossible - Rollbacks unreliable - Root-cause analysis ambiguous

Transforms exist to ensure: - Every change is attributable - Every output is explainable - Every error is traceable

Exam Insight: If a proposed solution allows silent mutation or undocumented logic, it is almost certainly wrong.

Chapter 6 — Types of Transforms (Deep, Comparative Understanding)

6.1 SQL Transforms

What SQL Transforms Are Best At

SQL transforms are ideal when: - Logic is declarative - Business rules must be transparent - Non-engineers may review logic

Examples: - Filtering invalid records - Aggregations - Dimension enrichment - Standard joins

Why Palantir Favors SQL When Possible

SQL has three properties Palantir values: 1. **Readability** — intent is obvious 2. **Governability** — logic is reviewable 3. **Determinism** — fewer hidden side effects

Limitations of SQL Transforms

SQL is not ideal for: - Complex procedural logic - Stateful algorithms - Advanced custom processing

Exam Judgment Rule: If SQL can express the logic clearly, SQL is the preferred answer.

6.2 Code Transforms (Python / PySpark)

When Code Is Necessary

Code transforms are appropriate when: - Logic is algorithmic - Multiple processing steps are required - Performance tuning is needed

Examples: - Complex deduplication - Sessionization - Feature engineering

Responsibilities That Come With Code

With power comes responsibility. Code transforms require you to: - Explicitly manage schemas - Handle nulls and edge cases - Ensure deterministic output

Foundry does **not** protect you from bad code.

Exam Trap: Choosing code “because it is more flexible” without justification is incorrect.

6.3 Pipeline Builder (Low-Code Transforms)

What Pipeline Builder Is For

Pipeline Builder is designed for: - Simple joins - Filters - Projections

It is intentionally constrained.

Why It Exists

Pipeline Builder allows: - Faster iteration - Broader participation

But it is **not** a replacement for engineering judgment.

Exam Insight: Pipeline Builder is rarely the correct answer for complex or critical logic.

Chapter 7 — Incremental Processing (The Most Important Chapter)

7.1 Why Incremental Processing Exists

Incremental processing exists to: - Reduce compute cost - Improve freshness - Scale pipelines

However, it introduces **correctness risk**.

Foundry treats incremental logic as **opt-in complexity**, not a default.

7.2 The Three Preconditions for Safe Incremental Logic

Incremental processing is only safe if **all three** conditions hold:

1. **Stable Primary Key**
2. Each real-world entity maps to exactly one key
3. Keys never change
4. **Reliable Change Detection**
5. Timestamps, version numbers, or CDC logs
6. **Idempotent Writes**
7. Reprocessing the same input does not duplicate data

If any condition fails, the pipeline must be full-refresh.

7.3 Common Incremental Patterns

Append-Only

- New rows are added
- Existing rows never change

Safe only when: - Data is immutable by nature (e.g., event logs)

Merge / Upsert

- Existing rows may be updated
- Requires careful deduplication

This is where most bugs occur.

7.4 Late-Arriving and Corrected Data

Late data breaks naive incremental logic.

Correct handling requires: - Identifying affected historical windows - Reprocessing those windows - Preserving original raw records

Exam Pattern: Questions often describe late data implicitly. The correct answer always preserves historical truth.

Chapter 8 — Deduplication (Often Underestimated)

8.1 Why Deduplication Is Hard

Duplicates arise from: - Retries - Source bugs - Event replay

Naive deduplication: - Drops valid records - Loses corrections

---

8.2 Correct Deduplication Strategy

A correct strategy: 1. Define the business key 2. Define recency or priority 3. Preserve all raw evidence

Deduplication belongs in **curated layers**, never raw.

Chapter 9 — Business Logic Placement

9.1 What Is Business Logic?

Business logic includes: - Classification rules - Thresholds - Eligibility criteria

It encodes **organizational belief**, not fact.

9.2 Where Business Logic Must Live

Business logic must: - Live in curated transforms - Be version-controlled - Be reviewable

Never embed business logic in: - Raw ingestion - Ontology actions (unless explicitly required)

Chapter 10 — Anti-Patterns (Exam Gold)

10.1 Common Mistakes

- Cleaning data during ingestion
- Using incremental logic without keys
- Exposing raw data to users
- Overusing code when SQL suffices

Each of these violates Foundry principles.

Chapter 11 — How This Appears in the Exam

Typical exam prompts: - “Design a pipeline for changing records” - “Optimize this pipeline for scale” - “Fix incorrect incremental logic”

Correct answers: - Emphasize safety - Preserve lineage - Prefer explicitness

---

End of Volume 2

Next volumes will cover: - Volume 3: Ontology, semantic modeling, actions - Volume 4: Data quality, governance, security - Volume 5: Debugging, operations, exam strategy