
Async SOQL Guide

Salesforce, Spring '18



CONTENTS

Async SOQL	1
Running Async SOQL Queries	4
Async SOQL Use Cases	10
Supported SOQL Commands	15
Index	18

ASYNC SOQL

Async SOQL is a method for running SOQL queries when you can't wait for the results in real time. These queries are run in the background over Salesforce entity data, standard objects, custom objects, and big objects. It 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.

EDITIONS

Available in: **Enterprise, Performance, Unlimited,** and **Developer** Editions

 **Note:** Async SOQL for standard and custom objects is in Pilot.

You can schedule multiple queries and monitor their completion status. The limit for Async SOQL queries is one concurrent query at a time.

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?

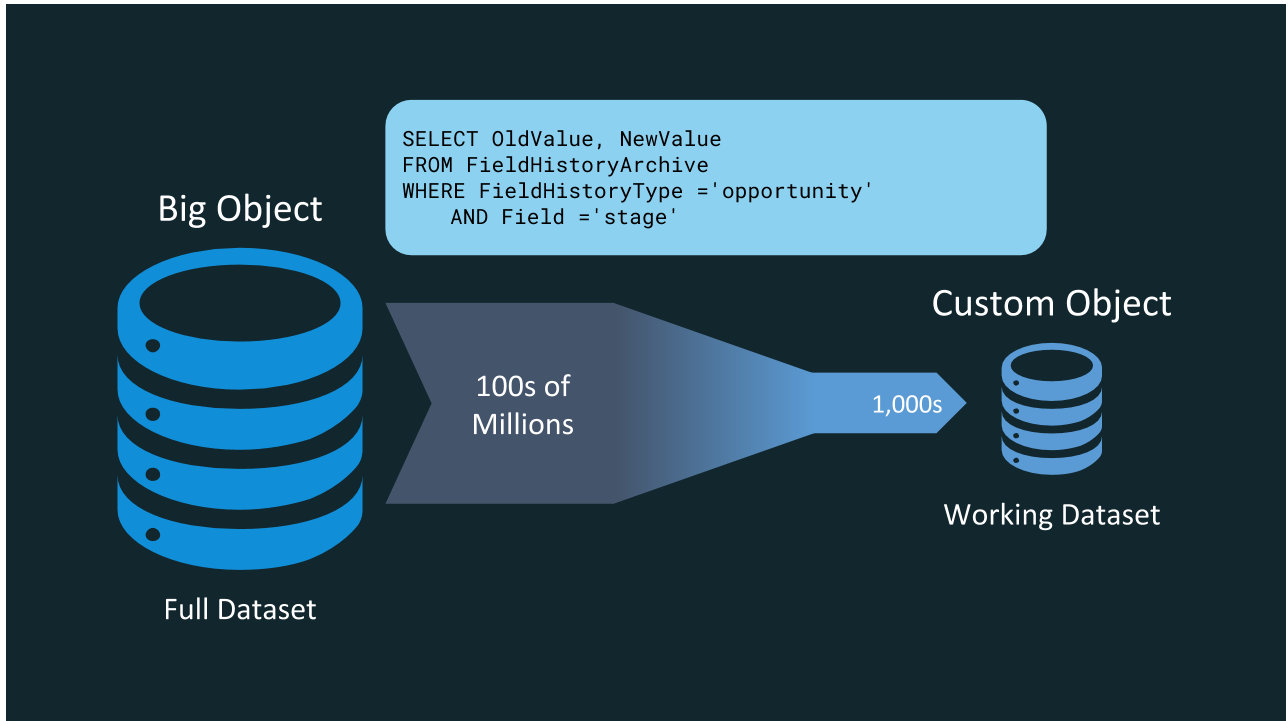
Use standard SOQL when:

- 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 will return a small amount of data.

Use Async SOQL when:

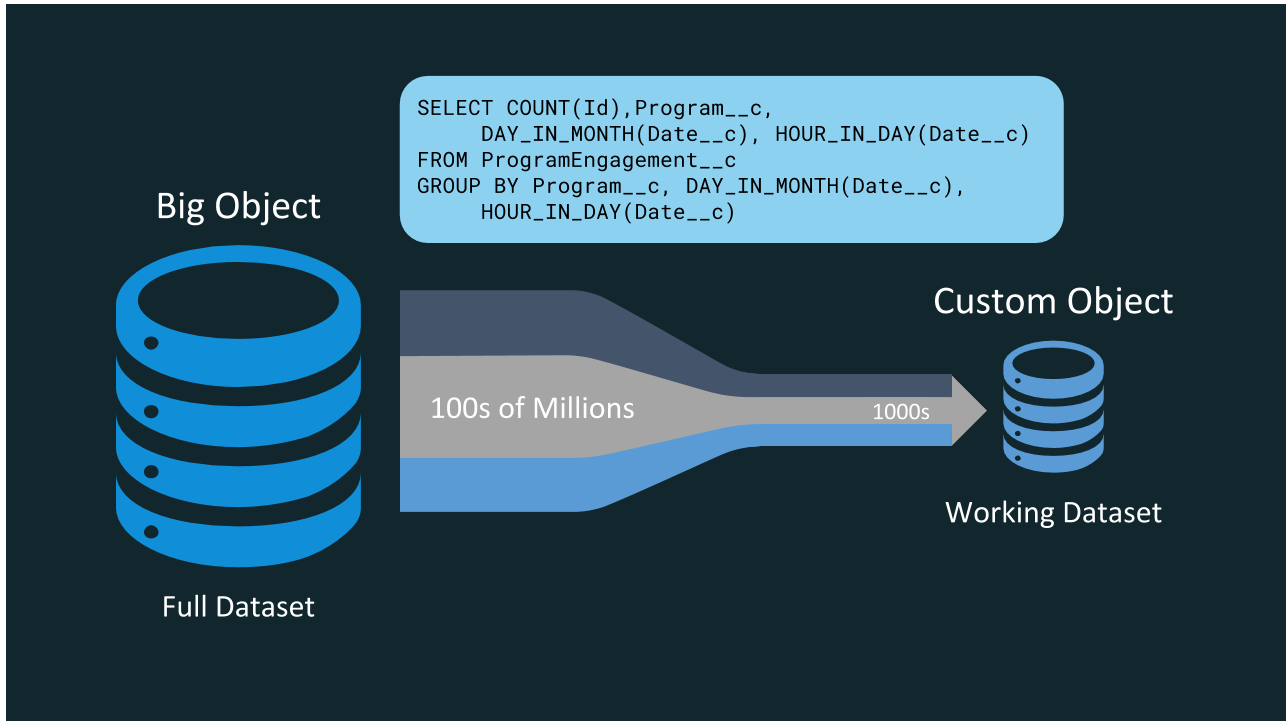
- You are querying against millions of records.
- You want to ensure that your query completes.
- You do not need to do aggregate queries or filtering outside of the index.

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 Force.com 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. With Async SOQL, you can aggregate that data by campaign and day, allowing you 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.



RUNNING ASYNC SOQL QUERIES

Learn how to run Async SOQL queries on your objects and check on the status of your query using the Chatter 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
<code>query</code>	String	Specifies the parameters for the SOQL query you want to execute.	Required	35.0
<code>operation</code>	String	Specify whether the query is an insert or upsert. If the record doesn't exist, an upsert behaves like an insert.  Note: Upsert is not supported for big objects	Optional	39.0
<code>targetObject</code>	String	A standard object, custom object, external object, or big object into which to insert the results of the query.	Required	35.0
<code>targetFieldMap</code>	Map<String, String>	Defines how to map the fields in the query result to the fields in the target object.  Note: When defining the <code>targetFieldMap</code> parameter, make sure that the field type mappings are consistent. If the source and target fields don't match, these considerations apply. <ul style="list-style-type: none">• Any source field can be mapped onto a target text field.• 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.• 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>	<p>Defines how to map static strings to fields in the target object. Any field or alias can be used as the <code>TargetValueMap</code> value in the SELECT clause of a query.</p> <p>You can map the special value, <code>\$JOB_ID</code>, 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.</p> <p>You can also include any field or alias in the SELECT clause of the <code>TargetValueMap</code>. They can be combined together to concatenate a value to be used.</p>	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__c`, a source custom object, and directs the result to `TargetObject__c`, another 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__c",
  "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 <code>targetValueMap</code> when <code>\$JOB_ID</code> is used. To get the status of an async query job, use this ID in an Async Query, Status request (<code>/async-queries/<i>jobId</i></code>).	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.  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.	Big, 35.0	35.0
status	String	Status of an async query job. <ul style="list-style-type: none"> • <code>Canceled</code>—The job was canceled before it could be run. • <code>Success</code>—The job was successfully completed. • <code>Failed</code>—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. • <code>Running</code>—The job is running successfully, and the org hasn't exceeded any limits. • <code>Scheduled</code>—The new job has been created and scheduled, but is not yet running. 	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
<code>targetFieldMap</code>	<code>Map<String, String></code>	<p>Defines how to map the fields in the query result to the fields in the target object.</p> <p> Note: When defining the <code>targetFieldMap</code> parameter, make sure that the field type mappings are consistent. If the source and target fields don't match, these considerations apply.</p> <ul style="list-style-type: none"> Any source field can be mapped onto a target text field. 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. 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
<code>targetValueMap</code>	<code>Map<String, String></code>	<p>Defines how to map static strings to fields in the target object. Any field or alias can be used as the <code>TargetValueMap</code> value in the SELECT clause of a query.</p> <p>You can map the special value, <code>\$JOB_ID</code>, 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.</p> <p>You can also include any field or alias in the SELECT clause of the <code>TargetValueMap</code>. They can be combined together to concatenate a value to be used.</p>	Big, 37.0	37.0
<code>targetObject</code>	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": "08PD0000000003kiT",
  "message": "",
  "query": "SELECT firstField__c, secondField__c FROM SourceObject__c",
  "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__c",
  "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__c",
  "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": "08PB0000000000NA",
  "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, independent of field history tracking. 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, as long as you specify them in that order.

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.

Event Monitoring

Login Forensics and Real-Time Events, currently in pilot, enable 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. This feature is useful for compliance with regulatory and audit requirements.

 **Note:** These features are available to select customers through a pilot program. To be nominated to join this pilot program, contact salesforce.com.

In the current pilot, you can monitor data accessed through API calls, which 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, UserId, UserAgent, and SourceIp, is stored in the ApiEvent object. You can then run SOQL queries on this object to find out details of user activity in your org.

For example, let's say you want to know everyone who viewed the contact record of your company's CEO, Jane Doe. The key to this query is the CEO's contact record ID: 003D000000QYVZ5. (You can also query the ID using SOQL: `SELECT Id FROM Contact WHERE Name = 'Jane Doe'`). You can use the following Async SOQL query to determine all users who saw their contact information, including when, how, and where they saw it.

Example URI

```

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

```

Example POST request body

```

{
    "query": "SELECT Soql, SourceIp, Username, EventTime FROM ApiEvent
WHERE RecordInfo Like '%003D000000QYVZ5%'",
    "targetObject": "QueryEvents__c",
}

```



```

    "targetFieldMap": {"Soql": "QueryString__c", "SourceIp": "IPAddress__c",
                      "Username": "User__c", "EventTime": "EventTime__c",
                      "UserAgent": "UserAgent__c"
    }
}


```

Example POST response body

```

{
  "jobId": "05PB000000001PQ",
  "message": "",
  "query": "SELECT Soql, SourceIp, Username, EventTime
           FROM ApiEvent WHERE RecordInfo Like '%003D000000QYVZ5%'",
  "status": "Complete",
  "targetObject": "QueryEvents__c",
  "targetFieldMap": {"Soql": "QueryString__c", "SourceIp": "IPAddress__c",
                    "Username": "User__c", "EventTime": "EventTime__c", "UserAgent": "UserAgent__c"
  }
}

```

 **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 Soql, SourceIp, UserAgent, Username, EventTime FROM ApiEvent WHERE
RecordInfo Like '%003D000000QYVZ5%'", "targetObject": "QueryEvents__c",
"targetFieldMap": {"Soql": "QueryString__c", "SourceIp": "IPAddress__c", "Username": "User__c",
"EventTime": "EventTime__c", "UserAgent": "UserAgent__c"}}'
"https://yourInstance.salesforce.com/services/data/v35.0/async-queries" -H
"Authorization: Bearer 00D3000000V88A!ARYAQCZOceABy29c3dNxRVtv433znH15gLWhLOUv7DVu.
uAGFhW9WMtGXCul6q.4xVQymfh4Cjxw4APbazT8bnIfx1RvUjDg"

```

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 and the records in which those numbers were exposed.

Example URI

```

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

```

Example POST request body

```

{
  "query": "SELECT Soql, Username, RecordIds, EventTime FROM ApiEvent
           WHERE Soql Like '%SSN__c%'",
  "targetObject": "QueryEvents__c",
  "targetFieldMap": {"Soql": "QueryString__c", "Username": "User__c",
                    "EventTime": "EventTime__c", "RecordIds": "Records_Seen__c"
  }
}

```


```
}  
}
```

Example POST response body

```
{  
  "jobId": "08PB000000001RS",  
  "message": "",  
  "query": "SELECT Soql, Username, RecordIds, EventTime FROM ApiEvent  
           WHERE Soql Like '%SSN__c'",  
  "status": "Complete",  
  "targetFieldMap": {"Soql": "QueryString__c", "Username": "User__c",  
                    "EventTime": "EventTime__c", "RecordIds": "Records_Seen__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(Id)
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(Id) 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(Id) c
FROM Contact
GROUP BY Account.ShippingState
```

Using Aliases with Aggregates

Examples

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

Supported SOQL Commands

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

```
{"query":"SELECT COUNT(Id), EventTime FROM LoginEvent group by EventTime",
"targetObject":"QueryEvents__c",
"targetFieldMap":{"expr0":"Count__c","EventTime" : "EventTime__c"}
}
```

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

INDEX

A

Async SOQL

Aggregate Functions [15](#)

Aliases [15](#)

Async SOQL (*continued*)

Commands [15](#)

Overview [1](#)

Queries [4](#)

Use cases [10](#)