Day 2 Exercises: Query DSL & ES|QL

Stack: `elk-single` | **Duration:** ~45 minutes total | **Kibana Dev Tools:** `http://localhost:5601`

Prerequisites

- Movies dataset loaded (`python data/load_data.py --dataset movies --size small`)
- Kibana sample data loaded (eCommerce orders)

Part A: Query DSL (7 tasks, ~25 min)

Task 1: match Query (Basic)

Find all movies with `"adventure"` in the `overview` field.

Then modify the query to require BOTH `"adventure"` AND `"space"` in the overview.

> **Hint:** Use the `match` query. For requiring both terms, add the `"operator": "and"` parameter inside the match object.

Task 2: multi_match with Boosting (Basic)

Search for `"dark knight"` across both `title` and `overview` fields. Boost the `title` field by 3x so title matches rank higher.

How many results do you get? What is the top result?

> **Hint:** Use `multi_match` with `"fields": ["title^3", "overview"]`. The `^3` syntax multiplies the score contribution of the title field by 3.

Task 3: match_phrase with Slop (Intermediate)

Find movies whose overview contains the exact phrase `"organized crime"`.

Then try the phrase `"crime family"` with `slop: 3`. Does "The Godfather" appear? Why or why not?

> **Hint:** Use `match_phrase` on the `overview` field. Slop allows terms to be N positions apart. Check The Godfather's overview -- does it contain "crime" and "family" within 3 words of each other?

Task 4: bool Query -- Filters and Scoring (Intermediate)

Write a `bool` query that finds movies matching ALL of these criteria:

1. The `overview` must contain `"love"` (scored -- contributes to relevance)
2. The `genres` must be `"Romance"` OR `"Drama"` (at least one)
3. The `vote_average` must be >= 7.0 (use filter -- not scored)
4. Must NOT be in the `"Horror"` genre

> **Hint:** Structure:
>
> - `must` : `match` on overview for "love"
> - `should` : two `term` queries for "Romance" and "Drama" with `minimum_should_match: 1`
> - `filter` : `range` on vote_average
> - `must_not` : `term` on genres for "Horror"

Task 5: Date Range + Bool Combo (Intermediate)

Find movies released in the **2000s decade** (2000-01-01 to 2009-12-31) that are in the `"Action"` genre and scored above 7.5.

Sort the results by `release_date` in ascending order and return only `title`, `release_date`, and `vote_average`.

> **Hint:** Combine `bool` with `filter` for the range and term queries (since we want date filtering, not scoring). Use `"sort": [{"release_date": "asc"}]` and `"_source": ["title", "release_date", "vote_average"]`.

Task 6: Highlighting (Intermediate)

Search for `"prison escape freedom"` in the `overview` field. Enable highlighting on the `overview` field with custom tags `<mark>` and `</mark>`.

Examine the highlighted fragments -- which terms actually matched?

> **Hint:** Add a `highlight` block with `"pre_tags": ["<mark>"]` and `"post_tags": ["</mark>"]`. Remember that the `english` analyzer stems words, so "prison" may match "imprisoned" and "freedom" may match "free".

Task 7: Complex Search Scenario (Bonus) -- Part 1

Build a search for a movie recommendation engine. The user types: `"epic science fiction"`.

Your query should:

1. Search `title` (boosted 3x) and `overview` using `multi_match` with `type: "best_fields"`
2. Boost results in `"Science Fiction"` genre (should, not required)
3. Filter to movies rated >= 7.0
4. Filter to movies released after 1990-01-01

Task 7: Complex Search Scenario (Bonus) -- Part 2

Your query should also:

5. Exclude `"Animation"` genre
6. Return 5 results with highlighting on `overview`
7. Return only `title`, `genres`, `vote_average`, `release_date`

> **Hint:** Nest the `multi_match` inside `bool.must`. Genre boost goes in `bool.should`. Date and rating go in `bool.filter` (array of two conditions). Animation exclusion in `bool.must_not`. Add `highlight`, `size`, and `_source` at the top level.

Part B: ES|QL (6 tasks, ~20 min)

Part B: Getting Started

All ES|QL queries can be run in Kibana Discover (ES|QL mode) or via Dev Tools:

```
POST /_query
{
  "query": """
    YOUR ESQL QUERY HERE
  """
}
```

Task 8: Basic Filtering (Basic)

Write an ES|QL query to find all movies with `vote_average >= 8.5`, sorted by rating descending. Show only `title`, `vote_average`, and `genres`. Limit to 10 results.

Hint:

```
FROM movies
| WHERE ...
| SORT ...
| LIMIT ...
| KEEP ...
```

Task 9: EVAL -- Computed Columns (Basic)

Write a query that:

1. Extracts the **year** from `release_date`
2. Creates a `rating_tier` column: "Masterpiece" (>= 8.5), "Great" (>= 7.5), "Good" (>= 6.0), "Okay" (< 6.0)
3. Shows `title`, `year`, `vote_average`, `rating_tier`
4. Sorts by `vote_average` DESC, limit 15

> **Hint:** Use `EVAL year = DATE_EXTRACT("year", release_date)` and `EVAL rating_tier = CASE(vote_average >= 8.5, "Masterpiece", vote_average >= 7.5, "Great", vote_average >= 6.0, "Good", "Okay")`.

Task 10: STATS...BY -- Genre Analysis (Intermediate)

Analyze movies by genre:

1. Expand the `genres` multi-value field
2. Group by genre
3. Calculate: count, average rating, max rating
4. Sort by count descending

Which genre has the most movies? Which has the highest average rating?

> **Hint:** Use `MV_EXPAND genres` before `STATS`. Remember that `MV_EXPAND` unnests the array so each movie appears once per genre.

Task 11: Decade Analysis (Intermediate)

Create a decade-based analysis:

1. Extract the year from `release_date`
2. Calculate the decade (1990, 2000, 2010, 2020...)
3. Group by decade
4. Show count, average rating, and min/max rating per decade
5. Sort by decade

Which decade produced the highest-rated movies on average?

Task 11: Decade Analysis (continued)

Hint:

```
EVAL year = DATE_EXTRACT("year", release_date)
EVAL decade = FLOOR(year / 10) * 10
```

Then `STATS ... BY decade`.

Task 12: eCommerce Data Exploration (Intermediate)

Switch to the `kibana_sample_data_ecommerce` index.

Write an ES|QL query that answers: **What is the average order value by day of the week?**

1. Extract the day of week from `order_date`
2. Group by day name
3. Calculate average and total `taxful_total_price`
4. Sort by average descending

> **Hint:** Use `DATE_EXTRACT("DAY_OF_WEEK", order_date)` to get day number (1=Monday, 7=Sunday). Use `CASE` to convert to names, or just use the number. The price field is `taxful_total_price`.

Task 13: ES|QL vs Query DSL Comparison (Bonus)

Write the **same query** in both ES|QL and Query DSL:

"Find the top 5 Drama movies released after 2000 with the highest vote_average"

Compare:

- Which is more readable?
- Which gives you relevance scoring?
- Which would you use for a search box? For a dashboard?

> **Hint:** ES|QL version uses `WHERE`, `SORT`, `LIMIT`. Query DSL version uses `bool.filter` with `term` + `range`, and `sort`. Note that ES|QL returns flat columnar data while Query DSL returns nested JSON with `_score`.

Cleanup

No cleanup needed -- keep the data for Day 3!