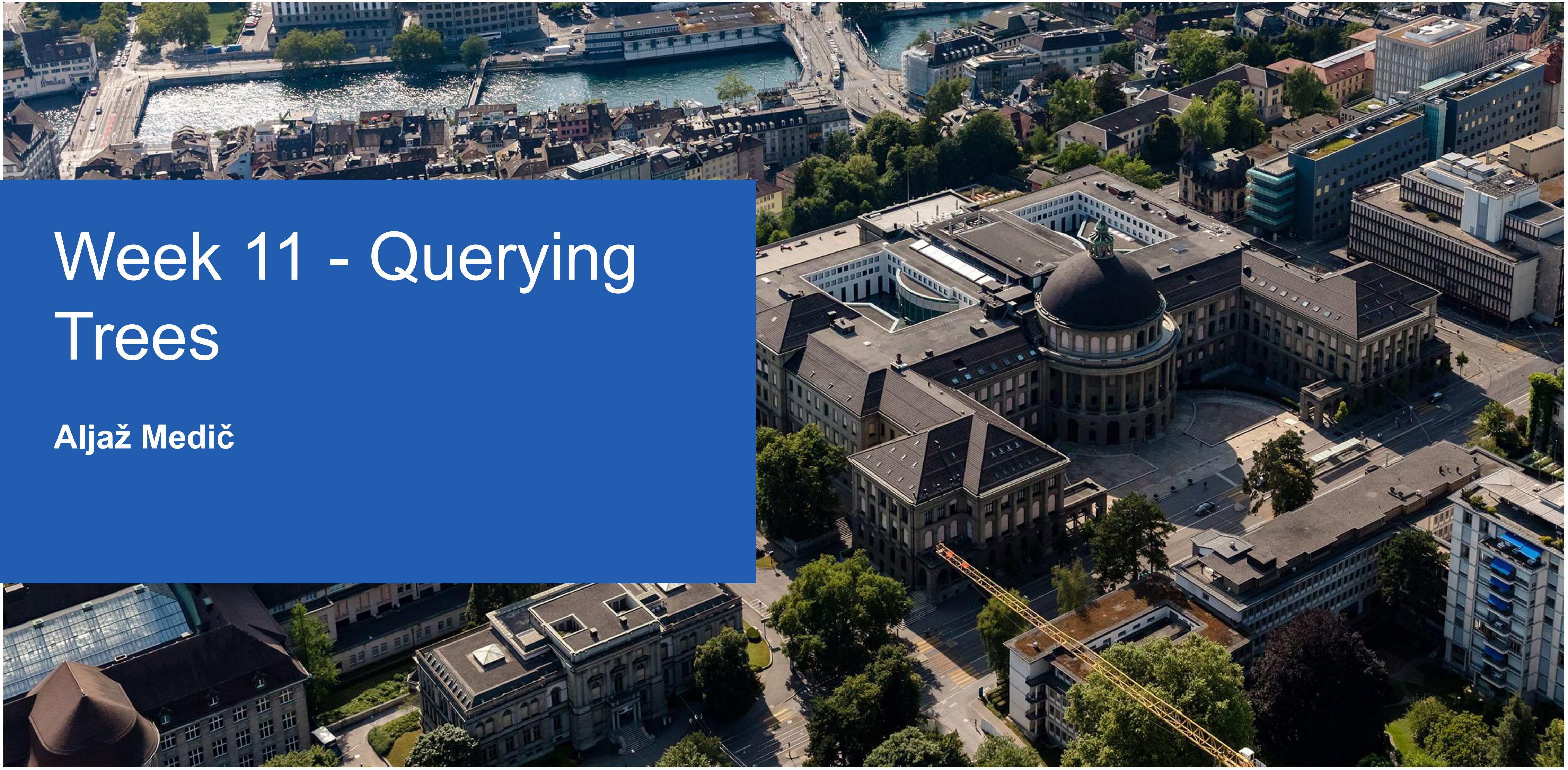


Week 11 - Querying Trees

Aljaž Medič

Plan for today

1. Quiz
2. Old exam questions
3. JSONiq queries

QUIZ

What is JSONiq?

A **query** language for querying complex, denormalized data.

What are desirable features for a high-level query language for denormalized data?

Being declarative, functional, set-based.

JSONiq expressions can return at most one item.

False. (The always return a sequence of items, which can have cardinality 0, 1, 2+ .)

JSONiq is based on XQuery and shares most of its semantics.

True.

QUIZ

Any valid JSON is also a valid JSONiq query.

True. (It returns itself in a 1-length sequence.)

Sequences can be nested.

False. (Sequences are always flat.)

An item in a sequence can be an object, a list, or an atomic item.

True.

You cannot join two different collections in JSONiq.

False. (Possible with nested/double for clause.)

QUIZ

Sequences are always homogeneous.

False. (They can contain a mixture of different items.)

Dot syntax discards any item in the sequence that is not an object.

True.

Array unboxing (`array[]`) preserves the original boundaries of subarrays.

False. (They get flattened into a single sequence.)

QUIZ

What is the result of the queries `([1,2], [3,4], 5)[2]` and `([1,2], [3,4], 5)[[2]]`?

1st: [3, 4] - Taking 2nd item from the sequence.

2nd: (2,4) - Taking 2nd item from the array(s).

In sequence filtering, we can use $$ to refer to the current item.

True. (e.g. `data[$$.name = “Albert”],` helper function: `data[position()<3]`)

What are FLWOR expressions?

For-Let-Where-OrderBy-Return expressions correspond to SQL’s SELECT-FROM-WHERE, but for nested data + additional flexibility.

QUIZ

We can have more than 1 return in FLWOR expression.

False.

Operation `1 + ()` evaluates to `()` and `1 + (1, 2)` throws an error.

True. (Arithmetic operations work inside “Maybe” monad.)

Comparison `(1 to 5 = 3 to 7)` returns `true`, and `(1 to 5 eq 3 to 7)` throws an error.

True. (=, !=, <, > , <=, >= are used for comparing sequences (existance + comparison)

eq, ne, le, ge, lt, gt are used for atomic items.)

QUIZ

The following query returns the object: `{“1”:2, “2”:4, “3”:6, “4”:8}`:

```
for $i in 1 to 4
return {string($i): $i * 2}
```

False. (It returns a sequence of objects. If we wrap the FLWOR with `{|...|}` it would.)

In a FLWOR expression, if you write `let $a := 1`, followed by `let $a := 2`, you reassigned the value of the variable `$a`.

False. (We created a new binding, that shadows the previous one. Variables are immutable.)

There is no strict order in which where, order by, count and let must appear in FLWOR (unlike SQL).

True.

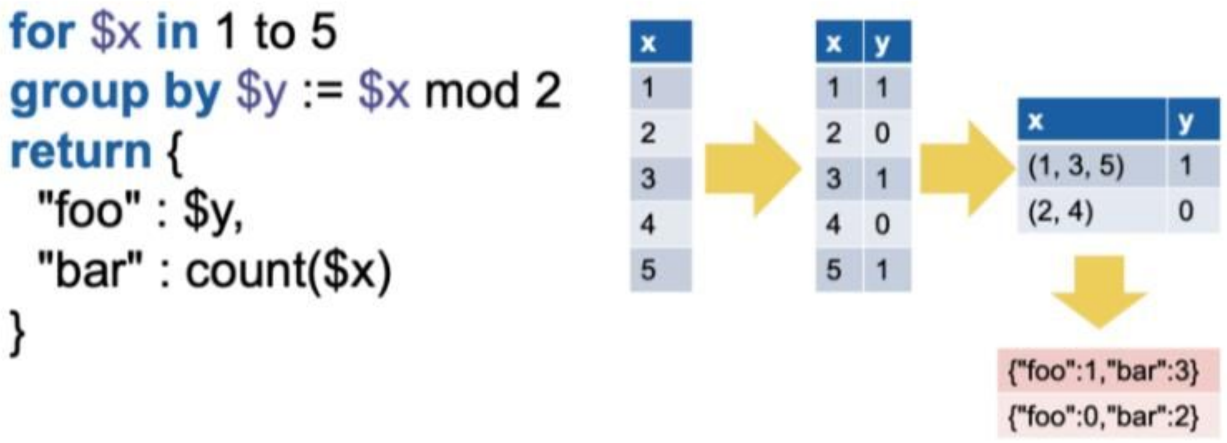
QUIZ

Does this return true, false or error: `count([1,1,2,3,5,8,13,21]) eq 8`.

count(...) operates on sequences, and this sequence has 1 element (array if size 8). The value 1 is then compared (as an item) to 8, which is **false**.

After a `group` by a variable that was previously bound to a single item will be bound to a sequence.

True.



```
for $x in 1 to 5
group by $y := $x mod 2
return {
  "foo" : $y,
  "bar" : count($x)
}
```

x
1
2
3
4
5

x	y
1	1
2	0
3	1
4	0
5	1

x	y
(1, 3, 5)	1
(2, 4)	0

```
{"foo":1,"bar":3}
{"foo":0,"bar":2}
```

QUIZ

JSONiq allows directly accessing S3, HDFS, http or local filesystem via `json-file()`.

True. (Just change the protocol in the URI.)

If the expression in the for clause evaluates into an empty sequence, the variable is bound to `null`.

False. (The FLWOR returns an empty sequence.)

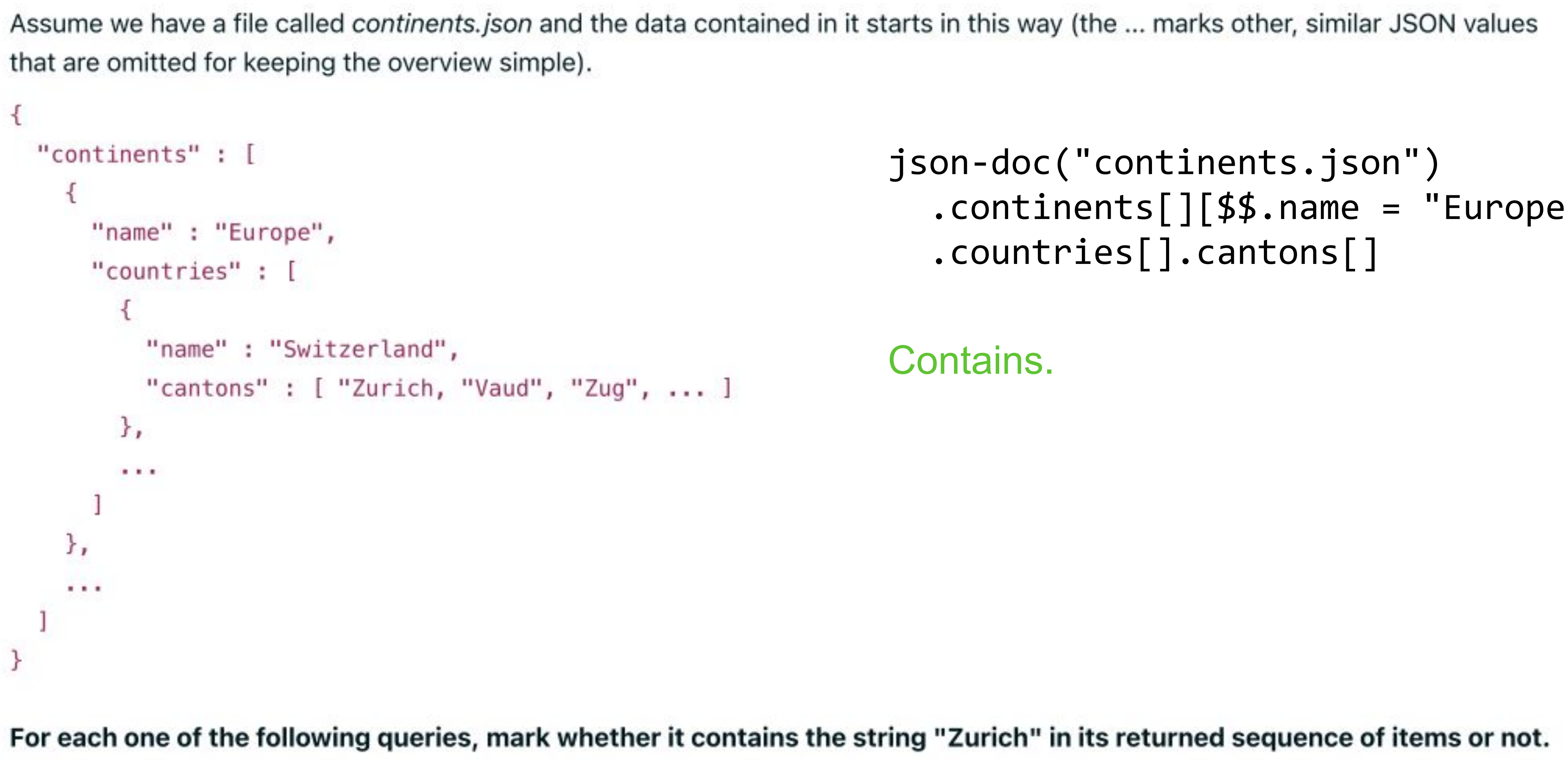
The mode of the execution is decided statically at compile time.

True. (We know enough before the query is actually runs.)

Each clause in FLWOR expression materializes the intermediate result and then continues processing it.

False. (Clauses can stream the tuples in batches, and pipeline processing. Not always possible.)

Old Exam Question - HS24, Q57



Assume we have a file called *continents.json* and the data contained in it starts in this way (the ... marks other, similar JSON values that are omitted for keeping the overview simple).

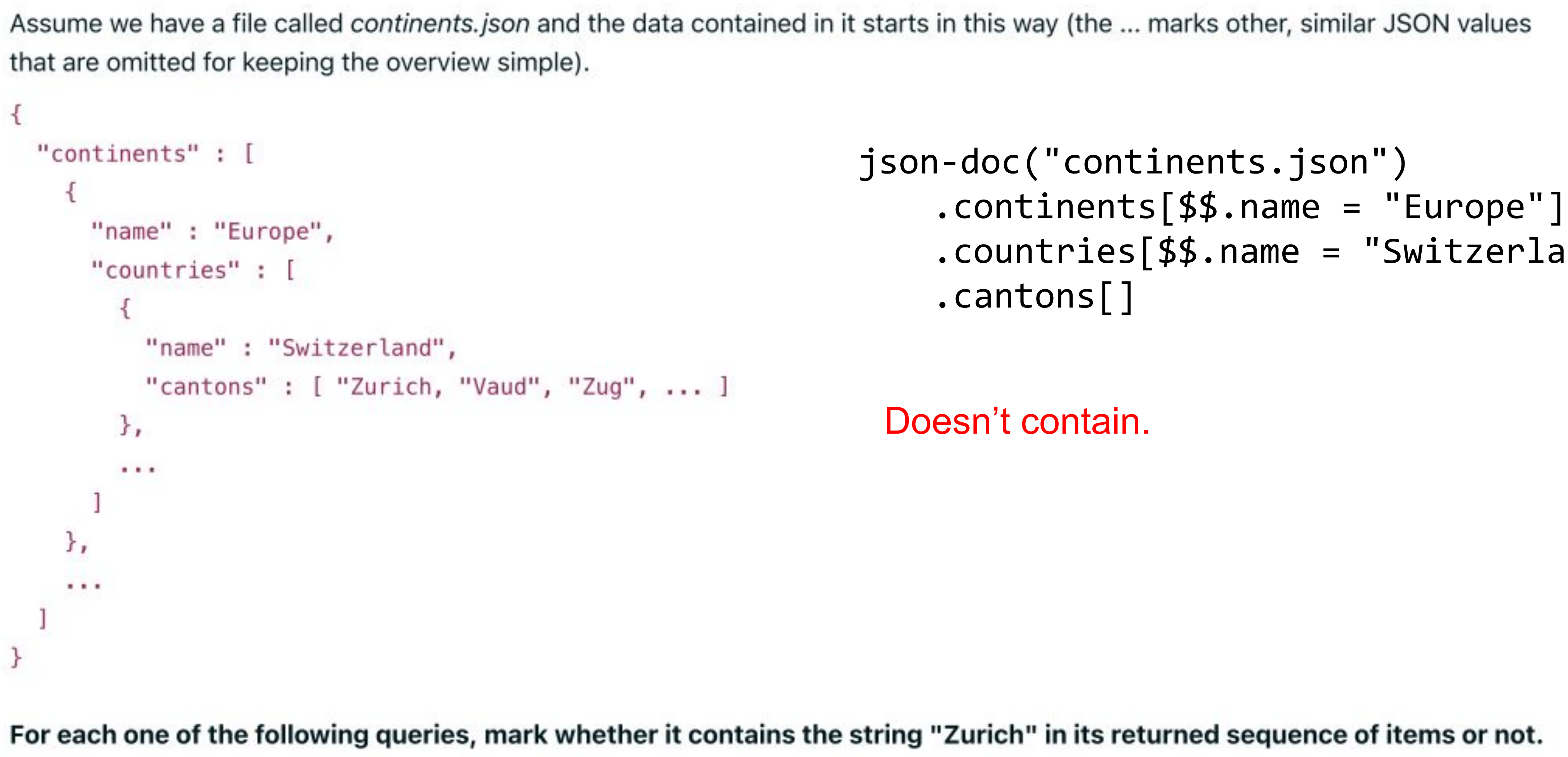
```
{
  "continents" : [
    {
      "name" : "Europe",
      "countries" : [
        {
          "name" : "Switzerland",
          "cantons" : [ "Zurich, "Vaud", "Zug", ... ]
        },
        ...
      ]
    },
    ...
  ]
}
```

For each one of the following queries, mark whether it contains the string "Zurich" in its returned sequence of items or not.

```
json-doc("continents.json")
  .continents[][$$.name = "Europe"]
  .countries[].cantons[]
```

Contains.

Old Exam Question - HS24, Q57



Assume we have a file called *continents.json* and the data contained in it starts in this way (the ... marks other, similar JSON values that are omitted for keeping the overview simple).

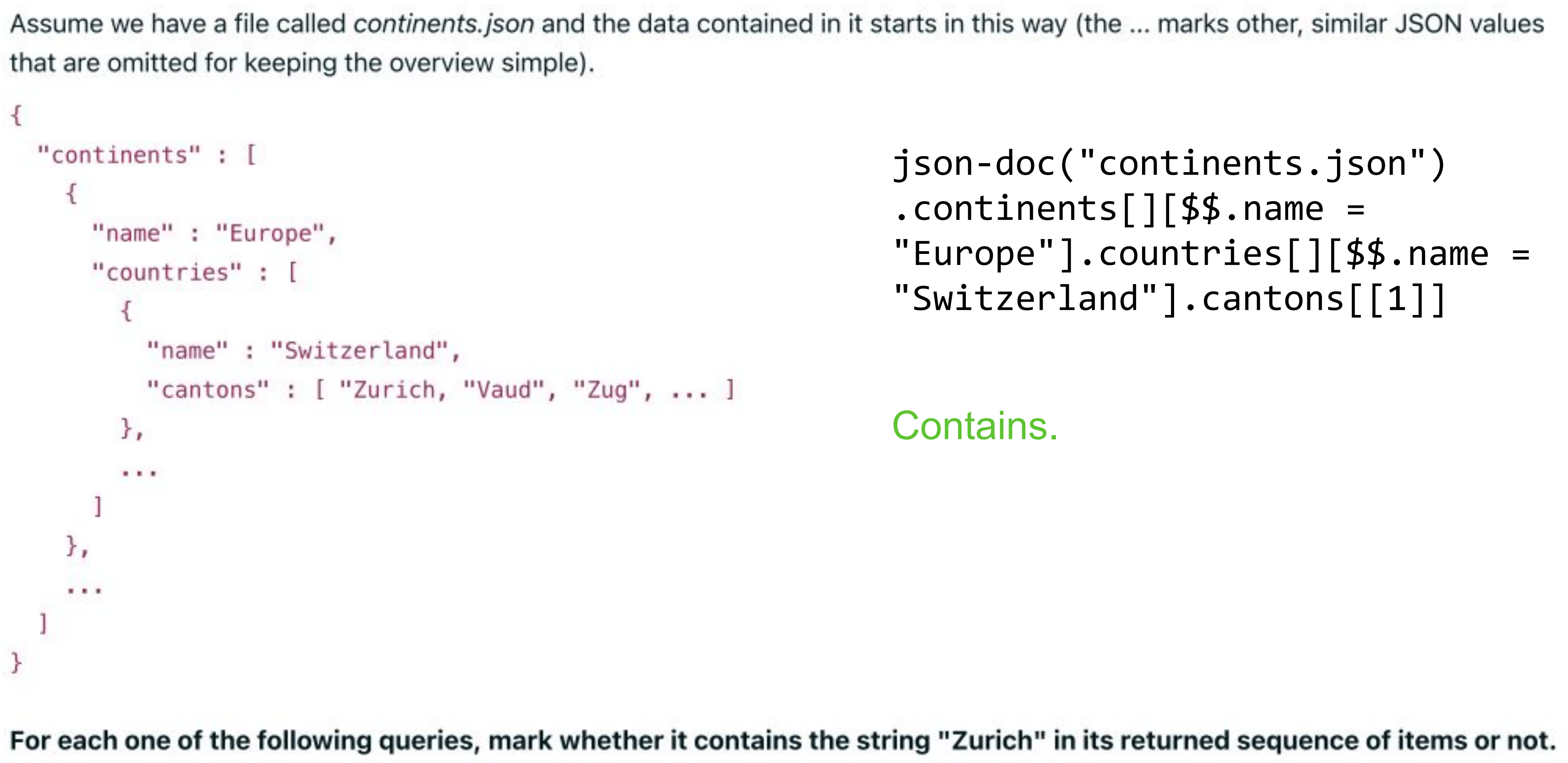
```
{
  "continents" : [
    {
      "name" : "Europe",
      "countries" : [
        {
          "name" : "Switzerland",
          "cantons" : [ "Zurich, "Vaud", "Zug", ... ]
        },
        ...
      ]
    },
    ...
  ]
}
```

For each one of the following queries, mark whether it contains the string "Zurich" in its returned sequence of items or not.

```
json-doc("continents.json")
    .continents[$$.name = "Europe"]
    .countries[$$.name = "Switzerland"]
    .cantons[]
```

Doesn’t contain.

Old Exam Question - HS24, Q57



Assume we have a file called *continents.json* and the data contained in it starts in this way (the ... marks other, similar JSON values that are omitted for keeping the overview simple).

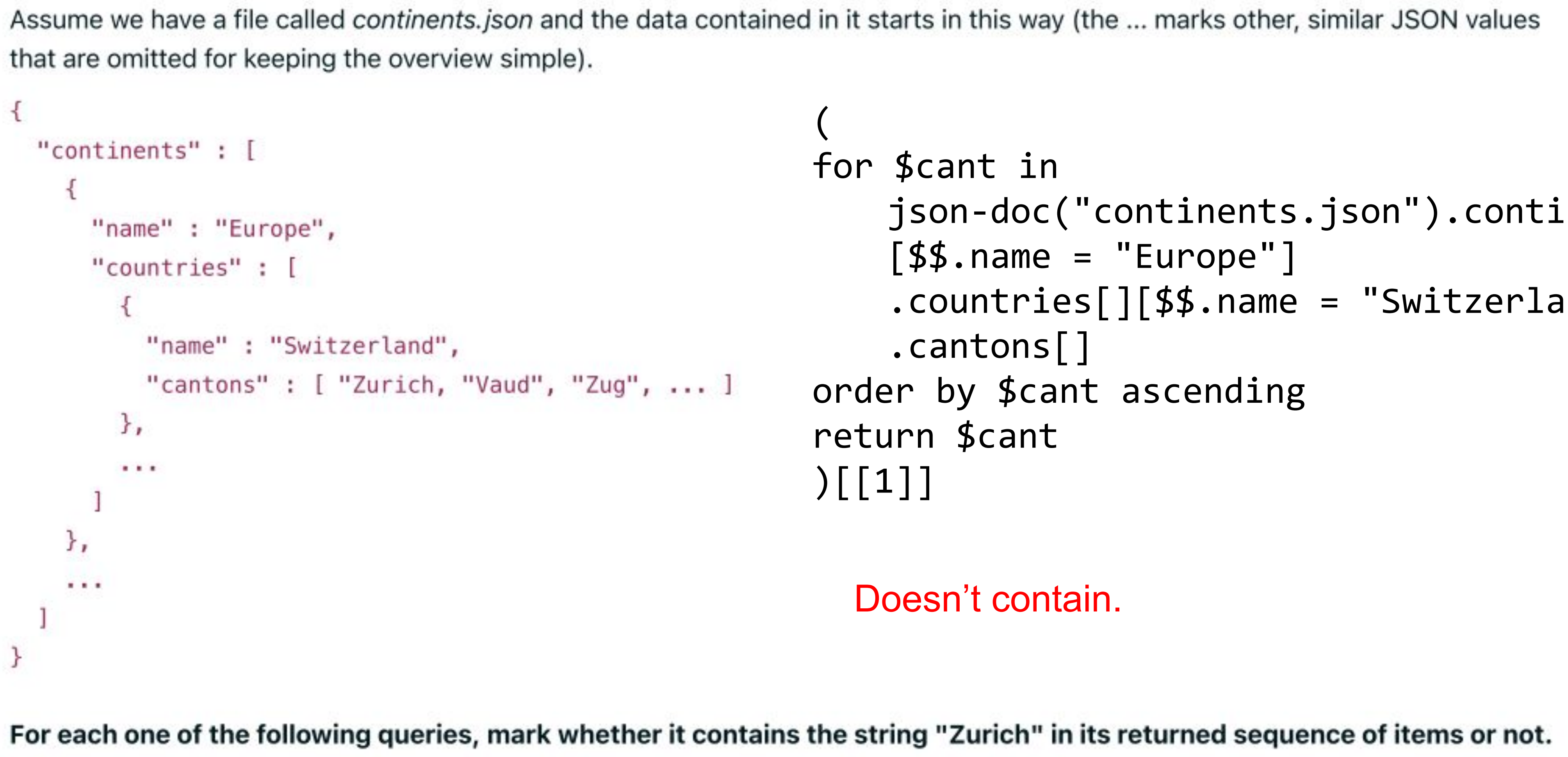
```
{
  "continents" : [
    {
      "name" : "Europe",
      "countries" : [
        {
          "name" : "Switzerland",
          "cantons" : [ "Zurich, "Vaud", "Zug", ... ]
        },
        ...
      ]
    },
    ...
  ]
}
```

For each one of the following queries, mark whether it contains the string "Zurich" in its returned sequence of items or not.

```
json-doc("continents.json")
.continents[][$$.name =
"Europe"].countries[][$$.name =
"Switzerland"].cantons[[1]]
```

Contains.

Old Exam Question - HS24, Q57



Assume we have a file called *continents.json* and the data contained in it starts in this way (the ... marks other, similar JSON values that are omitted for keeping the overview simple).

```
{
  "continents" : [
    {
      "name" : "Europe",
      "countries" : [
        {
          "name" : "Switzerland",
          "cantons" : [ "Zurich, "Vaud", "Zug", ... ]
        },
        ...
      ]
    },
    ...
  ]
}
```

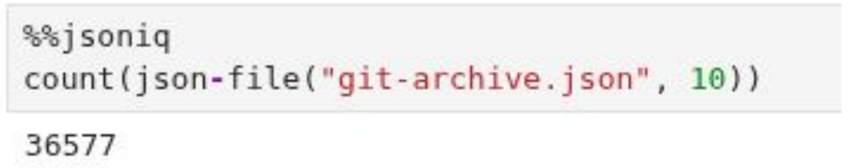
For each one of the following queries, mark whether it contains the string "Zurich" in its returned sequence of items or not.

```
(
for $cant in
    json-doc("continents.json").continents[]
    [$$.name = "Europe"]
    .countries[][$$.name = "Switzerland"]
    .cantons[]
order by $cant ascending
return $cant
)[[1]]
```

Doesn’t contain.

JSONiq programming questions

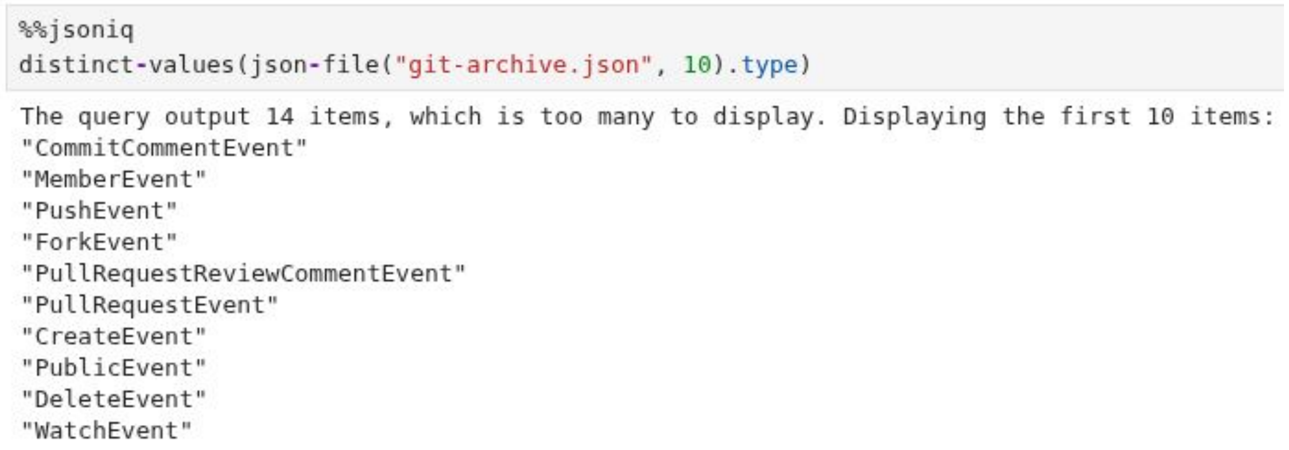
How many events are there in the dataset?



```
%%jsoniq
count(json-file("git-archive.json", 10))
```

```
36577
```

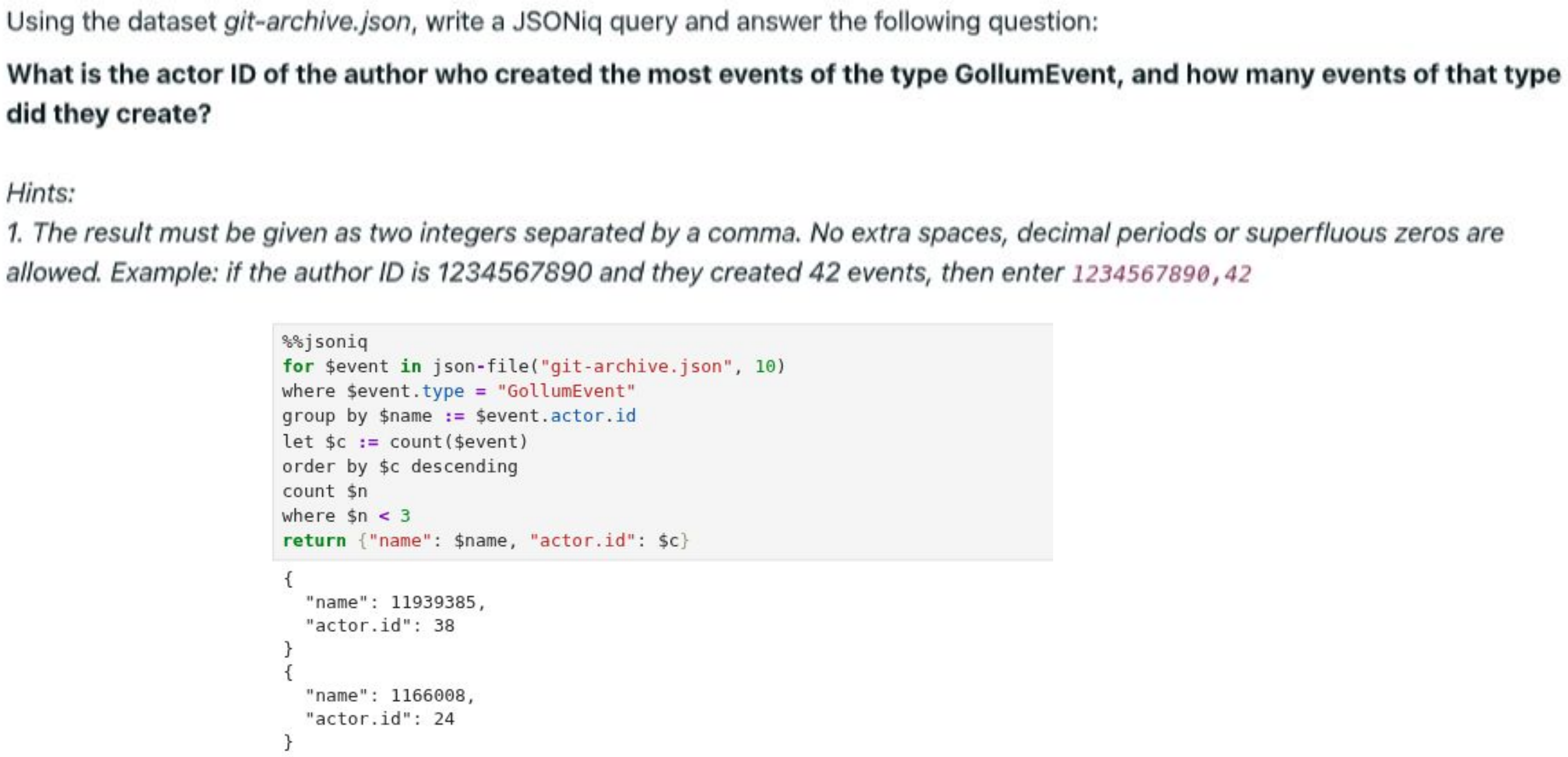
How many distinct event types are there in the dataset?



```
%%jsoniq
distinct-values(json-file("git-archive.json", 10).type)
```

```
The query output 14 items, which is too many to display. Displaying the first 10 items:
"CommitCommentEvent"
"MemberEvent"
"PushEvent"
"ForkEvent"
"PullRequestReviewCommentEvent"
"PullRequestEvent"
"CreateEvent"
"PublicEvent"
"DeleteEvent"
"WatchEvent"
```

Old Exam Question - HS24, Q60



Using the dataset *git-archive.json*, write a JSONiq query and answer the following question:

What is the actor ID of the author who created the most events of the type GollumEvent, and how many events of that type did they create?

Hints:
1. The result must be given as two integers separated by a comma. No extra spaces, decimal periods or superfluous zeros are allowed. Example: if the author ID is 1234567890 and they created 42 events, then enter `1234567890,42`

```
%%jsoniq
for $event in json-file("git-archive.json", 10)
where $event.type = "GollumEvent"
group by $name := $event.actor.id
let $c := count($event)
order by $c descending
count $n
where $n < 3
return {"name": $name, "actor.id": $c}
```

```
{
  "name": 11939385,
  "actor.id": 38
}
{
  "name": 1166008,
  "actor.id": 24
}
```

ETH zürich

See you next week!

Aljaž Medič
amedic@ethz.ch

Slides

Suggestions