# Database Design & ETL Automation — Project Documentation

**Students:** Oltean Simion, Ihos Iarina, Cuibus Dorin, Andreican Rares

October 18, 2025

# 1 Project Overview

**Goal.** Build a fully automated pipeline that ingests a CSV dataset into a *staging* area, then normalizes it into a *production* schema using idempotent, scheduled ETL jobs.

**Stack.** PostgreSQL 16, `pg_cron`, Docker.

# 2 Dataset Summary

The dataset (`dataset.csv`) models students and their academic-to-career journey. Fields used by the ETL are:

| | |
|---|---|
| `student_id` | Natural key for a student (string, unique). |
| `age` | Student's age (integer). |
| `gender` | Gender label (string). |
| `high_school_gpa` | HS GPA in [2.00, 4.00]. |
| `sat_score` | SAT score in [900, 1600]. |
| `university_gpa` | University GPA in [2.00, 4.00]. |
| `field_of_study` | Bachelor's major/field. |
| `internships_completed` | 0–4. |
| `projects_completed` | 0–9. |
| `certifications` | 0–5. |
| `soft_skills_score` | 1–10. |
| `networking_score` | 1–10. |
| `job_offers` | 0–5. |
| `starting_salary` | 25,000–1,000,000. |
| `career_satisfaction` | 1–10. |
| `years_to_promotion` | 1–5. |
| `current_job_level` | Entry/Mid/Senior/Executive. |
| `work_life_balance` | 1–10. |
| `entrepreneurship` | Yes/No. |

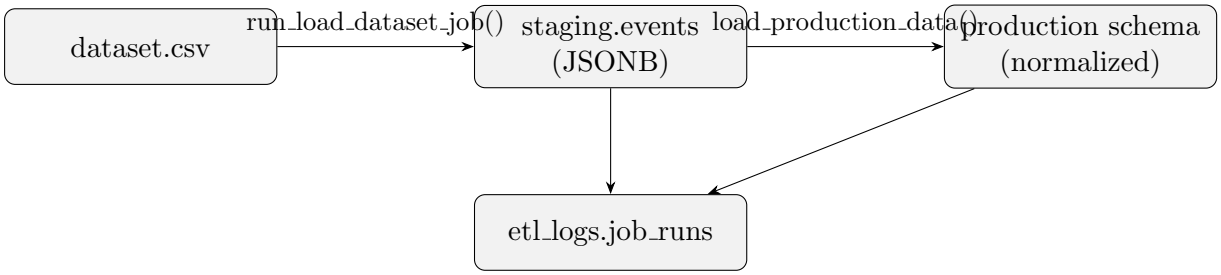
# 3 Architecture & Workflow

### High-level Flow

1. **Ingestion to Staging.** A scheduled job checks if `dataset.csv` changed (using file modification time) and **upserts** JSON rows into `staging.events`.

2. **Transform/Load to Production.** Another scheduled job materializes the normalized model into four production tables with `INSERT ...ON CONFLICT DO UPDATE` (idempotent).

3. **Logging.** All runs are captured in `etl_logs.job_runs` with status: *running*, *completed*, *skipped*, or *failed*.




# 4 Schemas

### Staging

`staging.events`(`stagEventId SERIAL PK`, `content JSONB`, `load_date TIMESTAMP`).
Raw rows are stored as JSONB for flexible ingestion. Upserts compare existing `content` by `student_id` and replace on change.

### ETL Logging

`etl_logs.job_runs`(`logId PK`, `jobname`, `status`, `error_message`, `start_date`, `end_date`, `records_count`).
Helper functions: `return_max_date()`, `update_job_run()`, `update_job_skipped()`, `update_job_failed()` track lifecycle and counts.

### Production (Normalized)

- `production.student`(`student_id PK`, `age`, `gender`)
- `production.academic_performance`(`student_id PK, FK`, GPA/SAT/Field)
- `production.skills_extracurriculars`(`student_id PK, FK`, internships/projects/certifications/soft & networking scores)
- `production.career_outcomes`(`student_id PK, FK`, offers/salary/satisfaction/promotion/job level/WLB/entrepreneurship)

All FKs cascade on delete from `student`. CHECK constraints enforce domain validity.

# 5 Normalization (1NF, 2NF, 3NF)

**1NF.** Atomic columns, no repeating groups. JSON is only in staging; production is scalarized per attribute.
**2NF.** All non-key attributes depend on the whole key. Every table uses the simple key `student_id`, avoiding partial dependencies.
**3NF.** No transitive dependencies: demographics, academics, skills, and career outcomes are factored into separate tables; each non-key attribute describes the table's subject (no attribute depends on another non-key attribute).

# 6 ETL Implementation Details

### Staging Loader (change-aware upsert)

**Function:** `run_load_dataset_job()`
**Logic:** compares `pg_stat_file('.../dataset.csv').modification` with the MAX(`end_date`)

for job "dataset-load". If newer, calls `upsert_dataset()`, which bulk loads into a temp table via `COPY`, then:

- *UPDATE* existing `staging.events` rows where content differs.
- *INSERT* missing `student_ids`.

This is idempotent and minimizes churn.

### Production Loader (idempotent upsert)

**Function:** `load_production_data()` performs four `INSERT ...ON CONFLICT DO UPDATE` statements from `staging.events` into normalized tables, preserving keys and enforcing constraints.

## 7 Automation (Scheduling with pg_cron)

**Extensions/config:** Install `postgresql-16-cron`; set in `postgresql.conf`: `shared_preload_libraries='pg_cron'` and `cron.database_name='test_db'`.
**Jobs:**

- `load-dataset`: every minute — `SELECT run_load_dataset_job();`
- `etl-job`: every two minutes — `SELECT load_production_data();`

These are created via `SELECT cron.schedule(...)` in the SQL scripts.

## 8 Deployment & Reproducibility

### Docker Image

**Dockerfile highlights:**

- Base: `postgres:16`; installs `postgresql-16-cron`.
- Enables `pg_cron` via `shared_preload_libraries` and sets `cron.database_name`.
- Copies `dataset.csv` and all SQL init scripts into `/docker-entrypoint-initdb.d/`.

### Build & Run

1. Build: `docker build -t dbd-pipeline .`
2. Run: `docker run --name dbd-pg -p 5432:5432 -e POSTGRES_PASSWORD=postgres dbd-pipeline`
3. Connect: `psql -h localhost -U postgres -d test_db`

The init scripts create schemas, tables, functions, and schedule `pg_cron` jobs automatically on first boot.

## 9 Logging & Monitoring

`etl_logs.job_runs` captures: job name, status, timestamps, error message, and processed record counts. Helper functions consistently transition statuses from *running* to terminal states. Use:

```
1 SELECT * FROM etl_logs.job_runs ORDER BY logId DESC LIMIT 20;
```

## 10 Analysis: Issues & Improvements (Requirement #7)

### What works well

- **Idempotent ETL:** Upserts in both staging and production prevent duplicates and safely re-run jobs.
- **Change-aware ingestion:** Skips loads when the CSV file is unchanged, reducing unnecessary writes.
- **Normalization:** Clear separation of concerns across four production tables with CHECK constraints.
- **Automation:** Fully scheduled via `pg_cron` at container start.

### Limitations Observed

- **CSV coupling / pathing:** The file is baked into the image at `/app/dataset.csv`; changing data requires rebuilding or mounting a volume.
- **Atomicity:** Staging and production loads are separate jobs; mid-run failures could leave a mismatch for a short window.
- **Validation at staging:** Staging accepts any JSON shape; malformed rows are only caught when casting to production types.

### Improvements (Future Work)

- **Mount dataset as a volume** instead of copying into the image; add checksum-based (SHA256) change detection.
- **Wrap production load in a transaction** and/or use a *swap table* pattern to ensure all-or-nothing refresh.
- **JSON schema validation in staging** (e.g., via `CHECK (content ? 'student_id' && (content->>'age')::INT > 0)` or PL/pgSQL validation) to fail fast.
- **Indexes for performance:** `CREATE UNIQUE INDEX ON staging.events ((content->>'student_id'));` plus indexes on FK columns in production tables.
- **Observability:** Add metrics tables for per-table row deltas; extend `job_runs` with duration milliseconds and more granular error codes.
- **Security:** Create a dedicated role for ETL with least privilege; avoid superuser for runtime jobs.
- **Scheduling cadence:** Switch to daily at 02:00 for production to match a conventional batch window.

## Appendix A: File Map

| | |
|---|---|
| `01_database_setup.sql` | Create DB, schemas (`staging, production, etl_logs`), enable extensions. |
| `02_staging_schema.sql` | Staging table, job log table, staging loader & helper functions, cron job for ingestion. |

| | |
|---|---|
| `03_production_schema.sql` | Production tables with constraints, production loader function, cron job for ETL. |
| `Dockerfile` | PostgreSQL 16 + pg_cron; copies dataset and init scripts; sets preload libraries. |
| `install.sh` | Convenience script to install Docker and Compose on Ubuntu. |

## Compilation Notes

This document uses standard LaTeX packages. Compile with `latexmk -pdf DBD_Documentation.tex`. If you see a "rerun" notice, compile a second time to refresh bookmarks/outlines.