

# Analytics en tiempo real con Rails

# Eduardo Hernández

Software Engineer @ Salesloft

Co-founder de Calzada Code

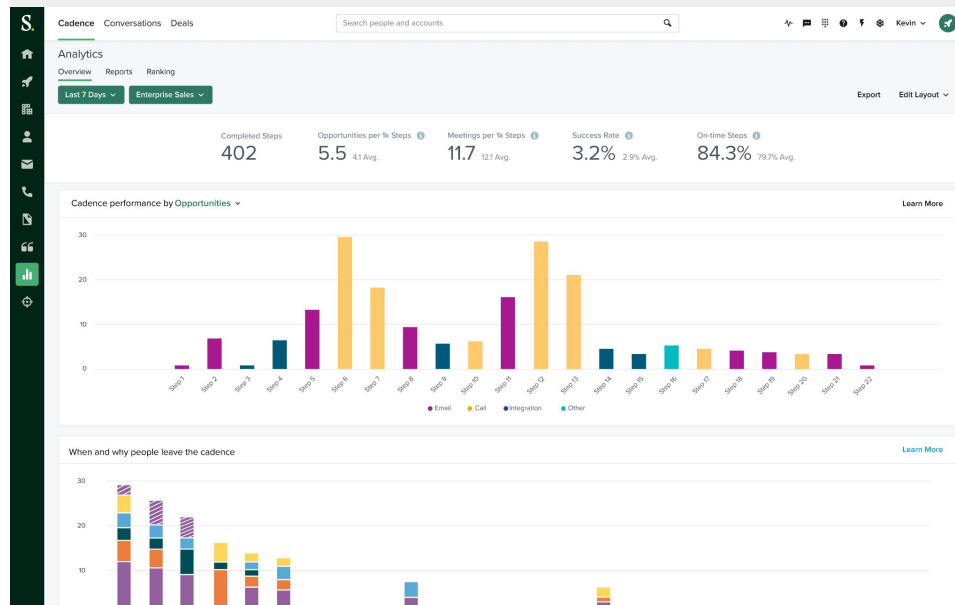
Colaborador en Codeando México

## Salesloft.



# Analytics APIs

Interfaz que facilite el acceso a información resumida



# REST para extraer métricas?

GET      /orders?<query>

¿Cuánto fue el total de las ventas (brutas) esta semana?

GET      /orders/:id

¿Cuánto fue el total de ventas por sucursal?

POST     /orders

PATCH   /orders/:id

¿Cuánto fue el total de impuestos a pagar este mes?

DELETE /orders/:id

# REST para extraer métricas?

GET      /orders?<query>

GET      /orders/:id

POST     /orders

PATCH   /orders/:id

DELETE   /orders/:id

¿Cuánto fue el total de ventas (brutas) esta semana?

¿Cuánto fue el total de ventas por sucursal?

¿Cuánto fue el total de cargos por envío de este mes?

Metrics:

- Ventas Brutas
- Impuestos

Dimensions:

- Sucursales
- Fecha

# Metrics & Dimensions



```
-- Metrics: Gross Sales, Shipping Rate
```

```
-- Dimensions: Date, Branch
```

```
SELECT sum(amount), sum(shipping_rate),
```

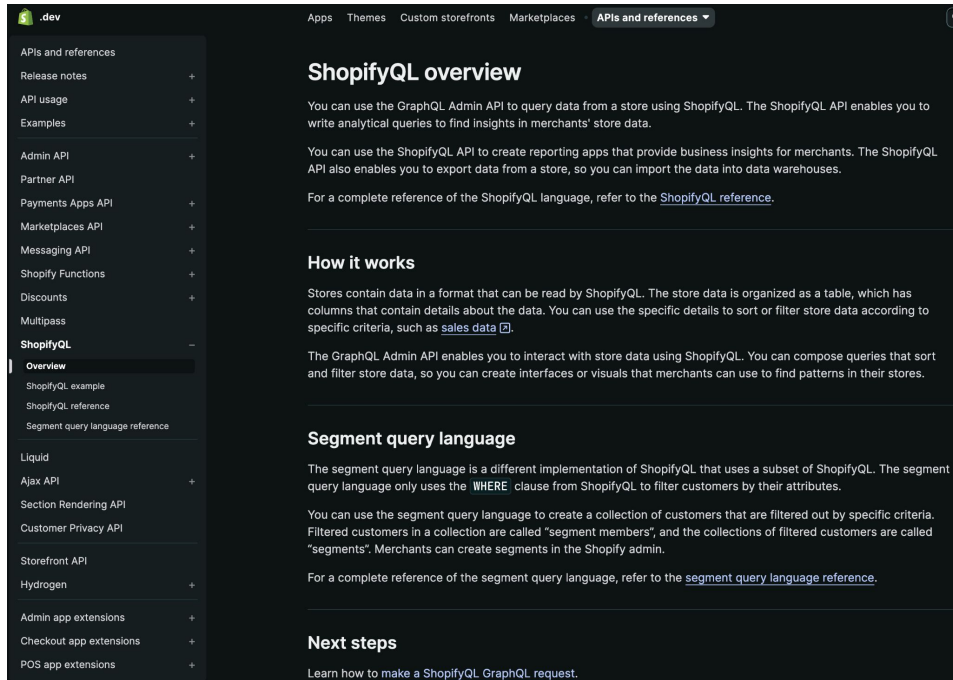
```
FROM orders
```

```
WHERE completed_at > beggining_of_week AND completed_at < end_of_week
```

```
GROUP BY date, branch;
```

# ShopifyQL

> The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.



The screenshot shows the ShopifyQL overview page on the .dev portal. The left sidebar contains a navigation menu with categories like 'APIs and references', 'Admin API', 'Payments Apps API', 'Marketplaces API', 'Messaging API', 'Shopify Functions', 'Discounts', 'Multipass', 'ShopifyQL', 'Liquid', 'Ajax API', 'Section Rendering API', 'Customer Privacy API', 'Storefront API', 'Hydrogen', 'Admin app extensions', 'Checkout app extensions', and 'POS app extensions'. The 'ShopifyQL' category is expanded, showing 'Overview', 'ShopifyQL example', 'ShopifyQL reference', and 'Segment query language reference'. The main content area is titled 'ShopifyQL overview' and contains three sections: 'How it works', 'Segment query language', and 'Next steps'. The 'How it works' section explains that the GraphQL Admin API is used to query data from a store using ShopifyQL. The 'Segment query language' section explains that it is a different implementation of ShopifyQL that uses a subset of ShopifyQL. The 'Next steps' section provides a link to learn how to make a ShopifyQL GraphQL request.

**ShopifyQL overview**

You can use the GraphQL Admin API to query data from a store using ShopifyQL. The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

You can use the ShopifyQL API to create reporting apps that provide business insights for merchants. The ShopifyQL API also enables you to export data from a store, so you can import the data into data warehouses.

For a complete reference of the ShopifyQL language, refer to the [ShopifyQL reference](#).

**How it works**

Stores contain data in a format that can be read by ShopifyQL. The store data is organized as a table, which has columns that contain details about the data. You can use the specific details to sort or filter store data according to specific criteria, such as [sales data](#).

The GraphQL Admin API enables you to interact with store data using ShopifyQL. You can compose queries that sort and filter store data, so you can create interfaces or visuals that merchants can use to find patterns in their stores.

**Segment query language**

The segment query language is a different implementation of ShopifyQL that uses a subset of ShopifyQL. The segment query language only uses the **WHERE** clause from ShopifyQL to filter customers by their attributes.

You can use the segment query language to create a collection of customers that are filtered out by specific criteria. Filtered customers in a collection are called "segment members", and the collections of filtered customers are called "segments". Merchants can create segments in the Shopify admin.

For a complete reference of the segment query language, refer to the [segment query language reference](#).

**Next steps**

Learn how to make a ShopifyQL GraphQL request.

Aggregates are **pre-defined aggregates of numeric values**, to replicate metrics as available throughout Shopify.

Aggregates can be grouped or filtered by any of the **dimensional attributes**.



# ShopifyQL

> The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

```
FROM {table_name}

SHOW {column1, column2, ...}

    AS {alias1, alias2, ...}

VISUALIZE {column1, column2, ...}

    TYPE {visualization_type}

BY {dimension|date}

OVER {date}

WHERE {condition}

SINCE {date_offset}

UNTIL {date_offset}

ORDER BY {column1 ASC|DESC, column2 ASC|DESC, ...}

LIMIT {number}
```

# ShopifyQL

> The ShopifyQL API enables you to write analytical queries to find insights in merchants' store data.

```
FROM sales
SHOW gross_sales
BY branch
SINCE '2022-06-20'
UNTIL '2022-06-25'
```

```
FROM {table_name}

SHOW {column1, column2, ...}

    AS {alias1, alias2, ...}

VISUALIZE {column1, column2, ...}

    TYPE {visualization_type}

BY {dimension|date}

OVER {date}

WHERE {condition}

SINCE {date_offset}

UNTIL {date_offset}

ORDER BY {column1 ASC|DESC, column2 ASC|DESC, ...}

LIMIT {number}
```

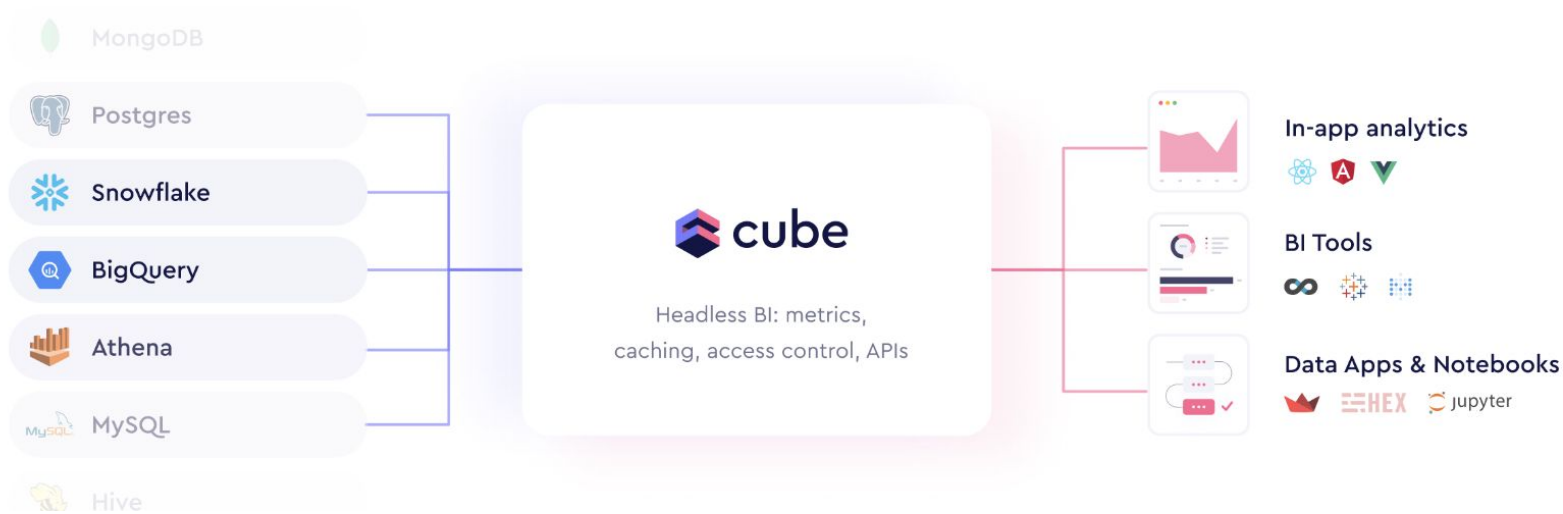
## POST /report

```
{
  "query": {
    "show": ["<aggregates>"],
    "from": ["<source>"],
    "by": ["<dimensions>"],
    "where": {"<dimesion>": "<value>"},
    "order": [{"<column>": "asc"}],
    "since": "<date>",
    "until": "<date>"
  }
}
```



cube.js

**Cube is the headless business intelligence platform.** It helps data engineers and application developers access data from modern data stores, organize it into consistent definitions, and deliver it to every application.



## Real-Time Dashboard

This live demo shows a [real-time data fetch](#) built with Cube, MongoDB, and React.

You can use it to update charts as new data comes in.

Follow the [guide](#) or explore the [source code](#) to learn more.

[Send Button Click Event](#)

Users Online

0

Total Button Clicks

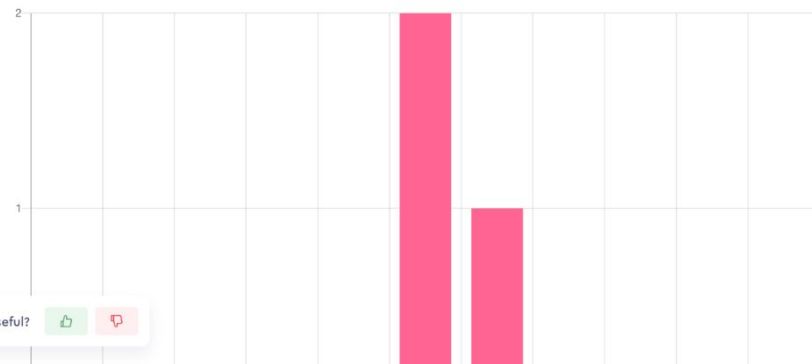
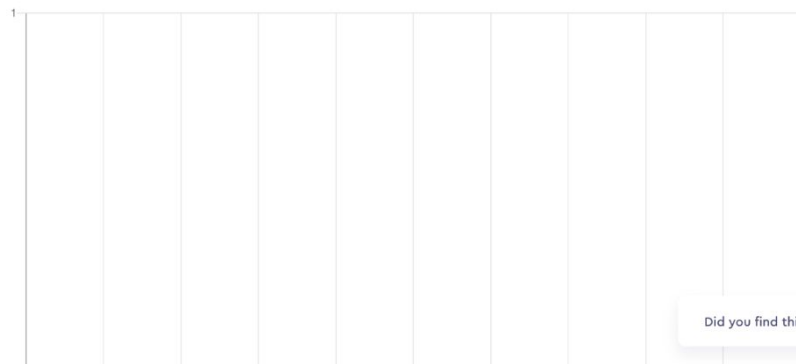
0

Total Page Views

7

Real Time Events

Events per Minute ago



Did you find this example useful?

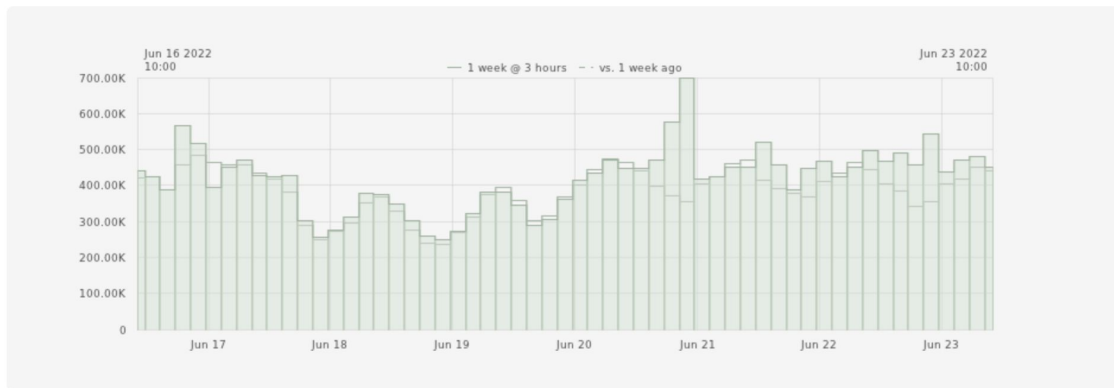


# Real-time analytics with PostgreSQL by Marco Slot



# Real-time analytics with PostgreSQL

- Direct Query to Raw Data
- Materialized Views
- Cache / Cascading aggregates



Open-source developers all over the world are working on millions of projects: writing code & documentation, fixing & submitting bugs, and so forth. GH Archive is a project to **record** the public GitHub timeline, **archive it**, and **make it easily accessible** for further analysis.

GitHub provides [20+ event types](#), which range from new commits and fork events, to opening new tickets, commenting, and adding members to a project. These events are aggregated into hourly archives, which you can access with any HTTP client:

Query	Command
Activity for 1/1/2015 @ 3PM UTC	<code>wget https://data.gharchive.org/2015-01-01-15.json.gz</code>
Activity for 1/1/2015	<code>wget https://data.gharchive.org/2015-01-01-{0..23}.json.gz</code>
Activity for all of January 2015	<code>wget https://data.gharchive.org/2015-01-{01..31}-{0..23}.json.gz</code>

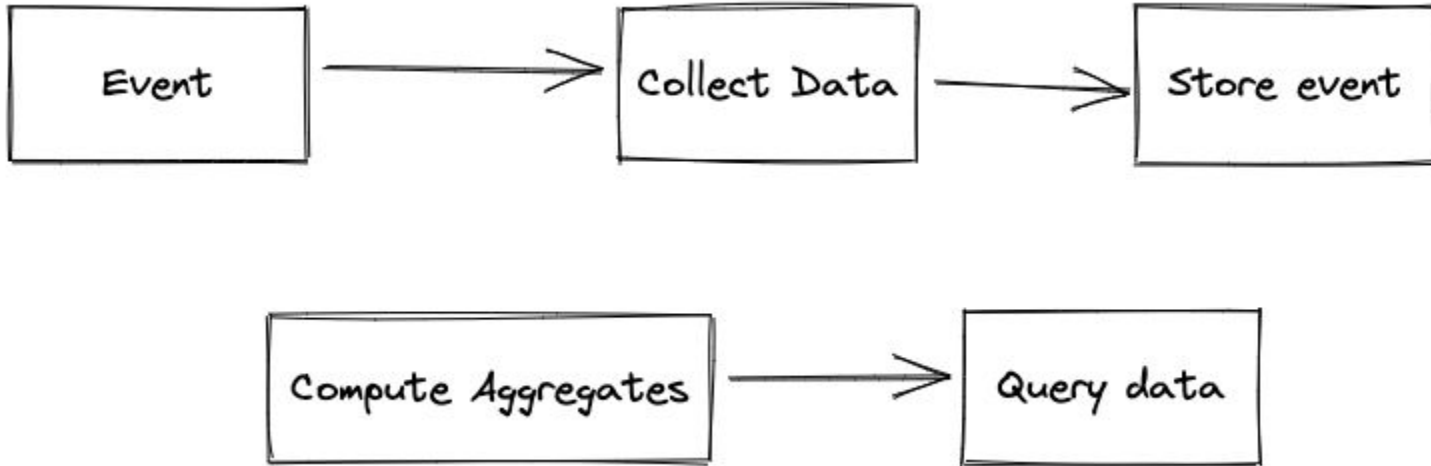


# GitHub::Event

- Raw
- Materialized
- Cascading Aggregates

```
{
  "id": "19543559179",
  "type": "PushEvent",
  "actor": {
    "id": 6274807,
    "login": "wazho",
    "display_login": "wazho",
    "gravatar_id": "",
    "url": "https://api.github.com/users/wazho",
    "avatar_url": "https://avatars.githubusercontent.com/u/6274807?"
  },
  "repo": {
    "id": 324390811,
    "name": "pmgo-professor-willow/data-youtuber",
    "url": "https://api.github.com/repos/pmgo-professor-willow/data-youtuber"
  },
  "payload": {
    "push_id": 8733866699,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/gh-pages",
    "head": "0f3c0f757ad0d42ff2a6a197c095d6d1bdf1e4af",
    "before": "1f7178fbf153b1ff5a22f2f827dd54227925094b",
    "commits": [
      {
        "sha": "0f3c0f757ad0d42ff2a6a197c095d6d1bdf1e4af",
        "author": {
          "email": "githubaction@fake.com",
          "name": "githubaction"
        },
        "message": "deploy to GitHub pages",
        "distinct": true,
        "url": "https://api.github.com/repos/pmgo-professor-willow/data-youtuber/commits/0f3c0f757ad0d42ff2a6a197c095d6d1bdf1e4af"
      }
    ]
  },
  "public": true,
  "created_at": "2022-01-01T12:00:00Z",
  "org": {
    "id": 77191076,
    "login": "pmgo-professor-willow",
    "gravatar_id": "",
    "url": "https://api.github.com/orgs/pmgo-professor-willow",
    "avatar_url": "https://avatars.githubusercontent.com/u/77191076?"
  }
}
```

# Data flow



## Segment Data Quality



Bronze

Raw Ingestion



Silver

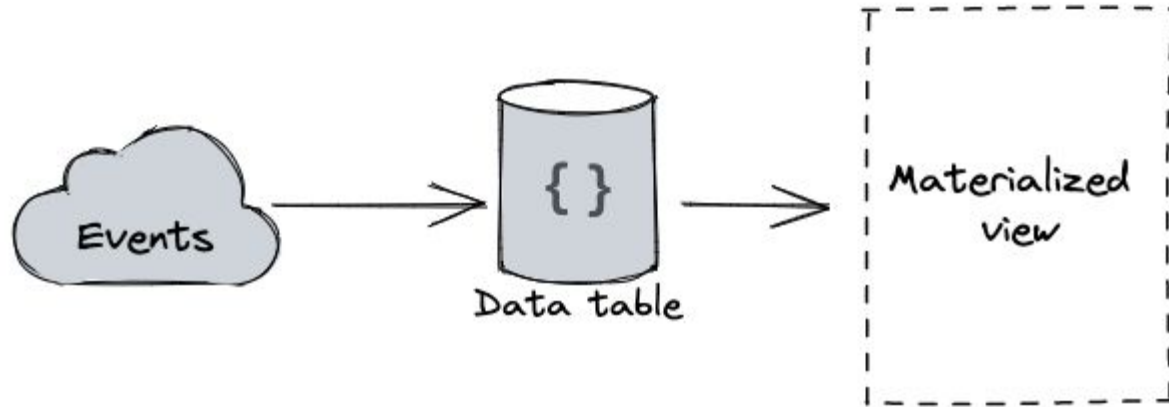
Filtered, cleaned,  
Augmented



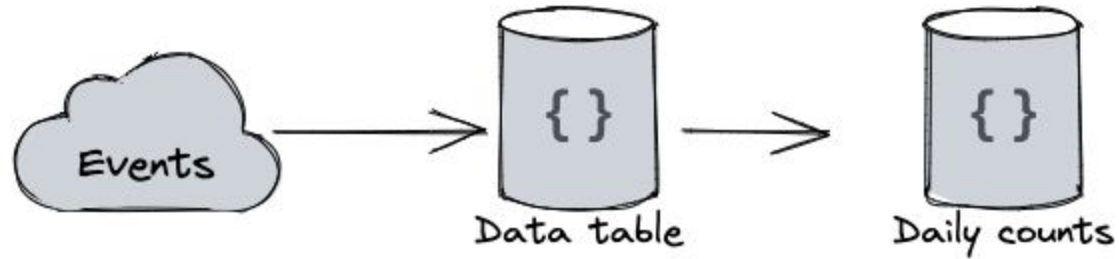
Gold

Business-level  
Aggregates

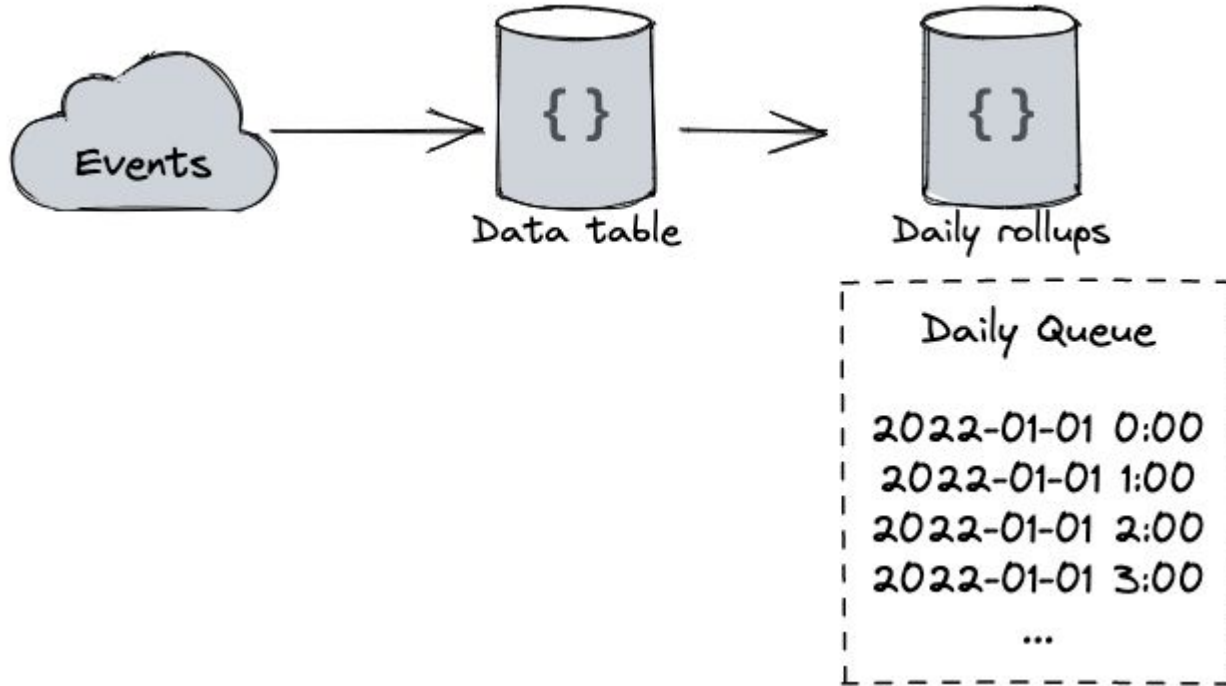
# Daily counts | Materialized views



## Daily counts | Actual DB table



# Cascading aggregates



# Cascading aggregates

```
# GitHub::Event.create(type: 'foo', created_at: '2022-01-01 1:05')
# GitHub::Event.create(type: 'foo', created_at: '2022-01-01 1:10')
# GitHub::Event.create(type: 'bar', created_at: '2022-01-01 2:25')
#
# Queues the following changes in git_hub_events_queues
#
# type | date | count
# -----+-----+-----+
# 'foo' | 2022-01-01 01:00:00 | 1 |
# 'foo' | 2022-01-01 01:00:00 | 1 |
# 'bar' | 2022-01-01 02:00:00 | 1 |
#
# Rollup the following changes in git_hub_events_rollups
#
# type | date | count
# -----+-----+-----+
# 'foo' | 2022-01-01 01:00:00 | 2 |
# 'bar' | 2022-01-01 02:00:00 | 1 |
```

Press ? for help

```

1 class GitHub::Event < ApplicationRecord
2   self.inheritance_column = nil
3 end
4 validates :type, presence: true
5 validates :public, presence: true
6
7 before_create -> do
8   GitHub::Event::HourlyQueue.create!(
9     type: type,
10    count: 1,
11    date: created_at.beginning_of_hour
12  )
13 end
14
15 before_destroy -> do
16   GitHub::Event::HourlyQueue.create!(
17     type: type,
18     count: -1,
19     date: created_at.beginning_of_hour
20   )
21 end
22 end
23

```

app/models/git\_hub/event.rb 3,0-1 All

```

1 class Rollup < GitHub::Event
2   def self.call(hour)
3     events = GitHub::Event::HourlyQueue.where(date: hour)
4
5     events.group(:type).count.each do |type, count|
6       GitHub::Event::HourlyRollup.create!(
7         type: type,
8         date: hour,
9         count: count
10      )
11    end
12
13    events.destroy_all
14  end
15 end

```

app/models/rollup/git\_hub/event.rb 1.1 All

```

1 class GitHub::Event::HourlyQueue < ApplicationRecord
2   self.inheritance_column = nil
3 end

```

app/models/git\_hub/event/hourly.rb 1,1 All

```

1 class GitHub::Event::HourlyRollup < ApplicationRecord
2   self.inheritance_column = nil
3 end

```

app/models/git\_hub/event/hourly\_rollup.rb 1.1 All



# Schedule Hourly rollups

- Clockwork
- Whenever

```
require 'yajl'

hours = 6
progress = ProgressBar.create(
  title: "2022-01-01 at 00:00 GitHub Events Import",
  total: hours
)

hours.times do |hour|
  gz = open("http://data.gharchive.org/2022-01-01-#{hour}.json.gz")
  js = Zlib::GzipReader.new(gz).read

  Yajl::Parser.new.parse(js) do |event|
    GitHub::Event.create(
      event_id: event.fetch('id'),
      type: event.fetch('type'),
      public: event.fetch('public'),
      created_at: event.fetch('created_at')
    )
  end

  time = Time.parse("2022-01-01 #{hour}:00 +0000")
  Rollup::GitHub::Event.call(time)

  progress.increment
end
```

# GitHub::Event::Hourly

Gem: Scenic

> Scenic adds methods to ActiveRecord::Migration to create and manage database views in Rails.

Repo: <https://github.com/scenic-views/scenic>

- Versiones views
- Views in SQL
- Maps to a model
- db/views

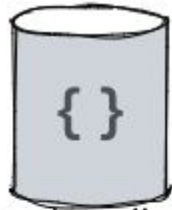
```
class GitHub::Event::Hourly < ApplicationRecord
  # ...
end
```

```
select
  type,
  date,
  sum(count)
from (
  select
    type,
    date,
    count
  from git_hub_event_hourly_rollups

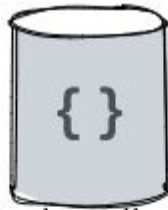
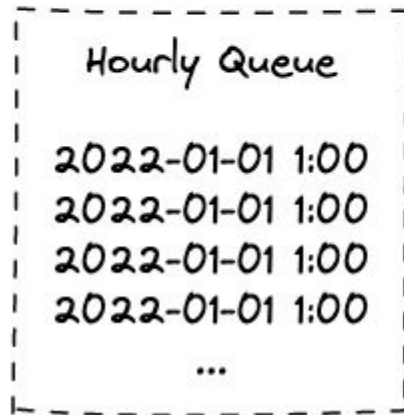
  union all

  select
    type,
    date,
    count
  from git_hub_event_hourly_queues
) git_hub_event_hourly
group by
  type,
  date
```

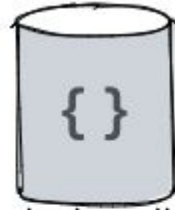
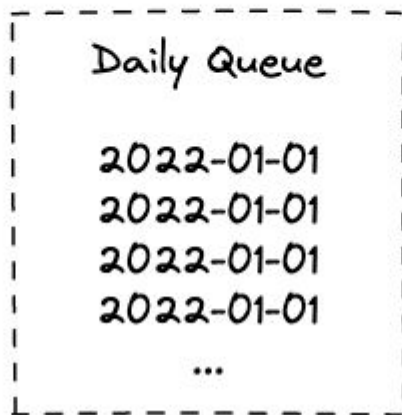
# Cascading Aggregates



Hourly rollups



Daily rollups



Monthly rollups



# Conclusiones

- Battle tested (reponses ~36ms-300ms)
- ¿Es esto realmente real-time?
- Soluciones qué se adapten a tus necesidades
- Mantener a largo plazo este diseño se vuelve manejable con Ruby/Rails
- Versionamiento de código
- Integración continua de la aplicación
- Testeable

# Gracias!

Preguntas? 😁