

Advanced Pre-aggregations Workshop



Adnan Rahic



Igor Lukanin



Ryan Pei

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 for “hands-on” demos.
- 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.
 - Instead of a traditional thank you gift for survey responses, you will have a chance to select a from a list charities that Cube will make a donation to.
 - [Doctors Without Borders](#), [UNHCR](#), [UNICEF](#), and [Save the Children](#)

What we will discuss today

- Recap of the first workshop and product updates
- Structuring, optimizing, and partitioning pre-aggregations
- Understanding pre-aggregation cardinality
- 1st Q&A Session
- Advanced strategies for refreshing pre-aggregations
- Rollup Joins and Multi-tenancy best practices
- 2nd Q&A Session

A quick recap of the first workshop

What is a pre-aggregation?

- Condensed versions of source data
- Partitioned, stored, and refreshed — optimized for efficient queries
- Most common type of pre-aggregation is **rollup**
- “Data condensing” mechanism of rollups == **GROUP BY** function in SQL

How do pre-aggregations work?

Raw Data

Order ID	Order Time	Product	Quantity
1	July 20, 2021 10:00 AM	Sugar Cubes	2
2	July 22, 2021 08:00 AM	Ice Cubes	4
3	July 26, 2021 03:00 PM	Ice Cubes	2
4	July 28, 2021 12:00 AM	Sugar Cubes	1
5	August 3, 2021 12:00 PM	Ice Cubes	4
6	August 5, 2021 09:00 AM	Ice Cubes	1

Row Count

~1M (5 years of data)

⚡ Pre-Aggregated Data

Order Time	Product	Quantity
July	Ice Cubes	6
July	Sugar Cubes	3
August	Ice Cubes	5

Row Count

~120

A recap blog post from the first workshop:
<https://cube.dev/blog/pre-aggregations-workshop-recap/>

Improvements in pre-aggregations since the first workshop

Updates

The screenshot shows a GitHub issue page for an Epic titled "Epic: New rollup pre-aggregations #3106". The repository is "cube-js / cube.js". The issue is marked as "Closed" and has 3 tasks, 0 comments, and is fixed by pull requests #3326, #3307, #3261, and #3165. The issue description, posted by user rpaik on July 13, 2021, aims to facilitate defining new rollup pre-aggregations and includes a checklist: "Iterate on the rollup designer", "Improve error messages", and "Contribute to increased adoption of pre-aggregations by Cube.js users". The issue is labeled with "Epic" and "Roadmap: 2021 Q3". The right sidebar shows the issue is assigned to no one, has no open projects, and is linked to a development commit: "feat: Mixed rolling window and regular measure ...".

Search or jump to... Pull requests Issues Marketplace Explore

cube-js / cube.js Watch Fork Starred 12.6k

<> Code Issues 359 Pull requests 266 Actions Projects 1 Settings Releases 657

Epic: New rollup pre-aggregations #3106

Edit New issue

Closed 3 tasks 0 comments · Fixed by #3326, #3307, #3261 or #3165

rpaik on Jul 13, 2021 Member

Facilitate defining new rollup pre-aggregations by working on:

- Iterate on the rollup designer.
- Improve error messages.
- Contribute to increased adoption of pre-aggregations by Cube.js users.

1 👍

rpaik added **Epic** **Roadmap: 2021 Q3** labels on Jul 13, 2021

rpaik added this to Epics in **Roadmap 2021 Q3** on Jul 13, 2021

rpaik changed the title **Epic: New rollup pre-aggregations** Epic: New rollup pre-aggregations on Jul 13, 2021

Assignees – assign yourself

Labels **Epic** **Roadmap: 2021 Q3**

Projects No open projects 1 closed project

Milestone

Development **feat: Mixed rolling window and regular measure ...** cube-js/cube.js **docs: Update using pre-aggs -> rollup only mode** cube-js/cube.js **feat(@cubejs-client/playground): rollup designe...** cube-js/cube.js

On the roadmap for the coming month

The screenshot shows a GitHub issue page for the repository 'cube-js / cube.js'. The issue title is 'Epic: Improve rollupJoins in pre-aggregations #3530'. It is categorized as an 'Epic' and has '2 tasks' and '0 comments'. The issue was created by user 'rpaik' on October 9, 2021. The description includes a list of work items, such as enabling rollupJoins across multiple data sources and improving error messages. The issue is labeled with 'Epic', 'Roadmap: 2021 Q4', and 'Roadmap: 2022 Q1'. The right sidebar shows the issue's metadata, including assignees, labels, projects, milestones, and development settings.

Search or jump to... Pull requests Issues Marketplace Explore

cube-js / cube.js Watch Fork Starred 12.6k

Code Issues 359 Pull requests 266 Actions Projects 1 Settings Releases 657

Epic: Improve rollupJoins in pre-aggregations #3530

Edit New Issue

Open 2 tasks 0 comments

rpaik on Oct 9, 2021

Continue with rollupJoin improvements (e.g. multiple sources).

Work items include:

- Enable rollupJoins across multiple data sources (e.g. Mongo DB & Snowflake): [rollupJoins on multiple DBs of different types #3432](#)
- Make rollupJoins easier to use with better error messages and improving documentation

rpaik added **Epic** **Roadmap: 2021 Q4** labels on Oct 9, 2021

rpaik added this to Epics in **Roadmap 2021 Q4** on Oct 9, 2021

rpaik added the **Roadmap: 2022 Q1** label on Jan 10

Assignees – assign yourself

Labels

- Epic**
- Roadmap: 2021 Q4**
- Roadmap: 2022 Q1**

Projects

- Roadmap 2022 Q1**
Epics

Milestone

Development – Create a branch

Notifications Customize

Cube Cloud – more insight into pre-aggregations

The screenshot shows the Cube Cloud interface for the 'cubejs-analytics' deployment. The left sidebar contains navigation items: Overview, Playground, Schema, Queries, Pre-Aggregations (highlighted), Metrics, and Settings. The main content area is titled 'WebEvents' and includes a 'Refresh' section with settings: 'Every 4 hour', 'Incremental refresh update window: last partition', and 'Automated refreshes'. Below this is a 'Partitions' table with columns for Definition, Preview, Used By, and Indexes. The table lists three partitions: 'March 2022', 'February 2022', and 'January 2022', each with a status icon and a 'Last started at' timestamp.

cube cloud Deployments / cubejs-analytics

master </> Enter Development Mode

Overview
Playground
Schema
Queries
Pre-Aggregations
Metrics
Settings

WebEvents

main

Refresh

Every 4 hour

- ✓ Incremental refresh update window: last partition
- ✓ Automated refreshes

Partitions Definition Preview Used By Indexes

Build range: Jan 1, 2019 - Mar 31, 2022

Build range start: `SELECT TIMESTAMP('2019-01-01')`

Build range end: `SELECT CURRENT_TIMESTAMP()`

	Last started at	Duration	Partition size
<input type="checkbox"/> ✓ March 2022	Mar 27 at 21:01:41	11.1s	1.67KB
<input type="checkbox"/> ✓ February 2022	> 3 days ago	-	-
<input type="checkbox"/> ✓ January 2022	> 3 days ago	-	-

The 'Add alert' dialog box is shown, allowing configuration of alerts for the 'WebEvents' pre-aggregation. It includes options for 'Event type', 'For deployments', and 'Send alerts to'.

Add alert

Event type

- API outages
- API/Database response timeouts
- Pre-aggregation build failures

For deployments

- All
- Specific

Send alerts to

- All users on this account
- Specific users

Add Cancel

Cube Store

Production checklist

- Cube Store 
- Dedicated Refresh Worker
- Dedicated Redis
- Batching and/or exporting data
- Pre-aggregation build/refreshes may need tuning

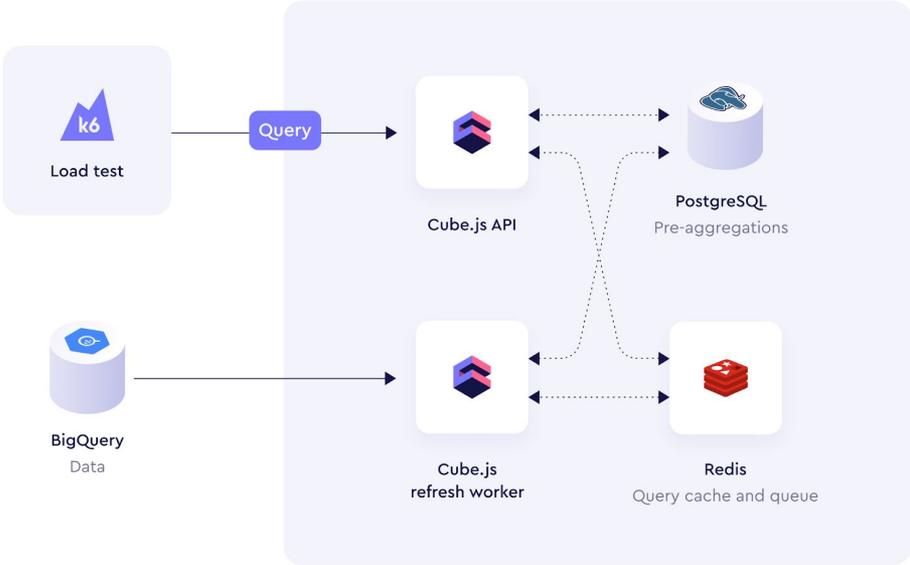
Before Cube Store

- Pre-aggregations stored in traditional databases
- **Pre-aggregations in traditional databases often won't allow for high concurrency and low latency of the analytical API**

Before Cube Store

Pre-aggregations in traditional databases

⚠ Acceptable with exceptions



Cube.js with pre-aggregations in PostgreSQL

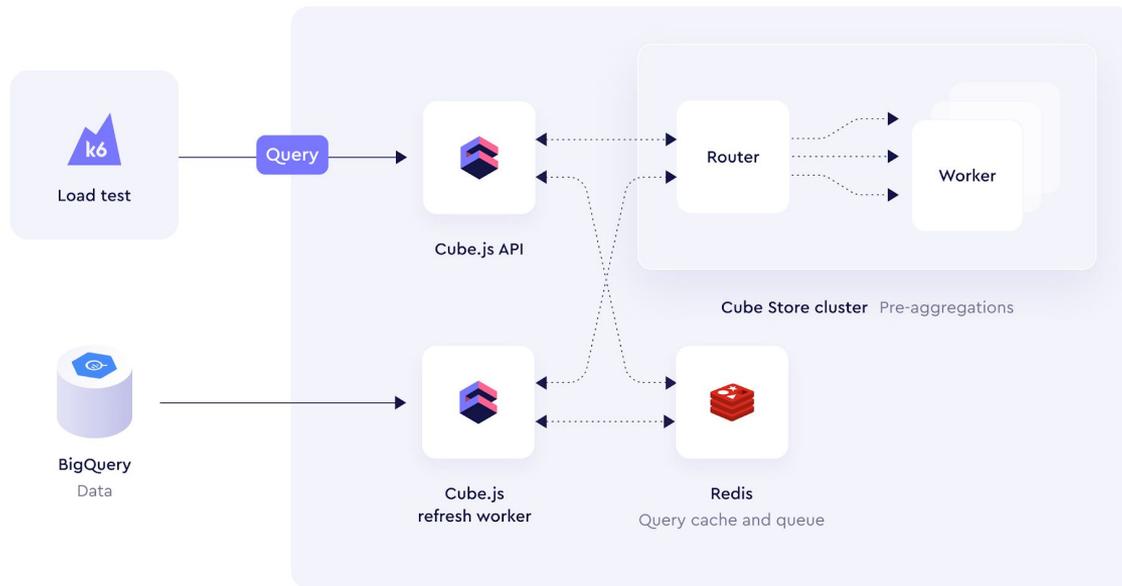
After Cube Store

- Cube Store is designed to resolve these issues and provide a performant pre-aggregation storage layer for Cube

After Cube Store

Pre-aggregations in Cube Store

✓ Recommended for production

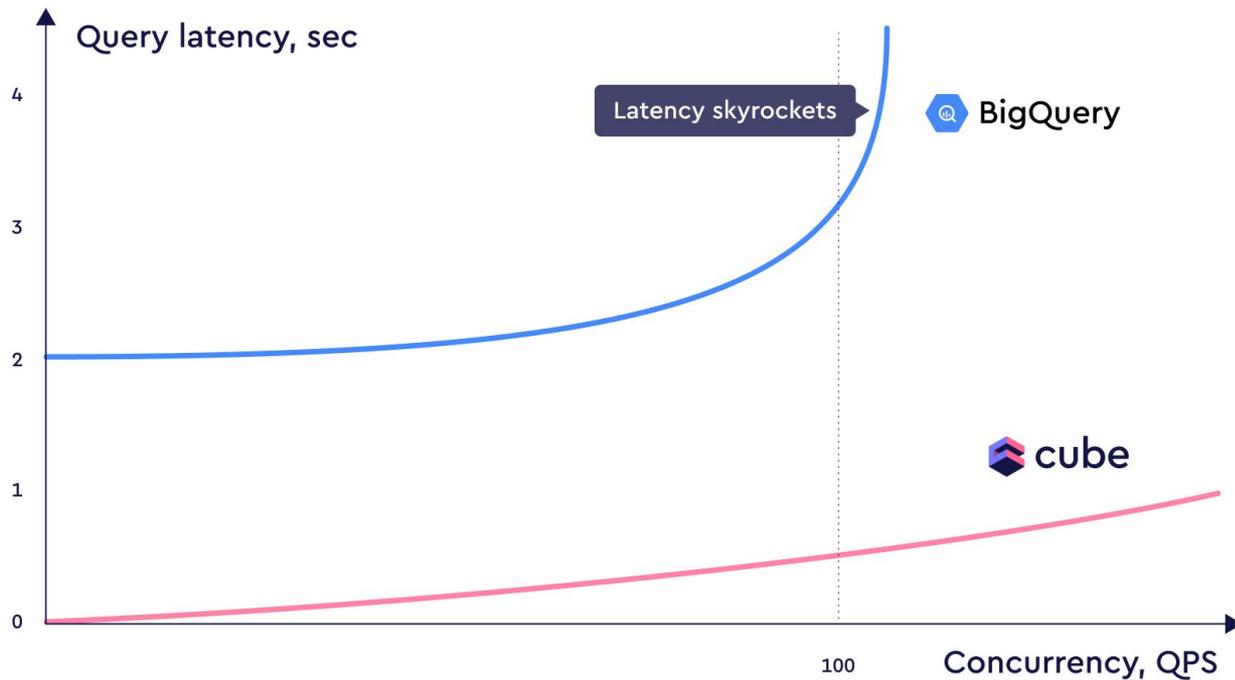


Cube.js with pre-aggregations in Cube Store

What and why Cube Store?

- Significant performance improvement
- Significantly lower latency
- Latency decreased by 5x in this use-case:
 - <https://cube.dev/blog/data-warehouse-performance-and-how-cube-can-help/>

Why Cube Store?



TPC-H BigQuery dataset with >150M entries

Prep Guide

- <https://descriptive-reply-0b7.notion.site/Workshop-Advanced-Pre-aggregations-Prep-Guide-5de1ae39a28849edb68d29f29728bc7c>

Code

- <https://descriptive-reply-0b7.notion.site/Workshop-Advanced-Pre-aggregations-Code-b08909ee5ecd43a1b7de4baa70f8cf28>

Download the GCP service account key

Use this key to access BigQuery

- <https://buff.ly/3qJFI16>

TPC-H x 100 dataset

- >150M entries
- BigQuery schema called `tpc_h` in the `cube-devrel-team` BigQuery instance

Demo: Step 0

BigQuery and TPC-H dataset

Matching Additive vs Non-Additive measures with Pre-aggregations

Additive vs Non-Additive measures

- Non-additive measures are average values or distinct counts
- Pre-aggregations with these measures are less likely to be selected to accelerate a query

Non-additive measures

```
totalPriceAvg: {  
  sql: `${CUBE}.O_TOTALPRICE`,  
  type: `avg`  
},
```

```
clerkCountDistinct: {  
  sql: `${CUBE}.O_CLERK`,  
  type: `countDistinct`  
},
```

Matching non-additive measures with pre-aggregations

```
totalPriceAvgClerkCountDist: {  
  measures: [  
    Order.totalPriceAvg,  
    Order.clerkCountDistinct  
  ],  
  dimensions: [  
    // Will NOT match without this dimension  
    Order.oOrderpriority  
  ]  
},
```

Turning non-additive into additive measures

```
totalPriceAvg: {  
  sql: `${CUBE.totalPriceSum} / ${CUBE.count}`,  
  type: `number`,  
},  
totalPriceSum: {  
  sql: `${CUBE}.O_TOTALPRICE`,  
  type: `sum`,  
},  
clerkCountDistinct: {  
  sql: `${CUBE}.O_CLERK`,  
  type: `countDistinctApprox`,  
},
```

Matching additive measures

```
totalPriceAvgClerkCountDist: {  
  measures: [  
    Order.totalPriceSum,  
    Order.count,  
    Order.clerkCountDistinct  
  ],  
  dimensions: [  
    // Will match even without this dimension  
    Order.o0rderpriority  
  ]  
}
```

Demo: Step 1

Matching non-additive and additive measures with pre-aggregations

Providing multiple pre-aggregations to solve issues with additive vs non-additive measures

Providing multiple pre-aggregations

- Dedicated pre-aggregations
- Definitions that fully match queries with non-additive measures
- Performance boost but longer build time and space consumed

Providing multiple pre-aggregations

```
/** With a dimension */
totalPriceAvgClerkCountDist1: {
  measures: [
    Order.totalPriceAvg,
    Order.clerkCountDistinct
  ],
  dimensions: [
    Order.oOrderpriority
  ]
},
/** Without a dimension */
totalPriceAvgClerkCountDist2: {
  measures: [
    Order.totalPriceAvg,
    Order.clerkCountDistinct
  ]
},
```

Demo: Step 2

Providing multiple pre-aggregations to solve issues with additive vs non-additive measures

Pre-aggregations with large queries that have joins

Pre-aggregations with large queries and joins

- Pre-aggregations across joined cubes
- Produces large result sets
- Produces large pre-aggregations

Let's run a large query

```
{
  "measures": [
    "Order.count",
    "Order.totalPriceAvg",
    "Order.totalPriceSum"
  ],
  "timeDimensions": [
    {
      "dimension": "Order.oOrderdate",
      "granularity": "day",
      "dateRange": [
        "1998-08-01",
        "1998-08-02"
      ]
    }
  ],
  "order": {
    "Order.count": "desc"
  },
  "dimensions": [
    "Customer.cName"
  ],
  "limit": 10
}
```

Pre-aggregation definition

```
dailyOrdersPerCustomer: {  
  measures: [  
    Order.count,  
    Order.totalPriceSum  
  ],  
  dimensions: [  
    Customer.cName  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`  
}
```

Demo: Step 3

Pre-aggregations with large queries that have joins

Demo: Step 3

- Add a join and a few dimensions first
- Add pre-aggregations
- Once the pre-aggregations start building we will see how long it will take due to the large dataset and no implemented partitions and optimization
- You'll see the warning:
The pre-aggregation "Order.dailyOrdersPerCustomer" has more than 100000 rows. Please consider using an export bucket. Learn more.

How to structure and optimize pre-aggregations

Partitioning

- `partitionGranularity` — Cube will generate separate tables for each partition of data
 - `day, month, etc...`
- In the demo we'll use:

```
partitionGranularity: `day`
```

What about incremental refresh?

- Incrementally refresh partitioned — `incremental: true`
- Defaults to `false`
- Building partitioned tables separately – slower than building one table
- Set `incremental: true` to refresh the last partition only
- Set `updateWindow: `7 day`` to refresh partitions where the end date lies within the `updateWindow` from the current time

Only refresh partitions within the last 7 days

```
refreshKey: {  
  every: `1 hour`,  
  incremental: true,  
  updateWindow: `7 day`,  
},
```

Pre-aggregation definition

```
dailyOrdersPerCustomer: {  
  measures: [  
    Order.count,  
    Order.totalPriceSum  
  ],  
  dimensions: [  
    Customer.cName  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
}
```

Build Strategies

- Simple
- Batching
- Export Bucket

Enable export bucket

```
CUBEJS_DB_EXPORT_BUCKET=cube_devrel_team_tpch  
CUBEJS_DB_EXPORT_BUCKET_TYPE=gcp
```

Demo: Step 4

How to structure and optimize pre-aggregations

Demo: Step 4

- Add the export bucket env vars
- Add a pre-aggregation with `partitionGranularity`
- Add incremental updates with `incremental: true`
- Build a few partitions to see how quick the builds are



Dedicated pre-aggregations for large queries

Dedicated pre-aggregations for large queries

- Multiple queries can match one larger pre-aggregation
- Split pre-aggregations based on measures and dimensions
- Make sure queries exactly match pre-aggregations

Dedicated pre-aggregations for large queries

- Individual size is smaller and the individual build time is shorter with two dedicated pre-aggregations
- Response times are faster with two dedicated pre-aggregations
- Single larger pre-aggregation is in total smaller than the two smaller combined
- Response times are slower with a single larger pre-aggregation

Let's run a large query

```
{
  "measures": [
    "Order.count",
    "Order.totalPriceAvg",
    "Order.totalPriceSum"
  ],
  "timeDimensions": [
    {
      "dimension": "Order.oOrderdate",
      "granularity": "day",
      "dateRange": [
        "1998-08-01",
        "1998-08-02"
      ]
    }
  ],
  "order": {
    "Order.count": "desc"
  },
  "dimensions": [
    "Customer.cName",
    "Customer.cAcctbal"
  ],
  "limit": 10
}
```

Pre-aggregation definition

```
dailyOrdersPerCustomer: {  
  measures: [  
    Order.count,  
    Order.totalPriceSum  
  ],  
  dimensions: [  
    Customer.cName,  
    Customer.cAcctbal  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
},
```

Let's run a smaller query

```
{
  "measures": [
    "Order.count"
  ],
  "timeDimensions": [
    {
      "dimension": "Order.oOrderdate",
      "granularity": "day",
      "dateRange": [
        "1998-08-01",
        "1998-08-02"
      ]
    }
  ],
  "order": {
    "Order.count": "desc"
  },
  "dimensions": [
    "Customer.cName"
  ],
  "limit": 10
}
```

Create multiple dedicated pre-aggregations

```
dailyOrderCountPerCustomer: {  
  measures: [  
    Order.count,  
  ],  
  dimensions: [  
    Customer.cName  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
},
```

```
dailyOrderPriceAvgPerCustomer: {  
  measures: [  
    Order.totalPriceAvg  
  ],  
  dimensions: [  
    Customer.cAcctbal  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
},
```

Demo: Step 5

Dedicated pre-aggregations for large queries

Demo: Step 5

- Add a compound pre-aggregation
- View build time, partition size, and response time
- Split it into two dedicated pre-aggregations
- View improved build time, smaller partition size, and quicker response time

Refresh tuning

Refreshing Pre-aggregations

Trigger refresh with `refreshKey` including:

- `sql`
- `every`
- both `sql` and `every`

Refresh options

- Interval with `every`
- Default value is 1 hour

```
refreshKey: {  
  every: `1 hour`, // This will refresh every hour  
  incremental: true,  
  updateWindow: `7 day`,  
},
```

Refresh options

- CRON syntax with `every`
- Default value is 1 hour

```
refreshKey: {  
  every: `0 * * * *`, // This will refresh every hour  
  incremental: true,  
  updateWindow: `7 day`,  
},
```

Refresh options

- Custom refresh check SQL with `sql` parameter
- Default `every` value is `2 minute` for BigQuery
- `MAX(updated_at_timestamp)` is generally best practice
- `FILTER_PARAMS` to filter values during SQL generation
- Mimic the incremental update with a custom `WHERE` clause

Refresh options

```
refreshKey: {  
  sql: `  
    SELECT  
      MAX(O_UPDATEDAT) FROM tpc_h.order  
    WHERE ${FILTER_PARAMS.Cube.createdAt.filter('O_CREATED_AT')}`  
  ,  
  every: `1 hour` ,  
}
```

Demo: Step 6

Refresh tuning

Demo: Step 6

- Show 3 approaches to using the `refreshKey` parameter

Rollup-joins

Rollup-joins

- Joins between separate pre-aggregations
- Use pre-aggregations instead of running SQL queries

Pre-aggregation with joined cubes

```
dailyOrderCountPerCustomer: {  
  measures: [  
    Order.count,  
  ],  
  dimensions: [  
    Customer.cName  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
},
```

Rollup-join pre-aggregation definition

- First the orders rollup

```
ordersRollup: {  
  measures: [  
    Order.count  
  ],  
  dimensions: [  
    Order.oCustkey  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `day`,  
  partitionGranularity: `day`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `7 day`,  
  }  
},
```

Rollup-join pre-aggregation definition

- Next the customers

```
customersRollup: {  
  dimensions: [  
    Customer.cCustkey,  
    Customer.cName,  
  ],  
}
```

Rollup-join pre-aggregation definition

- Join the rollups

```
ordersCustomersRollupJoin: {
  type: `rollupJoin`,
  measures: [Order.count],
  dimensions: [Customer.cName],
  timeDimension: Order.oOrderdate,
  granularity: `day`,
  partitionGranularity: `day`,
  refreshKey: {
    every: `1 hour`,
    incremental: true,
    updateWindow: `7 day`,
  },
  rollups: [
    Customer.customersRollup,
    Order.ordersRollup,
  ],
},
```

Run the same query – same response!

```
{
  "measures": [
    "Order.count"
  ],
  "timeDimensions": [
    {
      "dimension": "Order.oOrderdate",
      "granularity": "day",
      "dateRange": [
        "1998-08-01",
        "1998-08-02"
      ]
    }
  ],
  "order": {
    "Order.count": "desc"
  },
  "dimensions": [
    "Customer.cName"
  ],
  "limit": 10
}
```

Demo: Step 7

Rollup-joins

Demo: Step 7

- Transform a pre-aggregation with joined cubes into a rollup-join
- View the built pre-aggregations in the Cube Cloud UI
- Show how the queries are not hitting the database, instead only hitting Cube Store

Multi-tenancy with pre-aggregations

Enforcing tenant-aware filters on all queries

- `queryRewrite`
- Filter queries per tenant
- Tenant will be `region`

Run a query to count orders per customer for a particular region

```
{
  "measures": [
    "Order.count"
  ],
  "timeDimensions": [
    {
      "dimension": "Order.oOrderdate",
      "granularity": "month",
      "dateRange": [
        "1996-01-01",
        "1996-12-31"
      ]
    }
  ],
  "order": {
    "Order.count": "desc"
  },
  "dimensions": [
    "Customer.cName",
    "Region.rName"
  ],
  "limit": 10
}
```

Filter all queries based on the securityContext

```
queryRewrite: (query, { securityContext }) => {  
  // Ensure that the security context has the `rRegionkey` property  
  if (!securityContext.rRegionkey) {  
    throw new Error('No Region Key found in Security Context!');  
  }  
  
  // Apply a filter to all queries. Cube will make sure to join  
  // the `Region` cube to other cubes in a query to apply the filter  
  query.filters.push({  
    member: "Region.rRegionkey",  
    operator: "equals",  
    values: [ securityContext.rRegionkey ]  
  });  
  
  return query;  
},
```

Demo: Step 9 / Part 1

Enforcing tenant-aware filters on all queries

Demo: Step 9 / Part 1

- Filter queries based on the `securityContext`
- Add required joins
- Add `queryRewrite` in the `cube.js` file

Dynamic data schema with pre-aggregations separated by tenant

- Dynamically fetch tenants from the database
- Generate separate names for the tenant's pre-aggregations

Add a pre-aggregation

```
regionCustomerOrderCount: {  
  measures: [  
    Order.count  
  ],  
  dimensions: [  
    Customer.cName,  
    Region.rName,  
    Region.rRegionkey  
  ],  
  timeDimension: Order.oOrderdate,  
  granularity: `month`,  
  partitionGranularity: `month`,  
  refreshKey: {  
    every: `1 hour`,  
    incremental: true,  
    updateWindow: `1 month`,  
  },  
}
```

Fetch tenants from the DB with BigQuery SDK

```
const { BigQuery } = require('@google-cloud/bigquery');
const bigquery = new BigQuery();

async function fetchRegionKeys() {
  const regionsQuery = `
    SELECT DISTINCT R_REGIONKEY
    FROM `cube-devrel-team.tpc_h.region`
  `;

  const options = { query: regionsQuery, location: 'US' };
  const [ job ] = await bigquery.createQueryJob(options);

  const [ rows ] = await job.getQueryResults();
  const regionKeys = rows.map(row => row['R_REGIONKEY']);

  return regionKeys;
};
```

Configure dedicated pre-aggregation schemas

```
contextToAppId: ({ securityContext }) =>
  `CUBEJS_APP_${securityContext.rRegionkey}`,
preAggregationsSchema: ({ securityContext }) =>
  `pre_aggregations_${securityContext.rRegionkey}`,

scheduledRefreshContexts: async () => {
  const rRegionkeys = await fetchRegionKeys();

  function mapSecurityContext() {
    return rRegionkeys.map(rRegionkey => {
      return { securityContext: { rRegionkey } }
    })
  }
  return mapSecurityContext();
}
```

Third-party dependencies in Cube Cloud

In Cube Cloud

- `require(...)` is disabled
- Contact support if you need 3rd party deps
- Can require `@cubejs-backend/*-driver` packages

Workaround with Serverless Functions

- Use a function to fetch tenants from the database with BigQuery SDK

GCP Function to fetch tenants

The screenshot shows the Google Cloud Platform console interface for editing a Cloud Function. The top navigation bar includes the Google Cloud Platform logo, the user's team name 'cube-devrel-team', a search bar, and various utility icons. Below the navigation bar, the breadcrumb 'Cloud Functions' and the function name 'fetchRegionsFun' are visible. The main content area is divided into two tabs: 'Configuration' and 'Code', with 'Code' being the active tab. On the left side of the code editor, there are dropdown menus for 'Runtime' (set to 'Node.js 16') and 'Entry point' (set to 'fetchRegionsFun'). Below these, a 'Source code' dropdown is set to 'Inline Editor'. A file explorer on the left shows files 'index.js' and 'package.json'. The main code editor displays the following JavaScript code:

```
10
11 async function fetchRegionKeys() {
12   const regionsQuery = `
13     SELECT DISTINCT R_REGIONKEY
14     FROM `cube-devrel-team.tpc_h.region`
15   `;
16
17   const options = {
18     query: regionsQuery,
19     // Location must match that of the dataset(s) referenced in the query.
20     location: 'US',
21   };
22
23   // Run the query as a job
24   const [ job ] = await bigquery.createQueryJob(options);
25   console.log(`Job ${ job.id } started.`);
26
27   // Wait for the query to finish
28   const [ rows ] = await job.getQueryResults();
29   const regionKeys = rows.map(row => row['R_REGIONKEY']);
30
31   // Return regionKeys
32   return regionKeys;
33 };
34
35 exports.fetchRegionsFun = async (req, res) => {
36   const regionKeys = await fetchRegionKeys();
37   res.status(200).send(regionKeys);
38 };
39
```

At the bottom of the console, there are three buttons: 'PREVIOUS', 'DEPLOY', and 'CANCEL'.

cube.js config with GCP Function

```
const request = require('./utils/request');
async function fetchRegionKeysGCPFun() {
  const options = {
    host: 'us-central1-cube-devrel-team.cloudfunctions.net',
    path: '/fetchTpchRegions',
    port: '443'
  };
  const regionKeys = await request(options);
  return JSON.parse(regionKeys);
};
```

request.js helper

```
const https = require('https');
function request(options) {
  return new Promise((resolve, reject) => {
    https
      .request(options, function (response) {
        let responseBody = '';
        response.on('data', function (data) {
          responseBody += data;
        });
        response.on('end', function () {
          resolve(responseBody);
        });
        response.on('error', function (e) {
          throw e;
        });
      })
      .end();
  });
}
module.exports = request;
```

Demo: Step 9 / Part 2

Dynamic data schema with pre-aggregations separated by tenant

Demo: Step 9 / Part 2

- Add GCP Function in `cube.js` file
- Add dedicated pre-aggregation builds for each tenant
- Build pre-aggregations based on `securityContext`
- View the separate pre-aggregation tables in GCS

Resources

To learn more about pre-aggregations

- Documentation: <https://cube.dev/docs/caching>
- Blog posts:
 - <https://cube.dev/blog/high-performance-data-analytics-with-cubejs-pre-aggregations/>
 - <https://cube.dev/blog/data-warehouse-performance-and-how-cube-can-help/>
- Community support:
 - Discourse: <https://forum.cube.dev/>
 - Slack: <http://cube-js.slack.com/>



