Data Engineer Code Challenge: ClinicalTrials.gov → Neo4j Knowledge Graph

Goal

Build a small but well-architected data pipeline that ingests data from **ClinicalTrials.gov**, transforms it into a useful model, and loads it into a **Neo4j** graph database.

The graph must include, at minimum:

- **Clinical-stage drugs** referenced in trials
- **Companies / organizations** associated with the study
- **Route of administration** and **dosage form**

You may include additional entities/fields you judge relevant (e.g., trial phase, status, dates, study type, locations, endpoints, eligibility, etc.) as long as the core requirements are met.

Data source

Use the **ClinicalTrials.gov** downloads → https://aact.ctti-clinicaltrials.org/downloads

---

Scope & assumptions

This challenge is intentionally scoped so it can be completed without building a "perfect" pharma ontology.

- You must pick a reasonable approach to extracting **sponsor/collaborators** and **route/dosage form**.
- If a field is not available directly, you may infer it from textual fields, but clearly document your approach and limitations.
- Your solution should run end-to-end from a clean environment.

Functional requirements

1) Ingestion

- Fetch trials from ClinicalTrials.gov using a reproducible query (you define the query).
- Your query must return a **non-trivial dataset** (suggested: **≥ 500 studies**, but choose what makes sense and document it).

2) Transformation

Extract and normalize at least

- **Drug** name(s) (from interventions)
- **Company/Organization** (sponsor and collaborators)
- **Route of administration** (if present)
- **Dosage form** (if present)

Also capture trial identifiers and metadata needed for traceability:

- NCT ID
- Title (brief or official)
- Phase
- Study status
- Any additional key fields you choose

Handle:

- Missing fields gracefully
- Duplicates and naming inconsistencies (basic normalization is enough; document choices)

3) Graph modeling (Neo4j)

Create a graph model that at minimum supports these nodes and relationships:

Nodes (minimum)

- Trial
- Organization

Relationships (minimum)

- (Trial)-[:SPONSORED_BY]→(Organization) and/or [:COLLABORATES_WITH]
- Route + dosage form must be represented either as:
  - properties of a trial (preferred if you model an intermediate node), or
  - dedicated nodes like Route, DosageForm

Important: Route and dosage form are often trial- or arm-specific. Pick a model that makes sense and document it.

Constraints/Indexes

- Add uniqueness constraints (e.g., Trial.nct_id, and sensible keys for others)
- Add indexes where appropriate

4) Loading

- Load data into Neo4j using an automated script (no manual steps).
- Pipeline should be re-runnable without creating duplicates.

5) Query demonstration

Provide Cypher queries (in a .cypher file or README) that demonstrate the database works. Include at least:

1. **For a given company, list of associated clinical trials**
2. **Top companies by number of trials**
3. **Route and dosage form coverage** (e.g., how many trials have route/dosage form captured, or examples)

Deliverables (what you must submit)

1. **Source code repository** (GitHub link or zipped folder) containing:
   - Ingestion + transform + load pipeline
   - Neo4j schema setup (constraints/indexes)
2. **README.md** with:
   - How to run it end-to-end (commands)
   - Your data query choice and rationale
   - Graph model explanation (with a simple diagram or bullet list)
   - Assumptions/limitations (especially around route/dosage form extraction)
3. **Docker Compose setup** (preferred) that includes:
   - Neo4j container
   - Your pipeline container or run instructions
4. **Example outputs**
   - The Cypher queries listed above + sample results (screenshots or pasted output)

Technical expectations (how we'll evaluate)

Engineering quality

- Clear project structure (e.g., src/, tests/, scripts/, config/)
- Clean, readable code with sensible naming
- Strong error handling and logging
- Idempotent loads (re-running doesn't duplicate data)
- Config-driven design (env vars / config files), not hardcoded values
- Thoughtful use of batching/backpressure for API calls and Neo4j writes

Data modeling

- Pragmatic, consistent entity keys
- Reasonable normalization (e.g., case folding, trimming, dedupe strategy)
- Constraints/indexes used properly
- Clear justification for how route/dosage form are represented

Testing

- Include at least a few unit tests for:
  - Parsing/extraction logic
  - Normalization/deduping logic
- Bonus: integration test that loads a small sample dataset into Neo4j

Documentation

- README is complete enough that we can run it quickly
- Clear explanation of tradeoffs and limitations
- Clear “next steps” section: what you’d do with more time

Bonus points (optional)

- Incremental ingestion (e.g., “load only new/updated trials since last run”)
- Separate “raw → staged → graph” layers (data lake mindset)
- Use of a workflow tool (Airflow/Dagster/Prefect)
- Basic entity resolution improvements (e.g., org name variants)
- Metrics: counts ingested, nodes/edges created, missing-field stats
- Support for exporting a small “analysis view” (CSV/Parquet) alongside Neo4j load

Time guidance

Aim for a solution that demonstrates strong fundamentals and good judgment. We care more about clarity, correctness, and maintainability than about packing in every possible field.

We recommend no more than 6 hours to complete this challenge.