Ad Monitor Documentation

Rory White

2024-07-30

Table of contents

Intro

This guide explains our methodology for harvesting political ads shown in Canada from the Meta Ad Library and making the data accessible in a MySQL database.

What does each section cover?

Chapter 1 addresses all things API: how we harvested the data.

Chapter 2 explains any judgement calls we made around null values

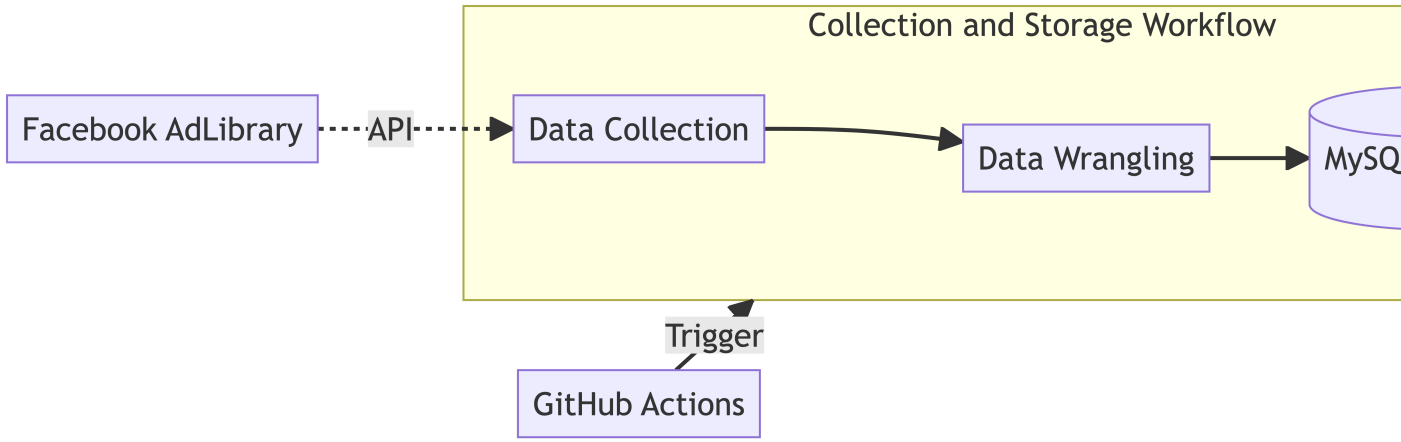
Chapter 3 introduces our MySQL database structure and provides tips for querying it

Chapter 6 contains a variety of useful queries for getting interesting data

Chapter 7 touches on important nuances when interpreting the ad library data

Overview

Below is a diagram of our overall process:



1 Data collection

1.1 Constructing our request

1.1.1 Example request

To request data from Ad Library API when need to construct a request url. Here is an example:

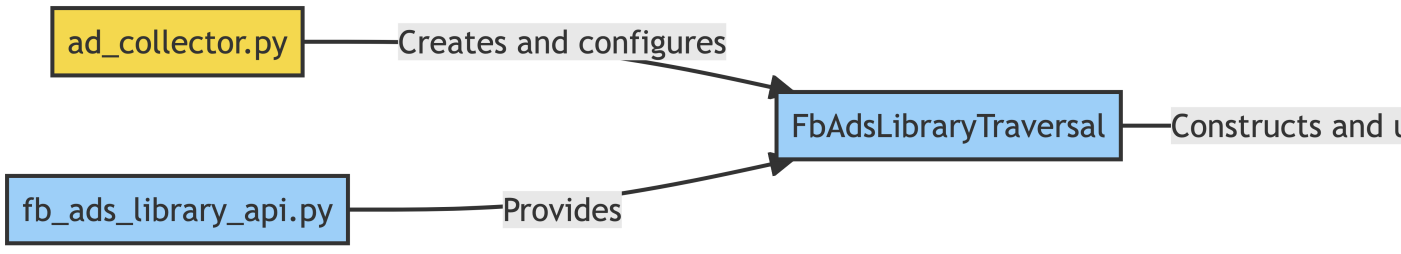
```
https://graph.facebook.com/v20.0/ads_archive?unmask_removed_content=true&ad_type=POLITICAL_A
```

Here are some of the key parameters of our request:

- `limit=500`: request 500 ads at a time.
- `ad_delivery_date_min=2020-05-22`: only ads shown to users since 2020-05-22
- `ad_type=POLITICAL_AND_ISSUE_ADS`: filter for only political ads
- `unmask_removed_content=true`: include ads that broke Facebook's content guidelines
- `ad_active_status=ALL`: ads don't have to be currently active
- `ad_reached_countries=CA`: only ads shown in Canada

1.1.2 How do we do this in our code?

To construct the url dynamically based on the current date we adapted some code provided by Meta.



In `ad_collector.py`, we use a class called `FbAdsLibraryTraversal` from `fb_ads_library_api.py`. We then make an instance of this class with the details we need for our API request.

Here is an example:

```
from fb_ads_library_api import FbAdsLibraryTraversal

collector = FbAdsLibraryTraversal(
        facebook_api_keys,
        "id,ad_creation_time,ad_creative_bodies,ad_creative_link_captions,ad_creative_link_d
        ".",
        "CA",
        ad_delivery_date_min=start_date,
        api_version="v20.0"
    )
```

Once our collector is set up, we make the API call using `collector.generate_ad_archives()`. This returns a list of dictionaries - one for each ad. The ads are returned in batches (of 500) and we wrangle and extract the key details for submission to the database.

1.2 Avoiding API errors

1.2.1 Too much data requested

Some quarto are exceptionally long (~90,000 characters!). This creates inconsistency around how many ads you can safely request at once. To solve this we implement the following sliding window strategy:

Step 1: Start by collecting large number of ads (500)

Step 2: If we encounter "Request less data" error, halve request size

Step 3: Repeat step 2 recursively

Step 4: If we successfully collect all 500, return to Step 1. Otherwise, request problematic ad without body text.

1.2.2 Too many requests made

We use multiple API keys on rotation by creating multiple Meta 'apps'. This means that if one key reaches the limit `total_time=100` then we can switch to a different key.

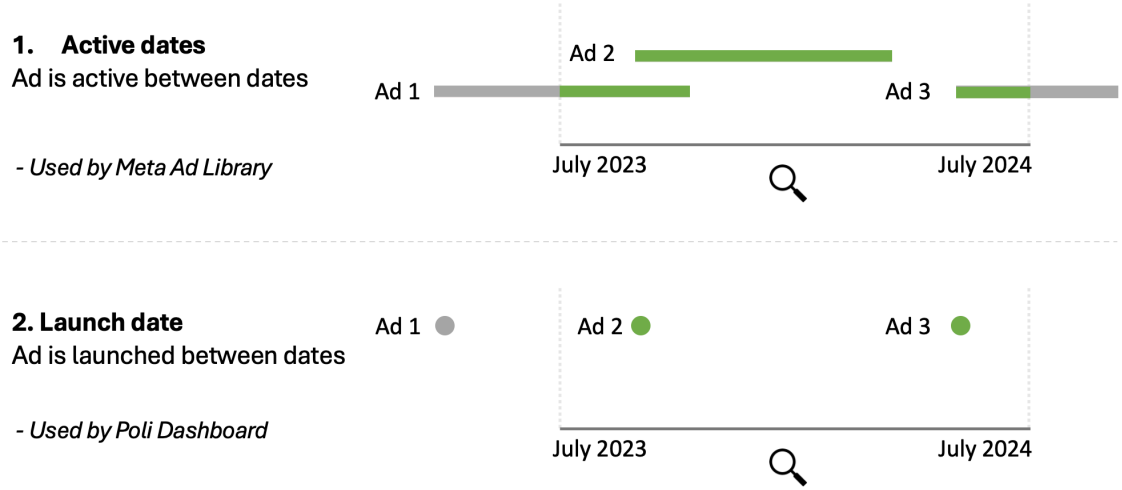
Renewing API access

Long term API tokens expire after three months so you have to renew them.

1.3 Active vs launched ads

Key distinction

Meta's API only allows us to request the ads active during a date range, not those launched. If you want launches, you can filter the data afterwards. We submit all of the data to our database, without filtering.

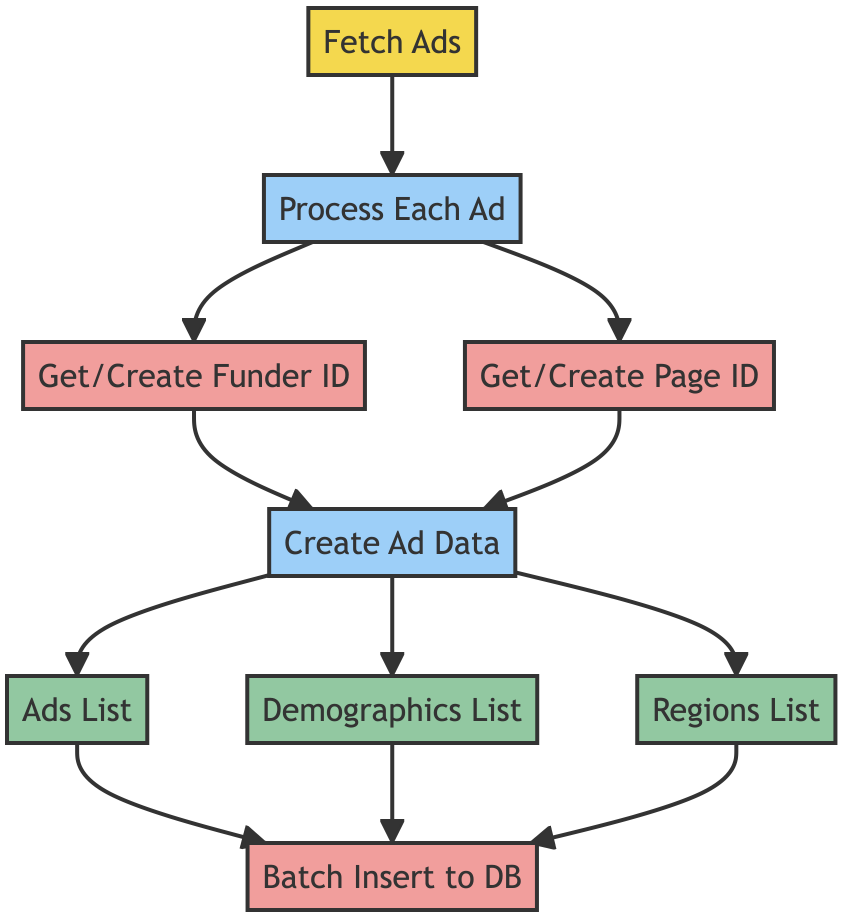


1.4 Daily data collection

- We use **Github Actions** to trigger the collection and storage of ads every day
- At 12AM EST, we collect and update all ads active in the past month
- At 8AM, 4PM and 8PM EST we collect and update all ads active on the current day

2 Data wrangling

2.1 Diagram



2.2 Dealing with empty values

Meta's API often returns ads with missing values, which can cause issues with data integrity. Here is how we deal with each one:

Missing Field	Action Taken
`page_id`	Use a hashing algorithm to generate one based on the page name. Derived ids are tracked by the `is_derived` property in the pages table.
`end_date`	Use the current date and store ongoing status in a boolean `is_active`
`estimated_audience_size` or `impressions` upper bound	Set equal to lower bound
`gender` or `age`	Set as 'unspecified'

2.3 Dealing with non-Canadian adverts

Often, advertisers based outside of Canada will include users within Canada for a very small fraction of their ad targeting. For comprehensiveness, we opted to collect these ads. However, any non-Canadian regions are aggregated as 'Overseas' in our database.

2.4 Property names

Here is a translation of Meta's API properties to our own database:

Meta API Property	Our Database Field
`id`	`id`
`page_id`	`page_id`
`funder_id`	`funder_id`
`ad_creation_time`	`created_date`
`ad_delivery_start_time`	`start_date`
`ad_delivery_stop_time`	`end_date`
	`is_active`
`ad_snapshot_url`	`ad_library_url`
`currency`	`currency`
`estimated_audience_size.lower_bound`	`audience_min`
`estimated_audience_size.upper_bound`	`audience_max`
`impressions.lower_bound`	`views_min`
`impressions.upper_bound`	`views_max`
`spend.lower_bound`	`cost_min`
`spend.upper_bound`	`cost_max`
`publisher_platforms`	`platforms`
`languages`	`languages`

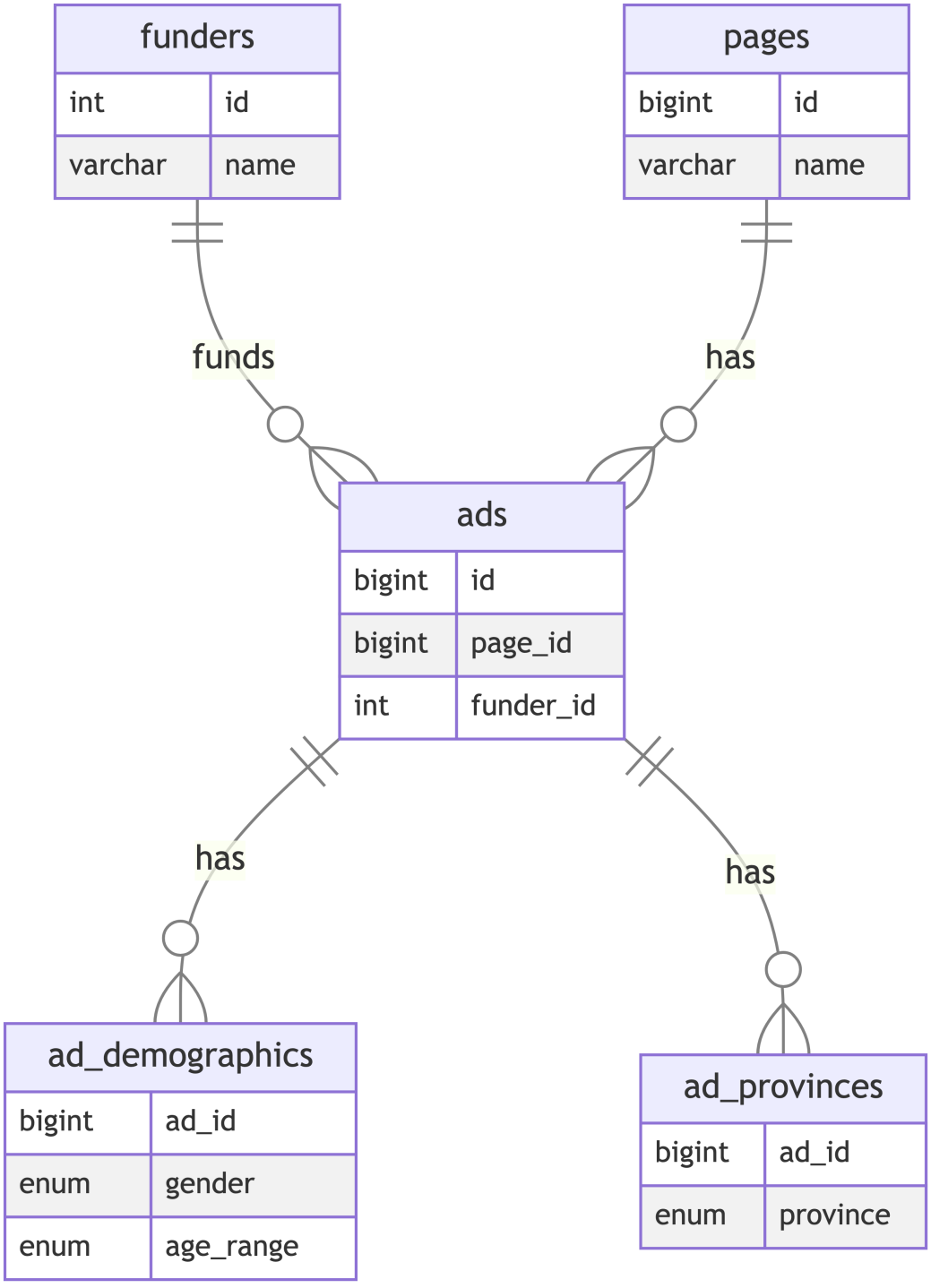
Meta API Property	Our Database Field
ad_creative_bodies	body
ad_creative_link_captions	link_url
ad_creative_link_descriptions	description
ad_creative_link_titles	link_title
delivery_by_region	provinces
demographic_distribution	demographics

3 MySQL database

3.1 Core database tables

Our MySQL database comprises five tables. The `ads` table is linked by `page_id` and `funder_id` to the `pages` and `funders` tables. Each ad in the `ads` table is linked to the `ad_demographics` and `ad_provinces` tables.

Here is the basic setup:


funders
int
id
varchar
name
pages
bigint
id
varchar
name
funds
has
ads
bigint
id
bigint
page_id
int
funder_id
has
has
ad_demographics
bigint
ad_id
enum
gender
enum
age_range
ad_provinces
bigint
ad_id
enum
province

3.2 Simplifying querying

Writing useful queries often involves joining many tables together and can be cumbersome for quick data analysis. We have designed `VIEWS` and `PROCEDURES` that make it easier to see interesting results.

3.2.1 Views

View Name	Description	Example Query
`ads_view`	Returns one row per advert, including `funder_name` and `page_name` for simpler querying.	`SELECT * FROM ads_view WHERE funder_name = 'Conservative Party of Canada - Parti conservateur du Canada' AND start_date > '2024-07-01';`
`ads_view_demographics`	Returns multiple rows per advert: one for each demographic targeted.	`SELECT * FROM ads_view_demographics WHERE...`
`ads_view_provinces`	Returns multiple rows per advert: one for each province targeted.	`SELECT * FROM ads_view_provinces WHERE...`

3.2.2 Pivot tables

The `get_pivot_table` stored procedure allows you to generate a pivot table that aggregates data about ads based on specific criteria.

```
CALL get_pivot_table('"climate change"', '2024-06-01', '2024-06-30', 'funder')
```

The procedure takes the following parameters:

- **p_keyword** (VARCHAR): A keyword to filter ads by their content, description, and link title. If `NULL`, no keyword filter is applied.
- **p_start_date** (DATE): The start date of the period for which data is to be aggregated.
- **p_end_date** (DATE): The end date of the period for which data is to be aggregated.
- **p_group_by** (VARCHAR): The dimension by which to group the results. Can be `funder`, `page`, or `both` (default).

This aggregates spending and views for each entity based on ads active in the period. For more information on how this is calculated see Section 7.1.

3.3 Table properties

Select a table to view its properties.

3.3.1 funders

Property
id
name

3.3.2 pages

Property
id
name
is_derived_id

3.3.3 ads

Property
id
page_id
funder_id
created_date
start_date
end_date
is_active
ad_library_url
currency
audience_min
audience_max
views_min
views_max
cost_min
cost_max
content_id
platforms

Property
languages
body
link_url
description
link_title
provinces
demographics

3.3.4 ad_demographics

Property
ad_id
gender
age_range
age_gender_percentage

3.3.5 ad_provinces

Property
ad_id
province
province_percentage

3.3.6 ads_view

Property
ad_id
page_id
page_name
funder_name
start_date
end_date
is_active
ad_library_url
currency

Property
views_min
views_max
cost_min
cost_max
platforms
languages
body
link_url
description
link_title
provinces
demographics

3.3.7 ads_view_demographics

Property
ad_id
page_name
funder_name
start_date
end_date
is_active
ad_library_url
currency
views_min
views_max
cost_min
cost_max
platforms
languages
body
link_url
description
link_title
gender
age_range
age_gender_percentage

3.3.8 ads_view_provinces

Property
ad_id
page_name
funder_name
start_date
end_date
is_active
ad_library_url
currency
views_min
views_max
cost_min
cost_max
platforms
languages
body
link_url
description
link_title
province
province_percentage

3.3.9 get_pivot_table

Property
group_id
group_name
ad_count
total_min_spend_for_period
total_max_spend_for_period
total_min_views_for_period
total_max_views_for_period

3.4 Top Tips for Querying

1. Include both the `start_date` and `end_date` properties as the query will be faster than filtering by just one.
2. For most queries, use the `ads_view` table rather than `ads`. It's faster even for counting rows.
3. If the database is slow, try to restrict your search. E.g. use a `WHERE` clause to search for a specific funder, keyword, date range combination.
4. Avoid trying to join unparsed demographic, regional and ad together together as you get row explosion.
5. There are two different strategies for filtering by date:

4 Querying by active date

```
end_date >= 2024-01-01 and start_date <= 2024-01-31
```

Returns all ads active in January 2024. This is the method used by the Facebook Ad Library and report.

5 Querying by launch date

```
start_date >= 2024-01-01 and start_date <= 2024-01-31
```

Returns all ads launched in January 2024.

6 Query Library

Discover useful queries.

```
<IPython.core.display.HTML object>
```

```
{
  const queries = {};

// Transform the original object
    data.Name.forEach((name, index) => {
    queries[name] = {
        query: data.Query[index],
        note: data.Notes[index]
    };
    });

  const styles = html`<style>
    #container {
      display: flex;
      height: 400px;
      background: #101b3d;
      font-family: sans-serif;
    }
    #sidebar {
      width: 30%;
      padding: 20px;
      box-sizing: border-box;
      display: flex;
      flex-direction: column;
      height: 400px;
      color: #8892b0;
      font-size: 0.85rem;
      font-weight: bold;
    }
    #search-box {
```

```
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #233554;
    border-radius: 5px;
    color: #8892b0;
    background-color: #172a45;
  }
  #query-list {
    flex-grow: 1;
    overflow-y: auto;
    border: 1px solid #233554;
    border-radius: 5px;
    max-height: calc(100vh - 100px);
    background-color: #172a45;
  }
  .query-option {
    padding: 10px;
    cursor: pointer;
    transition: background-color 0.3s;
    border-bottom: 1px solid #233554;
  }
  .query-option:hover {
    background-color: #1d3456;
  }
  #content {
    width: 70%;
    padding: 20px;
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    height: 400px;
  }
  #query-editor {
    width: 100%;
    flex-grow: 1;
    font-family: 'Courier New', monospace;
    padding: 20px;
    border: 1px solid #233554;
    border-radius: 5px;
    font-size: 18px;
    line-height: 1.5;
```

```
    background-color: #172a45;
    color: #8892b0;
    box-shadow: inset 0 0 10px rgba(0,0,0,0.1);
    overflow-y: auto;
    white-space: pre-wrap;
    word-wrap: break-word;
  }
  #query-editor:focus {
    outline: none;
  }
  #query-note {
    margin-top: 10px;
    padding: 10px;
    background-color: #233554;
    border-radius: 5px;
    color: #8892b0;
    font-size: 0.9rem;
  }
  .keyword { color: #ff79c6; }
  .function { color: #8be9fd; }
  .string { color: #f1fa8c; }
  .number { color: #bd93f9; }
  .comment { color: #6272a4; }
</style>`;

const container = html`<div id="container">`;

const sidebar = html`<div id="sidebar">`;
const searchBox = html`<input type="text" id="search-box" placeholder="Search queries...">
const queryList = html`<div id="query-list">`;

const content = html`<div id="content">`;
const queryEditor = html`<div id="query-editor" contenteditable="true" spellcheck="false"><
const queryNote = html`<div id="query-note"></div>`;

function renderQueryList(filter = '') {
  queryList.innerHTML = '';
  Object.keys(queries).forEach(key => {
    if (key.toLowerCase().includes(filter.toLowerCase())) {
      const option = html`<div class="query-option">${key}</div>`;
      option.onclick = () => {
        queryEditor.textContent = queries[key].query;
```

```
        queryNote.textContent = queries[key].note;
        highlightSyntax();
      };
      queryList.appendChild(option);
    }
  });
}

function highlightSyntax() {
  let text = queryEditor.innerText;
  text = text.replace(/\b(CALL|SELECT|FROM|WHERE|JOIN|ON|GROUP BY|HAVING|ORDER BY|UNION|CAS
  text = text.replace(/\b(AVG|SUM|COUNT|MAX|MIN)\b/gi, '<span class="function">$1</span>')
  text = text.replace(/'([^']*)'/g, '<span class="string">\'$1\'</span>');
  text = text.replace(/\b(\d+)\b/g, '<span class="number">$1</span>');
  text = text.replace(/--.*$/gm, '<span class="comment">$&</span>');

  // Save cursor position
  const selection = window.getSelection();
  const range = selection.getRangeAt(0);
  const preCaretRange = range.cloneRange();
  preCaretRange.selectNodeContents(queryEditor);
  preCaretRange.setEnd(range.endContainer, range.endOffset);
  const caretOffset = preCaretRange.toString().length;

  // Update content
  queryEditor.innerHTML = text;

  // Restore cursor position
  const newRange = document.createRange();
  newRange.setStart(queryEditor, 0);
  newRange.setEnd(queryEditor, 0);
  const nodeStack = [queryEditor];
  let node, foundStart = false, stop = false;
  let charCount = 0;

  while (!stop && (node = nodeStack.pop())) {
    if (node.nodeType === Node.TEXT_NODE) {
      const nextCharCount = charCount + node.length;
      if (!foundStart && caretOffset >= charCount && caretOffset <= nextCharCount) {
        newRange.setStart(node, caretOffset - charCount);
        foundStart = true;
      }
```

```
      if (foundStart && caretOffset >= charCount && caretOffset <= nextCharCount) {
        newRange.setEnd(node, caretOffset - charCount);
        stop = true;
      }
      charCount = nextCharCount;
    } else {
      let i = node.childNodes.length;
      while (i--) {
        nodeStack.push(node.childNodes[i]);
      }
    }
  }

  selection.removeAllRanges();
  selection.addRange(newRange);
}

searchBox.oninput = () => renderQueryList(searchBox.value);
queryEditor.oninput = highlightSyntax;

// Set placeholder text
queryEditor.dataset.placeholder = "Select a query from the list";

// Handle placeholder behavior
queryEditor.onfocus = function() {
  if (this.textContent.trim() === '') {
    this.textContent = '';
  }
};

queryEditor.onblur = function() {
  if (this.textContent.trim() === '') {
    this.textContent = '';
  }
};

sidebar.append(searchBox, queryList);
content.append(queryEditor, queryNote);
container.append(sidebar, content);

renderQueryList();
```

```
  return html`${styles}${container}`;
}
```

7 Nuances

7.1 What does spending really mean?

Spending on an ad is reported with an upper bound and lower bound by Meta.

It is the **total amount spent on the ad across its entire lifespan**. This is true even if you restrict your search with a date range. The same thing applies to views and audience size.

7.1.1 An exception: get_pivot_table()

The only place in the database where this differs is in the `get_pivot_table` procedure Section 3.2.2. Here, we estimate the spending and views **during the date range itself** by doing a couple things in the background:

1. We calculate the average amount per day for the ad
2. We multiply this by the number of days in the period

By doing this for both the upper and lower bounds, we can estimate a range for both spending and views.

`get_pivot_table` adds these results for all ads to get an idea of who is spending the most.

However, this is just an estimate. The Meta Ad Library Report provides the most reliable information on aggregated amounts spent by pages.

7.2 Pitfalls when harvesting historical data

For Meta, any ad that has no end date is generally regarded as currently active. This causes trouble when ads that clearly ended way back in 2021 have no end date.

Getting accurate end dates for these ads is tricky, so we wrote a script to query the API and manually identify the most recent date that an ad was active on.

This is a niche topic but it matters because, without treatment, these ads can creep into your calculations and analysis.

7.3 Who are they really targeting?

While regional and demographic targeting information provide some insight into the strategies of advertisers, there is also a much more opaque world.

Facebook allows pages to upload lists of people they want to include in or exclude from their adverts. These can also be used to find 'lookalike' users with similar characteristics. The Ad Library does not provide access to this granular information.

However, we can get an idea (targeted postcodes, interests, behaviors) by going on the Meta Ad Library website and viewing the 'Audience' of a page.