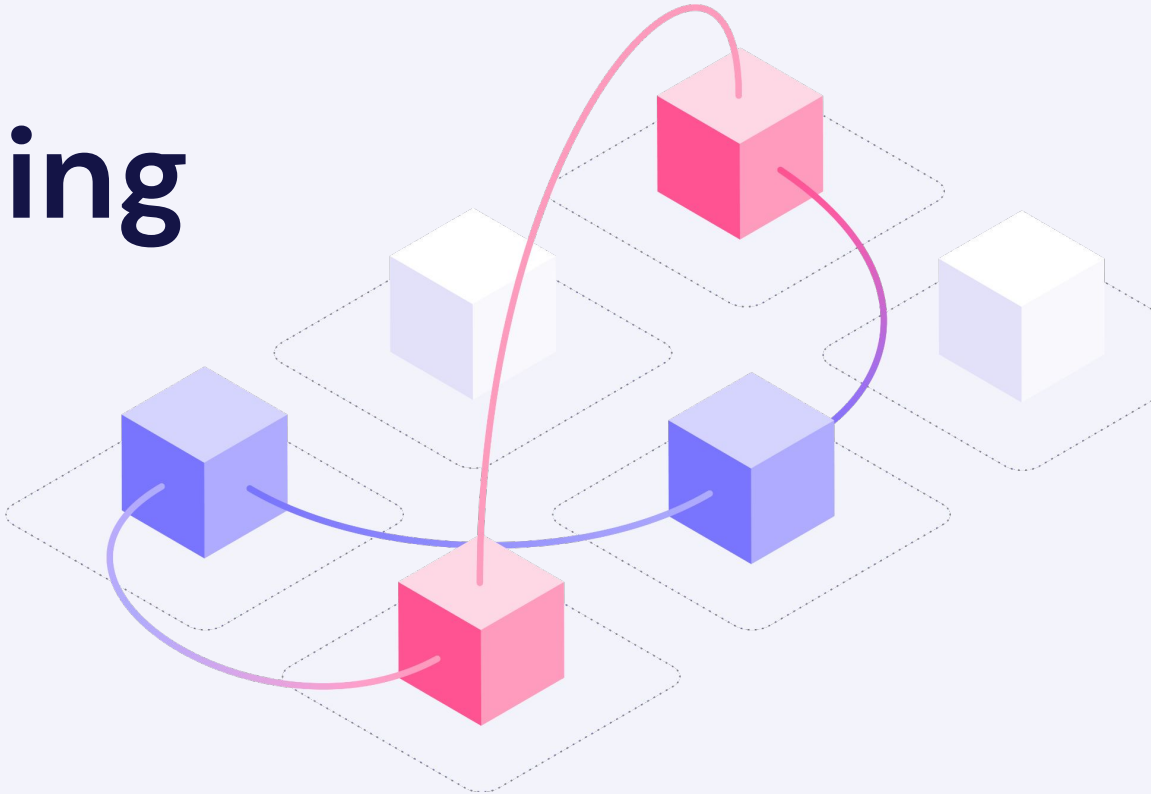
January 19, 2022

Data Modeling Workshop



cube


Ryan Pei


Igor Lukanin

Community Code of Conduct

- We want to foster an open and welcoming environment where everyone feels they belong in the Cube.js community.
- The full text of our Code of Conduct is available at https://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.

Some quick notes 🗒️

- If you have any questions during the workshop, please feel free to type them in the “Q&A”.
- We will be using Cube Cloud during the demo/hands-on lab.
- Recording of today’s workshop will be posted on the Cube Dev YouTube channel.
- All attendees will receive a post-event survey and we’d appreciate your feedback to help us with future events.
    - In the survey, you can also provide your address so we can mail your Cube swags 😊

What we will discuss today

- Data schema design
- Data schema compilation and versioning
- Dynamic data schema
- Joins
- Query-dependent data
- Resources
- Q&A

Data schema design

Data schema design

- Data schema elements are exposed via the Cube API, so **the data schema should reflect your business domain**
- Rule of thumb: *If an API query makes sense in your business domain, you're doing it right* 👌

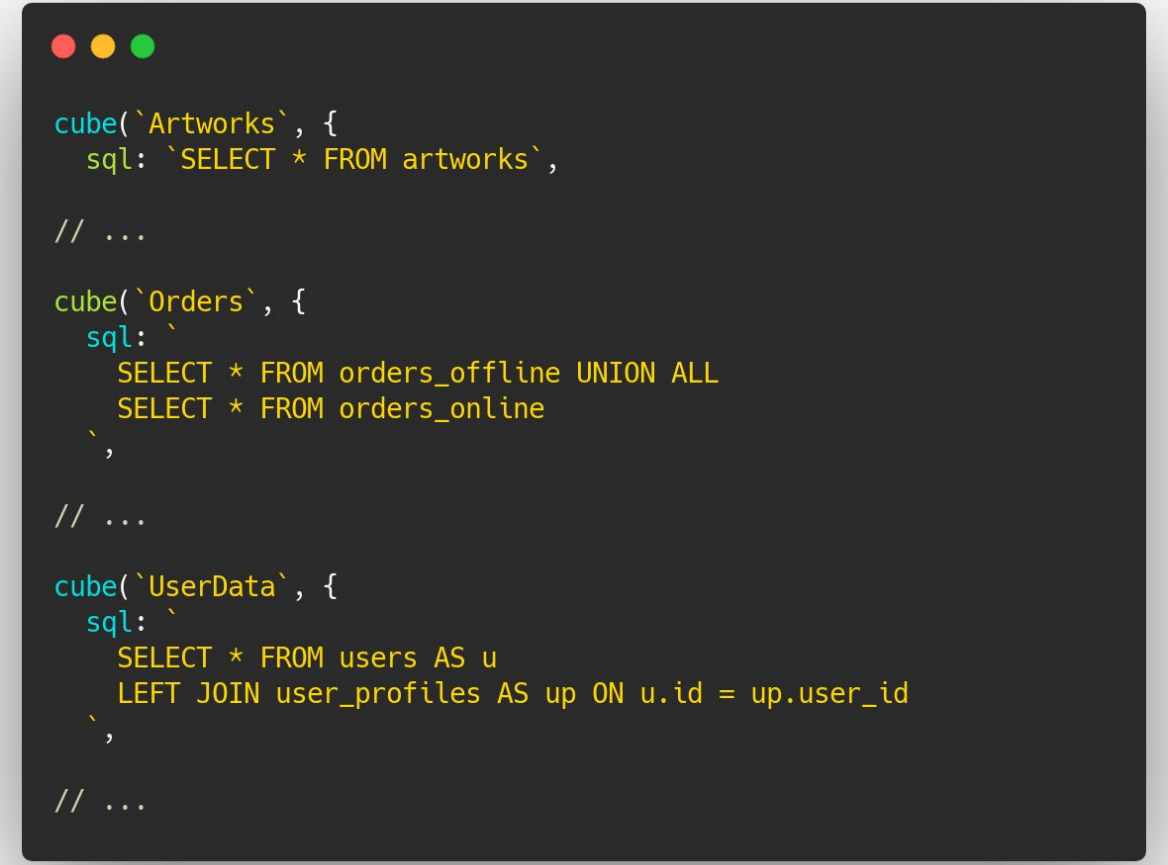
```
{
  measures: [
    "WebHits.count"
  ],
  dimensions: [
    "WebHits.path"
  ],
  filters: [
    {
      "member":   "WebHits.path",
      "operator": "equals",
      "values": [
        "/docs/getting-started/docker",
        "/docs/getting-started/nodejs"
      ]
    }
  ],
  timeDimensions: [
    {
      dimension: "WebHits.timestamp",
    }
  ]
}
```

Cubes vs. tables

- Often, tables reflect how your data is collected and stored
- Cubes reflect your business domain

Which one is a true statement?

- It’s okay if a cube is defined over a single table
- It’s okay if a cube is defined over a UNION of tables
- It’s okay if a cube is defined over a number of JOINed of tables
- It’s okay if a cube is defined over an arbitrary and complex SQL expression



```
cube(`Artworks`, {
  sql: `SELECT * FROM artworks`,

// ...

cube(`Orders`, {
  sql: `
    SELECT * FROM orders_offline UNION ALL
    SELECT * FROM orders_online
  `,

// ...

cube(`UserData`, {
  sql: `
    SELECT * FROM users AS u
    LEFT JOIN user_profiles AS up ON u.id = up.user_id
  `,

// ...
```

```
cube(`PullRequests`, {
  sql: `
    SELECT
      id,
      base.repo.full_name AS repo_name,
      html_url AS url,
      CAST(FROM_ISO8601_TIMESTAMP(created_at) AS TIMESTAMP) AS created_at,
      CAST(FROM_ISO8601_TIMESTAMP(updated_at) AS TIMESTAMP) AS updated_at,
      CAST(FROM_ISO8601_TIMESTAMP(closed_at) AS TIMESTAMP) AS closed_at,
      CAST(FROM_ISO8601_TIMESTAMP(merged_at) AS TIMESTAMP) AS merged_at,
      user.login AS user_login,
      COALESCE(state, 'Unknown') AS state,
      IF(merged, '1', '0') AS is_merged,
      CARDINALITY(labels) AS label_count,
      COALESCE(CAST(commits AS INTEGER), 0) AS commit_count,
      COALESCE(CAST(additions AS INTEGER), 0) AS addition_count,
      COALESCE(CAST(deletions AS INTEGER), 0) AS deletion_count,
      COALESCE(CAST(changed_files AS INTEGER), 0) AS changed_file_count,
      CASE WHEN user.login = base.repo.owner.login THEN '1' ELSE '0' END AS is_to_own_repo
    FROM cubedev_examples_hacktoberfest.api_accepted
  `,

// ...
```

Common table expressions (CTE)

- Really great way to simplify SQL expressions in your cubes

```
cube(`SlackUsers`, {
  sql: `
    WITH usersFirstMessages AS (
      SELECT user, MIN(ts) as joined_at
      FROM cubejs_slack.messages
      WHERE type = "message"
      AND subtype = "channel_join"
      GROUP BY 1
    ),

    users AS (
      SELECT * FROM cubejs_slack.users
      WHERE is_bot = false
      AND deleted = false
      AND users.id NOT IN (${Object.values(employeesOnSlack).map(x => `'${x}'`)})
    )

    SELECT users.*, usersFirstMessages.joined_at
    FROM users, usersFirstMessages
    WHERE users.id = usersFirstMessages.user
  `

// ...
```

Constants and helper functions (JavaScript)

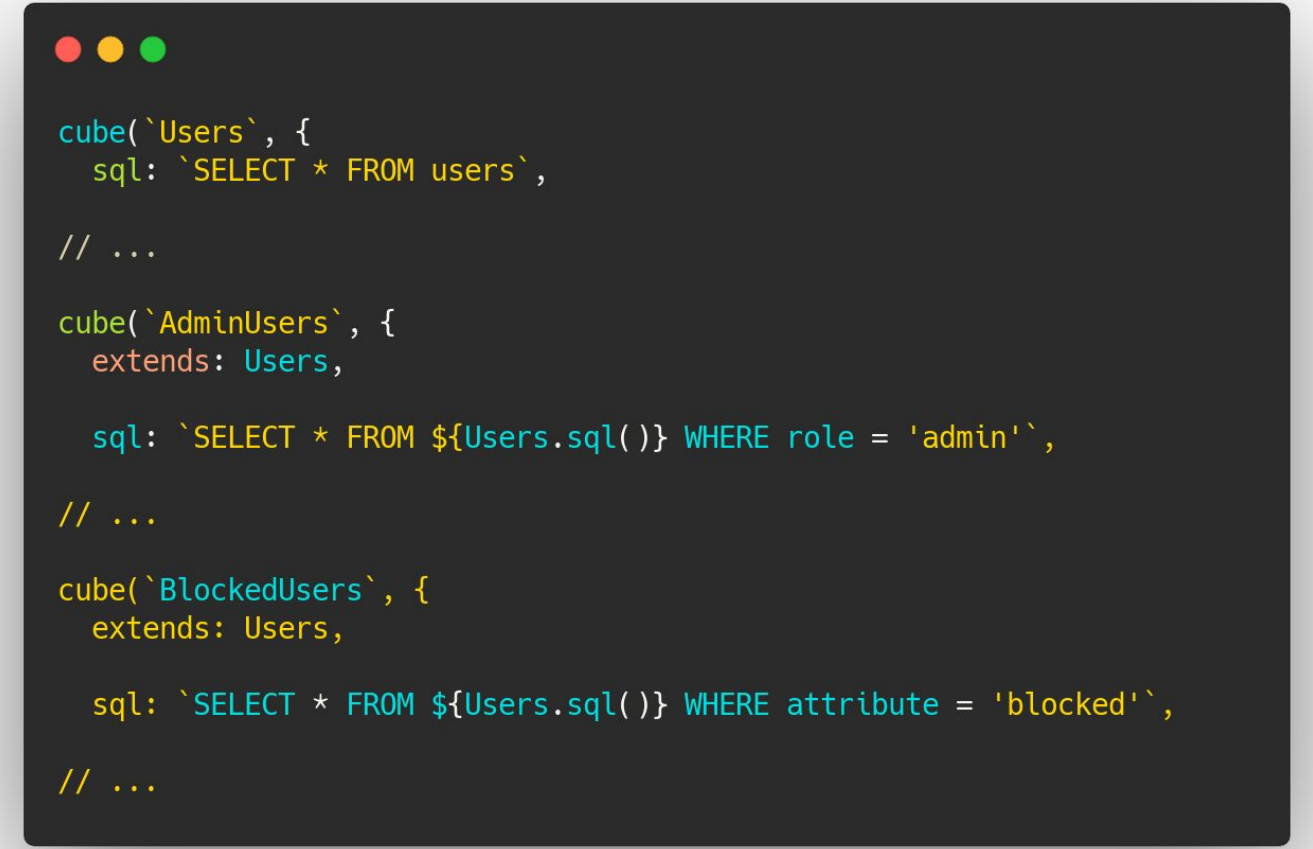
- Really great way to simplify your cubes and reuse code

```
/* commmon.js */

export const CUBE_DEV_EMPLOYEES = [
  {
    name: 'Pavel Tiunov',
    slackId: 'UC19HAUAK',
    githubId: 'paveltiunov',
  },
  {
    name: 'Artyom Keydunov',
    slackId: 'UC1HAFW4E',
    githubId: 'keydunov',
  }
];

/* Employees.js */

import { CUBE_DEV_EMPLOYEES } from './common';

cube('CubeDevEmployees', {
  sql: `
    WITH cubedev_employees AS (
      ${CUBE_DEV_EMPLOYEES.map(({ name, githubId, slackId }) =>
        `(SELECT "${name}" AS name, "${githubId}" AS github_id, "${slackId}" AS slack_id)`
      ).join('UNION ALL')}
    )

    SELECT * FROM cubedev_employees
  `,

// ...
```

sql(), extend, and polymorphic cubes

- Really great way to simplify your data schema and reuse code

```
/* Contributions.js */

cube(`Contributions`, {
  sql: `
    SELECT CAST(id as STRING) AS id, url, created_at as timestamp, user.id as user_id, "issue" as type
    FROM \`cubejs-analytics.cubejs_github.issues\`

    UNION ALL

    SELECT id, url, created_at as timestamp, user.id as user_id, "pull_request" as type
    FROM \`cubejs-analytics.cubejs_github.pull_requests\`
  `,

// ...

/* Contributors.js */

cube(`Contributors`, {
  sql: `
    SELECT DISTINCT(user_id) AS id
    FROM ${Contributions.sql()}
  `,

// ...
```



```
cube(`Users`, {
  sql: `SELECT * FROM users`,

// ...

cube(`AdminUsers`, {
  extends: Users,

  sql: `SELECT * FROM ${Users.sql()} WHERE role = 'admin'`,

// ...

cube(`BlockedUsers`, {
  extends: Users,

  sql: `SELECT * FROM ${Users.sql()} WHERE attribute = 'blocked'`,

// ...
```

Data schema compilation and versioning

Data schema compilation and versioning

- The data schema is not used by Cube “as is”
- When the first query comes, **the data schema is compiled into an internal representation and cached for performance**

Data schema versioning

- schemaVersion extension point
- When the returned value changes, Cube recompiles the data schema

Demo: schemaVersion

Data schema compilation

- SQL expressions are “compiled” from strings into functions
- If you’re extracting portions of a cube’s code and putting them outside the cube function, you should “compile” them yourself

Demo: compilation

Dynamic data schema

Dynamic vs. static data schema

- Often, the data schema is self-contained. However,
  a dynamic data schema **depends on an external data source**
- asyncModule service function
- The “dynamic” parts of the data schema would need to be “compiled”
  by yourself because they are defined outside the cube function

Demo: asyncModule

Joins

Joins in Cube

- How you define relationships between cubes (like tables, in SQL)
- `joins:{}` block within a cube are where joins with other cubes are defined
- All joins are `LEFT JOIN` (so whichever cube has the join defined is effectively the main table in that join relationship)
- Joins can have the following relationships:
  - `hasOne`
  - `hasMany`
  - `belongsTo`
- Every join statement must include at least one primary key
  - Primary key is the dimension set with `primaryKey: true` (only one can exist per cube!)
- Transitive property applies to multiple joins
  - e.g. `A` can join with `C` if joins exist between `A` and `B` and between `B` and `C`

Where to join data

In the base table — when you add a join in the main `sql` parameter of the Cube

vs.

Joins between cubes — joins in the `joins:{}` block

So which is better?

- Joins across cubes are more flexible when the relevant data is useful for more queries than just those using this particular join; otherwise it may be easier to add joins in the base table
- Consider your end-user: would they expect a separate cube or not?
- You can join pre-aggregations, too, on joins between cubes (`type: rollupJoin`)

Join direction

Let’s say you have `Orders` and `Customers` cubes. Which join relationship is better?

- Orders `belongsTo` Customers
- Customers `hasMany` Orders

Note: You should not define both relationships on the same cubes (effectively a bi-directional join), you should instead extend one of the cubes or otherwise use a new cube to add a join for the reverse relationship of the other join

Demo: switching join direction

Many-to-many joins

When multiple records are associated with multiple records (e.g. if we had multiple `Products` per `Order` and multiple `Orders` per `Product`)

Requires an associative table (or junction table, cross-reference table), either in your source data or as a new cube — examples of both cases are in our docs

Query-dependent data schema

Rule of thumb: Don’t let queries reshape your data in Cube

Need measures (e.g. counts) based on filters in your query?

- Not ideal: Calculating each count on the fly, depending on the filter
- Also not ideal: Adding one count measure per filter option

Why not?

- Prone to errors, unpredictable results
- Not easy to cover these cases with pre-aggregations

It’s better to prepare your data to account for the filter dimension, like so.

Multi-tenancy is another use-case of dynamic schema with query dependencies

We covered multi-tenancy using `queryRewrite` and `COMPILE_CONTEXT` in our previous workshop

Resources

Your resources for data modeling in Cube 📖

Please check out the Data Schema section of our documentation for most of what we discussed today:

- https://cube.dev/docs/schema/getting-started

Our recipes show Cube best practices for different use cases:

- https://cube.dev/docs/recipes#recipes-data-schema

Don’t hesitate to ask any questions on Slack.


cube