elasticsearch

By:- MAKOUNDOU Paule-Theodora
DE TROGOFF Guilhem
SHARMA Karan
LEDROLE Pierre

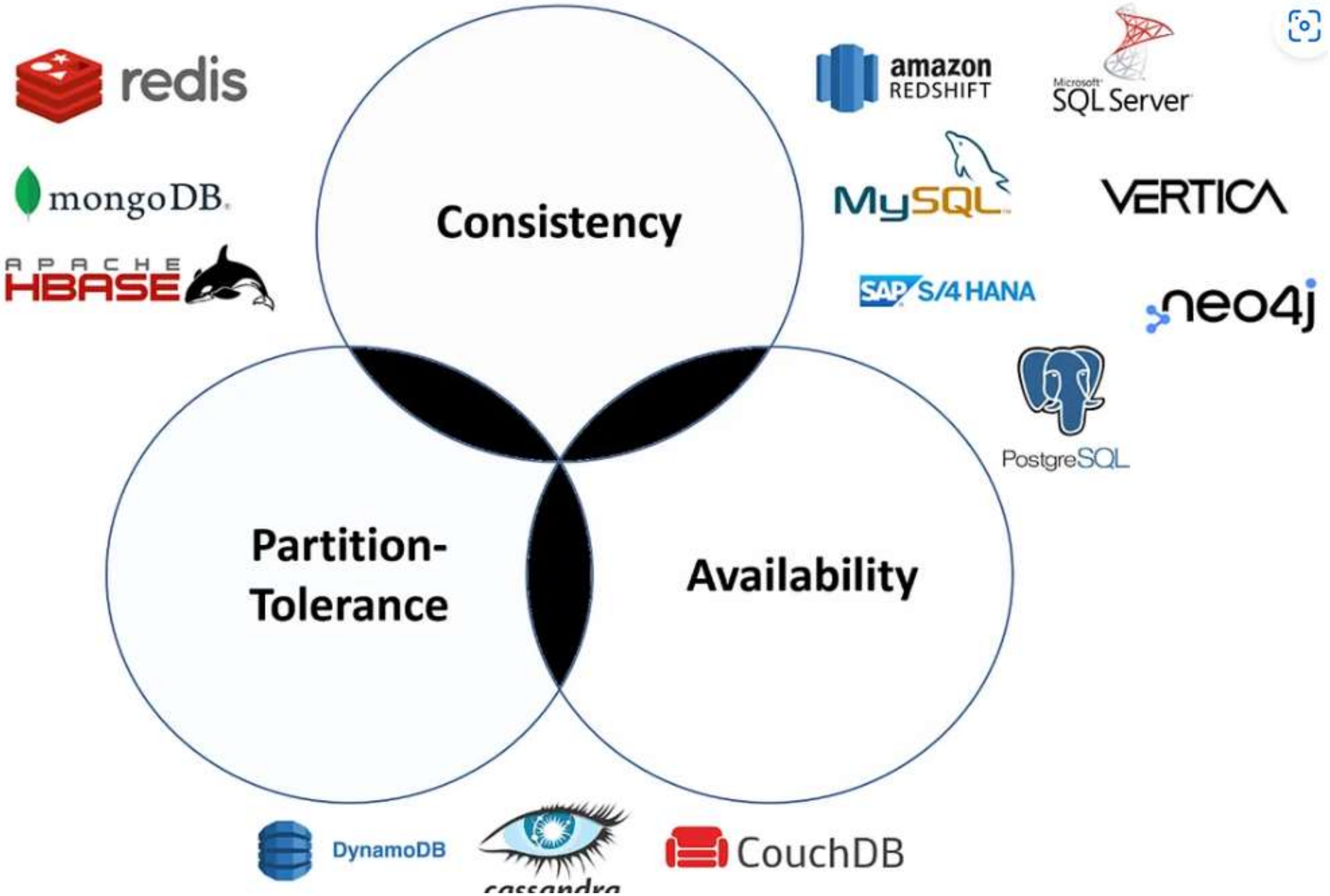
What is ElasticSearch ?

- Elasticsearch is a BASE (Basically Available, Soft State, Eventually Consistent) system.
- Prioritizes availability and scalability over strict consistency.
- Stores and indexes JSON documents for fast search and retrieval

Examples of Data Stored in Elasticsearch

- **Logs and Event Data:** System and application logs.
- **Full-Text Documents:** Articles, product descriptions.
- **Structured Data:** User information, product details.
- **Geospatial Data:** Geographic coordinates, maps.
- **Time-Series Data:** Metrics, financial data.
- **Social Media Data:** Posts, user interactions.
- **Machine Learning Data:** Training sets, predictions.

CAP theorem positioning



- Consistency: Ensures data across nodes is synchronized.
- Partition Tolerance: Continues operating despite network issues.
- Availability: Sacrifices some availability during network partitions to maintain consistency.

Examples of Uses

- Uses: Used by companies like Netflix and GitHub. Powers search engines for platforms like Wikipedia and Shopify.
- Issues: High resource usage (RAM, CPU), complex cluster management, risk of data loss from poor configuration.
- Cost: Free for the open-source version; paid options offer advanced features. Managed Elastic Cloud services simplify setup but increase costs

How Popular is this Tool?

- ElasticSearch is widely popular, especially in IT, e-commerce, and media.
- Major companies like Netflix, Uber, and Shopify rely on it
- Strong community support, frequent updates, and adaptability for real-time, large-scale search needs.

splunk>

amazon

Apache Solr

Main Competitors

Competitors: Apache Solr (open-source), Splunk (commercial), Amazon CloudSearch (managed), and Lucenefor custom search.

Timeline: Released in 2010, evolved with ELK integration, cloud offerings, and machine learning features.

Future and Market Changes

ElasticSearch is expected to expand:

in machine learning

in cloud services.

Benefits and limitations

- Benefits: Highly scalable, real-time search and analytics, flexible full-text search capabilities, open-source with strong integration options.

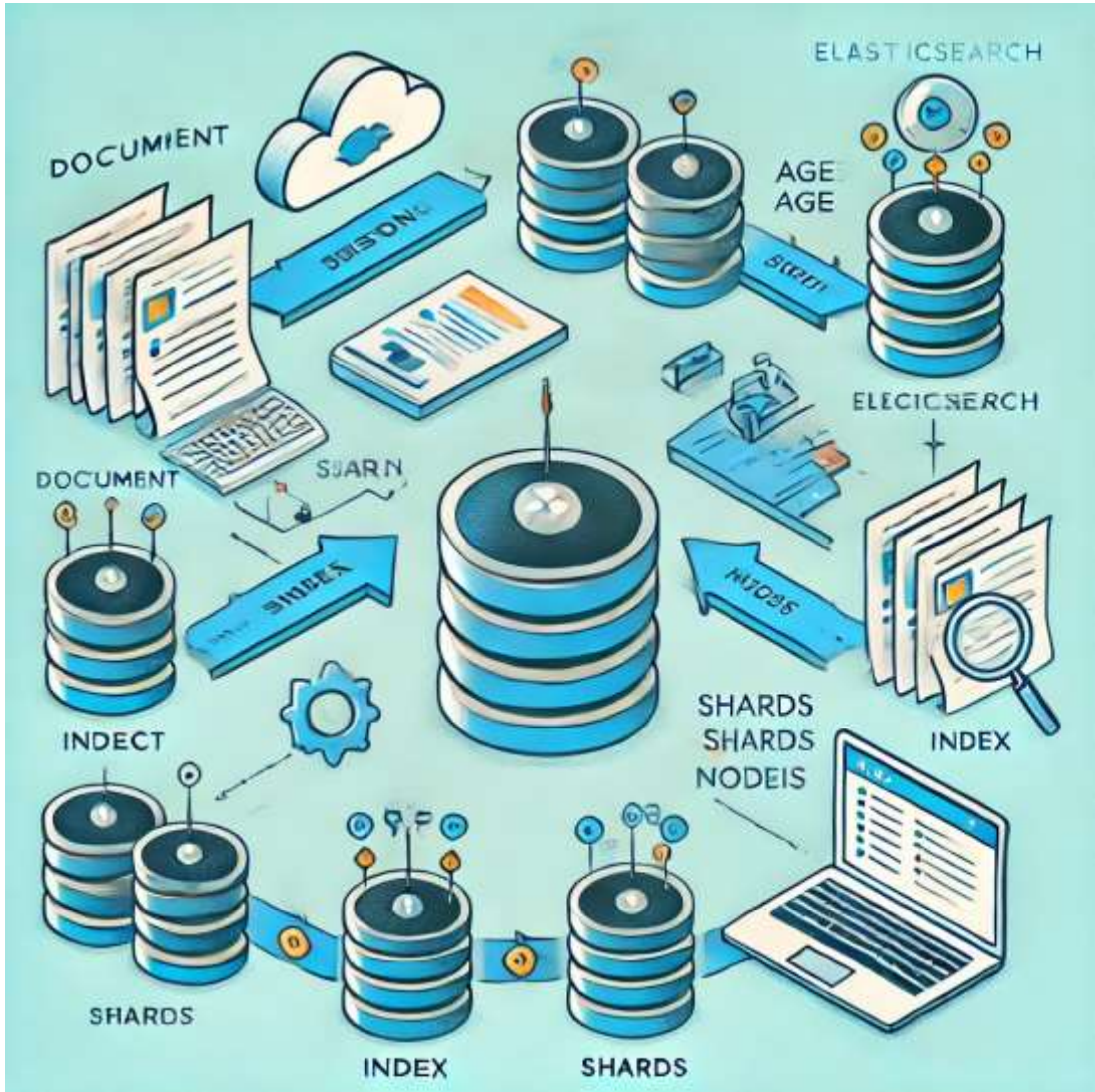
- Limitations: No ACID compliance (BASE model), complex cluster management, high resource demands increase operational costs.

Indexing in Elasticsearch and its code implementation

- Indexing in Elasticsearch is essentially the process of storing and organizing data so that it can be efficiently searched and retrieved. When we index a document, we add it to an **index** (like a table in SQL) where it becomes searchable.
- **Index**: A collection of documents (like a table in SQL).
- **Document**: The basic unit of information (like a row in SQL) containing various fields (key-value pairs).
- **Field**: Each key-value pair in a document (like columns in SQL).
- **Shards**: Indexes are split into smaller units called shards, which allow Elasticsearch to distribute data across multiple nodes for scalability and performance.

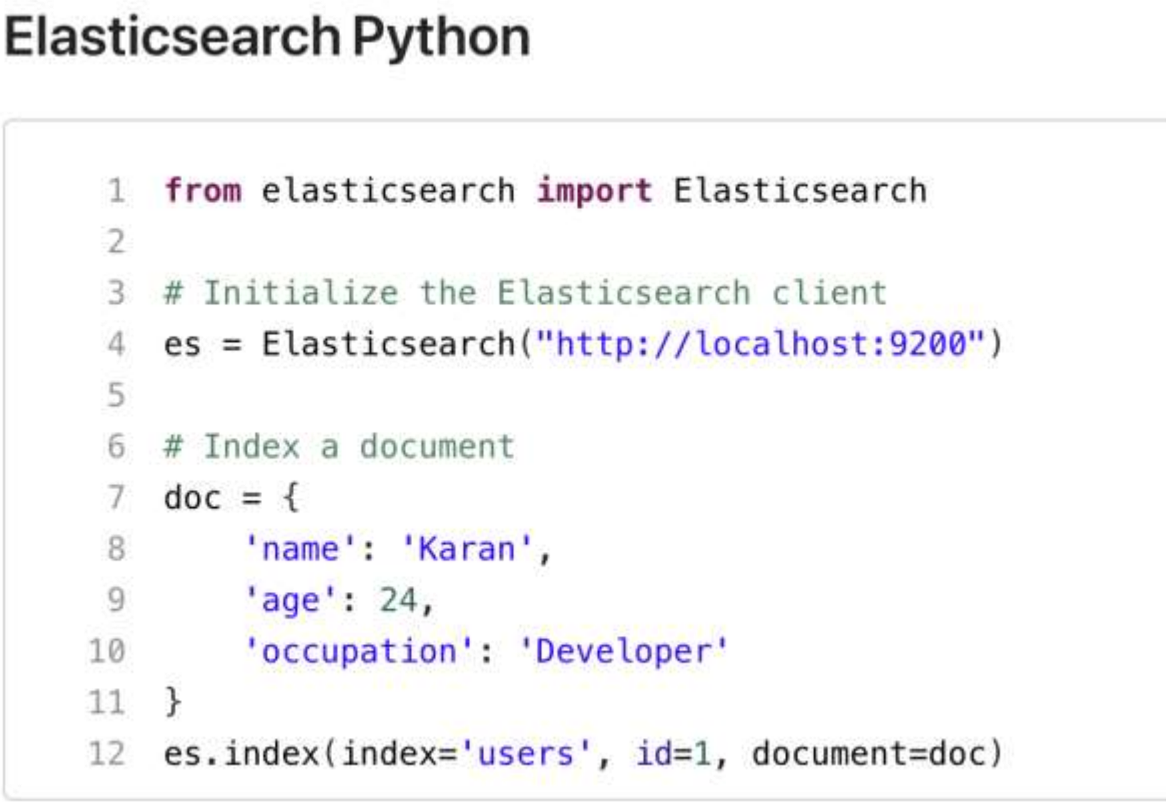
The Indexing Process:

1. **Parsing:** Elasticsearch breaks down a document into its individual fields during indexing.
2. **Mapping:** Elasticsearch automatically maps field types (or you can define your own mappings), telling Elasticsearch how to store and search the data.
3. **Inverted Index:** For text fields, Elasticsearch creates an inverted index that allows for fast full-text search by



Code Example (Python):

Indexing a document into Elasticsearch:



Elasticsearch Python

```python
from elasticsearch import Elasticsearch

# Initialize the Elasticsearch client
es = Elasticsearch("http://localhost:9200")

# Index a document
doc = {
    'name': 'Karan',
    'age': 24,
    'occupation': 'Developer'
}
es.index(index='users', id=1, document=doc)
```

Similarities with SQL and Code Comparison

Similarities with SQL:

- **Indexes vs. Tables:** In Elasticsearch, **indexes** are similar to **tables** in SQL. Both are used to store collections of documents (rows).
- **Documents vs. Rows:** In Elasticsearch, a **document** is like a **row** in SQL.
- **Fields vs. Columns:** Elasticsearch's **fields** correspond to **columns** in SQL.
- **Queries:** Both Elasticsearch and SQL allow filtering and searching based on conditions.
- **Aggregations:** Elasticsearch **aggregations** are similar to **SQL aggregate functions** (e.g., COUNT, SUM, AVG).

Key Differences:

Full-Text Search: Elasticsearch excels at full-text search, while SQL is not optimized for it.

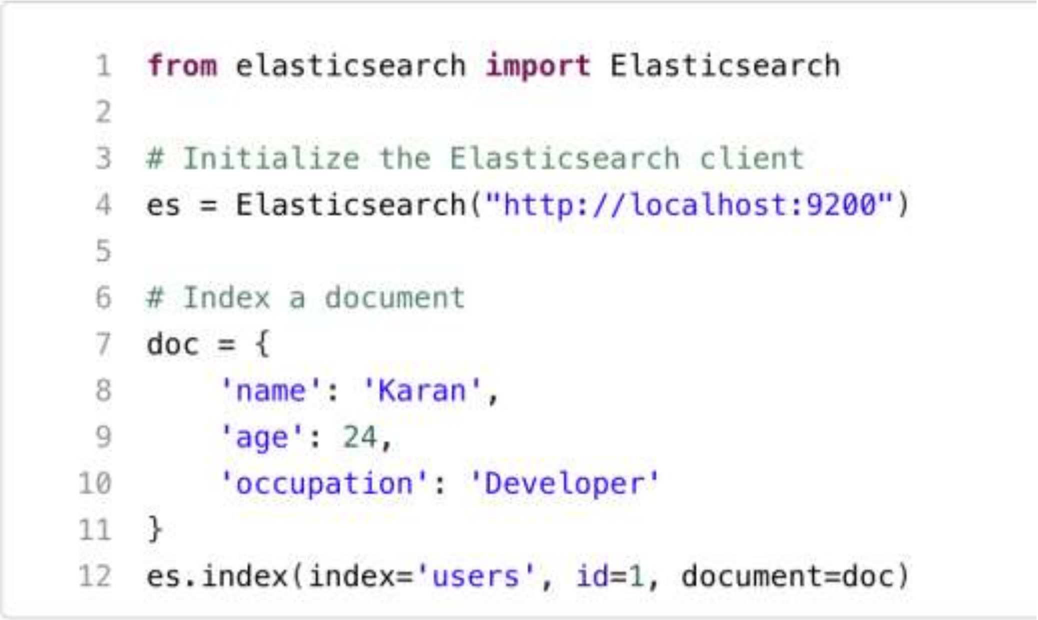
Schema: Elasticsearch is schema-less, whereas SQL databases require a predefined schema.

Joins: SQL supports complex joins between tables, while Elasticsearch is document-based and does not support traditional joins.

Code Comparison (Examples):

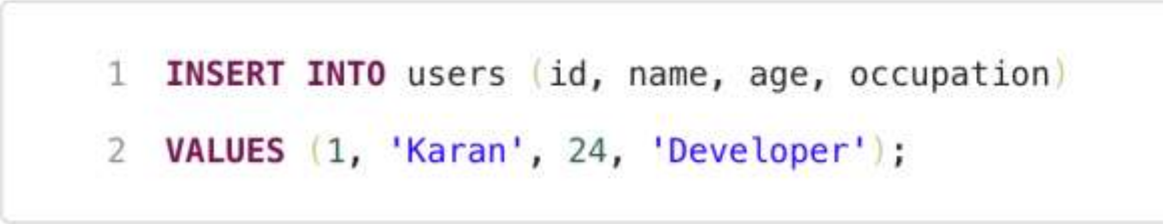
Insert a row:

Elasticsearch Python



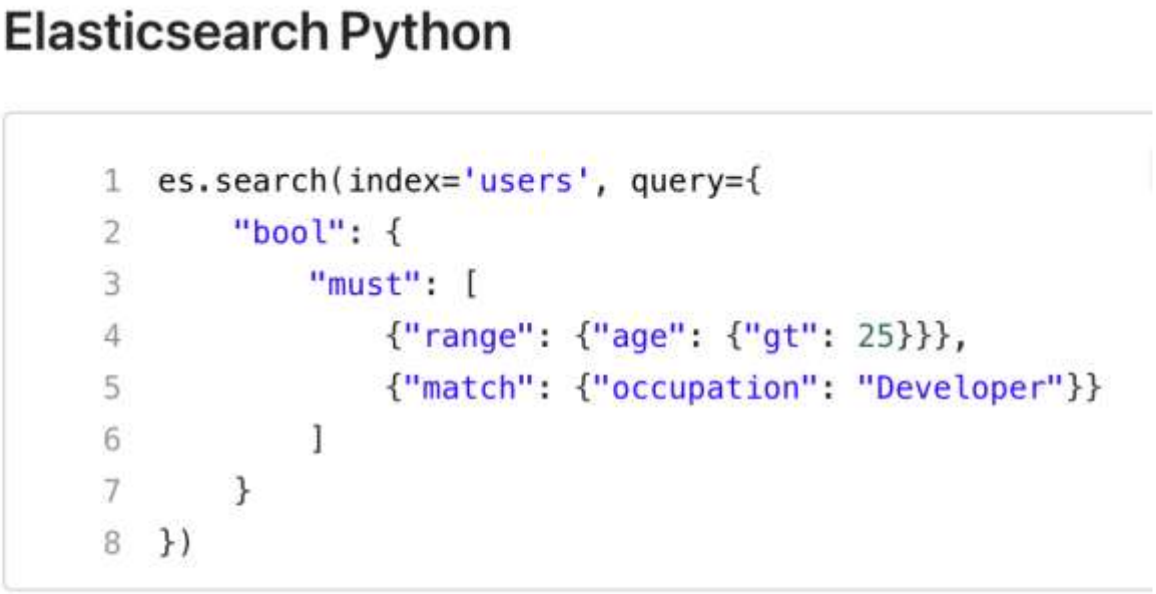
```python
from elasticsearch import Elasticsearch

# Initialize the Elasticsearch client
es = Elasticsearch("http://localhost:9200")

# Index a document
doc = {
    'name': 'Karan',
    'age': 24,
    'occupation': 'Developer'
}
es.index(index='users', id=1, document=doc)
```

SQL



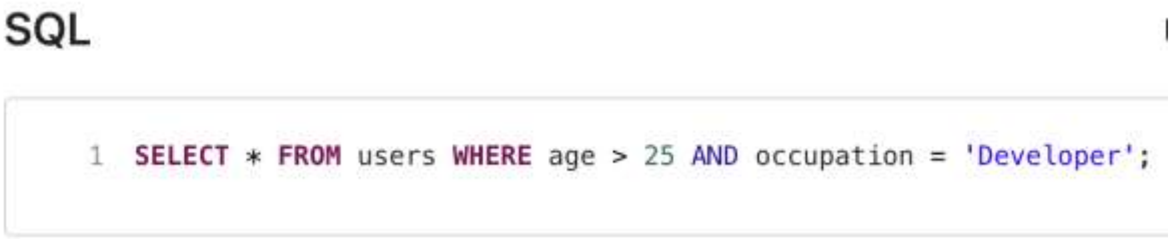
```sql
INSERT INTO users (id, name, age, occupation)
VALUES (1, 'Karan', 24, 'Developer');
```

Query with conditions:



Elasticsearch Python

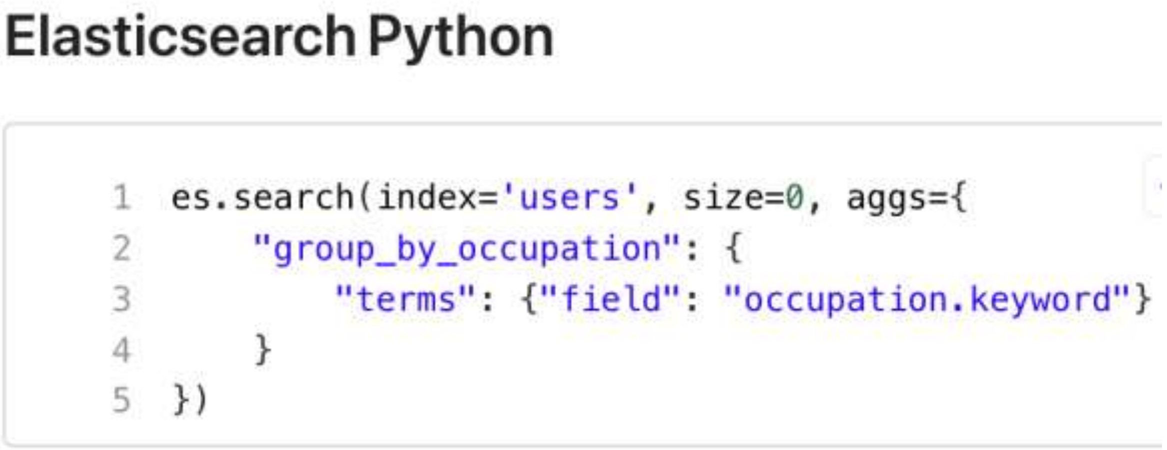
```python
es.search(index='users', query={
    "bool": {
        "must": [
            {"range": {"age": {"gt": 25}}},
            {"match": {"occupation": "Developer"}}
        ]
    }
})
```



SQL

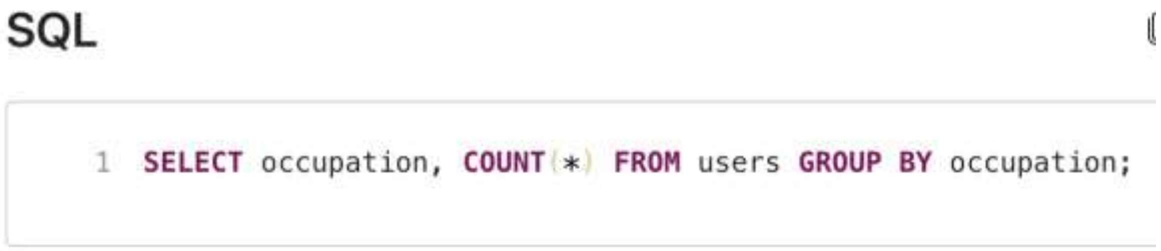
```sql
SELECT * FROM users WHERE age > 25 AND occupation = 'Developer';
```

Aggregate (COUNT) and group by:



Elasticsearch Python

```python
es.search(index='users', size=0, aggs={
    "group_by_occupation": {
        "terms": {"field": "occupation.keyword"}
    }
})
```



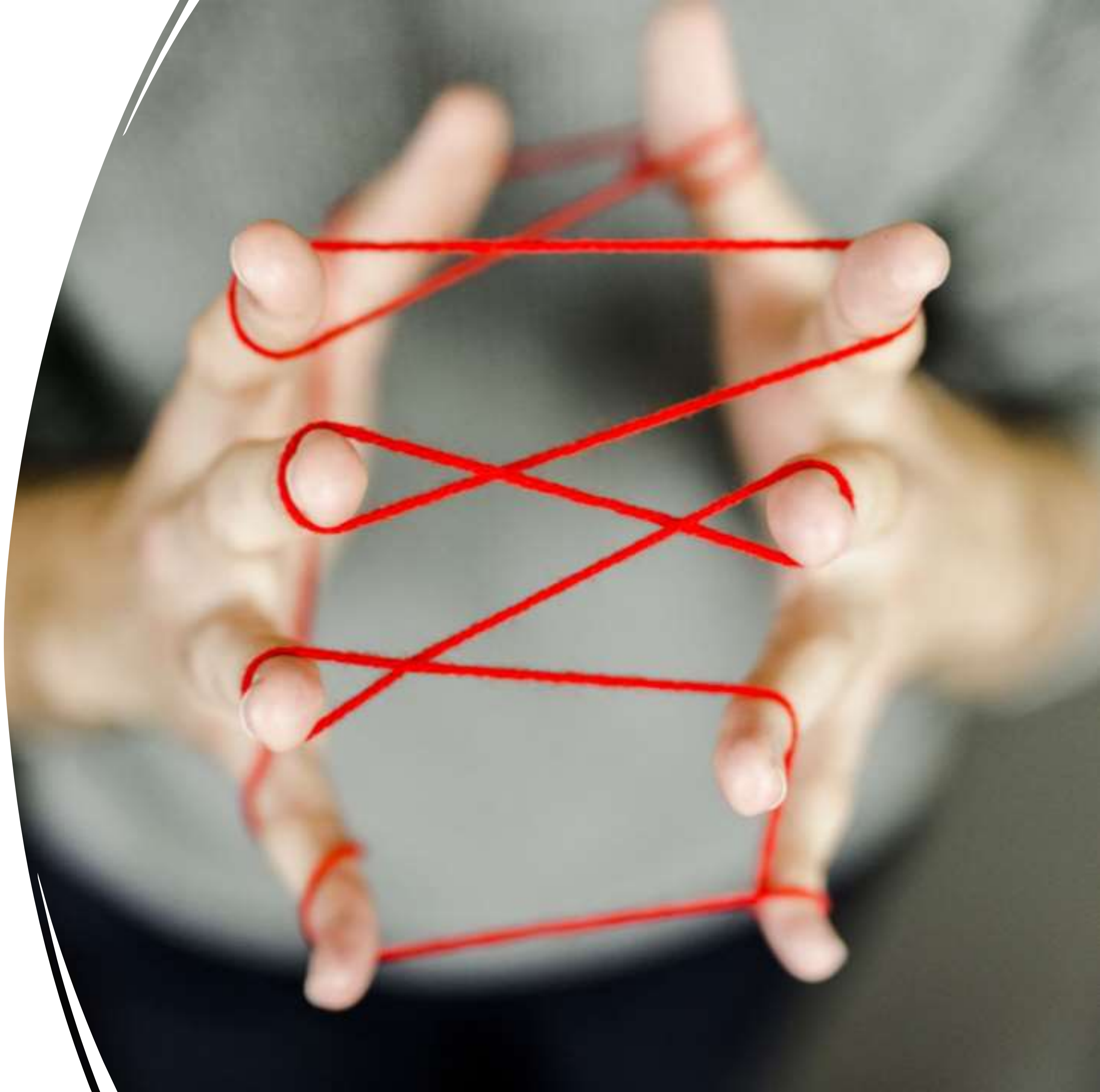
SQL

```sql
SELECT occupation, COUNT(*) FROM users GROUP BY occupation;
```

Scalability:

Elasticsearch is designed to be highly scalable. As your data grows, Elasticsearch can scale horizontally by adding more nodes to your cluster. Key features like **sharding** and **replication** make it resilient and able to handle large volumes of data efficiently.

- **Sharding**: Each index is divided into shards, which are distributed across multiple nodes.
- **Replication**: Shards are replicated to provide high availability. If a node goes down, Elasticsearch will use replica shards to ensure data availability.



Flexibility

One of Elasticsearch's greatest strengths is its flexibility. It handles structured, semi-structured, and unstructured data seamlessly. You can store:

- **Text** and perform full-text search.
- **Numbers**, **dates**, and **booleans** for structured queries.
- **Arrays** and **nested objects** for more complex data models.
- **Geospatial data** for location-based searches.
- Elasticsearch is also **schema-less** by default, meaning you don’t need to define a rigid structure upfront. You can index and search a wide variety of data types without setting up predefined mappings (though you can define mappings if needed).

Usability

RESTful API: You can interact with Elasticsearch using simple HTTP requests (e.g., GET, POST), making it easy to use and integrate with different tools.

Kibana: Kibana, part of the Elastic Stack, allows you to visualize and explore your data using a graphical interface. This makes Elasticsearch accessible to technical and non-technical users alike.

Real-Time Data Ingestion: Elasticsearch indexes data in near real-time, meaning new data becomes searchable almost immediately after it's ingested.

Q
A