# Async SOQL Guide

Salesforce, Winter '21


@salesforcedocs

Last updated: August 20, 2020

# CONTENTS

# ASYNC SOQL

Async SOQL is a method for running SOQL queries when you can't wait for immediate results. These queries are run in the background over Salesforce big object data. Async SOQL provides a convenient way to query large amounts of data stored in Salesforce.

Async SOQL is implemented as a RESTful API that enables you to run queries in the familiar syntax of SOQL. Because of its asynchronous operation, you can subset, join, and create more complex queries and not be subject to timeout limits. This situation is ideal when you have millions or billions of records and need more performant processing than is possible using synchronous SOQL. The results of each query are deposited into an object you specify, which can be a standard object, custom object, or big object.

The limit for Async SOQL queries is one concurrent query at a time.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

Available in: **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions for up to 1 million records

Extra record capacity and Async SOQL query available as an add-on license.

## Async SOQL Versus SOQL

SOQL and Async SOQL provide many of the same capabilities. So when would you use an Async SOQL query instead of standard SOQL?

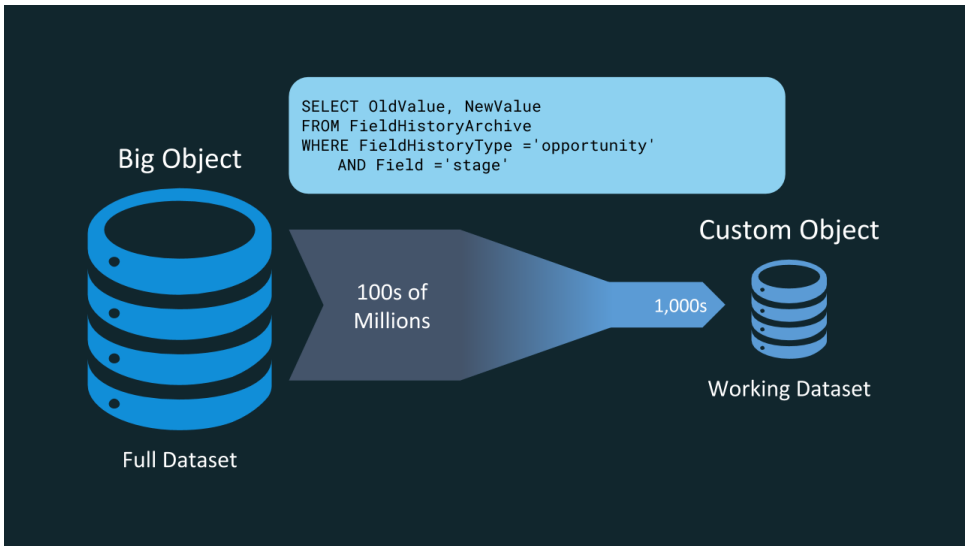
Use standard SOQL when:

- You want to display the results in the UI without having the user wait for results.
- You want results returned immediately for manipulation within a block of Apex code.
- You know that the query returns a small amount of data.

Use Async SOQL when:

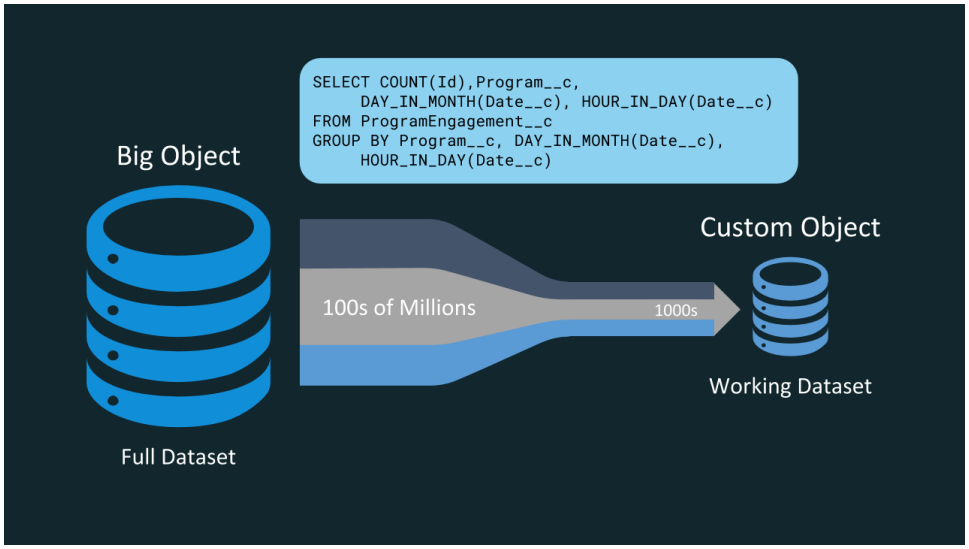
- You are querying against millions of records.
- You want to ensure that your query completes.
- You do not need to do aggregate queries or filtering outside of the index.

## Use Case: Create a Working Dataset with Filtering




For example, let's say that you want to analyze the years and years of opportunity history collected by Salesforce. The results could help you identify which current and future opportunities are more likely to close and give you a better picture of your forecast. But because the opportunity history data is stored with all the field history data across the application, the volume of data is too large to query directly. That's where Async SOQL comes in! You can use it to write a query that extracts a smaller, representative subset of the data that you're interested. You can store this working dataset in a custom object and use it in reports, dashboards, or any other Lightning Platform feature.

## Use Case: Create a Working Dataset with Coarse Aggregations




With big objects, you can now bring a much finer level of detail into your applications using data that you already have. For example, every interaction an individual has with your marketing campaign is stored as data that you can use, but it's unwieldy in its raw form. Async SOQL allows you to aggregate that data by campaign and day and to extract the relevant details of the full dataset into a smaller, usable dataset. As in the previous example, the smaller working set can live in a custom object and be used in your reports and dashboards.

# VIEW BIG OBJECT DATA IN REPORTS AND DASHBOARDS

When working with big data and billions of records, it's not practical to build reports or dashboards directly from that data. Instead, use Async SOQL to write a query that extracts a smaller, representative subset of the data that you're interested in. You can store this working dataset in a custom object and use it in reports, dashboards, or any other Lightning Platform feature.

1. Identify the big object that contains the data for which you need a report. In this example, the `Ride__b` big object contains the full dataset.
2. Create a custom object. This object holds the working dataset for the big object data that you want to report on. In this example, we use the `Bike_Rental__c` custom object.
   a. Under Optional Features for the custom object, click **Allow Reports**.
   b. Add custom fields to the object that match the fields that you want to report on from the big object.
3. Create an Async SOQL query that builds your working dataset by pulling the data from your big object into your custom object.

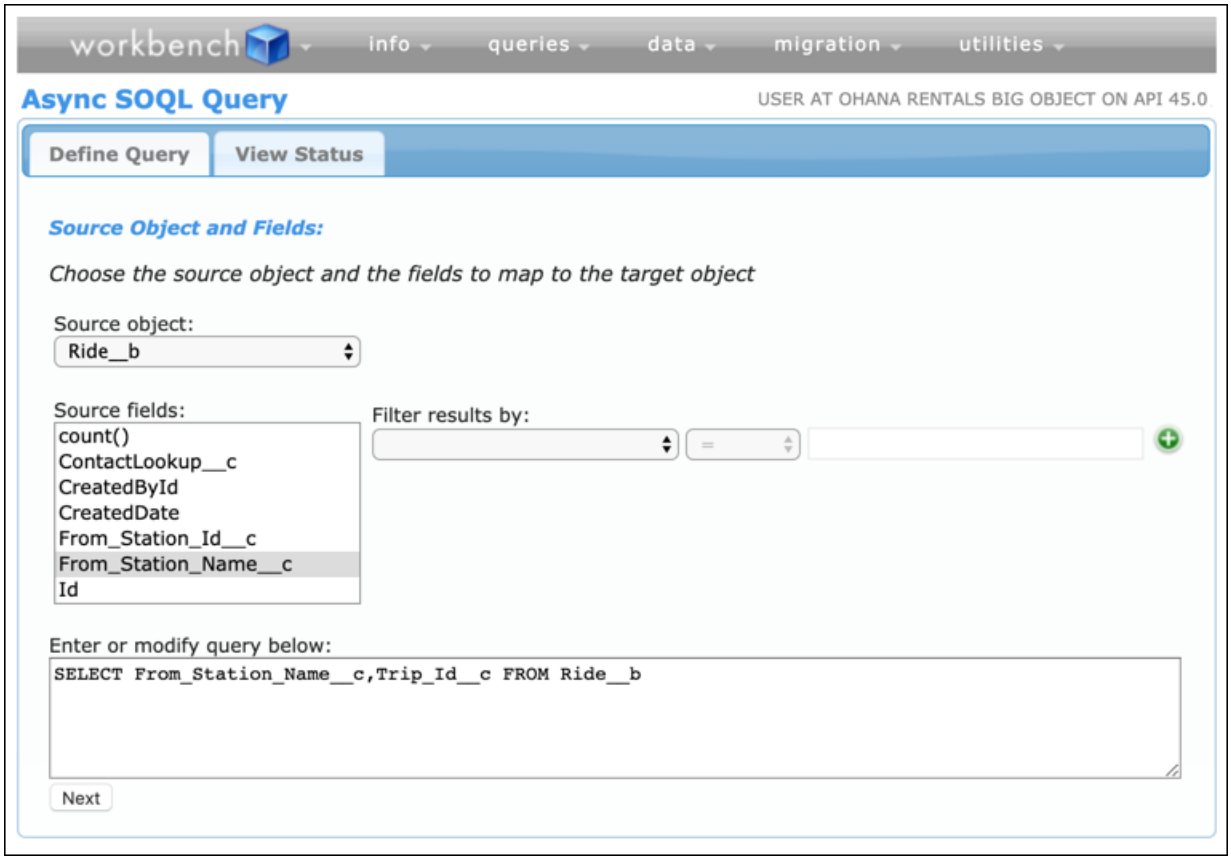
   Tip: To ensure that your working dataset is always up-to-date for accurate reporting, set this job to run nightly.

   a. Log in to Workbench.

      To access Workbench, log in to your org, then open a new browser tab and navigate to *`https://developer.salesforce.com/page/Workbench`*.

      Note: To see examples of running Async SOQL queries using REST API, see Running Async SOQL Queries.

   b. Select **Queries**, then **Async SOQL**.
   c. For the Source object, choose the big object from step 1. In this example, `Ride__b` is the source big object that holds the full dataset.

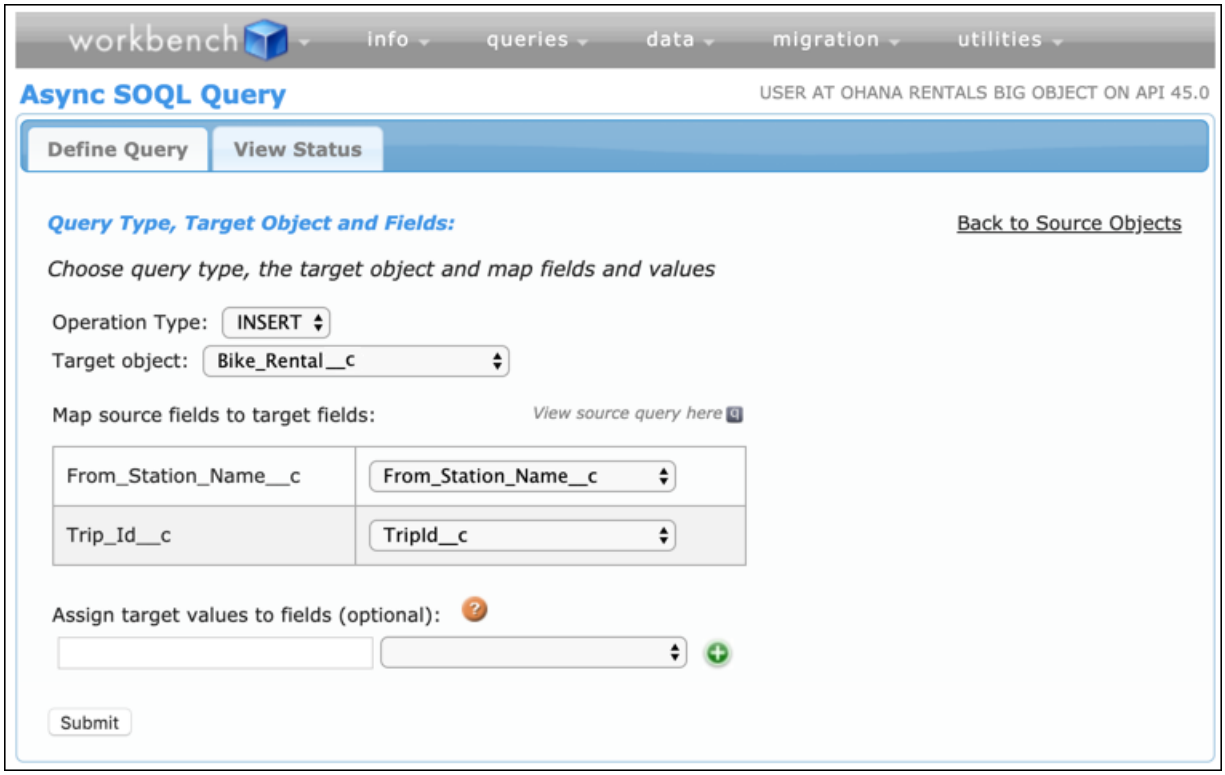



**d.** Select the source fields and filter criteria.

**e.** Set the operation type to **INSERT**.

**f.** For the Target object, choose the custom object from step 2, `Bike_Rental__c`.

**g.** Map the source fields from the big object to the target fields in the custom object. In this example, we are mapping the `From_Station_Name__c` and `Trip_Id__c` fields from the `Ride__b` source big object to the corresponding fields on the `Bike_Rental__c` target custom object.




**h.** Run the query and wait until it completes. Run time depends on how much data you have.

4. After the query runs, query the custom object in Workbench to see that the data is there.
5. Build a report using the working dataset you created.
   **a.** From Setup, enter *Report Types* in the Quick Find box, then select **Report Types**.
   **b.** Create a custom report type.
   **c.** For the Primary Object, select the custom object from step 2, `Bike_Rental__c`.
   **d.** Set the report to **Deployed**.
   **e.** Run the report.

You can now use the information from your working dataset not only in your reports, but also in dashboards or any other Lightning Platform feature.

SEE ALSO:

- Create Custom Objects
- Create Big Objects
- Create a Custom Report Type
- Using Workbench

# RUNNING ASYNC SOQL QUERIES

Learn how to run Async SOQL queries on your objects and check on the status of your query using Connect REST API.

## Formulating Your Async SOQL Query

To use Async SOQL effectively, it's helpful to understand its key component and other related concepts. Each query is formulated in the POST request as a JSON-encoded list of three or four key-value pairs.

**Request body for POST**

| Name | Type | Description | Required or Optional | Available Version |
|---|---|---|---|---|
| `query` | String | Specifies the parameters for the SOQL query you want to execute. The `FROM` object must be a big object. | Required | 35.0 |
| `operation` | String | Specify whether the query is an insert or upsert. If the record doesn't exist, an upsert behaves like an insert.<br>Note: Upsert is not supported for big objects | Optional | 39.0 |
| `targetObject` | String | A standard object, custom object, external object, or big object into which to insert the results of the query. | Required | 35.0 |
| `targetFieldMap` | Map<String, String> | Defines how to map the fields in the query result to the fields in the target object.<br>Note: When defining the `targetFieldMap` parameter, make sure that the field type mappings are consistent. If the source and target fields don't match, these considerations apply.<br>• Any source field can be mapped onto a target text field.<br>• If the source and target fields are both numerical, the target field must have the same or greater number of decimal places than the source field. If not, the request fails. This behavior is to ensure that no data is lost in the conversion.<br>• If a field in the query result is mapped more than once, even if mapped to | Required | 35.0 |

| Name | Type | Description | Required or Optional | Available Version |
|---|---|---|---|---|
| | | different fields in the target object, only the last mapping is used. | | |
| `targetValueMap` | Map<String, String> | Defines how to map static strings to fields in the target object. Any field or alias can be used as the `TargetValueMap` value in the SELECT clause of a query.<br>You can map the special value, $JOB_ID, to a field in the target object. The target field must be a lookup to the Background Operation standard object. In this case, the ID of the Background Operation object representing the Async SOQL query is inserted. If the target field is a text field, it must be at least 15–18 characters long.<br>You can also include any field or alias in the SELECT clause of the `TargetValueMap`. They can be combined together to concatenate a value to be used. | Optional | 37.0 |
| `targetExternalIdField` | String | The ID of the target sObject. Required for upsert operations. | Optional | 39.0 |

This simple Async SOQL example queries `SourceObject__b`, a source big object, and directs the result to `TargetObject__c`, a custom object. You can easily map the fields in the source object to the fields of the target object in which you want to write the results.

**Example URI**

```
https://yourInstance.salesforce.com/services/data/v38.0/async-queries/
```

**Example POST request body**

```
{
   "query": "SELECT firstField__c, secondField__c FROM SourceObject__b",

   "operation": "insert",

   "targetObject": "TargetObject__c",

   "targetFieldMap": {"firstField__c":"firstFieldTarget__c",
                      "secondField__c":"secondFieldTarget__c"
                      },
   "targetValueMap": {"$JOB_ID":"BackgroundOperationLookup__c",
                      "Copy fields from source to
target":"BackgroundOperationDescription__c"
                     }
}
```

The response of an Async SOQL query includes the elements of the initial POST request.

**Response body for POST**

| Property Name | Type | Description | Filter Group and Version | Available Version |
|---|---|---|---|---|
| `jobId` | String | The ID of the Async SOQL query. This ID corresponds to an entry in the Background Operation standard object. It matches the ID that is used in the `targetValueMap` when $JOB_ID is used. To get the status of an async query job, use this ID in an Async Query, Status request (`/async-queries/`***`jobId`***). | Big, 35.0 | 35.0 |
| `message` | String | A text message that provides information regarding the query, such as an error message if the query failed. | Big, 37.0 | 37.0 |
| `operation` | String | Specify whether the query is an insert or upsert. If the record doesn't exist, an upsert behaves like an insert.<br>Note: Upsert is not supported for big objects | Big, 39.0 | .39.0 |
| `query` | String | Specifies the parameters for the SOQL query you want to execute. The `FROM` object must be a big object. | Big, 35.0 | 35.0 |
| `status` | String | Status of an async query job.<br>• `Canceled`—The job was canceled before it could be run.<br>• `Complete`—The job was successfully completed.<br>• `Failed`—The job failed after the system submitted it or because the request exceeded the Async SOQL limits. The message field provides details on the reason for failure.<br>• `Running`—The job is running successfully, and the org hasn't exceeded any limits.<br>• `Scheduled`—The new job has been created and scheduled, but is not yet running.<br>• `New`—The job has been created but is not yet scheduled. | Big, 35.0 | 35.0 |
| `targetExternalIdField` | String | The ID of the target sObject. Required for upsert operations. | Big, 39.0 | 39.0 |

| Property Name | Type | Description | Filter Group and Version | Available Version |
|---|---|---|---|---|
| `targetFieldMap` | Map<String, String> | Defines how to map the fields in the query result to the fields in the target object.<br><br>Note: When defining the `targetFieldMap` parameter, make sure that the field type mappings are consistent. If the source and target fields don't match, these considerations apply.<br>• Any source field can be mapped onto a target text field.<br>• If the source and target fields are both numerical, the target field must have the same or greater number of decimal places than the source field. If not, the request fails. This behavior is to ensure that no data is lost in the conversion.<br>• If a field in the query result is mapped more than once, even if mapped to different fields in the target object, only the last mapping is used. | Big, 35.0 | 35.0 |
| `targetValueMap` | Map<String, String> | Defines how to map static strings to fields in the target object. Any field or alias can be used as the `TargetValueMap` value in the SELECT clause of a query.<br><br>You can map the special value, $JOB_ID, to a field in the target object. The target field must be a lookup to the Background Operation standard object. In this case, the ID of the Background Operation object representing the Async SOQL query is inserted. If the target field is a text field, it must be at least 15–18 characters long.<br><br>You can also include any field or alias in the SELECT clause of the `TargetValueMap`. They can be combined together to concatenate a value to be used. | Big, 37.0 | 37.0 |
| `targetObject` | String | A standard object, custom object, external object, or big object into which to insert the results of the query. | Big, 35.0 | 35.0 |

**Example POST response body**

```
{
   "jobId": "08PD000000003kiT",

   "message": "",

   "query": "SELECT firstField__c, secondField__c FROM SourceObject__b",

   "status": "New",

   "targetObject": "TargetObject__c",

   "targetFieldMap": {"firstField__c":"firstFieldTarget__c",
                      "secondField__c":"secondFieldTarget__c"
                     },
   "targetValueMap": {"$JOB_ID":"BackgroundOperationLookup__c",
                      "Copy fields from source to
target":"BackgroundOperationDescription__c"
                     }
}
```

# Tracking the Status of Your Query

To track the status of a query, specify its jobID with an HTTP GET request.

```
https://yourInstance.salesforce.com/services/data/v38.0/async-queries/<jobID>
```

The response is similar to the initial POST response but with updated status and message fields to reflect the status.

**Example GET response body**

```
{
"jobId": "08PD000000000001",
"message": "",
"query": "SELECT firstField__c, secondField__c FROM SourceObject__b",
"status": "Complete",
"targetObject": "TargetObject__c",
"targetFieldMap": {"firstField__c":"firstFieldTarget__c",
"secondField__c":"secondFieldTarget__c" }
}
```

You can get status information for all queries with the following HTTP GET request.

```
https://yourInstance.salesforce.com/services/data/v38.0/async-queries/
```

**Example GET response body**

```
{
"asyncQueries" : [ {
"jobId" : "08PD000000000002",
"message" : "",
"query" : "SELECT String__c FROM test__b",
```

```
"status" : "Running",
"targetFieldMap" : {
"String__c" : "String__c"
},
"targetObject" : "test__b",
"targetValueMap" : { }
}, {
"jobId": "08PD000000000001",
"message": "Complete",
"query": "SELECT firstField__c, secondField__c FROM SourceObject__b",
"status": "Complete",
"targetObject": "TargetObject__c",
"targetFieldMap": {"firstField__c":"firstFieldTarget__c",
"secondField__c":"secondFieldTarget__c" }
}
}
```

## Canceling a Query

You can cancel a query using an HTTP DELETE request by specifying its jobId.

```
https://yourInstance.salesforce.com/services/data/v38.0/async-queries/jobId
```

Note: Canceling a query that has already completed has no effect.

## Handling Errors in Async SOQL Queries

Two different types of errors can occur during the execution of an Async SOQL query.

- An error in the query execution
- One or more errors writing the results into the target object

Problems in executing the job cause some errors. For example, an invalid query was submitted, one of the Async SOQL limits was exceeded, or the query caused a problem with the underlying infrastructure. For these errors, the response body includes a status of Failed. The message parameter provides more information on the cause of the failure.

Other times, the query executes successfully but encounters an error while attempting to write the results to the target object. Because of the volume of data involved, capturing every error is inefficient. Instead, subsets of the errors generated are captured and made available. Those errors are captured in the BackgroundOperationResult object and retained for seven days. You can query this object with the Async SOQL query jobID to filter the errors for the specific Async SOQL query. Async SOQL job info is retained for a year.

# ASYNC SOQL USE CASES

Understand some of the common Async SOQL use cases.

## Customer 360 Degree and Filtering

In this use case, administrators load various customer engagement data from external sources into Salesforce big objects and then process the data to enrich customer profiles in Salesforce. The goal is to store customer transactions and interactions, such as point-of-sale data, orders, and line items in big objects and then process and correlate that data with your core CRM data. Anchoring customer transactions and interactions with core master data provides a richer 360-degree view that translates into an enhanced customer experience.

The following example analyzes the customer data stored in the Rider record of a car-sharing service. The source big object, Rider_Record_b, has a lookup relationship with the Contact object, allowing for an enriched view of the contact's riding history. You can see that the query includes Rider__r.FirstName, Rider__r.LastName, Rider__r.Email as part of the `SELECT` clause. This example demonstrates the ability to join big object data (Rider_Record__b) with Contact data (FirstName, LastName, Email) in a single Async SOQL query.

**Example URI**

```
https://yourInstance—api.salesforce.com/services/data/v38.0/async-queries/
```

**Example POST request body**

```
{
   "query": "SELECT End_Location_Lat__c, End_Location_Lon__c, End_Time__c,
                    Start_Location_Lat__c, Start_Location_Lon__c, Start_Time__c,
                    Car_Type__c, Rider__r.FirstName, Rider__r.LastName,
                    Rider__r.Email
              FROM Rider_Record__b WHERE Star_Rating__c = '5'",

   "targetObject": "Rider_Reduced__b",

   "targetFieldMap": {"End_Location_Lat__c":"End_Lat__c",
                      "End_Location_Lon__c":"End_Long__c",
                      "Start_Location_Lat__c": "Start_Lat__c",
                      "Start_Location_Lon__c": "Start_Long__c",
                      "End_Time__c": "End_Time__c",
                      "Start_Time__c": "Start_Time__c",
                      "Car_Type__c": "Car_Type__c",
                      "Rider__r.FirstName": "First_Name__c",
                      "Rider__r.LastName": "Last_Name__c",
                      "Rider__r.Email": "Rider_Email__c"
                     }
}
```

**Example POST response body**

```
{
   "jobId": "08PB000000000NA",

   "message": "",
```

```
   "query": "SELECT End_Location_Lat__c, End_Location_Lon__c, End_Time__c,
                    Start_Location_Lat__c, Start_Location_Lon__c, Start_Time__c,
                    Car_Type__c, Rider__r.FirstName, Rider__r.LastName,
                    Rider__r.Email
             FROM Rider_Record__b WHERE Star_Rating__c = '5'",

   "status": "New",

   "targetFieldMap": {"End_Location_Lat__c":"End_Lat__c",
                      "End_Location_Lon__c":"End_Long__c",
                      "Start_Location_Lat__c": "Start_Lat__c",
                      "Start_Location_Lon__c": "Start_Long__c",
                      "End_Time__c": "End_Time__c",
                      "Start_Time__c": "Start_Time__c",
                      "Car_Type__c": "Car_Type__c",
                      "Rider__r.FirstName": "First_Name__c",
                      "Rider__r.LastName": "Last_Name__c",
                      "Rider__r.Email": "Rider_Email__c"
                     },

    "targetObject": "Rider_Reduced__b"
}
```

# Field Audit Trail

Field Audit Trail lets you define a policy to retain archived field history data up to 10 years from the time the data was archived. This feature helps you comply with industry regulations related to audit capability and data retention.

You define a Field Audit Trail policy using the `HistoryRetentionPolicy` object for each object you want to archive. The field history data for that object is then moved from the History related list into the `FieldHistoryArchive` object at periodic intervals, as specified by the policy. For more information, see the Field Audit Trail Implementation Guide.

You can use Async SOQL to query archived fields stored in the `FieldHistoryArchive` object. You can use the `WHERE` clause to filter the query by specifying comparison expressions for the `FieldHistoryType`, `ParentId`, and `CreatedDate` fields.

Note: If platform encryption is enabled on the org, then AsyncSOQL on `FieldHistoryArchive` is not supported.

This example queries archived accounts created within the last month.

**Example URI**

```
https://yourInstance.salesforce.com/services/data/v38.0/async-queries/
```

**Example POST request body**

```
{
   "query": "SELECT ParentId, FieldHistoryType, Field, Id, NewValue, OldValue
             FROM FieldHistoryArchive WHERE FieldHistoryType = 'Account'
             AND CreatedDate > LAST_MONTH",

   "targetObject": "ArchivedAccounts__b",

   "targetFieldMap": {"ParentId": "ParentId__c",
                      "FieldHistoryType": "FieldHistoryType__c",
```

```
                        "Field": "Field__c",
                        "Id": "Id__c",
                        "NewValue": "NewValue__c",
                        "OldValue": "OldValue__c"
                       }
}
```

**Example POST response body**

```
{
              "jobId": "07PB000000006PN",
              "message": "",
             "query": "SELECT ParentId, FieldHistoryType, Field, Id, NewValue, OldValue

            FROM FieldHistoryArchive WHERE FieldHistoryType = 'Account' AND CreatedDate
 > LAST_MONTH",
              "status": "New",
              "targetObject": "ArchivedAccounts__b",
              "targetFieldMap": {"ParentId": "ParentId__c",
              "targetObject": "Rider_Reduced__b" }
              }
```

Note: All number fields returned from a SOQL query of archived objects are in standard notation, not scientific notation, as in the number fields in the entity history of standard objects.

# Real-Time Event Monitoring

Real-Time Event Monitoring enables you to track who is accessing confidential and sensitive data in your Salesforce org. You can view information about individual events or track trends in events to swiftly identify unusual behavior and safeguard your company's data. These features are useful for compliance with regulatory and audit requirements.

With Real-Time Events, you can monitor data accessed through API calls, report executions, and list views. The corresponding event objects are called ApiEvent, ReportEvent, and ListViewEvent. Querying these events covers many common scenarios because more than 50% of SOQL queries occur using the SOAP, REST, or Bulk APIs. Key information about each query—such as the username, user ID, rows processed, queried entities, and source IP address—is stored in the event objects. You can then run SOQL queries on the event objects to find out details of user activity in your org.

For more information, see Real-Time Event Monitoring.

Let's say you've created a custom object called Patent__c that contains sensitive patent information. You want to know when users query this object using any API. Use the following Async SOQL query on the ApiEvent object to determine when Patent__c was last accessed, who accessed it, and what part of it was accessed. The WHERE clause uses the `QueriedEntities` field to narrow the results to just API queries of the Patent__c object.

**Example URI**

```
https://yourInstance.salesforce.com/services/data/v48.0/async-queries/
```

**Example POST request body**

```
{
  "query": "SELECT EventDate, EventIdentifier, QueriedEntities, SourceIp, Username,
UserAgent FROM ApiEvent
            WHERE QueriedEntities LIKE '%Patent__c%'",
  "targetObject": "ApiTarget__c",
```

```
  "targetFieldMap": {
    "EventDate": "EventDate__c",
    "EventIdentifier": "EventIdentifier__c",
    "QueriedEntities": "QueriedEntities__c",
    "SourceIp": "IPAddress__c",
    "Username": "User__c",
    "UserAgent": "UserAgent__c"
  }
}
```

**Example POST response body**

```
{
  "jobId" : "08PB00000066JRfMAM",
  "message" : "",
  "operation" : "INSERT",
  "query" : "SELECT EventDate, EventIdentifier, QueriedEntities, SourceIp, Username,
UserAgent FROM ApiEvent
            WHERE QueriedEntities LIKE &#39;%Patent__c%&#39;",
  "status" : "Complete",
  "targetExternalIdField" : "",
  "targetFieldMap" : {
    "EventDate" : "EventDate__c",
    "SourceIp" : "IPAddress__c",
    "EventIdentifier" : "EventIdentifier__c",
    "QueriedEntities" : "QueriedEntities__c",
    "Username" : "User__c",
    "UserAgent" : "UserAgent__c"
  },
  "targetObject" : "ApiTarget__c",
  "targetValueMap" : { }
}
```

Note: All number fields returned from a SOQL query of archived objects are in standard notation, not scientific notation, as in the number fields in the entity history of standard objects.

If you ask this question on a repeated basis for audit purposes, you can automate the query using a cURL script.

```
curl -H "Content-Type: application/json" -X POST -d
'{"query": "SELECT EventDate, EventIdentifier, QueriedEntities, SourceIp, Username, UserAgent
 FROM ApiEvent WHERE QueriedEntities LIKE '%Patent__c%'",
  "targetObject": "ApiTarget__c",
  "targetFieldMap": {"EventDate": "EventDate__c","EventIdentifier":
"EventIdentifier__c","QueriedEntities": "QueriedEntities__c","SourceIp":
"IPAddress__c","Username": "User__c","UserAgent": "UserAgent__c"}}'
  "https://yourInstance.salesforce.com/services/data/v48.0/async-queries/" -H
  "Authorization: Bearer 00D30000000V88A!ARYAQCZOCeABy29c3dNxRVtv433znH15gLWhLOUv7DVu.
 uAGFhW9WMtGXCul6q.4xVQymfh4Cjxw4APbazT8bnIfxlRvUjDg"
```

Another event monitoring use case is to identify all users who accessed a sensitive field, such as Social Security Number or Email. For example, you can use the following Async SOQL query to determine the users who saw social security numbers.

**Example URI**

```
https://yourInstance.salesforce.com/services/data/v48.0/async-queries/
```

**Example POST request body**

```
{
   "query": "SELECT Query, Username, EventDate, SourceIp FROM ApiEvent
             WHERE Query LIKE '%SSN__c%'",
   "targetObject": "QueryEvents__c",
   "targetFieldMap": {
      "Query":"QueryString__c",
      "Username":"User__c",
      "EventDate":"EventDate__c",
      "SourceIp" : "IPAddress__c"
   }
}
```

**Example POST response body**

```
{
   "jobId": "08PB000000001RS",
   "message": "",
   "query": "SELECT Query, Username, EventDate, SourceIp FROM ApiEvent
             WHERE Query LIKE &#39;%SSN__c%&#39;",
   "status": "Complete",
   "targetFieldMap": {"Query":"QueryString__c", "Username":"User__c",
                      "EventDate":"EventDate__c", "SourceIp" : "IPAddress__c"
                      },
   "targetObject": "QueryEvents__c"
}
```

# SUPPORTED SOQL COMMANDS

Async SOQL supports a subset of commands in the SOQL language. The subset includes the most common commands that are relevant to key use cases.

Note: For details of any command, refer to the SOQL documentation.

## WHERE

Comparison operators

```
=, !=, <, <=, >, >=, LIKE
```

Logical operators

```
AND, OR
```

Date formats

```
YYYY-MM-DD, YYYY-MM-DDThh:mm:ss-hh:mm
```

**Example**

```
SELECT AnnualRevenue
FROM Account
WHERE NumberOfEmployees > 1000 AND ShippingState = 'CA'
```

## Date Functions

Date functions in Async SOQL queries allow you to group or filter data by time periods, such as day or hour.

| Method | Details |
|---|---|
| `DAY_ONLY()` | Returns a date representing the day portion of a dateTime field. |
| `HOUR_IN_DAY()` | Returns a number representing the hour in the day for a dateTime field. |
| `CALENDAR_MONTH()` | Returns a number representing the month for a dateTime field. |
| `CALENDAR_YEAR()` | Returns the year for a dateTime field. |

**Example**

```
SELECT DAY_ONLY(date__c), HOUR_IN_DAY(date__c), COUNT(fieldname__c)
FROM FieldHistoryArchive
GROUP BY DAY_ONLY(date__c), HOUR_IN_DAY(date__c)
```

# Aggregate Functions

```
AVG(field), COUNT(field), COUNT_DISTINCT(field), SUM(field), MIN(field), MAX(field)
```

Note: `MIN()` and `MAX()` do not support picklists.

**Example**

```
SELECT COUNT(field)
FROM FieldHistoryArchive
```

# HAVING

Use this command to filter results from aggregate functions.

**Example**

```
SELECT LeadSource, COUNT(Name)
FROM Lead
GROUP BY LeadSource HAVING COUNT (Name) > 100
```

# GROUP BY

Use this option to avoid iterating through individual query results. Specify a group of records instead of processing many individual records.

**Example**

```
SELECT COUNT(fieldname__c) count, CreatedById createdBy
FROM FieldHistoryArchive
GROUP BY CreatedById
```

# Relationship Queries

Single-level child-to-parent relationships are supported using dot notation. Use these queries with the SELECT, WHERE, and GROUP BY clauses.

**Example**

```
SELECT Account.ShippingState s, COUNT(fieldname__c) c
FROM Contact
GROUP BY Account.ShippingState
```

# Using Aliases with Aggregates

**Examples**

```
{"query":"SELECT COUNT(fieldname__c) c, EventTime t FROM LoginEvent group by EventTime",
 "targetObject":"QueryEvents__c",
```

```
 "targetFieldMap":{"c":"Count__c", "t" : "EventTime__c"}
}
```

```
{"query":"SELECT COUNT(fieldname__c), EventTime FROM LoginEvent group by EventTime",

 "targetObject":"QueryEvents__c",
 "targetFieldMap":{"expr0":"Count__c","EventTime" : "EventTime__c"}
}
```

```
{"query":"SELECT COUNT(fieldname__c) c , firstField__c f FROM SourceObject__c",
 "targetObject":"TargetObject__c",
 "targetFieldMap":{"c":"countTarget__c","f":"secondFieldTarget__c"}
}
```

# INDEX