EY Data Integration App — Technical Documentation

1) Purpose & Scope

This document explains how our app identifies tables/fields, creates a unified schema, transforms raw data, merges records, and produces audit-ready outputs. It's written so a new engineer can reproduce results end-to-end and so reviewers can trace every decision the system makes. It's aligned to EY's challenge brief and the documentation criteria (clarity, replicability, scalability).

---

2) System Overview

Inputs

- **Schema files**: Bank1_Schema.xlsx, Bank2_Schema.xlsx
- **Data files**: Bank1_Data.*, Bank2_Data.* (Excel/CSV; multiple tabs like Customers/Accounts/Transactions)
- **Model**: sentence-transformers/all-MiniLM-L6-v2 (for semantic similarity)
- **Config**: CONF_THRESHOLD=73.0 (confidence gate for "Confident Match")

High-level workflow

1. **Upload & classification** (frontend → backend)
2. **Parse schemas → JSON**
3. **AI mapping** (semantic similarity)
4. **Raw data ingestion** (adds bank_origin)
5. **Transform to unified schema** (rename, types, formats)

6. **Merge + conflict resolution**

---

3) Field & Table Identification — How It Works

3.1 Table name mapping (BankB → BankA)

Goal: normalize table names in BankB to BankA's names.

Process implemented in your code:

- Read two schema JSONs (FILE1, FILE2).
- Encode **table names** with Sentence-BERT; compute cosine similarity.
- Convert each similarity row to a confidence rating.
- **Decision rule:** if confidence ≥ CONF_THRESHOLD (73%) ⇒ **Confident Match**; else **Needs Review**.
- Build rename_dict for all confident table pairs and **rename BankB** tables accordingly.
- Persist:
    - bank2_renamed_schema.json (BankB schema with renamed tables)
    - table_name_mapping.json (each BankB table, its best BankA match, similarity, confidence, status)

Why this is transparent: every table pair is recorded with similarity & confidence; reviewers can spot low-confidence links quickly and override.

3.2 File-to-logical table inference (manifest builder)

Goal: connect actual data files (Excel/CSV) to logical tables determined above.

Process (your code):

- Load Table_name_mapping.json and **keep only "Confident Match"** entries.

- Normalize names (lower(), remove spaces/underscores).
- For each file in /BankA and /BankB, infer the **logical table** by checking if the cleaned filename contains the cleaned canonical table token.

Produce merge_manifest.json:

```
{
 "Customers": [
   { "source": "BankA/testing/BankA_Customers.xlsx", "bank": "BankA" },
   { "source": "BankB/testing/BankB_ClientList.xlsx", "bank": "BankB" }
 ],
  "Accounts": [ ... ]
}
```

Why this is transparent: the manifest explicitly lists which raw file contributes to which logical (unified) table.

3.3 Field-level mapping (Stage 3 in workflow)

Beyond table names, the **field identification** uses the same semantic approach at the **column level**:

- Build field strings (name + type + description + sample).
- Encode **BankA fields** and **BankB fields**, compute cosine similarity.
- For each BankA field, choose best BankB field; attach **confidence**.
- Construct a **Unified Schema** with unified_field and mappedFrom pairs (plus confidence).
  (That's how we justify each column rename/type alignment in Stage 5.)

Deliverables for auditors

- Table_name_mapping.json
- Field_name_mapping.json (same structure as table mapping, but for columns)
- Unified_Schema.json (canonical names, types, examples, lineage)

---

4) Transformation & Merge — Reproducible and Deterministic

4.1 Raw data ingestion (Stage 4)

- Load every sheet/CSV → pandas DataFrame.
- Add bank_origin = BankA or BankB.
- Write each to SQLite with unique names like BankA_<file>_<sheet>.
  (Exactly as in your script; logs list rows loaded per table.)

4.2 Transform to Unified Schema (Stage 5)

For each logical table group from merge_manifest.json:

1. **Rename** columns using Field_name_mapping.json (BankA and BankB → unified names).
2. **Standardize types**:
    - to_datetime for dates; define YYYY-MM-DD output.
    - Numeric coercion for amounts; **normalize currencies to CAD** (if applicable).
3. **Normalize formats**:
    - Trim whitespace; normalize casing for codes (e.g., str.upper()).
    - Normalize identifiers (CustomerID, account numbers) to canonical patterns.

The doc recommends a single transform_unified.py that:

- reads the manifest,
- loads per-source tables,
- applies per-column transforms driven by Unified_Schema.json rules,
- writes canonicalized DataFrames (e.g., Unified_Customers_staged).

(Your current script has these steps scaffolded; uncomment/expand the “STEP 4: Auto-merge” portion and swap “shared column intersection” for **explicit rename via field mappings**.)

4.3 Merge & conflict resolution (Stage 6)

- **Append** BankA + BankB unified DataFrames (same columns).
- **Duplicate detection**:
    1. Exact duplicate rows ⇒ keep one.
    2. Same key (CustomerID) with differing values ⇒ **conflict**.
- **Resolution rules** (configurable):
    1. Prefer non-null values.
    2. If still conflict ⇒ BankA overrides BankB (or "most recent by timestamp").
    3. Otherwise **flag** for manual review.
- **Log every conflict** to Conflicts table with full lineage (table, record_id, field, bankA_value, bankB_value, decision).

---

5) Outputs & Artifacts (What reviewers will see)

- **Table_name_mapping.json** — semantic matches, similarity, confidence, status.
- **Field_name_mapping.json** — per-column matches, confidence, status.
- **Unified_Schema.json** — canonical fields: name, type, format, description, source lineage.
- **merge_manifest.json** — raw files grouped by logical table with bank label.
- **merged_banks.db** — SQLite DB with raw-loaded tables + unified tables.
- **manifest.json** — run manifest (timestamp, tables loaded, merge summary, skips, errors).

---

6) Reproducibility — How to Run Locally

Prereqs: Python 3.10+, pandas, sqlite3, sentence-transformers, torch.

1. **Generate schema JSONs** (Stage 2):
   parse_schemas.py → outputs Bank1_Schema.json, Bank2_Schema.json.

2. **Run table mapping & rename** (your first script):
   - Set FILE1, FILE2, CONF_THRESHOLD.
   - Outputs Bank2_Renamed_Schema.json, Table_name_mapping.json.

3. **Generate field mappings** (Stage 3):
   map_fields.py → outputs Field_name_mapping.json, Unified_Schema.json. (Same embedding approach, field-level.)

4. **Build merge manifest** (your second script):
   Runs over /BankA and /BankB directories → outputs merge_manifest.json.

5. **Ingest and merge** (your third script; Stage 4–7):
   - Loads raw files into merged_banks.db.
   - Apply transforms (enable the section to **rename columns using Field_name_mapping.json**).
   - Merge, log conflicts, write unified tables.
   - Writes manifest.json.

6. **Export & report** (Stage 9):
   export_and_report.py → CSV/Excel/JSON exports + PDF report with mappings, confidences, conflicts, KPIs.

---

7) Scaling Guidelines (So it's easy to replicate & scale)

- **Storage**: switch SQLite → DuckDB (single-file analytics at scale) or **Postgres** (multi-user).
- **Embedding speed**: batch encode; cache embeddings per schema version; consider FAISS/Annoy for faster nearest-neighbor.

- **Bigger models**: allow MODEL_NAME override (e.g., all-MiniLM-L12-v2, bge-small-en).
- **Throughput**: parallelize file ingestion and per-table transforms with multiprocessing.
- **Observability**: structured logging (JSON), run IDs, and per-artifact checksums.
- **Config**: externalize thresholds & rules (conf.toml), incl. conflict strategies per field.
- **Data quality rules**: add a declarative ruleset (nullability, regex for IDs, currency/unit normalization).
- **Human-in-the-loop**: a UI to review "Needs Review" mappings and approve overrides (stored in Mappings_Overrides).

---

8) Data Lineage & Auditability

- Every mapping is **explainable**: we emit the similarity score, softmax confidence, and decision status.
- Every record logs its **origin** via bank_origin and can be traced back through merge_manifest.json.
- Every conflict records **inputs and chosen resolution** in Conflicts.
- The **Integration Report** consolidates this for non-technical reviewers.

---

9) Edge Cases & How We Handle Them

- **One-to-many fields** (e.g., BankB has FirstName/LastName, BankA has FullName)
  → rule-based compose/decompose during transform; document mapping notes in Unified_Schema.json.
- **Units/currencies**
  → convert to canonical units (e.g., CAD) with explicit conversion logs.
- **Low-confidence matches**
  → keep status = "Needs Review", never auto-rename; surface in UI/report.

- **Unexpected files/tabs**
  → logged in manifest.json as “skipped/unsupported” with reason.

---

10) Folder & Artifact Layout (suggested)

```
/project
  /BankA            # raw files
  /BankB
  /schemas          # parsed schema JSONs
  /mappings          # Table_name_mapping.json, Field_name_mapping.json, Unified_Schema.json
  /artifacts        # merge_manifest.json, manifest.json, IntegrationReport.pdf
  /db               # merged_banks.db
  /scripts          # parse_schemas.py, map_fields.py, transform_unified.py, export_and_report.py
  /config           # conf.toml (thresholds, conflict rules, formats)
  README.md         # quickstart + runbook
```