Junior Data Engineer Technical Exercise

Estimated effort: ~2 hours (plus any extra polish you choose to add)
Primary stack: Python 3 + SQLite (via sqlite3, SQLAlchemy, or similar)

---

1. Goal

Demonstrate your ability to ingest data from the public web, model and load it into a relational database, then answer business-style questions with a blend of SQL and Python.

---

2. Your Tasks

Phase	What you do	Notes
A. Pick & profile a data source	Choose **any** open, machine-readable dataset that can be downloaded without logging in (CSV, JSON API, etc.). Examples: NYC 311, OpenWeather, SpaceX launches, public GitHub API, etc.	• At least **10 MB** raw size (enough to be interesting, small enough to keep cloning time low).• Include a short explanation of why you picked it and a link to its documentation.
B. Design a SQLite schema	Create tables and indexes that make analytical queries efficient and keep the data normalized where sensible.	• Supply a .sql or Alembic migration (DDL) plus an ER diagram or a quick schema diagram generated by a tool of your choice.
C. Ingest the data	Write Python code that downloads (or paginates an API), parses, transforms, and inserts the data into SQLite.	• Aim for idempotency (running twice should not duplicate rows).
D. Analysis & insights	Answer **3–5** meaningful questions about the data. At least two questions must be answered **in pure SQL** (run via Python) and at least one with **Python data tooling** (e.g., pandas, matplotlib).	• Show both the raw SQL and a nicely formatted result (table or simple chart).• Include your rationale—why are these questions interesting?
E. Packaging	Make it easy for us to run your work end-to-end.	• Provide a README.md with setup and run instructions.• Include a requirements.txt or pyproject.toml (feel free to use virtualenv or Conda).• Your repository should contain **only code and small config**, not the raw dataset itself; script the download in step C.

---

3. Deliverables

1. **Source code** in a public Git repo (GitHub/GitLab/Bitbucket).

2. schema.sql (or migrations) plus the generated SQLite file **or** a script that builds it from scratch.
3. analysis.ipynb *or* a plain Python script with inline comments that:
    - executes the SQL queries
    - loads results into pandas (if applicable)
    - outputs tables/plots answering your questions
4. A concise README.md covering: prerequisites, setup, how to run ETL, and how to view answers.
5. (Optional, but appreciated) Automated tests or CI workflow.

---

4. Evaluation rubric (what we look for)

Category	Weight	What success looks like
Data ingestion	25 %	Clear, repeatable pipeline; good error handling; no hard-coded paths; sensible use of batching/transactions.
Schema design	20 %	Tables and datatypes reflect the source; appropriate primary/foreign keys; indexes accelerate the analytical queries.
SQL quality	20 %	Queries are correct, readable, and performant (no unnecessary sub-queries, good use of aggregates, etc.).
Python craftsmanship	15 %	Idiomatic code, docstrings, modular structure, optional tests.
Insight & communication	15 %	Chosen questions make sense; results are interpreted clearly; README is newcomer-friendly.
Polish & extras	5 %	CI, type hints, logging, lightweight Dockerfile, or small performance tweaks.

---

5. Submission checklist (for candidates)

- Git repo link with public access
- README.md with run instructions
- Scripts/notebooks produce the SQLite DB and analysis results in one command (make run, python main.py, or similar)
- All code is your own or properly cited (MIT/Apache-licensed tooling is fine)

Tips for candidates

- Keep it simple—depth over breadth.
- Commit early and often; we do look at history.
- If you hit an API rate limit or odd data quality issue, note it in the README.