ALT/FNDATA ACADEMY · PROBLEM SET

Problem set 12: Data engineering and the API

Module 12. Total: 100 points. Format: no-code sandbox discovery in Problem 1, with a substantial API component in Problems 2 through 5 using a shared class API key from your instructor. Problem 5's code component is optional; a written description earns full credit.

Before you start

This problem set moves from the no-code sandbox to the documented shape of the production API. The API exposes a table catalog and a schema endpoint for discovery, and a single query endpoint, POST /v1/tables/{name}/query, shared by every table. That endpoint takes a JSON body with four parts (fields, filters, sort, limit, offset) and returns an envelope of table, result_count, and data, meaning it returns matching rows, not a pre-computed aggregate. Choose one brand or category represented in any production table to use throughout this problem set. AI natural language search is in development and is not part of this problem set.

Problem 1 (15 points): Discovery in the sandbox

Open the sandbox's table list view and its schema tab for one table containing your chosen brand or category. In four to five sentences, state how many production tables the sandbox lists, name the documented fields shown for your chosen table, and explain why a client, whether working in the sandbox or against the production API, should always confirm a table's fields through discovery before writing a query rather than guessing at a field name.

Problem 2 (20 points): Write the query request body

Using only documented fields (designer, model, item_title, sale_date, usd_price_decimal, sale_estimates_high_usd_price, status, vendor, stock_ticker) and a valid filter operator, write the full endpoint, header, and JSON request body you would send to POST /v1/tables/{name}/query to retrieve every sold record for your chosen brand or category, sorted by sale date with the most recent first.

```
POST /v1/tables/<table_name>/query
Header: X-API-Key: <class key>
Body:
{
  ...
}
```

Problem 3 (20 points): Pagination on offset

Assume the query in Problem 2 matches more rows than fit on a single page. Write the second-page request body (showing how offset changes from the first page), and in three to four sentences explain why the API paginates on offset rather than a page number, and how you know, from the response alone, that you have retrieved every matching row rather than stopping partway through the result set.

```
POST /v1/tables/<table_name>/query
Header: X-API-Key: <class key>
Body:
{
  ...
}
```

Problem 4 (25 points): Design a small ETL

The query endpoint returns rows, not a server-side aggregate. Design, in writing, a small ETL that pulls every row for your chosen brand or category and computes one aggregate of your choosing (for example sold-lot count or average realized price) from the retrieved rows. Your design must specify the exact discovery calls you would make first, the exact query body or bodies, how you would handle pagination if the result exceeds one page, and the exact client-side calculation you would perform on the retrieved rows once every page has been collected.

Problem 5 (20 points, optional code component): The pagination loop

Describe, in plain language or in Python using the reusable client (altfndata_client.py), the loop a client would run to retrieve every page of your Problem 2 query and compute your Problem 4 aggregate. If you write code, it must show the loop condition that determines when the last page has been reached. If you describe it in writing instead, state that same condition explicitly.

```
# your loop here, or a written description of equivalent length
```

Submission. Turn in this file with your written answers, JSON bodies, and (optionally) your Python code filled in.