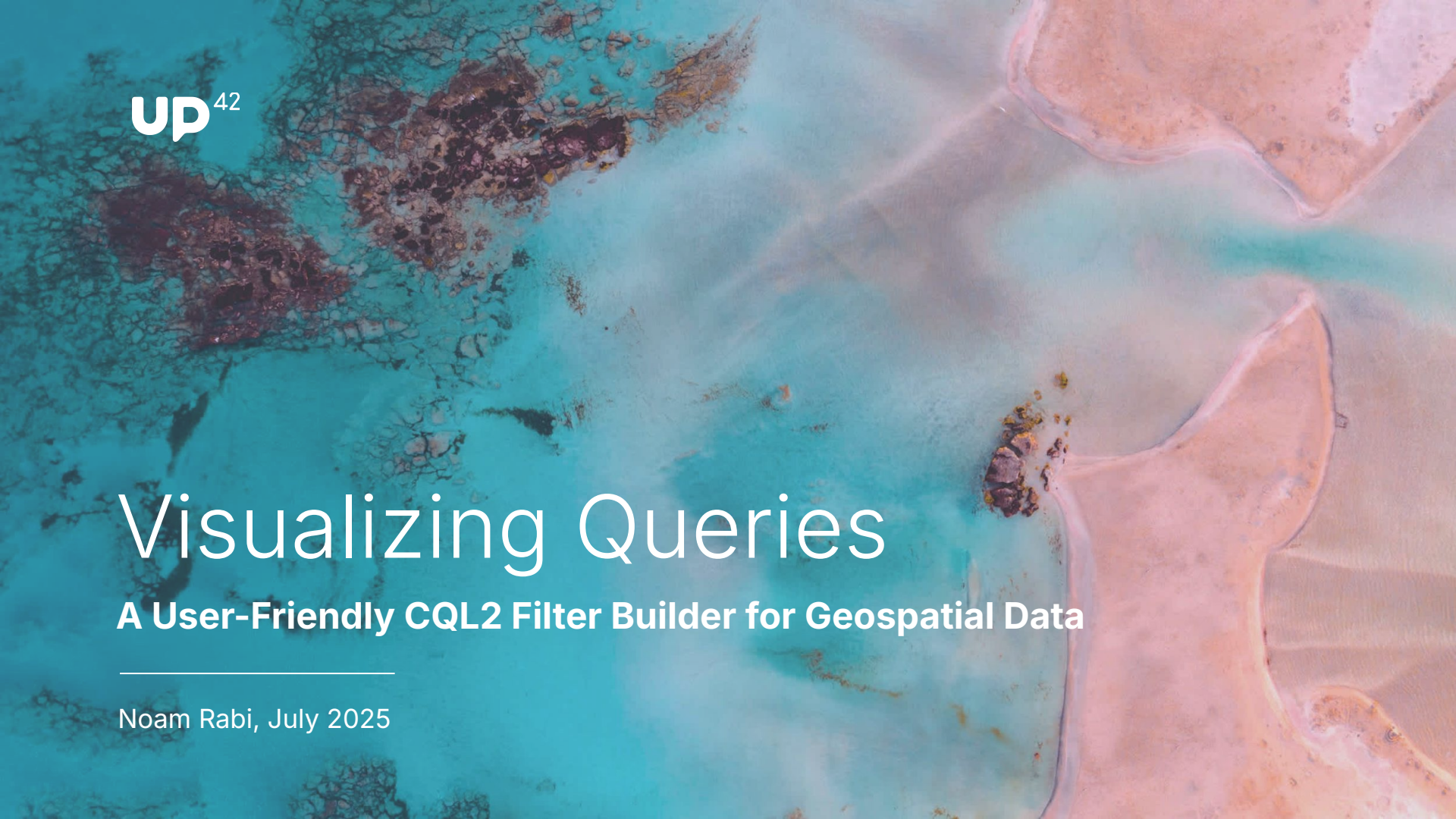


Visualizing Queries

A User-Friendly CQL2 Filter Builder for Geospatial Data

Noam Rabi, July 2025

Hi,
I'm Noam

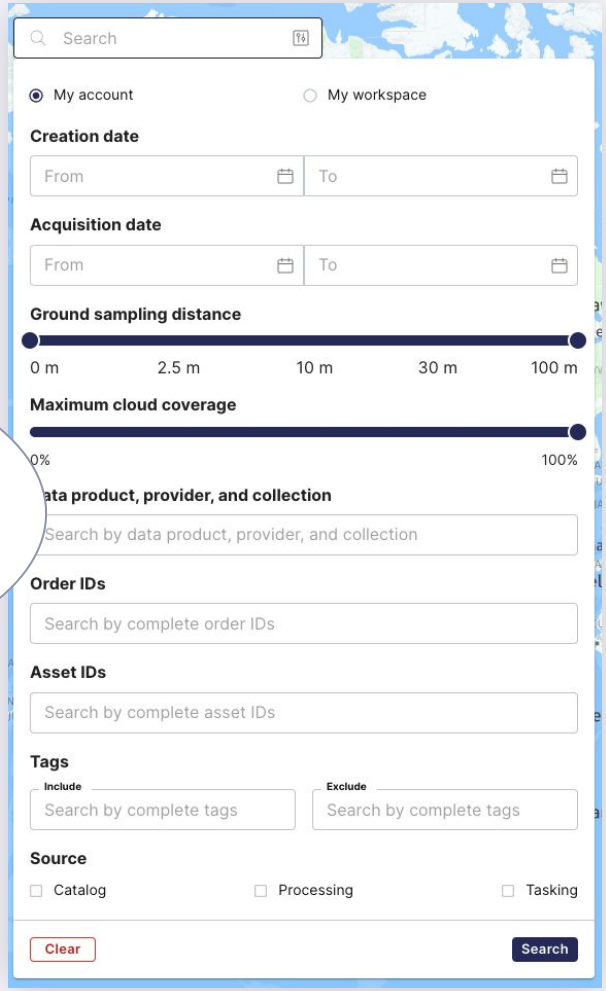
- Senior Front-End Engineer at UP42
- Based in **Berlin**
- My favourite ice cream flavour is **Chocolate**

INTRO

Why visualize queries?

We want to make exploring geospatial datasets easier.

Web interfaces *are good for that!*



UP42's Console

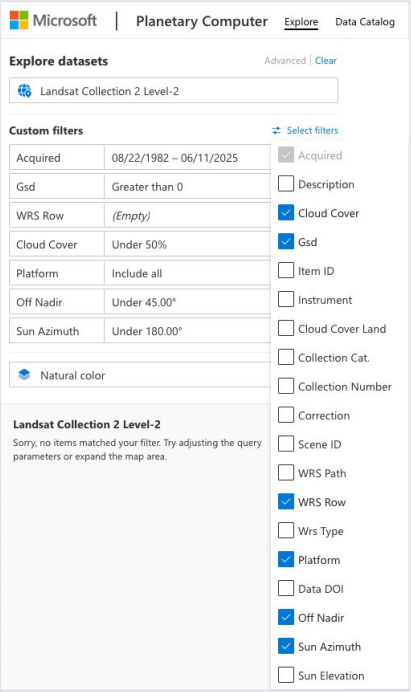
INTRO

Why visualize queries?

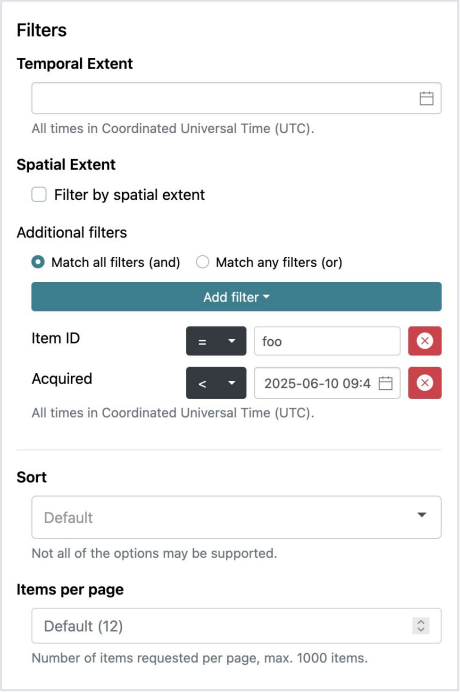
How to translate the form to a query?

Unfortunately, there was no JavaScript library to help me...

So I made one.



Planetary Computer



STAC Browser

INTRO

CQL2-filters-parser library

- **Unified** way to parse CQL2 Text and JSON encodings
- Runs in **Browsers** and **JavaScript** hosts
- **Extendable** to your needs

CQL2 text or JSON Empty

```
city='Mostar' AND T_EQUALS(talk_date, DATE('2025-07-16'))
```

Filter Builder

AND | Convert to rule

Property | city | = | String | Mostar | Convert to AND / OR

T_EQUALS (Property | talk_date , Date | 16.07.2025)

CQL2 filters

What is CQL2 and why is it helpful?

STAC vs CQL2

STAC
SpatioTemporal Asset Catalog

a way to organize geospatial data

- STAC Item
- STAC Catalog

Spec

CQL2
Common Query Language

a language to describe filter expressions for spatial and temporal data

Spec

Two encodings

CQL2 comes in two encodings,

Text and **JSON**

- Query language
- Only operators and operands
- No flow control, loops, recursion

```
// CQL2 Text encoding
city = 'Mostar'

// CQL2 JSON encoding
{
  "op": "=",
  "args": [
    { "property": "city" },
    "Mostar"
  ]
}
```

Two encodings

Differences:

- Audience
- Usage

```
// CQL2 Text encoding
city = 'Mostar'

// CQL2 JSON encoding
{
  "op": "=",
  "args": [
    { "property": "city" },
    "Mostar"
  ]
}
```

CQL2 FILTERS

Building blocks

- **Literals:** strings, numbers, booleans, etc.
- **Properties**: the variables of CQL2
- **Operators**: logical, comparison, arithmetic
- **Combining**: binary expression, function, etc.
- **Spatial**: bbox, geometries
- **Temporal:** timestamp, date, interval

All of these nodes expressions in a tree data structure.

```
// CQL2 Text encoding
city = 'Mostar'

// CQL2 JSON encoding
{
  "op": "=",
  "args": [
    { "property": "city" },
    "Mostar"
  ]
}
```

Parsing Text, Parsing JSON

Parsers

Text → tokenizer → parser → Expression tree

JSON → depth first parser → Expression tree

parse() → Expression tree

Parsers

```
import { parse } from "cql2-filters-parser";

const { encoding, expression } = parse("city='Mostar'");

console.log(encoding); // -> Text
console.log(expression.toText()); // -> "city = 'Mostar'"
console.log(expression.toJSON()); // ->
// {
//   op: '=',
//   args: [ { property: 'city' }, 'Mostar' ]
// }
```

Visitors welcome

Especially for this library :)

Visitor design pattern

Separates the operation from the object.
Allows defining new operations on data structure.

Visitor design pattern

Each node type has a corresponding visit function:

- Literal → visit**Literal**Expression()
- Operator → visit**Operator**Expression()
- Property → visit**Property**Expression()
- Binary expression → visit**Binary**Expression()

The **visitor object** is something that **implements visit functions**

HTML Builder Visitor

The **visitor object** is something that **implements visit functions**

```
const HTMLBuilderVisitor = {
  visitLiteralExpression(expr) { /* ToDo */ },
  visitOperatorExpression(expr) { /* ToDo */ },
  visitPropertyExpression(expr) { /* ToDo */ },
  visitBinaryExpression(expr) { /* ToDo */ },
};

const { expression } = parse("city='Mostar'");
const builderForm = expression.accept(HTMLBuilderVisitor);
document.getElementById("builder").appendChild(builderForm);
```

HTML Builder Visitor

```
const HTMLBuilderVisitor = {
  visitLiteralExpression(expr) {
    return createInputElement(expr.value, expr.type);
  },
  visitOperatorExpression(expr) {
    return createOperatorSelectElement(expr.text);
  },
  visitPropertyExpression(expr) {
    return createInputElement(expr.name, "text");
  },
  visitBinaryExpression(expr) {
    const left = expr.left.accept(BuilderVisitor);
    const op = expr.operator.accept(BuilderVisitor);
    const right = expr.right.accept(BuilderVisitor);
    return createBinaryPairElement(left, op, right);
  },
};

const builderForm = expression.accept(HTMLBuilderVisitor);
```

VISITORS WELCOME

HTML Builder Visitor

```
const HTMLBuilderVisitor = {
  visitLiteralExpression(expr) {
    return cre
  },
  visitOperat
    return c
  },
  visitPrope
    return c
  },
  visitBinary
    const le
    const op
    const ri
    return c
  },
};

const builderForm = expression.accept(HTMLBuilderVisitor);
```

CQL2 Text

```
city='Mostar' AND talk_duration > 20
```

Filter Builder

city	=	Mostar	and	talk_duration	>	20

Wrapping up

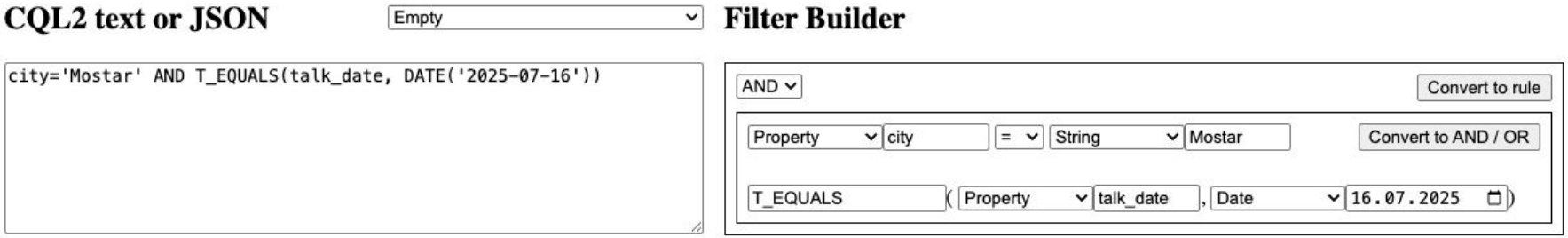
Something funny to add here

WRAPPING UP

CQL2-filters-parser library

GitHub repository

- **Unified** way to parse CQL2 Text and JSON encodings
- Runs in **Browsers** and **JavaScript** hosts
- **Extendable** to your needs



CQL2 Playground - https://noamra.github.io/ogc-cql2-filters

GitHub repository - https://github.com/NoamRa/ogc-cql2-filters

npm i **cql2-filters-parser**