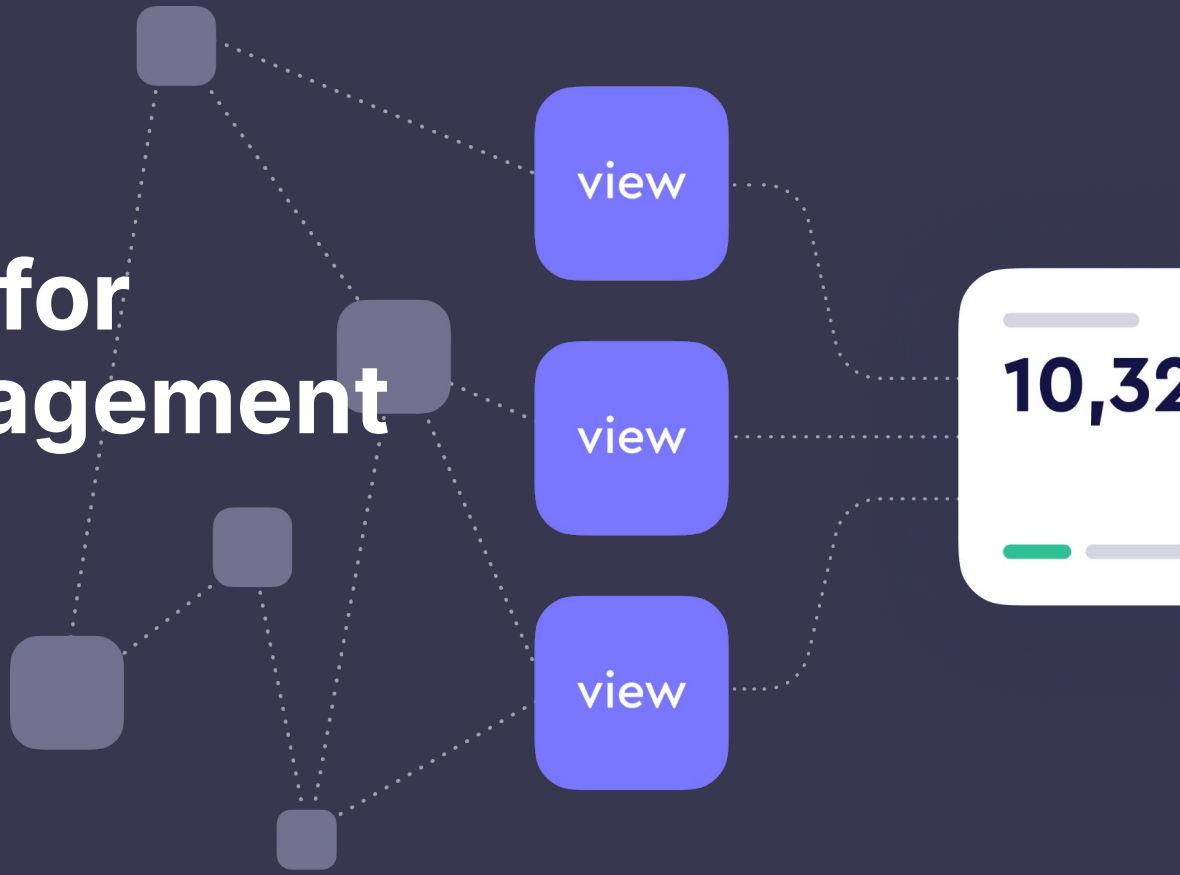
cube

Using Views for metrics management

Igor Lukanin
Head of Developer Relations



Community Code of Conduct

We want to foster an open and welcoming environment where everyone feels they belong in the Cube community.

The full text of our Code of Conduct is available at github.com/cube-js/cube.js/blob/master/CODE_OF_CONDUCT.md

Any instances of inappropriate/unacceptable behavior can be reported to conduct@cube.dev

Quick notes

- If you have any questions during the workshop, please type them in the “Q&A” section (on Zoom & YouTube)
- We will be using Cube Cloud for “hands-on” demos
- Recording of the workshop will be posted on the Cube’s YouTube channel
- All attendees will receive a post-event survey and we’d appreciate your feedback to help us with future events

What we will discuss today

- Recap of data modeling in Cube
- Views, the most recent addition to data modeling in Cube
- Use cases for Views
    a. Defining metrics
    b. Managing visibility and governance
    c. Controlling join paths
- Demo session
- Q&A session

Data Modeling

Cube — Headless Business Intelligence

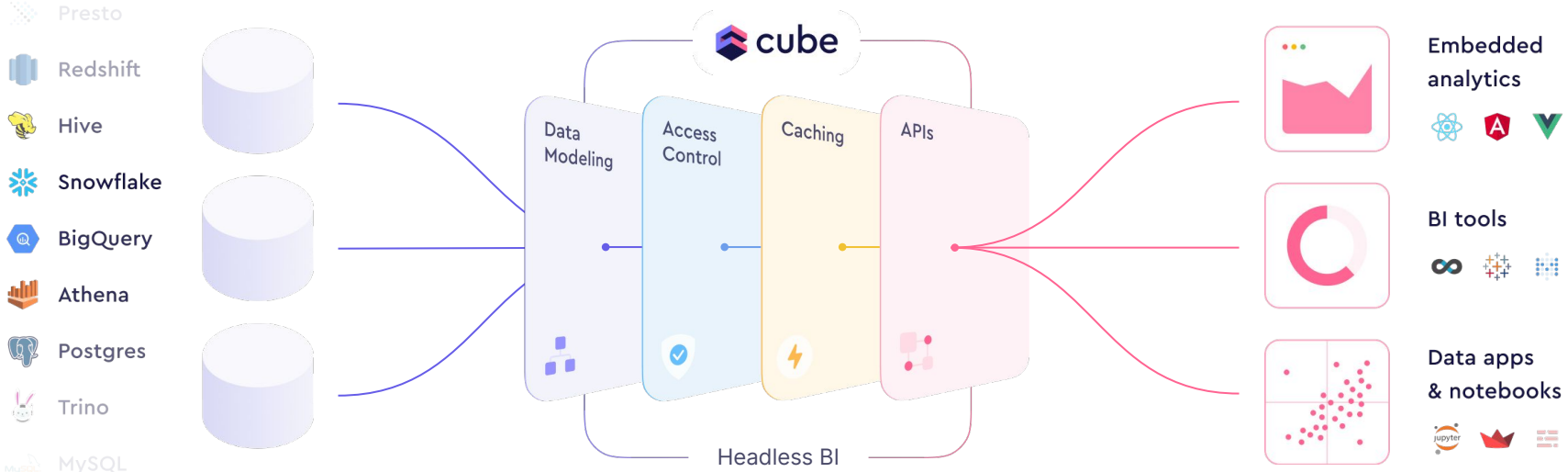
Cube helps to...

- consume data from modern data stores,
- organize it into consistent definitions, and
- deliver it to every application.

In-depth blog post: cube.dev/blog/headless-bi

Headless Business Intelligence

Four layers of [headless BI](): data modeling, access control, caching, and APIs.



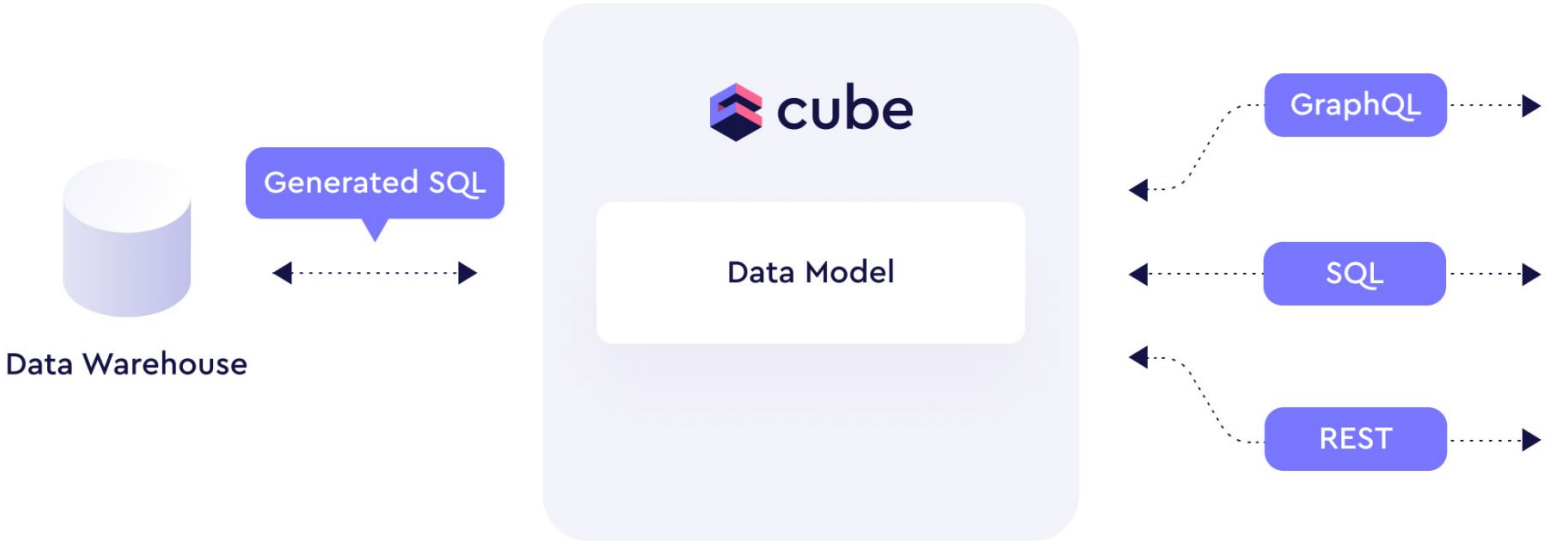
Data Modeling

Data modeling helps to...

- represent upstream data as entities in your business domain, and
- enable data access with semantic queries.

In-depth blog post: cube.dev/blog/open-source-data-modeling

Data Modeling



Elements of a Data Model (1/2)

Cubes are high-level entities, inspired by OLAP cubes.

They represent data tables that make sense in your business domain.

Examples:

- `Users`
- `Orders`
- `AdViews`
- `SupportQuestions`
- `ActivationFunnel`
- `Fortune1000`, etc.

Elements of a Data Model (2/2)

Cubes serve as namespaces for members: measures, dimensions, segments, joins, pre-aggregations, etc.

- *Measures* represent aggregated quantitative data,
  e.g., `Users.count`
- *Dimensions* represent qualitative data,
  e.g., `Orders.status`
- *Segments* represent pre-defined filters,
  e.g., `AdViews.ownedChannels`
- *Joins* represent relationships between cubes,
  e.g., each one of `Users` can have many `SupportQuestions`
- *Pre-aggregations* configure caching and query acceleration

Multiple APIs to Access the Data Model

- SQL API for BI tools, data notebooks, etc.
- REST API for front-end apps and programmatic access
- GraphQL API for front-end apps

All APIs provide data access with semantic queries.

Query Example

SQL API

```
SELECT country, SUM(totalValue)
FROM Orders
GROUP BY 1
```

REST API

```
{
  measures: [ Orders.totalValue ],
  dimensions: [ Orders.country ]
}
```

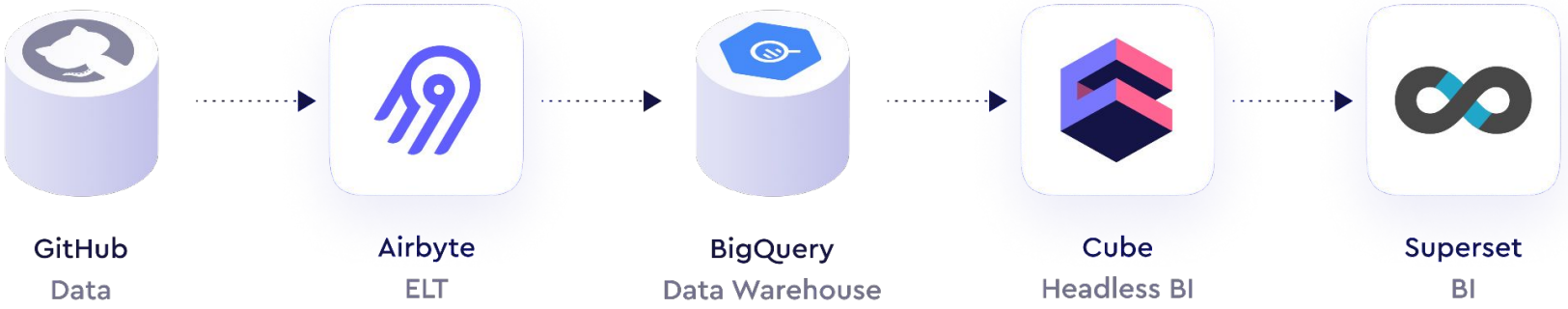
GraphQL API

```
{
  cube {
    orders {
      totalValue
      country
    }
  }
}
```

Demo 

Demo Data Pipeline

- Moving data from GitHub API to BigQuery with Airbyte.
- Organizing it into consistent definitions with Cube.
- Delivering it to Apache Superset, etc.

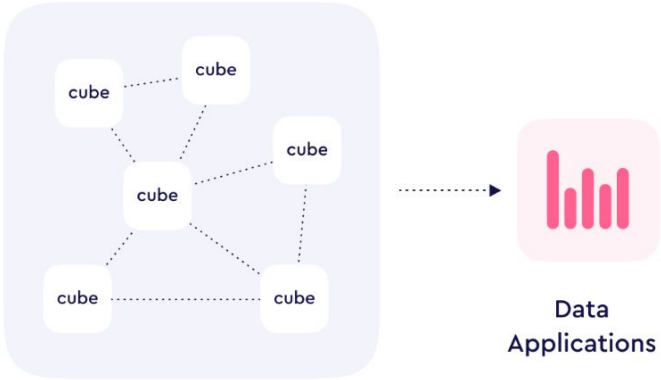


Q & A

Views

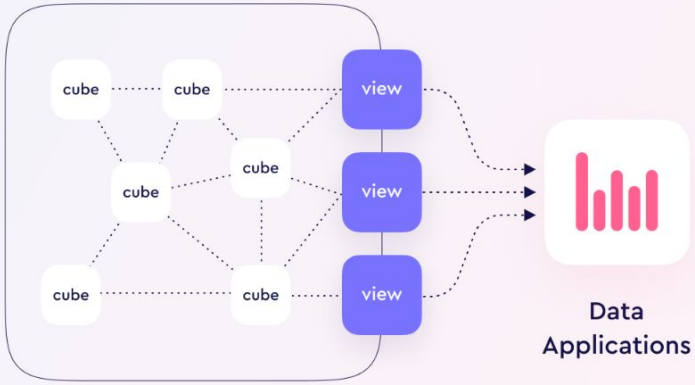
Views: New in Data Model

Before



How to define metrics? How to control join path?
How to manage visibility of cubes
to different teams/users?

Now



Views are the place to define metrics, control ambiguousjoin paths, and manage governance and accessibility for data consumers.

Views

Like cubes, *Views* are also high-level entities.

Views don’t define their own members. Instead, they let you

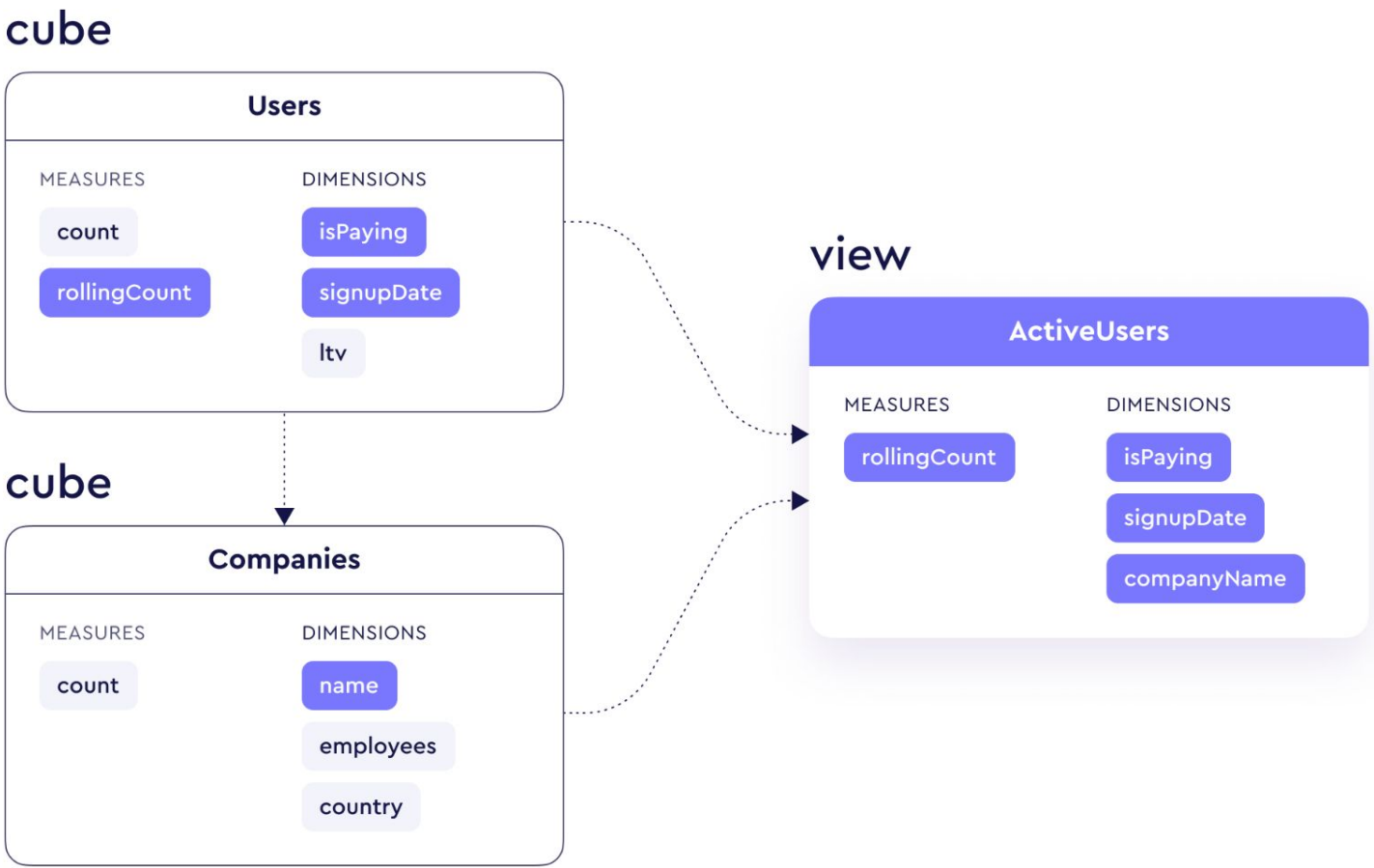
- expose select members of cubes
- you to manage the visibility
- control join paths

Views allow you to define the "public API" on the surface of your semantic layer.

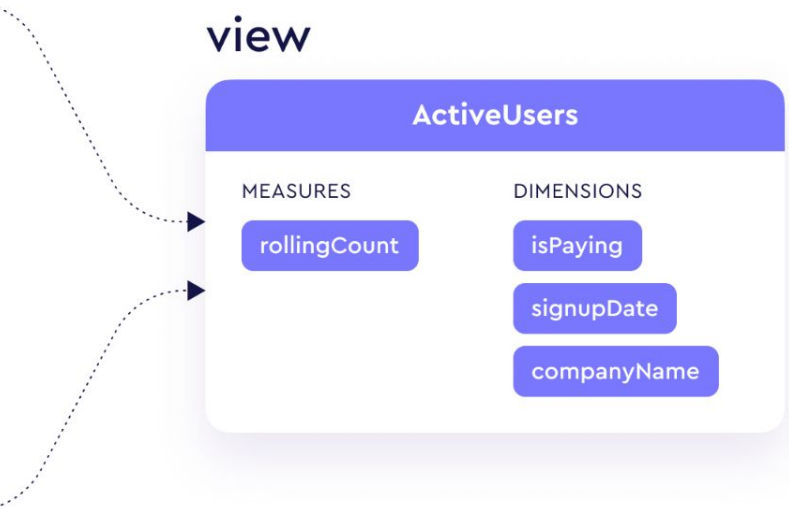
Views are optional: if you don’t need them, don’t use them.

Available starting from Cube v0.31.0.

Use Case: Defining Metrics


cube
Users
MEASURES
count
rollingCount
DIMENSIONS
isPaying
signupDate
ltv
cube
Companies
MEASURES
count
DIMENSIONS
name
employees
country
view
ActiveUsers
MEASURES
rollingCount
DIMENSIONS
isPaying
signupDate
companyName

Defining a View




```
view(`ActiveUsers`, {
  includes: [
    Users.rollingCount,
    Users.isPaying,
    Users.signupDate,
  ],

  dimensions: {
    companyName: {
      sql: `${Company.name}`,
      type: `string`
    },
  }
});
```

Exposing Select Members / Defining Metrics

You can...

- Use `includes` to reference members of cubes
- Use `measures` and `dimensions` to reference *and rename* them

You can also *define metrics* as single-measure views.

From dbt docs: “A metric is a time-series aggregation over a table that supports zero or more dimensions.”

Demo 

Q & A

Use Case: Managing Visibility

Managing Visibility of Views (and Cubes)

You can...

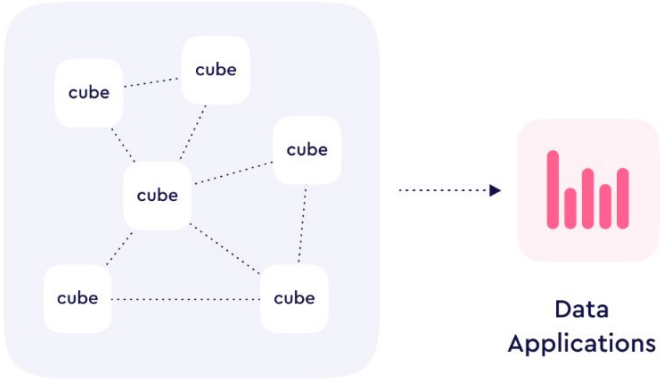
- Use `shown` to define whether a view should be available via APIs
- Use `COMPILE_CONTEXT` to show a view based on a condition
- Also works with cubes!

Possible scenario:

- hide all cubes,
- show views,
- but just to specific users.

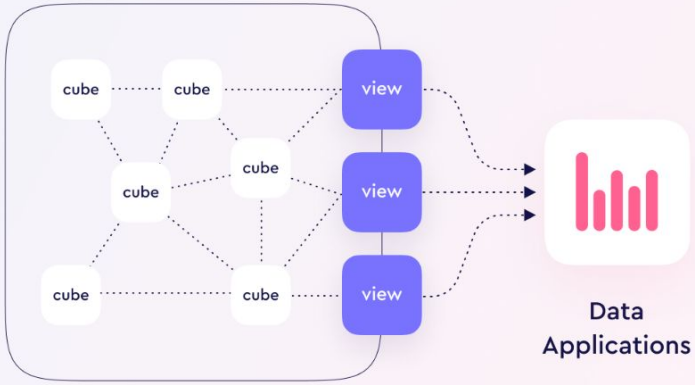
What If Only Views Are Shown in the API?

Before



How to define metrics? How to control join path?
How to manage visibility of cubes
to different teams/users?

Now



Views are the place to define metrics, control ambiguousjoin paths, and manage governance and accessibility for data consumers.

Conditionally Showing a View

```
view(`ARR`, {
  shown: COMPILE_CONTEXT.securityContext.finance,

  includes: [
    Revenue.arr,
    Revenue.date,
    Customers.plan
  ],
});
```

More in docs: cube.dev/docs/security/context#using-compile-context

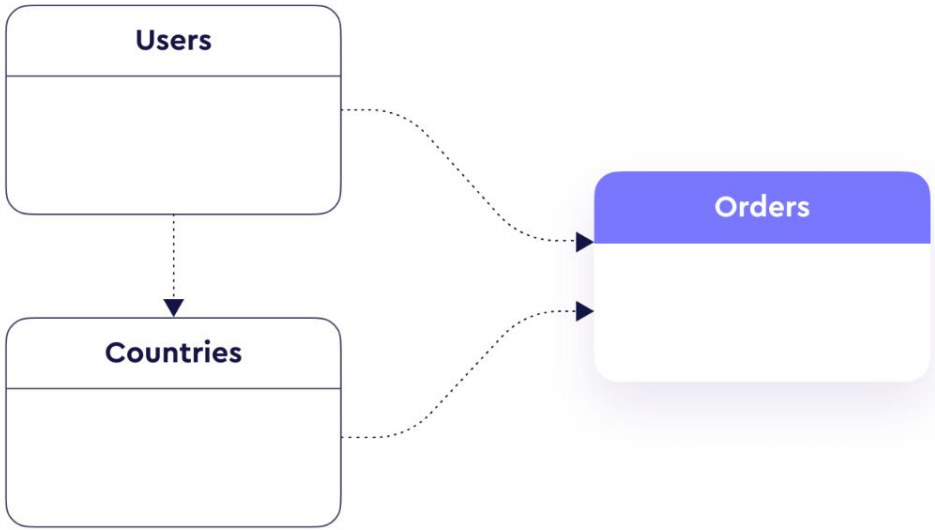
Demo 

Q & A

Use Case: Controlling Join Path

Multiple Join Paths from Cube X to Z

- `Orders` are delivered from specific `Countries`
- `Orders` are made by `Users` from specific `Countries`



Controlling Join Paths

You can...

- Use `measures` and `dimensions` to reference and rename members
- Reference members as `CubeX.CubeY.CubeZ.member`

Workaround from pre-views era:

- duplicate a cube with `extends`,
- add an additional join to it.

Conditionally Showing a View

```
view(`CompletedOrders`, {
  includes: [ Orders.count ],

  dimensions: {
    warehouseCountry: {
      sql: `${Orders.Countries.name}`,
      type: `string`
    },
    userCountry: {
      sql: `${Orders.Users.Countries.name}`,
      type: `string`
    }
  }
});
```

Demo 

Q & A

Learn more

Blog:

- cube.dev/blog/headless-bi
- cube.dev/blog/open-source-data-modeling
- cube.dev/blog/introducing-views

Documentation:

- cube.dev/docs/schema/getting-started
- cube.dev/docs/schema/reference/view

Slack community of ~7,000 data practitioners:

slack.cube.dev


cube