Exam : **OSCA**

Title : OpenSearch Certified Associate (OSCA)

Version : Demo

1.An online store is creating a new products index. Users must be able to run full-text searches on product_name. The same field must also support case-sensitive exact-value filtering, sorting, and terms aggregations without enabling field data on an analyzed field.
Which mapping best meets these requirements? Select the best answer.
A. {
  "product_name": {
    "type": "keyword"
  }
}
B. {
  "product_name": {
    "type": "text",
    "fielddata": true
  }
}
C. {
  "product_name": {
    "type": "text",
    "fields": {
      "raw": {
        "type": "keyword"
      }
    }
  }
}
D. {
  "product_name": {
    "properties": {
      "full_text": { "type": "text" },
      "exact": { "type": "keyword" }
    }
  }
}
Answer: C
Explanation:
A text field supports analyzed full-text search, while a keyword multi-field stores the same input as an unanalyzed value for exact matching, sorting, and aggregations. A keyword-only field does not provide analyzed full-text behavior, and enabling field data on a text field does not preserve each original product name as a single exact value.

2.A news search application has these mappings:
- body is a text field.
- status is a keyword field.
- published_at is a date field.

The application must rank articles by how well body matches distributed tracing. It must return only articles whose status is published and whose published_at value is within the last 30 days. The status and date restrictions must not affect _score.
Which query best meets these requirements? Select the best answer.
A. {
  "query": {
    "bool": {
      "must": [
        { "match": { "body": "distributed tracing" } },
        { "term": { "status": "published" } },
        { "range": { "published_at": { "gte": "now-30d", "lte": "now" } } }
      ]
    }
  }
}
B. {
  "query": {
    "bool": {
      "must": [
        { "match": { "body": "distributed tracing" } }
      ],
      "filter": [
        { "term": { "status": "published" } },
        { "range": { "published_at": { "gte": "now-30d", "lte": "now" } } }
      ]
    }
  }
}
C. {
  "query": {
    "bool": {
      "filter": [
        { "match": { "body": "distributed tracing" } }
      ],
      "must": [
        { "term": { "status": "published" } },
        { "range": { "published_at": { "gte": "now-30d", "lte": "now" } } }
      ]
    }
  }
}
D. {
  "query": {
    "bool": {

```
        "should": [
          { "match": { "body": "distributed tracing" } },
          { "term": { "status": "published" } },
          { "range": { "published_at": { "gte": "now-30d", "lte": "now" } } }
        ],
        "minimum_should_match": 1
      }
    }
}
```

Answer: B

Explanation:

The full-text match clause belongs in query context so that it contributes to _score. The exact status and bounded date restrictions belong in filter context, where they restrict matching documents without contributing to relevance scoring.

3.Applications search an index through the orders alias. The alias currently points to orders-v1, where customer_id is mapped as text. The team must change customer_id to keyword. Existing documents have _source enabled, writes can be paused during the migration, and search downtime must be minimized.

Which sequence best meets these requirements? Select the best answer.

A. Update the mapping of customer_id in orders-v1 from text to keyword, call the Refresh API, and keep the alias unchanged.

B. Close orders-v1, change the field type to keyword, reopen the index, and then force-merge its segments.

C. Create orders-v2 with the desired mapping. Pause writes, refresh orders-v1, and reindex into orders-v2; refresh and validate the destination, atomically switch orders to it with one Manage Aliases API request, and resume writes through the alias.

D. Create a snapshot of orders-v1, restore it under the name orders-v2 while supplying the new field mapping, and then delete the orders alias.

Answer: C

Explanation:

OpenSearch cannot change an existing field from text to keyword. Create the destination with the desired mapping, pause writes, and refresh the source before reindexing so its search snapshot includes all completed writes. After reindexing finishes, refresh and validate the destination, switch the alias with one atomic remove/add request, and resume writes through the alias. This keeps searches available on the original index until the switch.

4.An administrator receives this cluster health response:

```
{
  "status": "yellow",
  "active_primary_shards": 4,
  "active_shards": 4,
  "unassigned_shards": 1
}
```

Which statement best interprets the response? Select the best answer.
A. At least one primary shard is unassigned, so some indexed data is unavailable.
B. All primary shards are allocated, but at least one replica shard is unassigned, so the cluster is operational without full redundancy.
C. All primary and replica shards are allocated; yellow only indicates that cluster-manager tasks are pending.
D. The cluster-manager node has not been discovered, but shard allocation is otherwise complete.
Answer: B
Explanation:
A yellow cluster has all primary shards allocated but at least one replica shard unassigned, so searches can operate while redundancy is incomplete. An unassigned primary would make health red, while fully allocated primaries and replicas would make it green.

5.A team has instrumented its services to send traces to an OpenTelemetry Collector. Before the traces are stored in OpenSearch, the team needs a server-side component that can process and enrich span records, prepare trace-group and service-map data, and send the transformed results to OpenSearch for Trace Analytics.
Which design best meets these requirements? Select the best answer.
A. Send spans directly to OpenSearch Dashboards and use Dashboards to transform and index the trace data.
B. Apply an Index State Management policy to the incoming spans so it can build trace groups and service-map metadata before indexing.
C. Deploy Data Prepper between the OpenTelemetry Collector and OpenSearch, using trace-processing pipelines and an OpenSearch sink.
D. Send spans directly from the OpenTelemetry Collector to OpenSearch and rely on an index template to perform stateful trace processing and service-map calculations.
Answer: C
Explanation:
Data Prepper is the server-side ingestion component that receives OpenTelemetry trace data, processes and enriches spans for trace groups and service maps, and writes the results to OpenSearch. OpenSearch Dashboards visualizes the indexed data, while index templates and ISM policies do not perform the required stateful trace processing.