The Ultimate GAQL Workshop

Google Ads API Migration Workshops - 2021

Quick Introduction

In this session, our main goal will be getting a better understanding of building and validating queries using the Google Ads Query Language (GAQL).

The session will be broken down into 6 parts, as follows:

- Part 0 - Previous Class review
- Part 1 - Getting started
- Part 2 - Retrieving Available Fields with the `GoogleAdsFieldService`
- Part 3 - Dynamic Field Availability
- Part 4 - Creating Filtering Conditions
- Part 5 - Additional Validation
- Part 6 - Putting it all Together

This is meant to be an interactive session in which you can follow along with the demonstration by performing each of the steps below. Please post any questions you have to the Q&A forum, and our team will be standing by to help you out.

Part 0 - Previous Class Review

Why GAQL?

This is the new query language replacing the AdWords Query Language (AQWL). We've taken what we've learned from many years in supporting AWQL and put changes in place to better the developer experience. In the Google Ads API, all data retrieval (reports and Gets for Ads objects or data points like metrics) will be centralized through GAQL requests.

Structure

The syntax for GAQL is SQL-Like, but it is Not SQL. You can select data from a number of resources (like tables), with the ability to implicitly join against related resources, group by segmentable fields, filter by attributes/metrics, and other functionality that we'll cover below.

Making calls

The `GoogleAdsService` is used to retrieve data from the API when provided with a GAQL query. This service can use `Search` (paged responses) and `SearchStream` (streaming responses) methods, depending on your use case.

The `GoogleAdsFieldService` is used to retrieve metadata on a resource, or that resource's attributes/segments/metrics/etc. We'll use this later on to understand how to put together and validate parts of a GAQL query.

More Information

See the Reporting section of our developer pages, and be sure to watch our Session on *Flexible Reporting*.

Google Ads

Part 1 - Getting started

Part 1.0: Understanding the project

This presentation will center around 2 main goals of getting:

1. A better, deeper understanding of the Google Ads Query Language
2. Experience with the GAQL tools available to you

Whether building a full reporting interface, like the Interactive Query Builder, or trying to develop the right reports for your platform's needs, this session will get you better acquainted with how the query language works and set you on the path to becoming a reporting power user.

Part 1.1: Preparing the tools you'll need

This presentation will make calls to the API with cURL, walk through the UI of the Interactive Query Builder, and use the Query Validator to troubleshoot invalid queries.

You'll need:

- A terminal with cURL
- An active Google Ads API developer token
- A Google Cloud Platform project with the Google Ads API enabled (You'll need the ClientID, ClientSecret)
- A valid OAuth Refresh token

If you don't have these handy, please review the Quickstart section of our developer pages. Once you have all of this information, let's set some environment variables for the rest of this workshop:

1.1.0: Set Environment Variables

```
API_VERSION="8" \
DEVELOPER_TOKEN="<insert>" \
CLIENT_ID="<insert>" \
CLIENT_SECRET="<insert>" \
REFRESH_TOKEN="<insert>" \
OAUTH2_ACCESS_TOKEN="" # Populated in next step
```

The function below creates an Access Token and assigns it to the environment variable `OAUTH2_ACCESS_TOKEN`. After pasting this into your terminal, you can run `gen_access_token` to generate a new Access Token:

1.1.1: Function to Generate an Access Token and Assign to a Variable

```
gen_access_token() {
  OAUTH2_ACCESS_TOKEN=$(curl \
    --data "grant_type=refresh_token" \
    --data "client_id=${CLIENT_ID}" \
    --data "client_secret=${CLIENT_SECRET}" \
    --data "refresh_token=${REFRESH_TOKEN}" \
      https://www.googleapis.com/oauth2/v3/token | grep '"access_token": "' | sed
's/^.*: "//' | sed 's/",//')

  echo "Generated new access token: $OAUTH2_ACCESS_TOKEN"
}
```

Finally, the function below issues a search request against the `GoogleAdsFieldService` provided a query parameter:

1.1.2: Function to Issue a `GoogleAdsFieldService` Search Request

```
gafs() {
  if [ $# -eq 0 ]; then
    echo "No arguments supplied"
    return
  fi
  QUERY=${1}
  echo "Performing GoogleAdsFields search request with query: ${QUERY}"
  curl -f --request POST
"https://googleads.googleapis.com/v${API_VERSION}/googleAdsFields:search" \
    --header "Content-Type: application/json" \
    --header "developer-token: ${DEVELOPER_TOKEN}" \
```

```
    --header "Authorization: Bearer ${OAUTH2_ACCESS_TOKEN}" \
    --data "{ query:'${QUERY}',
          page_size:9999 }"
}
```

Now, you can issue a `SearchGoogleAdsFieldsRequest` simply by running `gafs` followed by a query. For example:

1.1.3: Example Command to Issue a `GoogleAdsFieldService` Search Request

```
gafs "SELECT segments, metrics WHERE name = \"campaign\""
```

Part 2 - Retrieving Available Fields with the `GoogleAdsFieldService`

Part 2.0: Choosing your "From Resource"

The root of every GAQL query is the FROM clause, which will always be a single resource. This is known as the "From Resource", the resource in the FROM clause. It defines all fields allowed in the SELECT, WHERE, and ORDER BY clauses. Let's begin by generating a list of the available resources that a user can select data from (or the possible values for the FROM clause) by issuing a query against the `GoogleAdsFieldService`:

2.0.0: Example Command to Issue a `GoogleAdsFieldService` Search Request

```
gafs "SELECT name, metrics WHERE category = \"RESOURCE\""
```

You'll notice this query looks similar to those we would use to retrieve data from the `GoogleAdsService` in a `SearchStream` or `Search` request; however there is no FROM clause. Instead, we're selecting across all Resources and Fields.

Part 2.1: Retrieving available attributes

The following fields are available given a resource in the FROM clause:

- Any attribute of that resource (example: `campaign.name` is an attribute of `campaign`)
- Any field listed in the resource's "`metrics`" or "`segments`" arrays
- Any attribute allowed from related `attributeResources`

Let's assume that we have chosen `campaign` as our resource in the FROM clause. We can find all attributes of the `campaign` resource with the following query because we know all of these attributes are prefixed with `campaign[dot]`.

2.1.0: List all Campaign Attributes

```
gafs "SELECT name WHERE name LIKE \"campaign.%\""
```

Because the `metrics`, `segments`, and `attributeResources` are present on all fields in a `GoogleAdsField`, we can retrieve those fields with the following query:

2.1.1: List all Metrics, Segments, and Attribute Resources on the Campaign Resources

```
gafs "SELECT metrics, segments, attribute_resources WHERE name = \"campaign\""
```

You may notice that the `attributeResources` array contains a list of resources rather than attribute fields. As discussed in the earlier *Flexible Reporting* session, all attribute fields on an attribute resource can be used in a GAQL query. In order to retrieve those fields, you would issue the same query as we did to retrieve the `campaign` resource's attribute fields, but replace `campaign` with the attribute resource. For example:

2.1.2: List all Attributes of an Attribute Resources (Campaign Budget)

```
gafs "SELECT name WHERE name LIKE \"campaign_budget.%\""
```

Part 2.2: Fields and clauses

We now have a full list of the fields that can be used in our GAQL query. The next step is to determine which fields can be placed in the SELECT, WHERE, and ORDER BY clauses. As a rule, an attribute can be placed in each of the aforementioned clauses if the following fields on the attribute have a value of true:

Clause	Field must be true
SELECT	`selectable`
WHERE	`filterable`
ORDER BY	`sortable`

You can retrieve this metadata from the `GoogleAdsFieldService`. As an example, let's dig deeper into the metadata of one of the `campaign` resource's attributes, `campaign.manual_cpm` with the following query:

2.2.0: Get Field Metadata for campaign.manual CPM

```
gafs "SELECT selectable, filterable, sortable WHERE name = \"campaign.manual_cpm\""
```

As you can see from the results below, `campaign.manual_cpm` can be placed in the SELECT and ORDER BY clauses because `selectable` and `sortable` are true. However, it cannot be used in the WHERE clause because `filterable` is false.

2.2.1: Field Metadata for campaign.manual CPM Response

```
{
  "resourceName": "googleAdsFields/campaign.manual_cpm",
  "selectable": true,
  "filterable": false,
  "sortable": false
}
```

Alternatively, you can find all fields of a given type by chaining filtering conditions together. For example, the following query produces a list of all campaign attributes that can be used in the WHERE clause:

2.2.2: List all Campaign Attributes that can be Used in the WHERE Clause

```
gafs "SELECT name WHERE name LIKE \"campaign.%\" AND filterable = true"
```

As before, the same query would work for any attribute resources.

2.2.3: List all Attribute Resource Attributes that can be Used in the WHERE Clause

```
gafs "SELECT name WHERE name LIKE \"campaign_budget.%\" AND filterable = true"
```

We can follow a similar pattern for `metrics` and `segments`. However, this would require two queries: one to retrieve all of the `segments` or `metrics`, and one to check if the fields meet the criteria to be used in a clause. For example, let's assume we want to find all `metrics` that can be used in the ORDER BY clause when `campaign` is in the FROM clause. Our first query will retrieve all of the `metrics` on the `campaign` resource, clean up some formatting, and save the result to a variable called `METRICS`.

2.2.4: Get Campaign Metrics and Store as Environment Variable

```
METRICS=$(gafs "SELECT metrics WHERE name = \"campaign\"" | grep "metrics\." | sed 's/"/\\"/g')
```

Then, we can chain together filtering conditions using the IN notation to find all `metrics` that are `sortable`.

2.2.5: List all Metrics that can be Used in the ORDER BY Clause

```
gafs "SELECT name WHERE name IN ($METRICS) AND sortable = true"
```

Conversely, by changing `sortable` to `false`, you can see that there is only one `metric` on the `campaign` resource, `metrics.interaction_event_types`, that cannot be placed in the ORDER BY clause:

2.2.6: List all Metrics that cannot be Used in the ORDER BY Clause

```
gafs "SELECT name WHERE name IN ($METRICS) AND sortable = false"
```

Part 3 - Dynamic Field Availability

We now have a list of all of the attribute fields, attribute resource fields, metrics, and segments that can be inserted into each of the SELECT, WHERE, and ORDER BY clauses given a resource in the FROM clause. However, the fields that you can use in a GAQL query are dynamic in that their availability for use in a query/clause is also based on the state of the query itself.

Step 3.0: Field Compatibility

Segments are fields that, when added to the SELECT clause, automatically bucket your selected metrics like a "GROUP BY" statement might in SQL. Since segments perform this special function, not every segment will work with every metric, resource, or other segment. For this reason, we need to perform a validation step to ensure that any field we would like to add to our GAQL query is compatible with every field that has already been added to our query. We can use the `selectable_with` field to make this determination. As an example, let's get the metadata for `segments.device` with the following query:

3.0.0: List Fields Selectable with `segments.device`

```
gafs "SELECT selectable_with WHERE name = \"segments.device\""
```

The `selectableWith` field in our results set lists all fields that are compatible with the given field, and therefore can be present in a GAQL query at the same time.

To demonstrate this visually, let's turn to the Google Ads Query Builder.

1. Navigate to the query builder for the campaign resource. *(Note: all resources have a query builder UI)*

2. Add `metrics.clicks` to the SELECT clause. You can find `metrics.clicks` by either using the search bar or expanding the *Metrics* container.
3. Now, expand the *Segments* container (you may need to clear any search input first).

As you can see, several segments are grayed out and are no longer selectable. If you hover over the  badge on any of those segments, you'll see a message indicating that the segment is not compatible with `metrics.clicks`.

Let's take a look at the metadata for one of those fields, `segments.conversion_attribution_event_type`. If you click the  icon next to the segment name, and scroll down to the *Incompatible Fields* section, you'll see `metrics.clicks` listed.

Now, select add `segments.ad_destination_type` to the SELECT clause of your query. Notice how `segments.conversion_value_rule_primary_dimension` now shows two errors. If you hover over the error badge, now showing the  icon, you'll see that this segment is incompatible with both of the fields we've selected, so you would need to remove each of those fields from the query in order to use `segments.conversion_value_rule_primary_dimension` in your query.

Part 3.1: Fields required in SELECT

There is one more rule that dictates whether or not a field can be added to a clause in a GAQL query. Certain fields must be present in the SELECT clause before they can be added to either the WHERE or ORDER BY clauses.

The WHERE Clause

All segments (other than "core date segments") and segmenting resource attributes must be added to the SELECT clause before being added to the WHERE clause. The following "core date segments" can be added to the WHERE clause without being present in the SELECT clause:

- `segments.date`
- `segments.week`
- `segments.month`
- `segments.quarter`
- `segments.year`

Let's demonstrate again with the query builder for the campaign resource:

- Clear your selections.
- Switch to the WHERE tab.
- Select any segment.
- Notice the pop-up prompts you to first add that segment to the SELECT clause.
- Click *Yes*.
- Now, deselect that segment from the SELECT clause and notice how it is automatically removed from the WHERE clause as well.

The ORDER BY Clause

The ORDER BY clause has similar rules, but they apply to a broader scope of fields. The following fields must be present in the SELECT clause before they can be added to the ORDER BY clause:

- Segments (other than core date segments)
- Segmenting resources
- Metrics

In addition, at least one field on an attribute resource field must be present in the SELECT clause in order for any field on that attribute resource to be inserted into the ORDER BY clause. Let's look at an example in the campaign query builder:

- Clear your selections.
- Switch to the ORDER BY tab.
- Select `campaign_budget.amount_micros`. You'll see the same pop-up as before. Click *Yes*.
- Add another field on the `campaign_budget` resource to the ORDER BY clause (for example, `campaign_budget.delivery_method`).
- Notice how there is no pop-up, the new field is immediately added to the ORDER BY clause.
- Remove `campaign_budget.amount_micros` from the SELECT clause. Assuming that was the only field on `campaign_budget` in your SELECT clause, you'll notice how all of the `campaign_budget` fields have automatically been removed from the ORDER BY clause

Google Ads

Part 4 - Additional Validation

We now know how to find all fields that are available to be placed in a GAQL query, determine whether or not they can be added to a given clause, and construct filtering conditions in the WHERE clause. Each query you submit also undergoes a few additional steps of validation. For example, the SELECT and FROM clauses are required, the LIMIT clause must be a positive integer, if used, and each clause must be syntactically correct. In addition, we briefly covered resource-specific validation in the *Flexible Reporting* session. We are going to focus on one specific type of validation relating to the use of core date segments.

Part 4.0: Core Date Segments

If a core date segment is present in any clause of a GAQL query, the WHERE clause must contain filtering conditions on core date segments that combine to form a valid, finite date range.

Valid Date Range

A date range is valid if there is at least one date that meets all of the core date-related filtering conditions. For example:

- Clear any selections and add `segments.date` to the SELECT clause. There is now an error message indicating that we are missing a valid, finite date range.
- Add the following filtering condition to the WHERE clause: `segments.date = “2021-01-01”`, and notice how there is no longer an error message.
- Add the following filtering condition to the WHERE clause: `segments.date DURING LAST_7_DAYS`. Now, the error message has reappeared because “2021-01-01” falls outside of the LAST_7_DAYS,

and therefore, there are no dates that meet all of the filtering conditions.

Finite Date Range

A date range is finite if it has a beginning and an end. In other words, the date range must not be open-ended. For example:

- Clear any selections and add `segments.date` to the SELECT clause. There is now an error message indicating that we are missing a valid, finite date range.
- Add the following filtering condition to the WHERE clause: `segments.date > “2021-01-01”`. This time, the error hasn’t gone away because this is an open-ended date range.
- Add the following filtering condition to the WHERE clause: `segments.date < “2021-02-01`. Now, the error message has disappeared because the filtering conditions combine to form a finite date range.

Part 5 - Creating Filtering Conditions

The WHERE clause is completely optional. Each filtering condition in the WHERE clause is composed of three parts:

- **“Filter Field” -** The field to be filtered on
- **“Filter Operator” -** The operator used for comparison
- **“Filter Value” -** The value we’re using to constrain the Filter Field

The operators allowed in a given filtering condition are dependent on the Filter Field’s `data_type`, and the format of the filter value is dependent on the combination of the `data_type` and selected operator. Let’s look at some examples in the campaign query builder:

- Clear any selections and switch to the WHERE tab.
- Select `segments.date`.
- Click on the *Choose an operator* dropdown to see the available operators for `segments.date`, which has a `data_type` of `DATE`.
- Select the “=” operator, and you’ll notice a hint that says the input must be formatted as “YYYY-MM-DD”.
- Switch the operator to “BETWEEN”. Now, there are two date input fields, both of which must be formatted as “YYYY-MM-DD”. When the BETWEEN operator is used on a DATE. Enter two dates to see how the final filtering condition must be formatted.
- Remove the filtering condition from your query, and click `segments.date` again. This time, choose the DURING operator.
- The input for Filter Values is now a dropdown. When the DURING operator is used, there is a predetermined set of available options for the Filter Values.

As another example, the `segments.day_of_week` field is an “Enum” type. This would tell us that we need to look at the accompanying “enumValues” array on the Filter Field to restrict the values allowed for filtering:

- Clear your selections in the query builder and switch to the WHERE tab.
- Select `segments.day_of_week` and click *Yes* when prompted.
- Click the operator dropdown, and you’ll notice a completely different set of options than we saw with `segments.date`, which are applicable to ENUM `data_types`.
- Select the “!=” operator, and click the Filter Value dropdown. You’ll now see enum values for `segments.day_of_week`, or the days of the week. Choose a value and add the filtering condition to the query.
- Click *Add filtering condition on segments.day_of_week* to add another filtering condition.
- This time, choose the IN operator.
- You’ll still see the same enum values in the Filter Value dropdown, but now you have the option to select multiple values.
- Select one or more values and add the filtering condition to the query.
- When IN is used as the operator, you can see that the filter value must be a comma separated list of those enum values wrapped in parentheses.

Part 6 - Putting it all Together

Now that we've learned all about constructing and validating GAQL queries, let's use the Query Validator to see if we can take what we've learned to fix some invalid queries:

1. `SELECT campaign.id, campaign.name, segments.slot, segments.hour, metrics.clicks FROM campaign`
   - **Reason**:
     segments.slot is not selectable with segments.hour.
   - **Valid**:
     ```
     SELECT campaign.id, campaign.name, segments.hour, metrics.clicks FROM campaign
     ```
2. `SELECT ad_group.id, metrics.clicks, segments.conversion_action FROM ad_group ORDER BY campaign.advertising_channel_type`
   - **Reason**:
     segments.conversion_action is incompatible with metrics.clicks, and campaign.advertising_channel_type is used in the ORDER BY but not in the SELECT
   - **Valid**:
     ```
     SELECT ad_group.id, metrics.clicks, campaign.advertising_channel_type FROM ad_group ORDER BY campaign.advertising_channel_type
     ```
3. `SELECT ad_group.id, metrics.clicks FROM ad_group WHERE metrics.clicks != (1, 2, 3)`
   - **Reason**:
     Incorrect operator and filter value combination in the WHERE clause
   - **Valid**:
     ```
     SELECT ad_group.id, metrics.clicks FROM ad_group WHERE metrics.clicks NOT IN (1, 2, 3)
     ```

Google Ads

Conclusion

Hopefully you found this informative, and you're leaving with a better understanding of GAQL than you started with. Please keep in mind that when you're writing reports against the API, much of the complexity outlined in this presentation can be boiled down to a more digestible format by using the Interactive Query Builder directly.