ALT/FNDATA ACADEMY · PROBLEM SET

Problem set 2: Data science and SQL

Module 2. Total: 100 points, plus a 10 point optional bonus. Format: no-code sandbox SQL editor; the bonus problem introduces the API for students who want the code extension.

Before you start

Work in the sandbox SQL editor at sandbox.altfndata.com. Pick one category table to work in for the whole problem set (for example watches, handbags, or jewelry and gems) and confirm the exact table name and column names in the schema tab before you begin. Every query in this problem set should use only these documented fields: designer, model, item_title, sale_date, usd_price_decimal, sale_estimates_high_usd_price, status, vendor, stock_ticker.

Problem 1 (15 points): Filtering with WHERE and LIKE

Write a query that returns item_title, designer, sale_date, and usd_price_decimal for all sold records where the designer field contains a brand name of your choice (use the like operator, which is a case-insensitive substring match). Order the results by usd_price_decimal descending and limit to 20 rows.

```
-- your query here
```

Problem 2 (20 points): Aggregation with GROUP BY

Write a query that counts the number of sold records per vendor for your chosen category, and returns the 10 vendors with the highest count. Then write a second query that computes the average usd_price_decimal per designer, for designers with at least 20 sold records, returning the 10 designers with the highest average.

```
-- query 2a: sold record count per vendor

-- query 2b: average usd_price_decimal per designer, min 20 records
```

Problem 3 (20 points): Pricing power

Pricing power is defined as the median of realized price divided by high estimate, across sold records: median(usd_price_decimal / sale_estimates_high_usd_price). A pricing power above 1.0 means buyers are, in the aggregate, paying above the auction house's own high estimate.

a. Write a SQL query that computes this ratio for every sold, non-null record for one designer of your choice, and returns the median (or, if your sandbox's SQL dialect does not expose a median function, the average, clearly labeled as an approximation). b. In two to three sentences, explain what a pricing power figure above 1.0 versus below 1.0 tells an analyst about that designer's demand at auction.

```
-- your query here
```

Problem 4 (20 points): Sell-through rate

Sell-through rate is the share of offered lots that actually sold: count of status = 'sold' divided by count of all offered lots (sold plus unsold) for a given designer or vendor.

Write a single SQL query that computes sell-through rate for one designer of your choice, returning the designer name, the sold count, the total offered count, and the sell-through rate as a decimal.

```
-- your query here
```

Problem 5 (25 points): Building a demand index (methodology exercise)

This problem asks you to build a simple demand index for one designer by bucketing cleared (sold) prices into calendar quarters using sale_date. This is a methodology exercise, not a market forecast.

a. Write a SQL query that groups sold records for one designer by year and quarter of sale_date and returns the count of sales and the average or median usd_price_decimal per quarter, ordered chronologically. b. Important caveat: the most recent one to two quarters in this dataset are typically under-represented because ingestion of new auction results lags behind the sale date. In three to four sentences, explain why this means a drop in the count or average price in the most recent quarters of your result should not be read as a signal that demand for the brand is falling, and describe one thing you would do before drawing any conclusion from a quarterly series like this.

```
-- your query here
```

Bonus problem (10 points, optional): From SQL to the API

Your instructor has provided a shared class API key for this problem only. Translate the query you wrote in Problem 4 (sell-through rate inputs) into a request body for the production API endpoint POST /v1/tables/{name}/query. You do not need to actually run the request; write the JSON body and the endpoint path.

```
POST /v1/tables/<table_name>/query
Header: X-API-Key: <class key>
Body:
{
  "fields": [...],
  "filters": [...],
  "sort": [...],
  "limit": ...
}
```

Note that the API response envelope is {"table": ..., "result_count": ..., "data": [...]}, where result_count is the number of rows on the current page, not the total number of matching rows, so remember to paginate on offset if you need more than one page.

Submission. Turn in this file with your SQL and written answers filled in.