SQLBITES.NET

Stop Borrowing Contoso

Build realistic demo data for Power BI.

Jonathan Stewart · SQLLocks / SQLBites

JONATHAN STEWART · SQLLOCKS@SQLBITES.COM

About Me

25+ Years in Data & Analytics

Helping enterprises migrate to and scale on Microsoft Fabric. From architecture and data modeling to Power BI and production governance.

→ **Creator/Founder: SQLBites Community**

Building the next generation of data practitioners.

→ **Migrating Enterprises to Fabric**

Real production migrations across healthcare, finance, retail, and manufacturing.

→ **Hundreds of Technical Talks Globally**

International speaker on data architecture and analytics engineering.

Sound BI

"IS THIS CONTOSO?"

The stakeholder recognized the product names

The conversation shifted from your report to your data. The decision got deferred.

IT WASN'T THE REPORT'S FAULT.

Your visuals were fine. Your DAX was fine.

The fake data broke the spell.

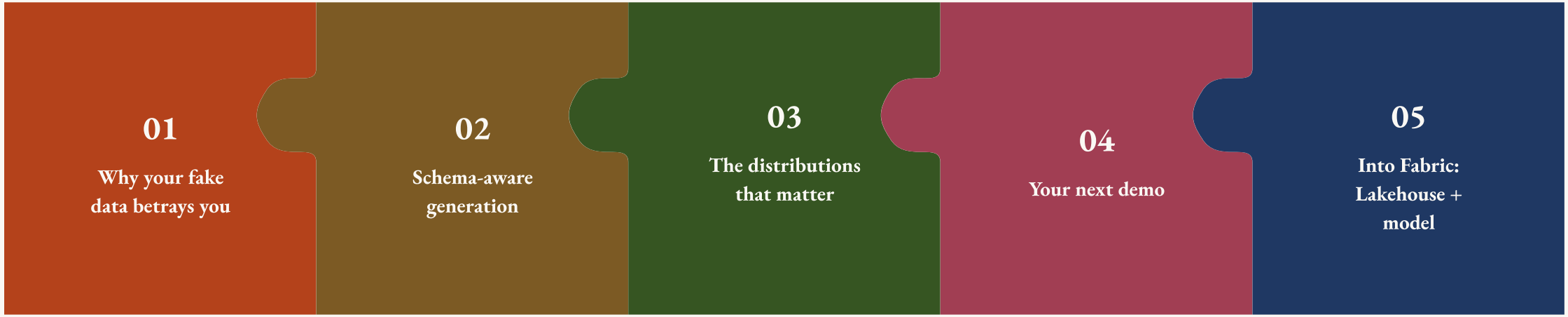
YOU'VE REBUILT THE SAME POC THREE TIMES WITH PRETTIER FAKE DATA.

The fix isn't polish. It's plausibility

That's a data problem, and it's solvable.

WHAT WE'LL COVER

Why fake data fails, and how to fix it *in three lines.*



```
PS> SELECT * FROM customers; -- Contoso,
```

CHAPTER ONE

WHY FAKE DATA BETRAYS YOU

01

The tells they catch before you finish the sentence.

PLAUSIBLE INDIVIDUAL VALUES. ZERO REALISTIC RELATIONSHIPS.

A name, an address, a number

Fine alone, nonsense together. Faker has no idea this customer's region should affect what they buy.

What your audience actually sees

01

Uniform distributions

Flat bars where real data has a long tail.

01

02

No correlation

Florida and Minnesota buy identical mixes.

02

03

Broken FK integrity

Orphan orders, customers with no region.

03

04

No temporal pattern

Dead-flat trend lines, no seasonality.

04

Plus random lifecycle statuses. 50% "Cancelled" orders.

YOUR AUDIENCE READS CHARTS FOR A LIVING.

They feel the fake before they can name it

And once they distrust the data, they distrust the platform.

You need the third one.

PHASE I · LEVEL 1

Random

1

Garbage values.

PHASE II · LEVEL 2

Plausible

2

Faker-grade.

PHASE III · LEVEL 3

Statistically realistic

3

The sweet spot.

PHASE IV · LEVEL 4

Production-mirrored

4

Can't have it (PII).

CHAPTER TWO

SCHEMA-AWARE GENERATION

02

Generate a believable instance of your whole schema.

GENERATE A SCHEMA, NOT COLUMNS.

Define tables + relationships + dependency order

The engine builds in the right order. Stop thinking "fill this column". Start thinking "generate a believable instance of my schema."

Parents before children.

You can't generate an order for a customer who doesn't exist yet. The engine topologically sorts the schema.

EVERY FOREIGN KEY POINTS AT A REAL PRIMARY KEY.

0 orphans. Validated

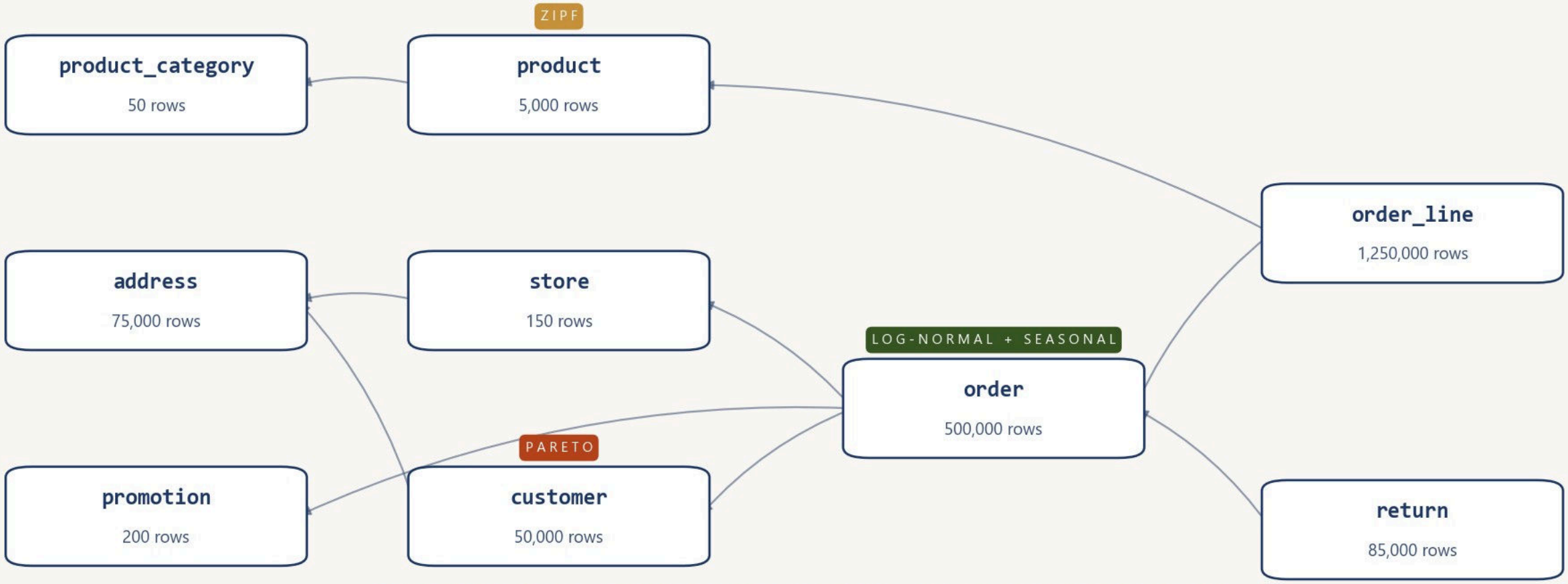
Including composite keys. This is the difference between "looks real in one table" and "holds together across a star schema." Your relationships actually work in the model.

SCHEMA-AWARE GENERATION

The retail schema.

9 tables, real FKs, parents generated before children.

BADGES = WHERE DISTRIBUTIONS ATTACH



GENERATION ORDER: PARENTS BEFORE CHILDREN, FKS VALID BY CONSTRUCTION

SQLLOCKS-SPINDLE 2.14.0 . SCALE=MEDIUM . SEED=42 . 1,965,400 ROWS . COMPUTED FROM THIS RUN

Three lines to generate it

```
from sqllocks_spindle import Spindle, RetailDomain

tables = Spindle().generate(
    domain=RetailDomain(),
    scale="medium",   # 2s, 500k orders
    seed=42,          # identical every run
).tables             # 9 DataFrames
```

- *scale is a preset: small / medium / large*
- *seed makes rehearsal == live*
- *tables = dict of 9 DataFrames, real FKs*

[LIVE DEMO 1] pip install, three lines, 1.97M rows in ~2s.

CHAPTER THREE

THE DISTRIBUTIONS THAT MATTER

03

Skew is the difference between real and random.

DISTRIBUTIONS ARE THE WHOLE GAME.

Uniform data breaks every aggregate

Every chart, every KPI. Real business data is skewed. If the distribution is wrong, no amount of row volume saves you.

Skew, on purpose: the four that matter for retail

Pareto

Revenue concentration. Top 10% of customers carry about 35% of revenue, the curve your CFO expects to see.

1

Zipf

Product popularity. Top SKU is 39% of order lines, top 10 are 77%. A few heroes, a long tail.

2

Log-normal

Order and return amounts. Mostly small, a few big, never a neat bell curve.

3

Seasonal

December peak, 9% weekend dip, weekday rhythm. Time intelligence has something to find.

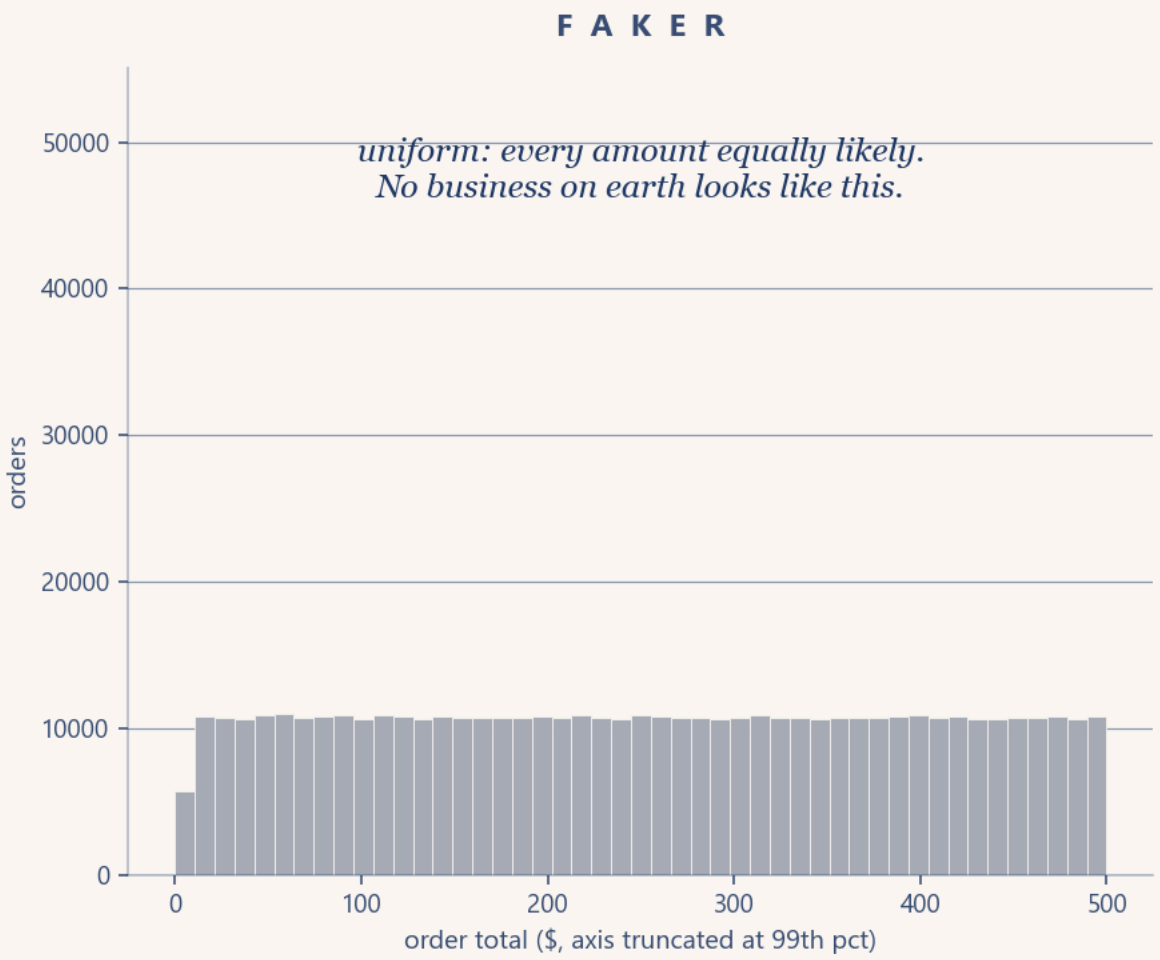
4

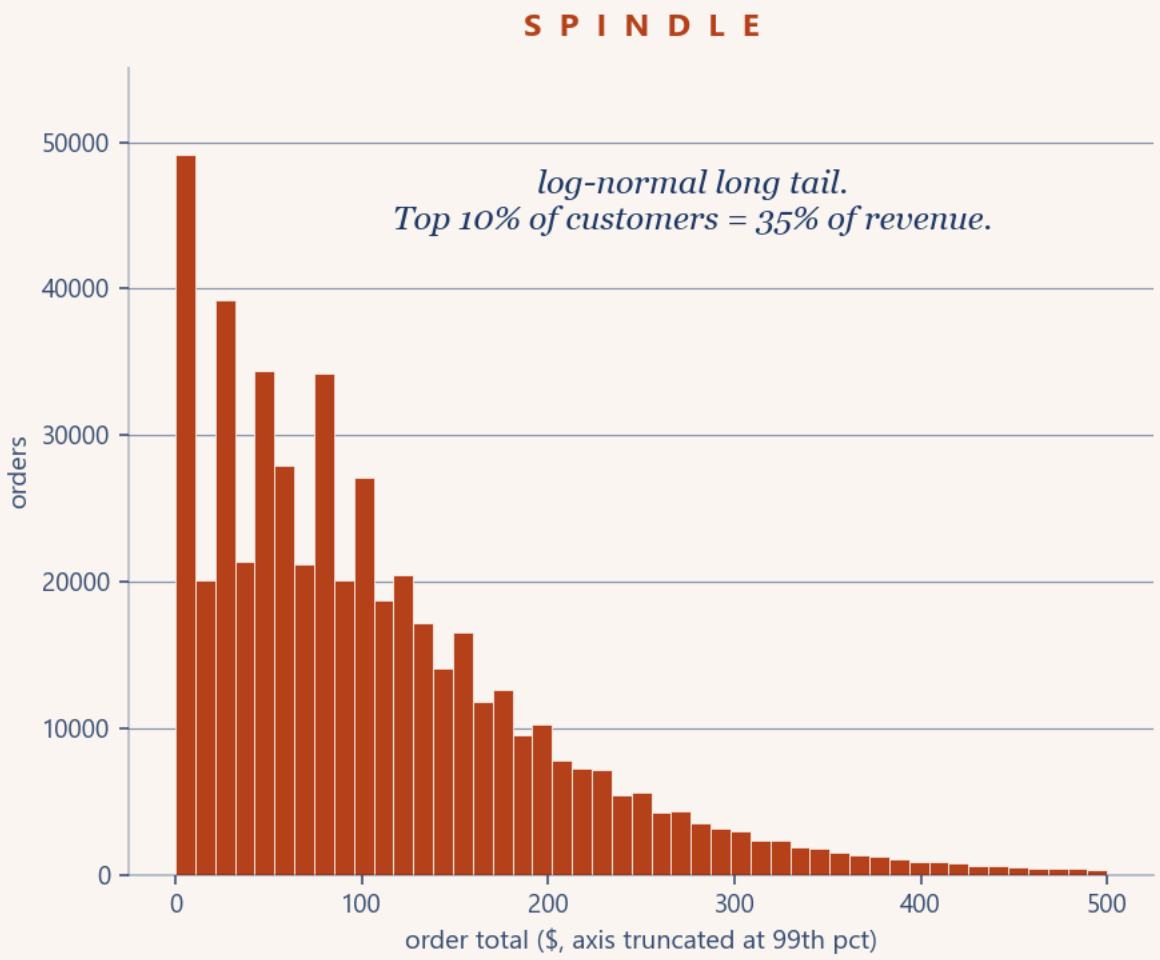
All validated by running the engine.

THE PERSUASION SHOT

Revenue: flat vs Pareto.

The same 500,000 orders each. Completely different believability.





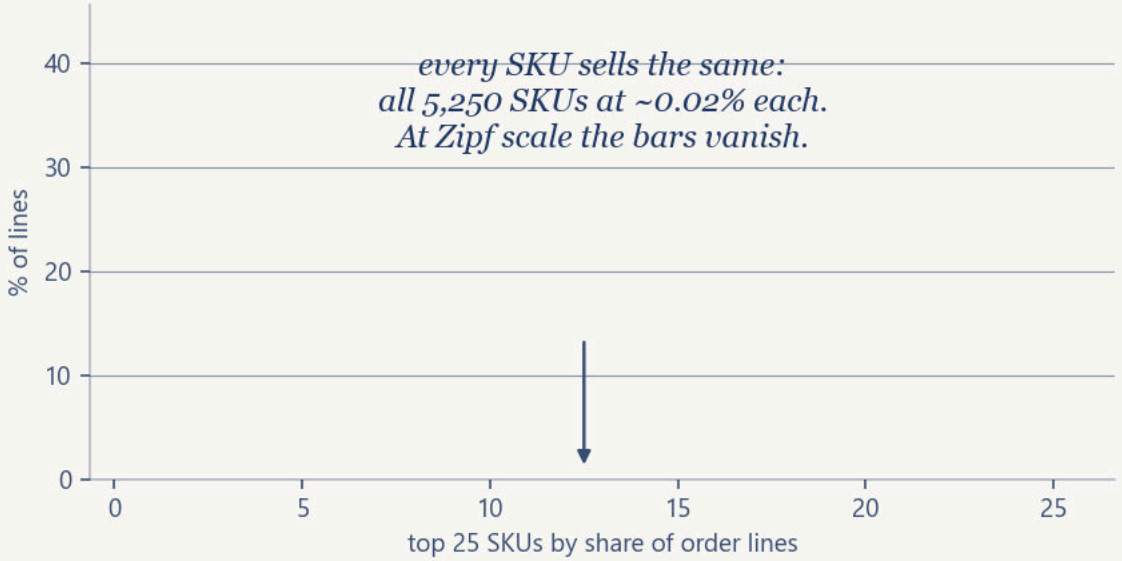
SQLLOCKS-SPINDLE 2.14.0 . SCALE=MEDIUM . SEED=42 . 1,965,400 ROWS . COMPUTED FROM THIS RUN

THE SECOND PUNCH

Popularity and integrity.

Uniform vs Zipf on top. Orphans vs zero orphans below.

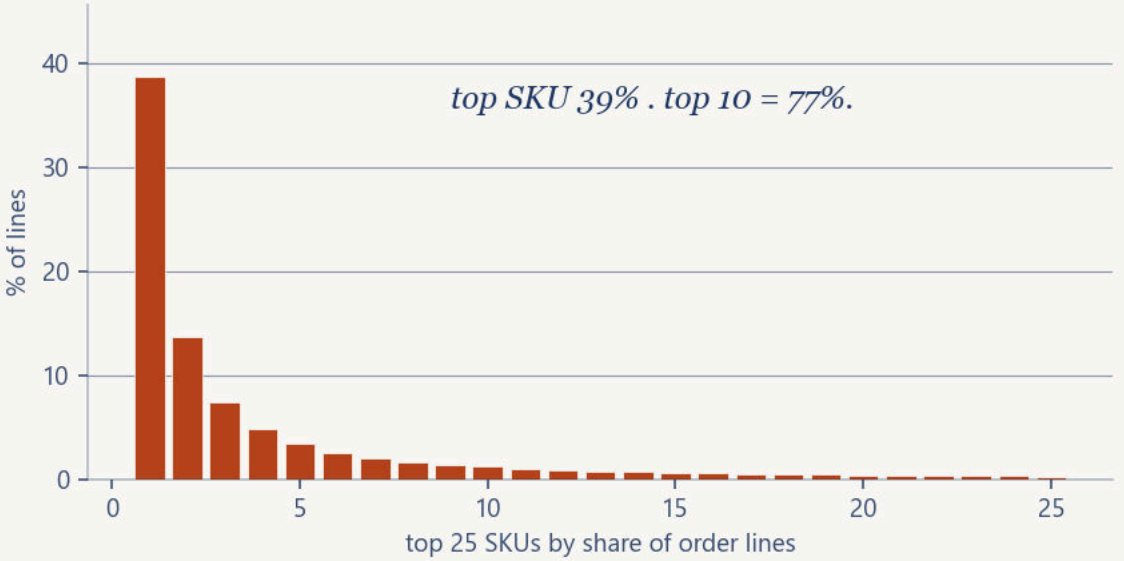
FAKER



✗ **59,775 orphan order_lines (4.8%)**

naive random keys, no parent check

SPINDLE



✓ **0 orphans in 1,250,000 lines**

FKs valid by construction (checked this run)

THIS IS WHAT MAKES A SLICER INTERACTION FEEL REAL.

Geography → product mix · tier → order value · season → category

Click Florida, the product mix changes the way it would in real life. Faker can't do this.

LIFECYCLE + TEMPORAL

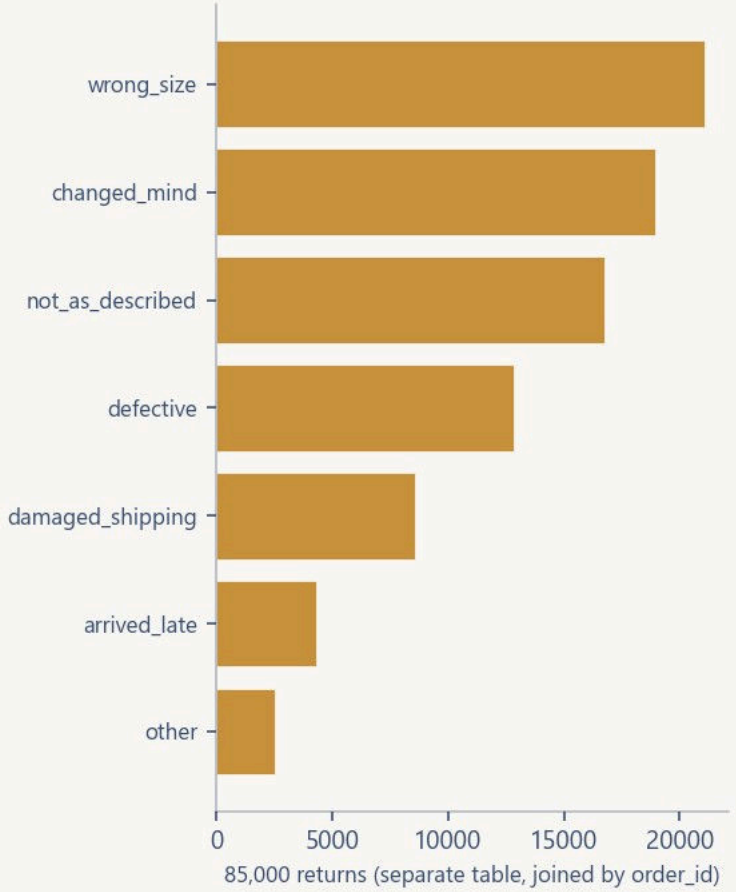
The data has a pulse even standing still.

Status mix, returns by reason, December peaks, weekend dip. All generated.

ORDER STATUS

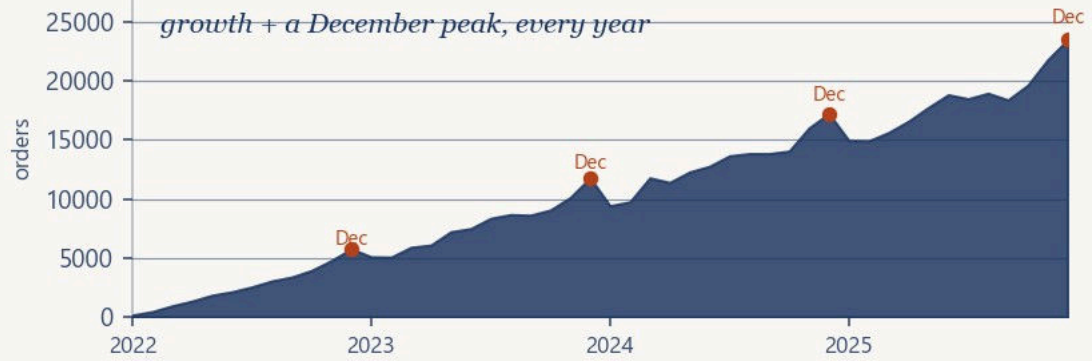
completed 77% returned 9% shipped 8%
cancelled 4% processing 2%

RETURNS BY REASON

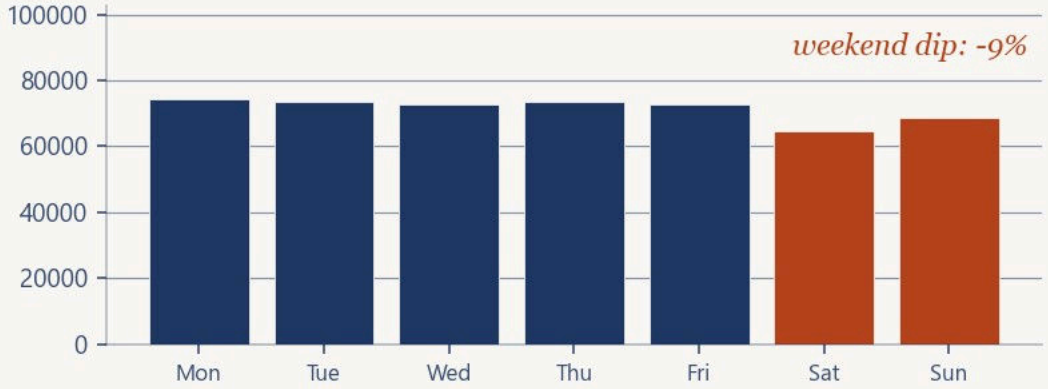


85,000 returns (separate table, joined by order_id)

ORDERS BY MONTH



ORDERS BY WEEKDAY



THE CONTOSO EVERYONE BORROWS STOPPED IN APRIL 2024.

~26 months stale, today

Don't even have to ditch it. Spindle can bring the borrowed data up to date. Watch.

Contoso, brought *current*.

STALE

26mo

Last sale: Apr 2024.

CURRENT

0days

Filled to today, valid FKs.

Same schema. Same distributions. The date range now runs to today.

Spindle continues Contoso

Profile the real data, then generate Day-2 rows that match it.

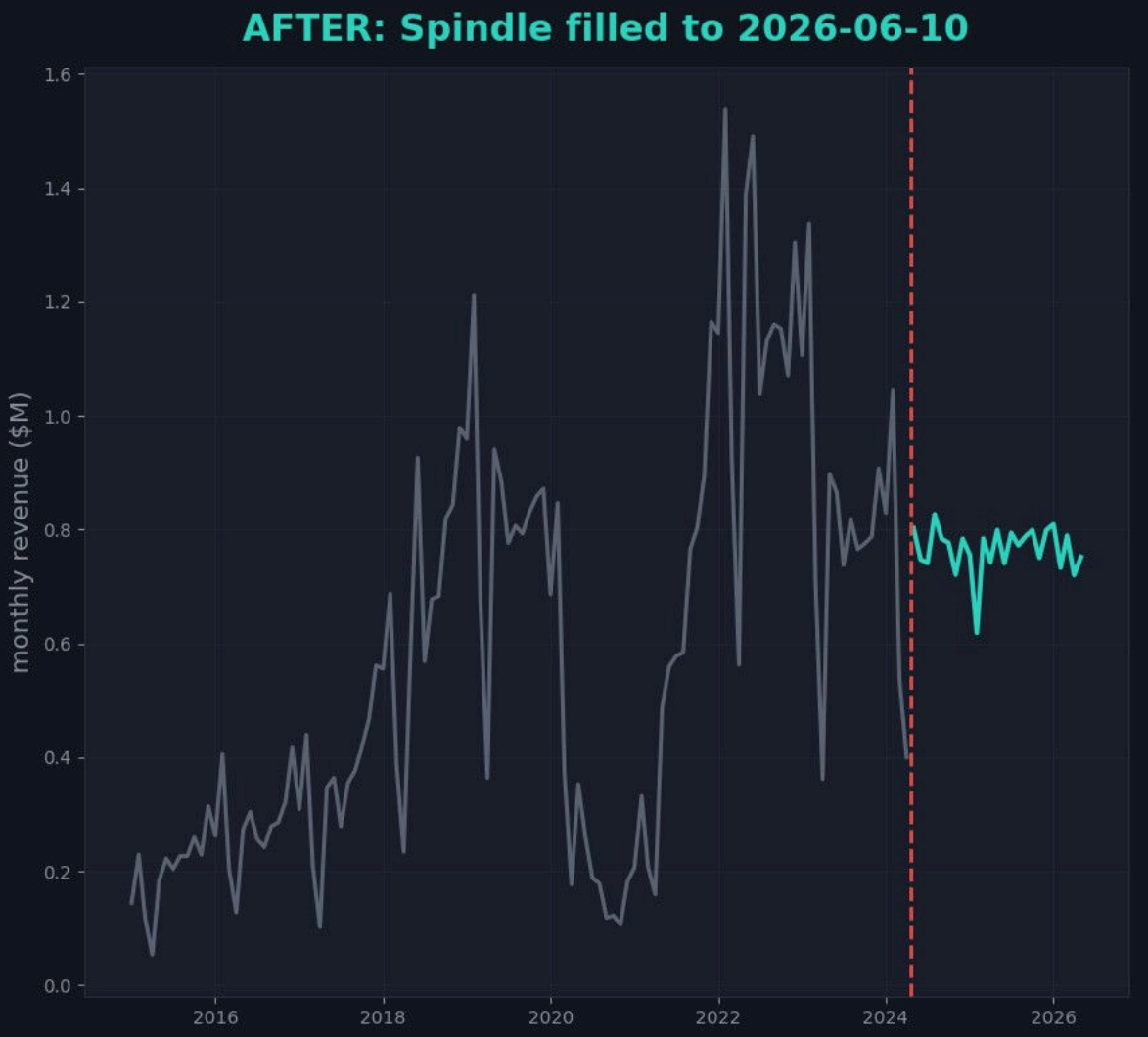
```
prof   = DataProfiler().profile_dataset(contoso)
schema = SchemaBuilder().build(prof, domain_name="contoso")
schema.tables["sales"].primary_key = ["OrderKey"]   # keys marked FK → kept valid

delta = ContinueEngine().continue_from(
    contoso, schema=schema,
    config=ContinueConfig(insert_count=60_000, seed=11))
…
```

- *FK columns excluded from perturbation, 100% valid keys*
- *Measures jitter within Contoso's own distribution*

Spindle matched Contoso, it didn't fake it.

SCRIPTED DEMO · REAL CONTOSO V2 100K + ContinueEngine(seed=11)

Contoso, brought current: the curve continues past the gap



left: last sale 2024-04-20 (stale 26 months) · right: +60,000 Spindle Day-2 rows, FKs 100% valid, NetPrice mean 324 vs 327

YOU SOURCE-CONTROL YOUR SCHEMA. YOU'VE NEVER SOURCE-CONTROLLED THE SHAPE.

Distributions, correlations, fan-out, null rates

DDL, dbt, migrations: all in git. The statistical shape of your data, where "it broke in prod" actually lives, is invisible to git. Schema-as-code is solved. Shape-as-code is new.

The shape is a file

Capture any dataset's distributions as a committable artifact. Captured, not canned.

```
$ spindle profile capture ./prod_extract -o prod.profile.json


# prod.profile.json
{ "distributions": {
    "customer.loyalty_tier": {
      "Basic": 0.55, "Silver": 0.25, "Gold": 0.13, "Platinum": 0.07 } },
  "ratios": { ... } }    # FK fan-out, null rates
```

- *SafeProfile by construction: winsorized bounds, k-anonymized categories, PII gate*
- *Safe by DEFAULT, kilobytes, zero raw rows. Human-readable and git-diffable*

The same file is the contract AND the generator.

What the shape file unlocks in production

01	**Shape-drift CI gate**	*profile diff prod staging --threshold 0.2 exits 1. Real run: Continent NA 0.53 → 0.00, drift 3.19. Build fails.*
02	**Repro prod bugs with no prod data**	*Ship the kilobyte profile, regenerate the failing shape locally. Nothing sensitive moves.*
03	**git diff for your data's behavior**	*A PR shows distribution changes, not just column changes.*
04	**Prove the twin: a fidelity score**	*spindle compare real synth scores every column 0-100: KS, chi-squared, null rates, cardinality. The receipt for "looks real."*

Tonic and Gretel generate data. Neither hands you a committable, diffable shape file.

Break it on purpose: the chaos generator

Late arrivals

Out-of-order timestamps, DST boundaries, facts that show up before their dimensions.

1

Broken keys

Duplicate business keys, orphaned foreign keys. The bugs that pass clean tests.

2

Hostile values

Nulls in "never null" columns, out-of-range numbers, wrong types.

3

Volume chaos

10x spikes, empty batches, single-row batches. Same seed, same disaster, every rehearsal.

4

Your pipeline passed every test. It has never met data like this.

CHAPTER FOUR

YOUR NEXT DEMO

04

Three lines on Monday morning.

YOU STOPPED POLISHING THE REPORT AND FIXED THE DATA UNDERNEATH IT.

Same effort. Completely different reception.

The credibility problem from the opening is solved at the data layer.

PIP INSTALL SQLLOCKS-SPINDLE

12 domains · retail today · extensible schema

Healthcare, financial, supply chain, and more. And profile capture learns YOUR data's shape tonight.

3 things for Monday

01	**pip install sqllocks-spindle**	*Generate retail with a fixed seed.*
02	**Load to a Lakehouse**	*Point a Direct Lake model at it.*
03	**Rebuild your stalled POC**	*On data that behaves like production.*

Go re-run the demo that stalled. On data that behaves.

CHAPTER FIVE

INTO FABRIC: LAKEHOUSE, MODEL, PULSE

05

From DataFrames to a Direct Lake model, and a dashboard that breathes.

Generated → Lakehouse → semantic model

01	**DataFrames**	*9 pandas frames from Spindle.*
02	**Lakehouse Delta tables**	*One Spark write loop.*
03	**Direct Lake model + report**	*Relationships auto-detect: FKs are real.*
04	**Pulse**	*A live dashboard ticking on top.*

Generate and load

The loop that already landed the tables. Pre-loaded; show, don't re-run.

```
tables = Spindle().generate(
    domain=RetailDomain(), scale="medium", seed=42).tables

for name, df in tables.items():
    (spark.createDataFrame(df)
          .write.mode("overwrite")
          .format("delta").saveAsTable(name))
```

- *9 Delta tables appear in the Lakehouse*
- *`order` / `return` are reserved: backtick them*

The Lakehouse already has the tables. Flash the SQL endpoint verify if time allows.

BUILD THE DIRECT LAKE MODEL.

Relationships auto-detect. The FKs are real

Fact order + order_line, dims around them. Because the FKs hold, the model just works. Tie back to schema-awareness.

Pulse: watch it tick

01	**Notebook loop**	*Four stores generating live, store1 to store4, continuously.*
02	**Eventhouse**	*Streaming ingest, sub-second arrival.*
03	**Live dashboard**	*The report breathes while you talk.*

[LIVE DEMO 2] Generated data behaving like production, in real time.

THIS IS THE DEMO THAT WOULDN'T HAVE STALLED.

It looks like production because it behaves like production

Live data, real distributions, valid keys. Nobody asks "is this Contoso?"

Right size, every time

small (slides)

21.7k rows in 0.08s. Screenshots, static slides, unit tests.

1

medium (the live demo)

500k orders in about 2s. Fast enough to run in front of a room without losing it.

2

large (rehearsal only)

19.6M rows in about 42s. Never run it live; 42 seconds of silence stalls the room.

3

Always seed, so rehearsal == live.

Questions I always get

Q.01

Can I define my own schema/domain?

Yes. Domains are extensible.

Q.02

Is it PII-safe for training data?

Synthetic, no real PII.

Q.03

Streaming?

Batch. Pulse loops seeded micro-batches, which is what you just watched.

Q.04

How big can it go?

Scale presets; large = 19.6M rows in ~42s.

Take it with you.

SQLLOCKS

pip install sqllocks-spindle · sqllocks@sqlbites.com · SQLBites.net