Introduction to Analytics Engineering

Building Trust in Data Systems

Harsh Punjabi
Senior Analytics Engineer
@
Lightspeed Commerce

19 - Feb - 2026

What You'll Learn Today

ORU

1. Where Analytics Engineering fits in data organizations and why it exists
2. How to make grain decisions in analytical data modeling
3. How to define canonical metrics that handle financial edge cases
4. Career pathways into Analytics Engineering
5. **Plus: Portfolio-grade take-home project**

The Value You'll Take Home

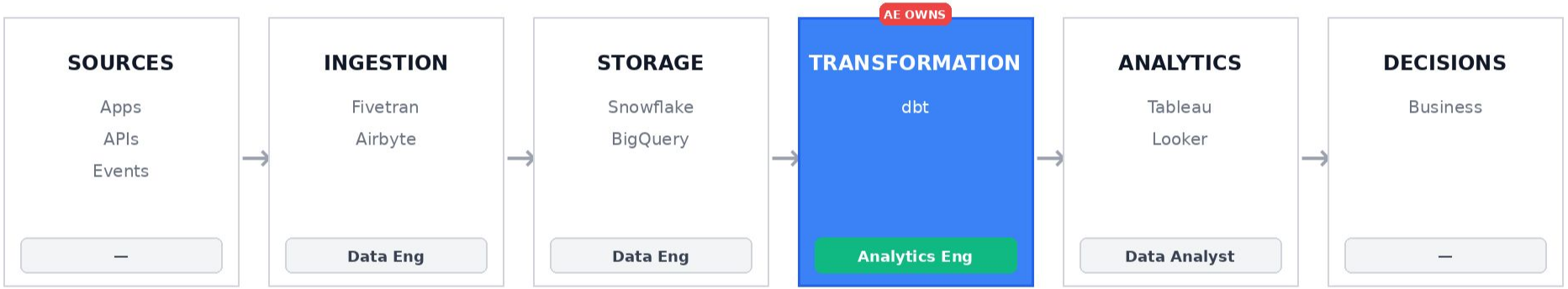
- Understanding of the AE role and organizational fit
- Framework for modeling decisions (grain, metrics)
- Real life FinTech project with 50K+ transactions
- GitHub-ready portfolio project (6-10 hours to complete)
- Clear career guidance for Toronto/Montreal market

The Data Team - Who Does What

Data Infrastructure - Where AE Lives

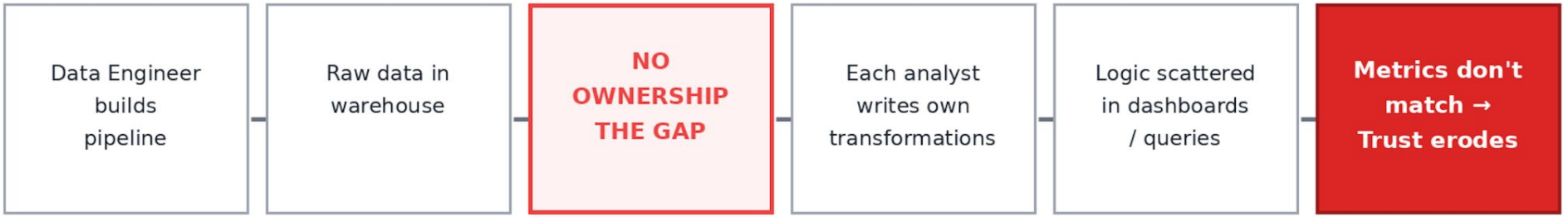
The AE Layer:

- Staging models (clean raw data)
- Business logic transformations
- Fact tables & dimensions
- Canonical metrics
- Data quality tests



The Gap That Created Analytics Engineering

Before AE existed:



Three problems:

1. **Decentralized logic** = Metric chaos (Finance and Product report different numbers)
2. **Undocumented transformations** = Black boxes (analyst leaves, knowledge lost)
3. **No ownership** = Silent failures (dashboard breaks, nobody knows why)

What AE Owns (and Doesn't Own)

✅ Analytics Engineer OWNS	❌ Analytics Engineer DOESN'T OWN
Transformation logic (SQL models)	Data pipelines/infrastructure
Metric definitions	Source data ingestion
Analytical data models	Exploratory analysis
Data quality for analytics	Production ML models
Business logic documentation	BI tool administration

Accountability: "If the metric is wrong, it's on AE"

Why FinTech Especially Needs AE

Four reasons:

1. **Regulatory Requirements**
   - Financial metrics must be auditable
   - Clear lineage: "How did we calculate this number?"
2. **High Stakes Accuracy**
   - Money involved - mistakes cost real dollars
   - Example: Miscalculated interchange fees = $100K loss
3. **Complex Financial Logic**
   - Multi-currency, fee structures, refunds, chargebacks
   - Transaction date ≠ settlement date ≠ recognition date
4. **Multi-Stakeholder Needs**
   - Finance needs GAAP-compliant revenue
   - Product needs user behavior metrics
   - Risk needs fraud indicators
   - **Everyone needs to trust the numbers**

Exercise - The NorthPay Metric Mess

Scenario: PaymentCo

- Canadian payment processor, $50M volume/month
- Growing fast (Series B)

The Problem (Monday exec meeting):

Three leaders present different "December Transaction Volume":

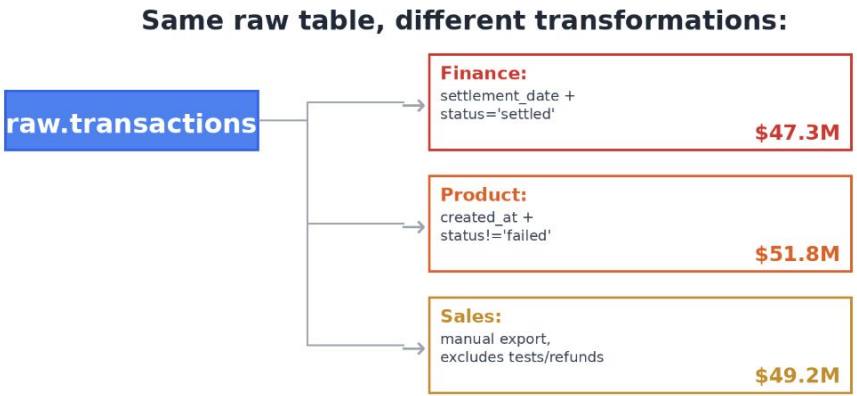
- **CFO (Finance):** $47.3M
- **VP Product:** $51.8M
- **Head of Sales:** $49.2M

Think About:

1. Where in the data infrastructure does this problem live?
2. Why do the three numbers differ?
3. Who should own the fix?
4. What would the AE solution look like?

The Diagnosis (Debrief)

Why numbers differ:



Key differences:

- **Date field:** settlement vs created (timing)
- **Status logic:** 'settled' vs '!= failed' (different exclusions)
- **Test data:** included vs excluded
- **Refunds:** netted out vs not

The AE Solution:

1. ✅ Canonical transaction model (one source, all date fields)
2. ✅ Multiple defined metrics (each with clear business logic)
3. ✅ Governed access (marts, not raw tables)
4. ✅ Documentation (when to use which metric)

Three Types of Data

Raw Data → Analytical Data → Decision-Ready Data

Raw Data (What DE delivers)

- As received from sources
- Minimal transformation
- Example: `payment_gateway_events` with JSON

Analytical Data (What AE builds) ← **Focus today**

- Modeled with business logic
- Queryable structure (facts & dimensions)
- Example: `fct_transactions` with proper grain

Decision-Ready Data (What DA consumes)

- Aggregated metrics, KPIs
- Example: "Daily Revenue by Payment Method"

Key Point: You can't skip Raw → Decision-Ready. AE builds the foundation.

What Makes a Metric Canonical?

A metric is **canonical** when:

1. ✅ ONE authoritative definition
2. ✅ Centrally maintained & versioned
3. ✅ Edge cases handled explicitly
4. ✅ Business logic documented
5. ✅ Everyone uses the same calculation

Why FinTech needs this:

- Regulatory reporting requires auditability
- Financial metrics affect investor decisions
- Mistakes are expensive
- Multi-stakeholder orgs need alignment

Good vs Bad Metric Definition

Example: Monthly Recurring Revenue (MRR)

❌ **Bad Definition:**

MRR = SUM(subscription_amount)
WHERE active = TRUE

Problems: What's "active"? Annual vs monthly subs? Free trials? Mid-month changes?

✅ **Good Definition:**

Metric: monthly_recurring_revenue
Owner: Finance
Definition: Sum of normalized monthly subscription value for active subscriptions as of month-end
Includes:
- Paid subscriptions (status: active, past_due) - Annual subscriptions normalized to monthly (/12)
Excludes:
- Free trials, test accounts - Canceled subscriptions - One-time payments
Edge Cases:
- Mid-month cancels: count until month-end - Mid-month upgrades: use new amount from change date - Annual prepay: normalize to monthly equivalent

Edge Cases That Break Financial Metrics

Scenario	Naive Approach	Problem	AE Solution
Refunds	`SUM(amount)` includes negatives	Refunds in different month skew revenue	Separate gross/refunds/net metrics
Multi-currency	Convert at today's rate	Historical revenue changes with FX	Store original + CAD, convert at transaction time
Failed transactions	Include or exclude?	Unclear definition	Define separate metrics: "attempted volume" vs "settled volume"
Partial refunds	Single refund field	Can't track multiple partials	Separate refunds table, join to calculate net
Backdated transactions	Use transaction date	Settlement timing matters for cash	Store both created_at and settled_at

Key Point: Edge cases aren't edge cases in FinTech - they're the business!

The Metric Drift Problem

What happens without AE:

Month 1: Analyst A builds "Active Users" in Dashboard 1 **Logic:** ≥1 transaction in last 30 days

Month 3: Analyst B builds "Active Users" in Dashboard 2 **Logic:** ≥1 login in last 30 days (didn't know about Dashboard 1)

Month 6: Product Manager builds Dashboard 3 **Logic:** account status = 'active' (different again!)

Month 12: Executive asks "How many active users?" → 3 dashboards, 3 answers, zero trust

How AE prevents this:

- Canonical metrics in code (dbt metrics, LookML)
- ONE definition, consumed everywhere
- Changes go through review
- Documentation explains when to use which metric

The Analytics Engineering Skillset

ORU

1. Technical Rigor

- SQL (CTEs, window
- functions, joins)
- Git/version control
- dbt, data warehouses
- Testing frameworks

2. Business Impact

- Understand
- stakeholder needs
- Translate to data
- models
- Think about edge
- cases
- Anticipate data usage

3. Systems Thinking

- See how pieces
- connect
- Understand
- dependencies
- Think about failure
- modes
- Long-term
- maintainability

What makes you great:
All three, not just #1

Career Path & Canadian Market

Entry Points

- From Data Analyst: Add engineering rigor
- (Git, dbt, testing)
- From Data Engineer: Add business context
- From Bootcamp: Build portfolio projects
- From Career Switch: Leverage domain
- expertise + learn SQL

Progression

Junior AE (0-2 yrs) → AE (2-4 yrs) →
Senior AE (4-7 yrs) → Staff/Principal
↘ Manager

Toronto/Montreal Market

- Industries: FinTech (Wealthsimple, Neo),
- SaaS (Shopify), E-commerce

- Salary (Toronto, 2024):
- Junior $70-90K
- Mid $90-120K
- Senior $120-160K

- Demand: Growing rapidly
- (dbt adoption driving this)

How to Stand Out

- Portfolio projects (like today's take-home!)
- Contributions (dbt Slack, blog posts)
- Network (Toronto meetups, LinkedIn)

Additional Resources

Your Portfolio Project - NorthPay

What You'll Build: Complete analytics layer for a FinTech payment processor

Dataset: 50K+ realistic transactions

- Multi-currency (CAD, USD, EUR)
- Refunds (full and partial)
- Multiple merchants with fee structures
- Edge cases: failed retries, same-day refunds, currency conversion

What You'll Deliver:

1. **Diagnosis:** Analyze current data problems
2. **Design:** Model your analytical layer (grain decisions, ERD)
3. **Implementation:** SQL models (staging → marts), metrics
4. **Quality:** Tests, documentation, README

Portfolio-Grade Means:

- GitHub repo with professional README
- Demonstrates all 3 skills: technical + business + systems thinking
- Something you can show in interviews

Project Details & Next Steps

You'll Receive:

- 4 CSV files (transactions, refunds, merchants, fee plans)
- Setup guide (DuckDB - free, local / BigQuery sandbox / Snowflake trial)
- Project brief with requirements
- Evaluation rubric
- Templates (README, metrics, SQL structure)
- FAQ document

Time Commitment: 6-10 hours

How to Use This:

- Complete it as portfolio project
- Put on GitHub + LinkedIn
- Reference in job applications
- Use as conversation starter in interviews

Key Takeaways & Action Plan

What We Covered:

1. **AE role:** Owns transformation layer, builds trust in data
2. **Grain:** Most important modeling decision (atomic, documented)
3. **Metrics:** Canonical definitions with edge cases handled
4. **Career:** Multiple entry points, strong demand in Toronto/Montreal

Your Next Steps:

This Week:

- Complete NorthPay project
- Join dbt Slack community

This Month:

- Put project on GitHub with great README
- Reach out to 3 AEs for informational interviews

This Quarter:

- Build second portfolio project
- Apply to AE roles or talk to your manager about transition

Resources shared in project package. Questions?

What is Grain?

Definition: The level of detail in your fact table.
Answers: "What does one row represent?"

Example - FinTech Transactions:

❌ **Bad Grain:**

Table: daily_transactions
Grain: One row per day
Problem: Can't see individual transactions, can't debug, can't filter by payment method

✅ **Good Grain:**

Table: fct_transactions
Grain: One row per transaction attempt
Benefit: Can aggregate any way, can debug issues, can handle all edge cases

The Golden Rule:
Choose the most atomic grain that serves your questions.
You can always **aggregate UP**, but you can never **disaggregate DOWN**.

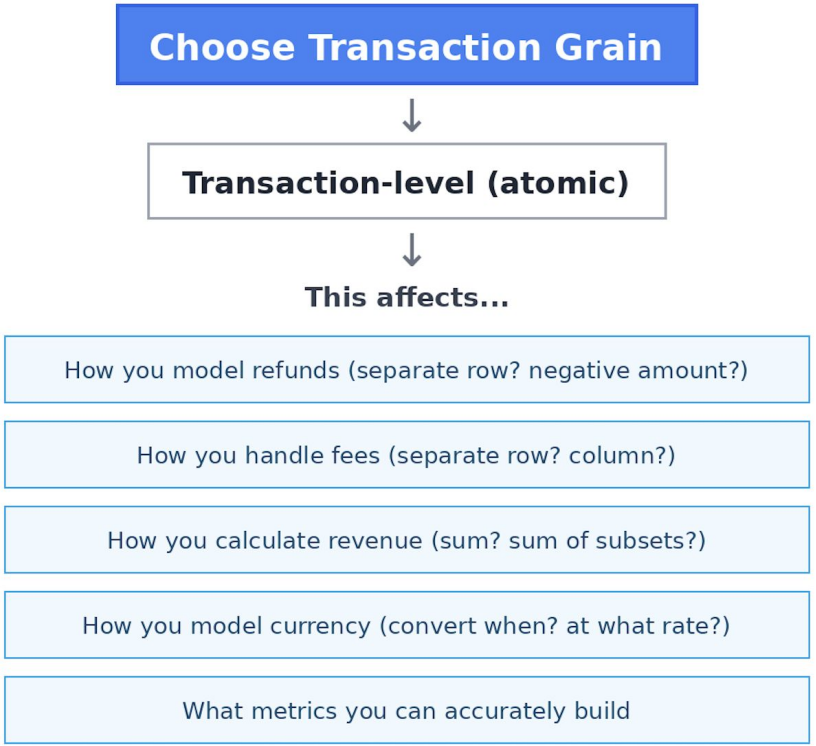
Why Grain is THE Most Important Decision

Grain decisions cascade into everything:

Wrong grain = Every downstream metric is compromised

Common Mistakes:

1. Aggregating too early (daily table → can't debug spikes)
2. Mixing grains (transactions + refunds → double-counting)
3. Unclear documentation ("user transactions" → per user? per day? what?



Exercise - LendTech Grain Decisions

Context: Personal lending platform (loans, payments, fees)

Your Task:
For each business question, choose the right grain and justify:

Question 1: "What's our loan portfolio balance as of any date?" (Think: Do you need daily snapshots? Can you calculate from transactions?)

- One loan, One loan on one specific day, or One payment transaction. Pick One & Explain Why

Question 2: "What % of payments are made on time vs. late?" (Think: Loan-level or payment-level grain?)

- One Loan, or One Individual Payment. Pick One & Explain Why

Question 3: "What's total fee revenue from late fees this month?" (Think: Fees as separate transactions or columns?)

- One Loan, or One Fee Charge. Pick One & Explain Why.

Deliverable: Grain choice + 2-3 reasons why

Exercise Solutions (Debrief)

Q1: Portfolio Balance

- ✅ **Grain:** One loan on one specific day - Daily loan snapshot (loan_id + date)
- **Why:** Point-in-time accuracy (regulatory requirement), trend analysis
- **Why NOT transaction-level:** Complex windowing to reconstruct balance

Q2: Payment Performance

- ✅ **Grain:** One individual payment - Payment-level (one row per scheduled payment)
- **Why:** Track on-time %, identify patterns, support collections
- **Why NOT loan-level:** Loses "which payments were late", can't see trends

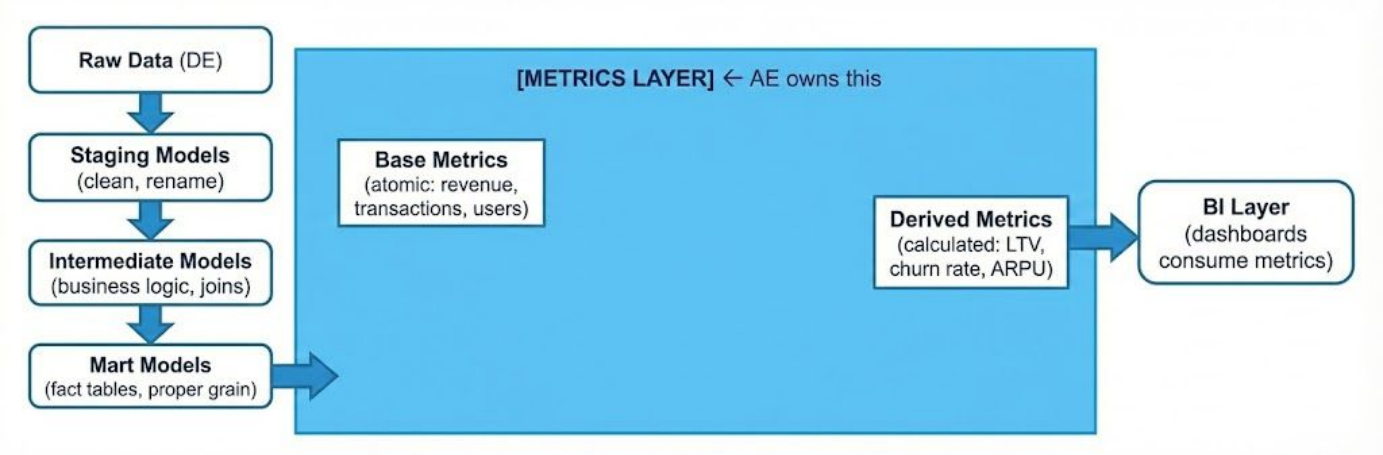
Q3: Fee Revenue

- ✅ **Grain:** One fee charge - Fee transaction-level
- **Why:** Clear revenue recognition (assessed vs collected), auditable, handles refunds
- **Why NOT column:** Multiple fee types, can't track fee-specific refunds separately

Key Takeaway: Grain depends on your questions. Document it explicitly!

Metrics Layer Architecture

The Pattern:



Self-Serve Without Chaos:

- Analysts query marts freely ✅
- Analysts build custom dashboards ✅
- Analysts combine metrics ✅
- Analysts CANNOT change metric definitions ❌

Result: Freedom within guardrails