

Noise: A new search index

Volker Mische
@vmx

Search Meetup Munich 2018-04-23, Munich, Germany

About me

- Volker Mische (@vmx)
- Geo + databases
 - Apache CouchDB (GeoCouch)
 - Couchbase
 - RocksDB
- Rust, Python, JavaScript, Erlang
- "Open Sourcing Coven Leader"



Noise



What is Noise?

- JSON
- Embedded search index/engine
- Full-text, ranges, geo
- Query by example
- <https://noisearch.org/>



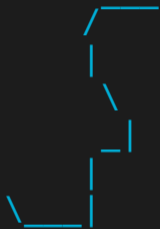
Demo



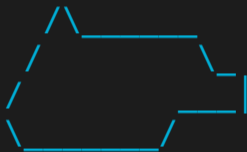
What's so special about geo data?

- Multi-dimensional
=> No total order
- Many tricks to work around the multi-dimensionality
- Data vs. space partitioning algorithms and data structures
- R-trees can be used for any numeric ranges

Bounding box



Line



Polygon

Bounding box



Line



Polygon

Usage

- Insert any JSON
- Automatic indexing:
 - Strings -> Full-text search
 - Numbers -> Ranges
 - GeoJSON -> Bounding boxes
- Query: JSON + operator + value

Technologies

- JavaScript API, Python prototype (more to come)
- Coded in Rust: Fast, safe, better C/C++
- RocksDB: Key-value store from Facebook

How is it different?

- Traditional FTS: flatten data with mappings
- Document databases with SQL: flatten data when queried
- Noise: preserve structure



How does Noise work?



How does the indexing work?

- Shredder: JSON -> Key-value:

```
{"info": {"event": "Heat"}}
```



```
info.event!Heat#123
```

- Indexes are sorted by Internal ID (even the R-tree)

How does the indexing work?

- Shredder: JSON -> Key-value:

```
{"info": {"event": "Heat"}}
```



```
info.event!Heat#123
```

- Indexes are sorted by Internal ID (even the R-tree)

How does the indexing work?

- Shredder: JSON -> Key-value:

```
{"info": {"event": "Heat"}}
```



```
info.event!Heat#123
```

- Indexes are sorted by Internal ID (even the R-tree)

How does the indexing work?

- Shredder: JSON -> Key-value:

```
{"info": {"event": "Heat"}}
```



```
info.event!Heat#123
```

- Indexes are sorted by **Internal ID**
(even the R-tree)

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244  
...  
info.description!risk#244  
...  
info.parameter.thunderstorm!möglich#720
```

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244  
...  
info.description!risk#244  
...  
info.parameter.thunderstorm!möglich#720
```

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244  
...  
info.description!risk#244  
...  
info.parameter.thunderstorm!möglich#720
```

Same word,
different
documents

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244  
...  
info.description!risk#244  
...  
info.parameter.thunderstorm!möglich#720
```

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244           multiple  
...                                   words, same  
info.description!risk#244           document  
...  
info.parameter.thunderstorm!möglich#720
```

How does the indexing work?

```
...  
info.category!Met#1218  
...  
info.category!Met#1560  
...  
info.description!high#244  
...  
info.description!risk#244  
...  
info.parameter.thunderstorm!möglich#720
```

How does the indexing work?

...
info.category!Met#1218

...
info.category!Met#1560

...
info.description!high#244

...
info.description!risk#244

**deeply
nested**

...
info.parameter.thunderstorm!möglich#720

How does the querying work?

- Similar to traditional DBs (Volcano Iterator Model)
- Parse query
- Build up Query Engine
- Iterate through conditions

Example Query

```
find {
  info: {
    description: ~= "high",
    area: {
      geometry: && [10.9, 48.2, 11.0, 48.4]
    }
  }
}
```

```
find {
  info: {
    description: ~= "high",
    area: {
      geometry: && [10.9, 48.2, 11.0, 48.4]
    }
  }
}
```

Example Query

```
find {  
  info: {  
    description: ~= "high"  
  }  
}
```

Shred based on operator

Example Query

```
find {  
  info: {  
    description: ~="high"  }  
}
```

**Shred based on
operator**

=> info.description!high

```
}
```

```
find {
  info: {
    description: ~= "high",
    area: {
      geometry: && [10.9, 48.2, 11.0, 48.4]
    }
  }
}
```

```
find {
  info: {
    description: ~= "high",
    area: {
      geometry: && [...]
    }
  }
}
```

Example Query

`info.description!high`

`geometry: && [...]`

Example Query

`info.description!high`

`geometry: && [...]`

Example Query

`info.description!high`

`geometry: && [...]`

```
...
info.description!few#333
info.description!high#244
info.description!high#333
info.description!high#720
info.description!humidity#244
...
```

**Find first
match**

Example Query

info.description!high

geometry: && [...]

...

info.description!few#333

▶ info.description!high#244

info.description!high#333

info.description!high#720

info.description!humidity#244

...

**Found first
match**

Example Query

info.description!high

geometry: && [...]

...

info.description!few#333

▶ info.description!high#244

info.description!high#333

info.description!high#720

info.description!humidity#244

...

Example Query

info.description!high
244

geometry: && [...]

Example Query

```
info.description!high  
244
```

```
geometry: && [...]
```

Example Query

info.description!high
244

geometry: && [...]

120

184

210

333

387

415

**All IIDs
within bbox**

Example Query

info.description!high

244

geometry: && [...]

120

184

210

▶ 333

387

415

Find IID

>= 244

Example Query

info.description!high

244

geometry: && [...]

333

**IIDs don't
match =>
No Result
(yet)**

Example Query

info.description!high

geometry: && [...]
333

...
info.description!few#333
info.description!high#244
info.description!high#333
info.description!high#720
info.description!humidity#244
...

Find IID
>=333

Example Query

info.description!high

geometry: && [...]
333

...

info.description!few#333

info.description!high#244

▶ info.description!high#333

info.description!high#720

info.description!humidity#244

...

Example Query

info.description!high

geometry: && [...]
333

...

info.description!few#333

info.description!high#244

▶ info.description!high#**333**

info.description!high#720

info.description!humidity#244

...

Example Query

info.description!high

geometry: && [...]
333

...

info.description!few#333

info.description!high#244

▶ info.description!high#**333**

info.description!high#720

info.description!humidity#244

...

**Found exact
match =>
Return result**

Example Query

info.description!high

geometry: && [...]

...

info.description!few#333

info.description!high#244

info.description!high#333

▶ info.description!high#720

info.description!humidity#244

...

**Go on with
next match**

Example Query

```
info.description!high  
720
```

```
geometry: && [...]
```

```
...  
info.description!few#333  
info.description!high#244  
info.description!high#333  
▶ info.description!high#720  
info.description!humidity#244  
...
```

Example Query

info.description!high

720

geometry: && [...]

120

184

210

333

387

415

Example Query

info.description!high

720

geometry: && [...]

120

184

210

333

387

415

No match

>=720

=> Done

Summary

If one condition narrows down the result a lot, the query should perform well

- Do what people need
- Mapping for including/excluding parts of JSON
- Support for more programming languages
- Scaling up and/or down
- InterPlanetary File System (IPFS)/
InterPlanetary Linked Data (IPLD)

Thanks !

<https://try.noisearch.org/>

Volker Mische

<http://vmx.cx/>

volker.mische@gmail.com
@vmx