Vintage Pipeline Specification

Specification for the vintage parquet store, data ingestion lifecycle, and panel construction pipeline. Covers the current state, known issues, and design for connecting the Hive-partitioned vintage database to the model-ready observation panel.

---

1. Vintage Store: Current State

1.1 Layout

```
data/raw/vintages/series/
  source=ces/
    seasonally_adjusted=true/
      befb4d4e…-0.parquet   (747,752 rows, 8 cols - no series_id)
      2bc608b4…-0.parquet   (747,752 rows, 9 cols - has series_id)
    seasonally_adjusted=false/
      befb4d4e…-0.parquet   (795,603 rows, 8 cols)
      2bc608b4…-0.parquet   (795,603 rows, 9 cols)
  source=qcew/
    seasonally_adjusted=false/
      befb4d4e…-0.parquet   (84,510 rows, 8 cols)
      2bc608b4…-0.parquet   (82,746 rows, 9 cols)
```

Hive partitioning by `source` and `seasonally_adjusted`. Each partition currently contains **two files** — an older generation without `series_id` and a newer generation with it. Reading all files together yields 3,253,966 rows; after deduplication (ignoring `series_id`) only **1,569,497 unique rows** remain.

1.2 Schema (Target)

Column	Type	Description
`source`	Utf8	`'ces'` or `'qcew'` (Hive partition key)
`seasonally_adjusted`	Boolean	Hive partition key
`series_id`	UInt32	Unique series identifier (optional, newer files only)

Column	Type	Description
`geographic_type`	Utf8	`'national'`, `'region'`, `'division'`, `'state'`
`geographic_code`	Utf8	`'00'` (national), FIPS for state, etc.
`industry_type`	Utf8	`'national'`, `'domain'`, `'supersector'`, `'sector'`
`industry_code`	Utf8	CES-aligned 2-digit code
`ref_date`	Date	12th of reference month
`revision`	UInt8	0–4 = prints, 9 = benchmark
`vintage_date`	Date	Publication date of this vintage
`employment`	Float64	Employment level (**units inconsistent — see §1.3**)

Uniqueness constraint (from README): `(source, ref_date, industry_type, industry_code, seasonally_adjusted, geographic_type, geographic_code, revision)` — each combination should appear once.

1.3 Known Issues

These are documented in `AUDIT.md` and confirmed by inspection.

#	Issue	Severity	Rows Affected
1	**Duplicate files per partition.** Older files (no `series_id`) and newer files (with `series_id`) coexist. Reading with `hive_partitioning=True` returns ~2× expected rows.	High	1,684,469 duplicates

#	Issue	Severity	Rows Affected
2	**QCEW employment in raw headcount, CES in thousands.** CES national total nonfarm: 130k–160k. QCEW: 126M–314M. Off by ~1,000×.	High	All QCEW rows
3	**8,478 rows with null `vintage_date`.** All CES subnational for `ref_date=2025-09-12`, revision 1. Appear to be "latest available" snapshots where the vintage date was not yet assigned.	Medium	8,478
4	**1,764 QCEW rows with null `industry_code`.** All `industry_type='sector'` at division-level geography — mapping gap during harmonization.	Medium	1,764

#	Issue	Severity	Rows Affected
5	**Retail trade sector code mismatch.** CES uses `42`, QCEW uses `44` (NAICS 44-45). Needs remapping for cross-source joins.	Medium	Sector `42`/`44` rows
6	**CES sector `89` (unclassified) has no QCEW equivalent.** 7,080 rows at division level with ~170 employment.	Low	7,080
7	**CES has 0.0 employment values.** `employment` min is 0.0 for CES, which will produce -inf in log-growth.	Medium	TBD

1.4 Remediation Plan

Before the store can be consumed by the ingest pipeline, the following corrections must be applied (either in-place or as a materialization step):

1. **Remove duplicate files.** Delete the older `befb4d4e…` files from each partition, keeping only the `2bc608b4…` files that include `series_id`. Alternatively, consolidate each partition into a single file.
2. **Normalize QCEW employment to thousands.** Divide all QCEW `employment` values by 1,000 to match CES units. Document the convention in the README.
3. **Fill or drop null `vintage_date` rows.** Either infer the vintage date from the revision number and publication schedule, or drop these rows and re-ingest when actual vintage dates are available.

4. **Fill null `industry_code` rows.** Map the missing QCEW division-level sector codes using the NAICS-to-CES crosswalk, or drop if unmappable.
5. **Harmonize retail trade sector code.** Remap QCEW `44` → `42` to align with CES conventions, or vice versa. Pick one convention and document it.
6. **Filter zero-employment rows.** Drop or flag CES rows with `employment == 0.0` before growth computation.

---

2. Data Ingestion Lifecycle

2.1 Data Sources and Release Schedules

Source	Frequency	Lag	Revisions	Notes
CES National	Monthly	~5 weeks	3 prints + annual benchmark	First Friday of month
CES State	Monthly	~7 weeks	2 prints + annual benchmark	3rd Friday after national
QCEW	Quarterly	~5 months	2–5 vintages (asymmetric by quarter)	Quarterly CSV files
Payroll providers	Monthly	~3 weeks	Not revised	CSV/API delivery

2.2 Ingestion Triggers

```
BLS Release Calendar

           CES Employment Situation (monthly, 1st Friday)
                → fetch CES national + state current
                → identify which ref_dates are new or revised
                → append/update vintage store

           QCEW Quarterly Release (~5 months after quarter end)
                → fetch QCEW quarterly data
                → identify new quarter + revisions to prior quarters
                → append/update vintage store

           Annual Benchmark (February)
```

```
→ fetch benchmarked CES series
→ tag with revision=9, append to vintage store
```

2.3 Update Protocol

When new BLS data becomes available, the pipeline must:

1. **Fetch** the new release from the BLS API (via `bls/` download layer).
2. **Classify** each observation's revision stage using the publication calendar and elapsed time since reference period.
3. **Assign vintage metadata**: `revision` number and `vintage_date` (actual publication date from the calendar, or computed from structural lag).
4. **Normalize** employment units (QCEW ÷ 1,000) and harmonize codes.
5. **Deduplicate** against the existing store — if `(source, ref_date, industry_type, industry_code, seasonally_adjusted, geographic_type, geographic_code, revision)` already exists, skip or warn.
6. **Append** new rows to the appropriate Hive partition.

Write Strategy

Two options for updating the partitioned store:

Option A — Append-only (recommended for simplicity): Write new rows as additional parquet files within existing partitions. Polars `write_parquet` with `use_pyarrow=True` can target a specific partition path. Periodically compact partitions (merge small files into one per partition).

Option B — Rewrite partition: Read the existing partition, concat new rows, deduplicate, write back as a single file. Safer uniqueness guarantee but requires read-modify-write.

Recommended: Option A with periodic compaction

```python
def append_to_vintage_store(
    new_rows: pl.DataFrame,
    store_path: Path = DATA_DIR / "raw" / "vintages" / "series",
) -> None:
    """Append new vintage observations to the Hive-partitioned store.

    Parameters
    ----------
    new_rows : pl.DataFrame
        Must conform to the vintage store schema (§1.2).
```

```python
    store_path : Path
        Root of the Hive-partitioned parquet store.
    """
    for (source, sa), partition in new_rows.group_by(
        ["source", "seasonally_adjusted"]
    ):
        partition_dir = store_path / f"source={source}" / f"seasonally_adjusted={sa}"
        partition_dir.mkdir(parents=True, exist_ok=True)
        # Write with a deterministic filename based on vintage_date range
        vmin = partition["vintage_date"].min()
        vmax = partition["vintage_date"].max()
        fname = f"v_{vmin}_{vmax}.parquet"
        partition.drop(["source", "seasonally_adjusted"]).write_parquet(
            partition_dir / fname
        )
```

2.4 Reading from the Vintage Store

The current ingest pipeline (`ces_national.py`, `qcew.py`) reads from **old flat parquet files** (`ces_vintages.parquet`, `qcew_vintages.parquet`) with source-specific schemas (`CES_VINTAGE_SCHEMA`, `QCEW_VINTAGE_SCHEMA` in `base.py`). These are separate from the new Hive store.

A new reader function is needed to replace both `load_ces_vintages()` and `load_qcew_vintages()`:

```python
def read_vintage_store(
    store_path: Path,
    source: str | None = None,
    seasonally_adjusted: bool | None = None,
    geographic_type: str | None = None,
    industry_type: str | None = None,
    ref_date_range: tuple[date, date] | None = None,
) -> pl.LazyFrame:
    """Read from the Hive-partitioned vintage store with predicate pushdown.

    Partition filters (source, seasonally_adjusted) are pushed to the
    scan level. Additional filters are applied lazily.
    """
    lf = pl.scan_parquet(
        store_path / "**/*.parquet",
        hive_partitioning=True,
    )
```

```python
    if source is not None:
        lf = lf.filter(pl.col("source") == source)
    if seasonally_adjusted is not None:
        lf = lf.filter(pl.col("seasonally_adjusted") == seasonally_adjusted)
    if geographic_type is not None:
        lf = lf.filter(pl.col("geographic_type") == geographic_type)
    if industry_type is not None:
        lf = lf.filter(pl.col("industry_type") == industry_type)
    if ref_date_range is not None:
        lf = lf.filter(
            pl.col("ref_date").is_between(ref_date_range[0], ref_date_range[1])
        )
    return lf
```

---

3. Vintage Store → Observation Panel

3.1 The Two Schemas

The system uses two distinct schemas at different pipeline stages:

```
   VINTAGE STORE SCHEMA                    PANEL_SCHEMA
   (data/raw/vintages/)               (model-ready panel)

 source            Utf8              period            Date
 seasonally_adj.   Bool              industry_code     Utf8
 geographic_type   Utf8              industry_level    Utf8
 geographic_code   Utf8              source            Utf8
 industry_type     Utf8              source_type       Utf8
 industry_code     Utf8              growth            Float64
 ref_date          Date              employment_level  Float64
 revision          UInt8             is_seasonally_adj Bool
 vintage_date      Date              vintage_date      Date
 employment        Float64           revision_number   Int32
 series_id         UInt32            is_final          Bool
                                     publication_lag   Int32
                                     coverage_ratio    Float64
```

Key differences: - **Vintage store** holds employment *levels*; **panel** holds log *growth rates*. - **Vintage store** has full geography; **panel** is typically filtered to one geographic scope before model ingestion. - **Panel** adds derived columns: `growth`, `source_type`, `is_final`, `publication_lag_months`, `coverage_ratio`. - **Panel** splits CES into `ces_sa` / `ces_nsa` source tags; the vintage store uses the `seasonally_adjusted` boolean.

3.2 Transformation: Vintage Store → Panel

The transformation from the Hive vintage store to `PANEL_SCHEMA` involves:

```
read_vintage_store()

       Filter: geographic scope (e.g. national only)
       Filter: industry scope (e.g. supersector + national)
       Filter: drop nulls in vintage_date, industry_code, employment
       Filter: drop zero employment

       Normalize: QCEW employment ÷ 1,000
       Harmonize: sector code remapping (QCEW 44 → 42)

       Derive source tag:
          ces + SA=true  → source='ces_sa',  source_type='official_sa'
          ces + SA=false → source='ces_nsa', source_type='official_nsa'
          qcew           → source='qcew',    source_type='census'

       Compute growth:
          For each (source, industry_code, revision) group,
          sort by ref_date, compute log(emp_t / emp_{t-1})

       Derive metadata:
          revision_number = revision (UInt8 → Int32; map 9 → -1 for benchmark)
          is_final = (revision == max_revision for that source/quarter)
          publication_lag_months = months(vintage_date - ref_date)
          industry_level = industry_type (rename)

       Output: pl.DataFrame conforming to PANEL_SCHEMA
```

3.3 Combining with Provider Data

Payroll provider data lives outside the vintage store (providers are not revised and have no vintage dimension):

```
data/raw/providers/
  pp1/
     alt_nfp_index_1.csv
  pp2/
      alt_nfp_index_2.csv

data/
  alt_nfp_index_1.csv    (legacy location, still used as fallback)
  alt_nfp_index_2.csv
```

Provider ingestion (`ingest_provider()`) reads a CSV, computes log growth, and assigns `source_type='payroll'`, `revision_number=0`, `is_final=True`. Providers are always national total-private (`industry_code='05'`).

The full panel is the vertical concatenation:

```
observation_panel = concat([
    transform_vintage_store(ces_vintages),     # from Hive store
    transform_vintage_store(qcew_vintages),    # from Hive store
    ingest_provider(pp1_config),               # from CSV
    ingest_provider(pp2_config),               # from CSV
])
```

3.4 Vintage Views for Model Consumption

The model does not consume the full panel directly. Instead, `vintages/views.py` provides filtered views:

- **`real_time_view(panel, as_of)`** — For each `(period, source, industry_code)`, keep the highest `revision_number` whose `vintage_date <= as_of`. Simulates the information set available at date `as_of`.
- **`final_view(panel)`** — Keep only rows where `is_final == True`. Used for evaluation against "truth".
- **`specific_vintage_view(panel, source, revision_number)`** — Single revision stage of a single source. Used for studying revision properties.

The `build_noise_multiplier_vector()` function in `vintages/evaluation.py` converts a materialized panel view into a per-observation noise scaling array using the revision schedules from `lookups/revision_schedules.py`.

---

4. End-to-End Pipeline

BLS Release

fetch_ces
(bls/ layer)

fetch_qcew
(bls/ layer)

classify
revision +
normalize

classify
revision +
normalize

append_to_vintage_
store()
(Hive parquet write)

read_vintage_store()
(lazy, filtered scan)

transform to
PANEL_SCHEMA
(growth, metadata)

ingest_provider()
(PP1, PP2, …)

pl.concat()
validate_panel()

observation_panel
(PANEL_SCHEMA)

real_time_
view(as_of)

final_view()

specific_
vintage_view

noise_multiplier_vector()
→ PyMC model

---

5. Implementation Gaps

The following work items bridge the current codebase to this specification.

5.1 Vintage Store Cleanup (prerequisite)

Task	Description	Files
5.1.1	Remove duplicate parquet files per partition (keep `2bc608b4…` files)	`data/raw/vintages/series/`
5.1.2	Normalize QCEW employment to thousands	`data/raw/vintages/series/source=qc`
5.1.3	Fill or drop null `vintage_date` rows (8,478 CES subnational)	`data/raw/vintages/series/source=ce`
5.1.4	Fill null `industry_code` rows (1,764 QCEW division-sector)	`data/raw/vintages/series/source=qc`

Task	Description	Files
5.1.5	Harmonize retail trade sector code (QCEW 44 → 42)	`data/raw/vintages/series/source=qc`
5.1.6	Filter rows with zero employment	All partitions

5.2 New Reader Module

Task	Description	Files
5.2.1	Implement `read_vintage_store()` as a lazy Polars scan with partition pushdown	New: `ingest/vintage_store.py`
5.2.2	Implement `transform_to_panel()` — vintage store → `PANEL_SCHEMA` (growth computation, metadata derivation, unit normalization)	New: `ingest/vintage_store.py`
5.2.3	Deprecate `load_ces_vintages()` and `load_qcew_vintages()` in favor of unified reader	`ingest/ces_national.py`, `ingest/qcew.py`
5.2.4	Deprecate `CES_VINTAGE_SCHEMA` and `QCEW_VINTAGE_SCHEMA` in `base.py` — replace with a single `VINTAGE_STORE_SCHEMA`	`ingest/base.py`

5.3 Update Pipeline

Task	Description	Files
5.3.1	Implement `append_to_vintage_store()` with deduplication check	New: `ingest/vintage_store.py`
5.3.2	Implement `compact_partition()` — merge small files within a partition	New: `ingest/vintage_store.py`
5.3.3	Wire CES fetcher (`fetch_ces_current`) to produce vintage-store-schema rows and call `append_to_vintage_store()`	`ingest/ces_national.py`
5.3.4	Wire QCEW fetcher (`fetch_qcew_current`) to produce vintage-store-schema rows and call `append_to_vintage_store()`	`ingest/qcew.py`

5.4 Panel Builder Refactor

Task	Description	Files
5.4.1	Update `build_panel()` to use `read_vintage_store()` + `transform_to_panel()` instead of separate CES/QCEW loaders	`ingest/panel.py`
5.4.2	Add `geographic_code` to `PANEL_SCHEMA` (currently lost in transformation; needed for state-level modeling)	`ingest/base.py`, `ingest/panel.py`
5.4.3	Update `validate_panel()` to account for new geographic scope	`ingest/base.py`

5.5 Tests

Task	Description	Files
5.5.1	Test `read_vintage_store()` with a synthetic Hive-partitioned fixture	`tests/ingest/test_vintage_store.py`
5.5.2	Test `transform_to_panel()` round-trip (known levels → expected growth)	`tests/ingest/test_vintage_store.py`
5.5.3	Test `append_to_vintage_store()` deduplication semantics	`tests/ingest/test_vintage_store.py`
5.5.4	Test end-to-end `build_panel()` from synthetic Hive store + provider CSVs	`tests/ingest/test_panel.py`

---

6. Open Design Questions

1. **Revision number encoding.** The vintage store uses `UInt8` with `9` for benchmark. The panel uses `Int32` with `-1` for benchmark. Should the store adopt `-1`, or should the mapping live in `transform_to_panel()`?
2. **Geographic scope in the panel.** The current `PANEL_SCHEMA` has no geography columns. If the model expands to state-level, should `geographic_type` and `geographic_code` be added to `PANEL_SCHEMA`, or should separate panels be built per geographic scope?

3. **`series_id` usage.** The newer parquet files include a `UInt32` `series_id` (14,731 unique values). Is this intended as a foreign key to a series catalog, or is it redundant with the composite key `(source, seasonally_adjusted, geographic_type, geographic_code, industry_type, industry_code)`? If redundant, it can be dropped during compaction.
4. **Compaction frequency.** Monthly appends will create ~12 small files per partition per year. Should compaction run after every append, quarterly, or on-demand?
5. **Provider data in the vintage store.** Payroll providers are currently stored as flat CSVs outside the vintage store. If a provider ever introduces revisions, should they move into the Hive store? Or keep the separation since providers are fundamentally different (no BLS provenance, no revision chain)?
6. **Benchmark revision handling.** When the annual CES benchmark drops in February, it revises ~5 years of history. Should the store accept bulk overwrites for `revision=9`, or append new `revision=9` rows alongside prior benchmark vintages?