Technical Walkthrough

A system that tells an underwriter which productions should cost more to insure. Six stages, 45K raw rows in, one number I can defend.

STACK

Pandas · TMDB API · PySpark
Postgres · dbt · Airflow · Power BI

PRESENTED BY

Paul Mandap

The portfolio could not be priced as it arrived

45,486	75%	84%	0
RAW ROWS RECEIVED	BUDGETS STORED AS ZERO	REVENUES STORED AS ZERO	DIRECTOR COLUMNS IN SOURCE

- You cannot price a risk when you don't know what it cost or what it earned.
- Every design decision downstream came from **what was broken here**.

Log then clean — nothing is removed silently

```python
"""Collects every removed row with the reason."""
def add(self, df, reason: str, source: str):
    if df.empty:
        return
    tagged = df.copy()
    tagged["quarantine_reason"] = reason
    tagged["source_file"] = source
    self._frames.append(tagged)
```

scripts/pandas/clean_data.py — class Quarantine

REASON	ROWS	WHAT BROKE
SHIFTED_ROW	3	Columns shifted left — a date landed in id
DUPLICATE_EXACT	6	Byte-identical copy
DUPLICATE_AFTER_NORM	44	Same film — visible only after normalising

A production company called "The"

```
# 227 films store "Jim Henson Company, The"
# instead of "The Jim Henson Company"
_INVERTED_ARTICLE = re.compile(
    r'([^,]+),\s*(The|A|An|La|Le|Der|Die)(?=,|$)')
```

scripts/pandas/clean_data.py:194

- I checked first: no genre value contains a comma.
- The check was true. **The conclusion was wrong.**
- Found by an aggregation query — **"The" ranked 25th largest studio by box office**
- Not by eyeballing the CSV.

> **Fixed at the source.** The repair lives in the cleaning layer, so the marts and dashboard can never inherit it.

Zero means "not reported" — not "cost nothing"

```python
for col in ("budget", "revenue"):
    numeric = pd.to_numeric(out[col], errors="coerce")
    out[col] = numeric.where(numeric != 0)
    out[f"{col}_scale_suspect"] = (
        out[col].notna() & (out[col] < threshold))
```

scripts/pandas/clean_data.py — fix_missing_budget()

- **0** becomes **null** → never left as literal 0
- Never **imputed** — guessing a budget invents risk data

> **The tradeoff:** only about 1 in 5 films end up priceable. That's the honest size, disclosed on every page.

I had the English title. I chose not to use it.


```
{
  "id": 197057,
  "title": "And in the Morning They Woke Up",
  "original_title": "А поутру они проснулись",
  "original_language": "ru"
}
```


data/cache/tmdb/57/197057.json

Scope discipline. The TMDB sweep was scoped to money. Title never enters the enriched record.

CHECK	VALUE
Films with non-Latin titles	38 of 45,433
Of those, priceable	2
Loss rate with them	35.7255%
Loss rate without them	35.7111%
Impact on spine metric	**0.014 pts**

Every measure keys on **movie_id** — never on the title.

Only change what is provably wrong

OURS	TMDB	OUTCOME	KEPT	COUNT
Missing	Real	FILLED	TMDB's	10,465
Below $1,000	Real	CORRECTED	TMDB's	133
Real	Differs >5%	DISCREPANCY	**Ours — logged**	1,469
Missing	Missing	UNVERIFIED	null	—

Why disagreement keeps our value: TMDB is live and has drifted. Toy Story reads $962M there vs $373M here, because TMDB now counts the 3D re-release. This model prices original theatrical performance.

4,332	8,890	+105%	44,793
PRICEABLE BEFORE	PRICEABLE AFTER	USABLE BOOK	DIRECTORS RECOVERED · 0 EXTRA CALLS

Three dbt layers, one door in

RAW
PySpark loads · explicit schemas

STAGING
Rename + cast only. No joins.

INTERMEDIATE
Joins, unnesting, all business logic

MARTS
Star schema — the only layer Power BI reads

```sql
-- Rename and cast only, no joins and no business
-- logic: that is what intermediate is for.
select
    movie_id,
    budget::numeric(15, 2)   as budget,
    revenue::numeric(15, 2)  as revenue
from source
```

dbt_project/models/staging/stg_movies.sql

- **20 models · 104 tests** — every model has a description
- **source()** only in staging — everything else uses ref()
- **2 custom tests** guard the spine metric
- **Not medallion** — this is dbt's own layering, feeding a Kimball star

Hashed keys — a rebuild refreshes, it never breaks

- Each key is built from the film's name, not a row number
- Rebuild the whole warehouse, and the keys come out exactly the same
- **A row-number key would silently break every relationship on rebuild**

REBUILD DRILL · 20 AUG	RESULT
Schemas dropped	5
Airflow triggers	1
Rebuild time	1m 22s
dbt test	PASS=104 · ERROR=0
Verification diff	byte-identical

> **It even proved the retries work.** One task failed because Postgres was still booting. retries=3 absorbed it with no human intervention.

The obvious index did nothing. I measured it.

STAGE	INDEX ON THE TABLE	ROWS DISCARDED	BUFFERS	TIME	SIZE
0	None	6,691	935	20.5 ms	—
1	Plain release_year	6,691	935	17.3 ms	336 kB
2	**Partial: WHERE is_priceable**	**0**	**753**	**14.9 ms**	**80 kB**

```
CREATE INDEX idx_movies_priceable_year
    ON movies (release_year) WHERE is_priceable;
```

scripts/sql/schema.sql — measured in queries/03 §3.4

- The query filters on **two** predicates — a plain index covers one, Postgres refused it
- The index that works is **76% smaller** than the one that didn't
- Honest limit: 27%, not 10×. The table is 4 MB and fully cached — the gap widens with scale.

Three questions, asked at every decision

- What does this data actually mean?
- What happens if I get it wrong?
- How can I prove I got it right?

> I'm still an intern, not a full data engineer.
> :)