BigObjects Guide

Version 38.0, Winter '17

@salesforcedocs

Last updated: December 9, 2016

CONTENTS

CHAPTER 1 BigObjects

In this chapter ...

BigObjects let you store and manage very large amounts of data on the Salesforce platform.

You can use a BigObject to store very high volumes of customer event history such as weblog data, loyalty data, customer event history or archive data. You create BigObjects by using the Metadata API, and then you can see them in the Setup UI. BigObjects use the suffix `__b` and because they have their own key prefix range they don't affect other object limits. You can insert and query data via the Bulk API and SOAP API.

EDITIONS

Available in: Salesforce Classic

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

BigObjects support the following field types.

- Text
- Number
- DateTime
- Lookup

You can create a Lookup field on a BigObject referring to a standard or custom object so that you can write `JOIN` queries.

The Field Audit Trail feature, based on BigObjects, lets you define a policy to retain archived field history data up to 10 years, independent of field history tracking.

Use Cases

- Data archiving—If you have a large amount of data stored in standard or custom objects in Salesforce, use BigObjects to store historical data.
- Compliance—Retain records on Salesforce for long periods of time to comply with regulatory requirements.
- Analytics—Perform sophisticated analysis on historical data, including correlating it with live data in Salesforce.
- Customer Loyalty—Store and analyze customer data to refine the customer relationship.
- Event Monitoring—Track who is accessing sensitive data in your Salesforce org, to detect unusual behavior and improve security.

Considerations

- Once you define a BigObject's metadata, you can't add, change, or remove fields.
- You can't change data in a BigObject, but you can make a copy with different data.
- Because data in a BigObject never changes, there is no field history and you can't use triggers.
- BigObjects don't support sharing rules. You can use permission sets and field-level security to control who can view BigObjects and who has create and read permissions.

- You can't define a BigObject from a sandbox and deploy to a production environment. BigObjects don't have an In Development state.
- List Views aren't supported for BigObjects.
- Permissions on BigObjects are restrictive by default. You must explicitly grant permissions using permission sets or profiles so that users can create and view BigObjects.
- The only SOQL relationship queries available are based on a lookup field from a BigObject to a standard or custom object.
- You can't insert into BigObjects via Apex.
- BigObjects don't support transactions.
- You can create up to 100 BigObjects per organization. The limits for BigObject fields are similar to the limits on custom objects, and depend on your organization's license type.
- BigObjects don't appear in the Setup UI until they are deployed.
- BigObjects don't appear in Salesforce1.

Note: This feature is available to select customers through a pilot program. To be nominated to join this pilot program, contact salesforce.com. Additional terms and conditions may apply to participate in the pilot program. Please note that pilot programs are subject to change, and as such, we cannot guarantee acceptance into this pilot program or a particular time frame that this feature can be enabled. Any unreleased services or features referenced in this or other press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make their purchase decisions based upon features that are currently available.

Define and Deploy BigObjects

You can define BigObjects by using the Metadata API. After you define and deploy a BigObject, you can view it in the Setup UI.

EDITIONS

Available in: Salesforce Classic

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

Define a BigObject

You define BigObjects by creating XML files that define the fields for each object and the field-level security for each user profile.

- `object` files—create one for each object to define the BigObject's fields.
- `profile` files—create one for each user profile to specify the permissions that the profile has for each field. By default, access to a BigObject is restricted. Use the `profile` file to grant read and edit permissions to users who need them. You can deploy `profile` files separately to specify permissions after the BigObject has been deployed.
- `package` file—create one for the metadata package to specify the contents.

BigObjects are simpler than other custom objects, and only support a subset of the XML used to define them. Use the following tags in a `CustomObject` to define a BigObject.

deploymentStatus
BigObject's deployment status ("Deployed" for all BigObjects)

fields
The definition of a field in the object

label
The object name as displayed in the UI

pluralLabel
The object plural name as displayed in the UI

fullName
The unique API name of a field

label
A field name as displayed in the UI

length
The length of a field in characters (text fields only)

referenceTo
The related object type for a lookup field (lookup fields only)

relationshipName
The name of a relationship as displayed in the UI (lookup fields only)

type
A field type: dateTime, lookup, or text

Example: **Create Metadata Files for Deployment**

The following XML excerpts show how to create metadata files that you can deploy as a package. In this example, each Customer Interaction object represents customer data from a single session in an online video game. A lookup field relates the Customer Interactions to the Account object.

Customer_Interaction__b.object

```
<?xml version="1.0" encoding="UTF-8"?>
<CustomObject xmlns="http://soap.sforce.com/2006/04/metadata">
    <deploymentStatus>Deployed</deploymentStatus>

    <fields>
        <fullName>In_Game_Purchase__c</fullName>
        <label>In-Game Purchase</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>

    <fields>
        <fullName>Level_Achieved__c</fullName>
        <label>Level Achieved</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>

    <fields>
        <fullName>Lives_This_Game__c</fullName>
        <label>Lives Used This Game</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>

    <fields>
        <fullName>Game_Platform__c</fullName>
        <label>Platform</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>

     <fields>
        <fullName>Score_This_Game__c</fullName>
        <label>Score This Game</label>
        <length>16</length>
        <required>false</required>
        <type>Text</type>
        <unique>false</unique>
    </fields>

    <fields>
        <fullName>Account__c</fullName>
        <label>User Account</label>
```

```
        <referenceTo>Account</referenceTo>
        <relationshipName>Game_User_Account</relationshipName>
        <type>Lookup</type>
    </fields>

    <fields>
        <fullName>Play_Date__c</fullName>
        <label>Date of Play</label>
        <required>false</required>
        <type>DateTime</type>
    </fields>

    <fields>
        <fullName>Play_Duration__c</fullName>
        <label>Play Duration</label>
        <required>false</required>
        <type>Number</type>
        <scale>2</scale>
        <precision>18</precision>
    </fields>


    <label>Customer Interaction</label>
    <pluralLabel>Customer Interactions</pluralLabel>
</CustomObject>
```

package.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <types>
        <members>*</members>
        <name>Profile</name>
    </types>
    <version>35.0</version>
</Package>
```

Admin.profile

```
<?xml version="1.0" encoding="UTF-8"?>
<Profile xmlns="http://soap.sforce.com/2006/04/metadata">

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Play_Date__c</field>
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.In_Game_Purchase__c</field>
```

```
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Level_Achieved__c</field>
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Lives_This_Game__c</field>
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Game_Platform__c</field>
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Play_Duration__c</field>
        <readable>true</readable>
    </fieldPermissions>

    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Score_This_Game__c</field>
        <readable>true</readable>
    </fieldPermissions>

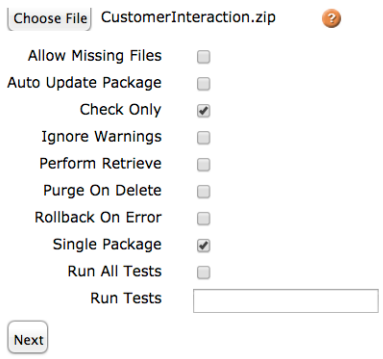
    <fieldPermissions>
        <editable>true</editable>
        <field>Customer_Interaction__b.Account__c</field>
        <readable>true</readable>
    </fieldPermissions>

</Profile>
```

Deploy BigObjects as a Metadata Package

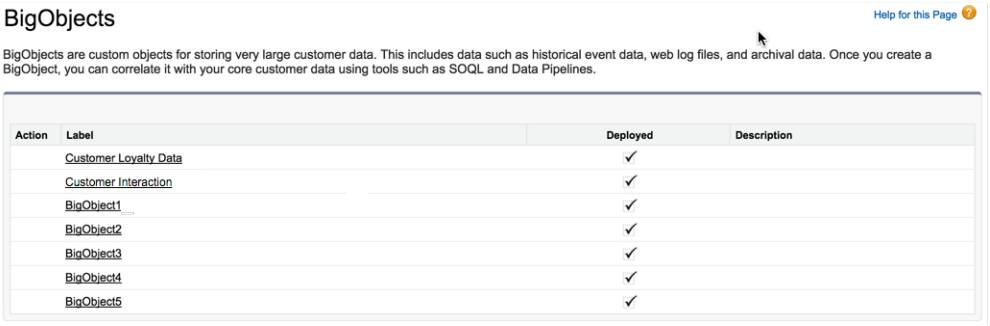
After you create the XML files, you package them as a ZIP file. You then deploy the package to your organization by using Workbench, the Force Migration Tool, or the Metadata API.

Note: Because you can't delete a BigObject after it's created, test to make sure that it's defined correctly before you deploy to a production organization. You can run a test deployment by using the `checkOnly` deployment option. In Workbench, select the **Check Only** option on the Deploy screen.



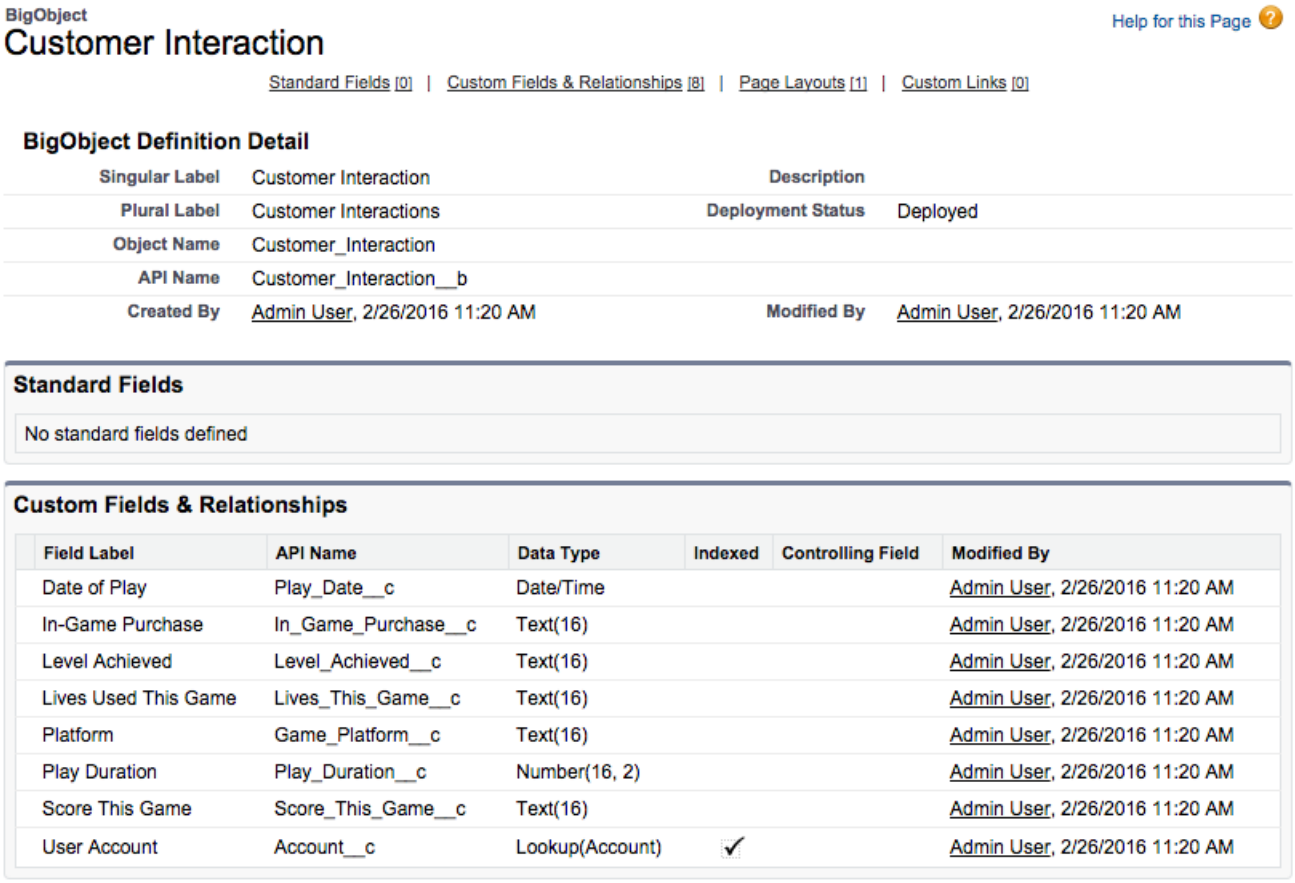
View a BigObject in Setup

After you've deployed your BigObject, you can view it by logging in to your organization and, from Setup, entering `BigObjects` in the `Quick Find` box, then selecting **BigObjects**.



The action column has no **Edit** and **Del** links because after you define a BigObject you can't change it. There are no **New BigObject** or **Schema Builder** buttons because you can only define BigObjects by using XML.

Click the name of a BigObject, to see its fields and relationships.



BigObject
Customer Interaction

Help for this Page

Standard Fields [0] | Custom Fields & Relationships [8] | Page Layouts [1] | Custom Links [0]

BigObject Definition Detail

Singular Label	Customer Interaction	Description	
Plural Label	Customer Interactions	Deployment Status	Deployed
Object Name	Customer_Interaction		
API Name	Customer_Interaction__b		
Created By	Admin User, 2/26/2016 11:20 AM	Modified By	Admin User, 2/26/2016 11:20 AM

Standard Fields

No standard fields defined

Custom Fields & Relationships

Field Label	API Name	Data Type	Indexed	Controlling Field	Modified By
Date of Play	Play_Date__c	Date/Time			Admin User, 2/26/2016 11:20 AM
In-Game Purchase	In_Game_Purchase__c	Text(16)			Admin User, 2/26/2016 11:20 AM
Level Achieved	Level_Achieved__c	Text(16)			Admin User, 2/26/2016 11:20 AM
Lives Used This Game	Lives_This_Game__c	Text(16)			Admin User, 2/26/2016 11:20 AM
Platform	Game_Platform__c	Text(16)			Admin User, 2/26/2016 11:20 AM
Play Duration	Play_Duration__c	Number(16, 2)			Admin User, 2/26/2016 11:20 AM
Score This Game	Score_This_Game__c	Text(16)			Admin User, 2/26/2016 11:20 AM
User Account	Account__c	Lookup(Account)	✓		Admin User, 2/26/2016 11:20 AM

Note: The *scale* and *precision* of a number are the number of digits to the right of the decimal and the total number of digits, respectively. For example, the number 256.99 has a precision of 5 and a scale of 2. When displaying a number field in the UI, its data type is displayed as Number (*length*, *scale*), where *length* = the number of digits to the left of the decimal, so *precision* = *length* + *scale*. For example, if you define a number field with a precision of 18 and a scale of 2, its data type will be displayed as Number(16, 2). This does not mean there was a loss of data, only that the parameters used to define a number field in metadata (*precision* and *scale*) are different from those displayed in the UI (*length* and *scale*).

Populate a BigObject

Using a tool like Workbench is the easiest way to populate a BigObject.

You can use Workbench to load data from a CSV file into a BigObject via the Bulk API. Make sure that the first row in the CSV file contains field labels that you can use to map the CSV data to the fields in the BigObject during import. You can also populate a BigObject via SOAP API or Bulk API.

Note: This feature is available to select customers through a pilot program. To be nominated to join this pilot program, contact salesforce.com.

Table 1: Ingest Rates by API and Batch Size

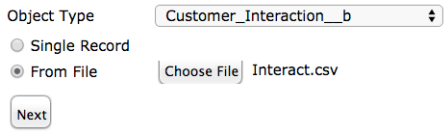
API	Approximate Data Ingest Rates
Bulk API	• 50 fields, 1,000 rows/batch: 2,000 batches/hour

API	Approximate Data Ingest Rates
	• 100 fields, 500 rows/batch: 1,500 batches/hour
SOAP API	• 50 fields, 200 rows/batch: 17,000 batches/hour • 100 fields, 100 rows/batch: 8,500 batches/hour

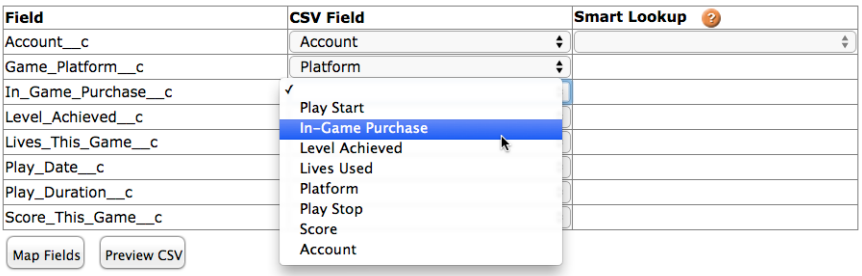
For example, this CSV file contains data for import into a Customer Interaction object.

```
Play Start,In-Game Purchase,Level Achieved,Lives Used,Platform,Play Stop,Score,Account
2015-01-01T23:01:01Z,A12569,57,7,PC,2015-01-02T02:27:01Z,55736,001R000000302D3
2015-01-03T13:22:01Z,B78945,58,7,PC,2015-01-03T15:47:01Z,61209,001R000000302D3
2015-01-04T15:16:01Z,D12156,43,5,iOS,2015-01-04T16:55:01Z,36148,001R000000302D3
```

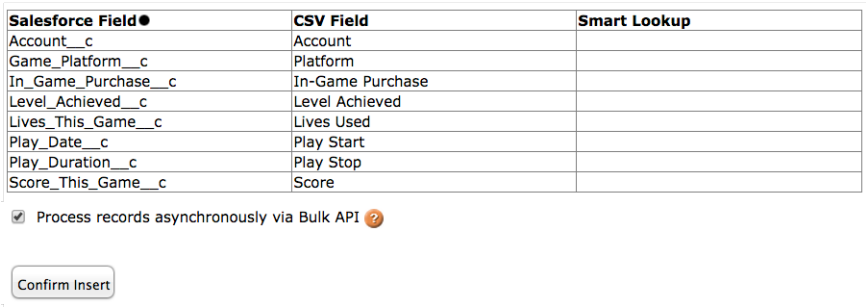
1. Log in to your organization from Workbench.
2. Select **Data** > **Insert** to insert new records.
3. Choose the CSV file containing your data, then click **Next**.



4. Map the fields in the BigObject to the corresponding fields in the CSV file.



5. Confirm that the fields are mapped properly.



6. If you have many records, select **Process records asynchronously via Bulk API**.
7. Click **Confirm Insert**.

Query BigObjects

Query a stored BigObject with a limited set of SOQL keywords.

Query a BigObject by Using SOQL

You can use a subset of SOQL keywords to query a BigObject.

```
SELECT Account__c, Game_Platform__c, Id, In_Game_Purchase__c, Level_Achieved__c,
Lives_This_Game__c,Play_Date__c, Play_Duration__c, Score_This_Game__c FROM
Customer_Interaction__b
```

You can use the `LIMIT` clause to limit the number of returned results. If you don't use the `LIMIT` clause, a maximum of 2,000 results are returned. You can retrieve additional batches of results by using `queryMore()`.

You can use the `WHERE` clause to filter by `Id`.

```
SELECT Account__c, Game_Platform__c, Id, In_Game_Purchase__c, Level_Achieved__c FROM
Customer_Interaction__b WHERE Id = "Z08R00000001IAA"
```

You can query BigObject relationships. For example, the following query returns information about the Account associated with the Customer Interaction BigObject.

```
SELECT Account__c, Game_Platform__c, Account__r.BillingAddress FROM Customer_Interaction__b
```

CHAPTER 2 Async SOQL

In this chapter ...

Async SOQL is a new 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, BigObjects, and external objects (accessed via Salesforce Connect). It provides a convenient way to query large amounts of data stored in Salesforce.

Note: This feature is generally available as a part of the Field Audit Trail product only for queries against the FieldHistoryArchive object. For queries against other objects, this feature is available to select customers through a pilot program. To have this feature enabled for Field Audit Trail or to be nominated to join this pilot program, contact salesforce.com.

EDITIONS

Available in: Salesforce Classic

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

Although many different data object types are supported, the release of the BigObjects feature is what drove the need for Async SOQL. BigObjects makes it possible to keep billions of records on the Force.com platform, orders of magnitude larger than what was previously possible. Async SOQL allows you to combine BigObject data with your core CRM and App Cloud business data.

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, external object, or BigObject.

You can run multiple queries in the background, and monitor their completion status. The limit for Async SOQL queries is 2 concurrent queries, and a maximum of 10 in a 24-hour period.

Async SOQL Versus SOQL

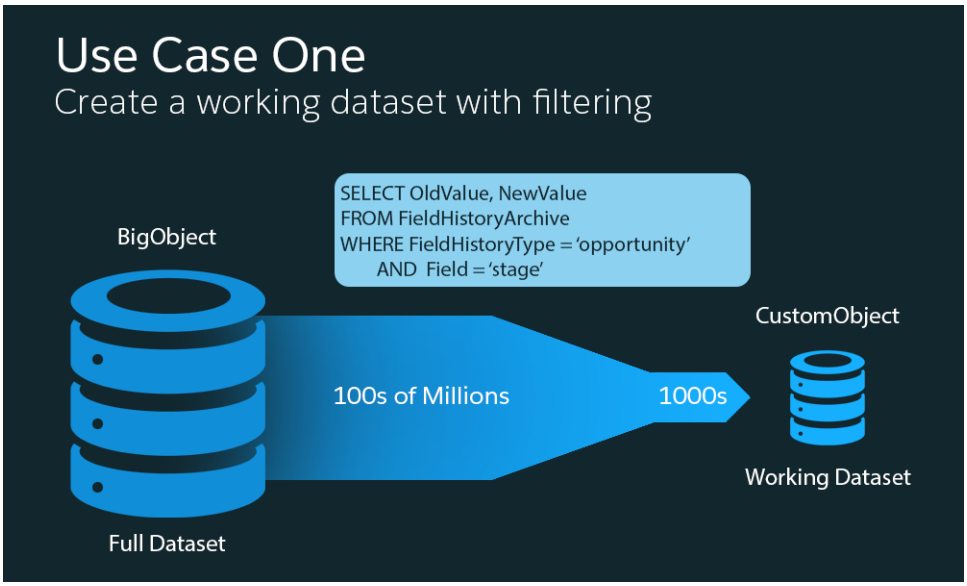
SOQL and Async SOQL provide many of the same capabilities: They are both ways of querying Force.com data. So when would you use an asynchronous SOQL query instead of standard SOQL?

Use standard SOQL when:

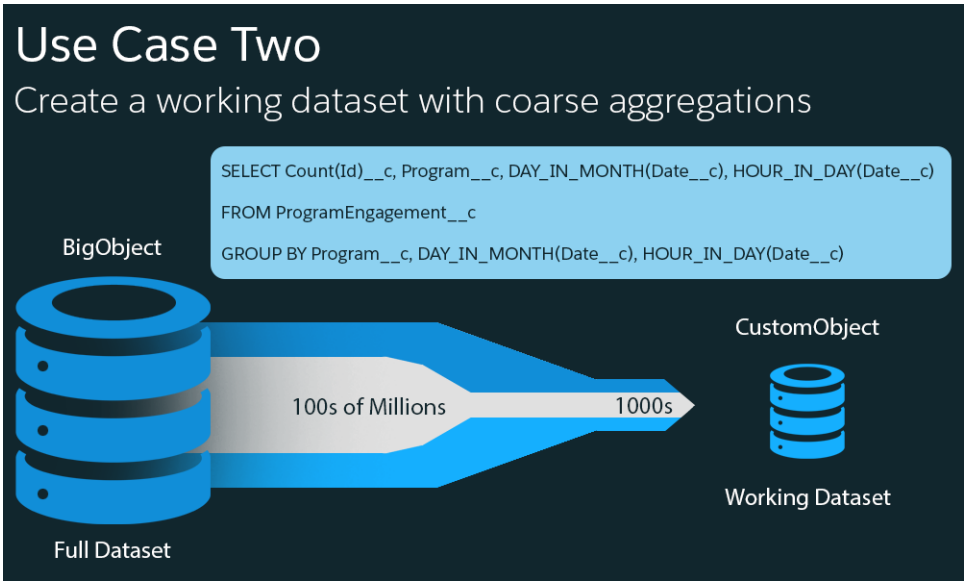
- You want to display the results in the UI without having the user wait for results.
- You want results returned immediately for manipulation within a block of Apex code.
- You know that the volume of data won't cause your query to time out.

Use Async SOQL when:

- You are querying against millions of records and you expect the query to time out.
- You are querying against hundreds of millions of records and you know the query will time out.
- You want to ensure that your query completes reliably.
- You don't need immediate access to the results.



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.



With BigObjects, 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 leveraged in your reports and dashboards.

Running Async SOQL Queries

The Force.com REST API provides a powerful, convenient, and simple web services API for interacting with Force.com. REST API allows for ease of integration and development.

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.

Parameter	Description
`query`	Required. Specifies the parameters for the SOQL query you want to execute. Note: In this pilot, only a limited subset of SOQL commands is supported.
`targetObject`	Required. A standard object, custom object, external object, or BigObject into which to insert the results of the query.
`targetFieldMap`	Required. Defines how to map the fields in the query result to the fields in the target object. 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. • 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.
`targetValueMap`	Optional. Defines how to map static strings to fields in the target object. Note: 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.

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",
```

```
   "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, including the query, `targetObject`, `targetFieldMap`, and `targetValueMap`. It also includes:

Parameter	Description
`jobId`	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.
`status`	The status of the Async SOQL query. See "Tracking the Status of Your Query" for more information.
`message`	A text message that provides information regarding the query, such as an error message if the query failed.

Example POST response body

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

   "message": "",

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

   "status": "New",

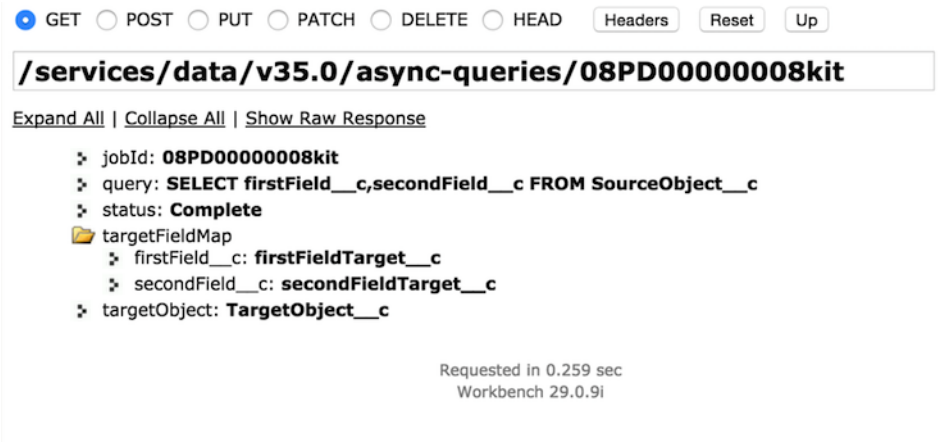
   "targetObject": "TargetObject__c",

   "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

Here's how the previous command looks executed in Workbench.

To track the status of a query, specify its jobID in a GET request. The jobID is the first field in the response.



You can also get more detailed information on a query by specifying the jobID after the BackgroundOperation entity.

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" : "08PD00000000002",
```

```
"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

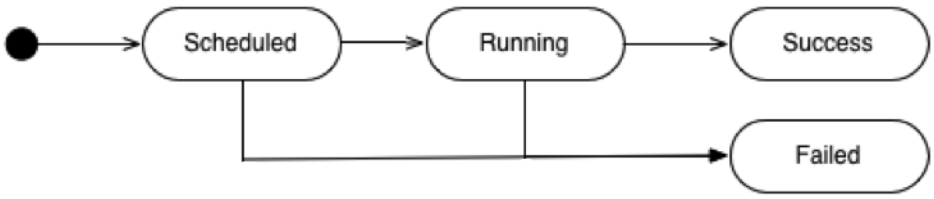
You can cancel a query using an HTTP DELETE request by specifying its jobId.

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

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

States of an Async SOQL Query

Your Async SOQL job goes through several state transitions that allow you to track its progress. The following table describes the possible states for an Async SOQL query.



State	Description
Scheduled	The new job has been created and scheduled, but is not yet running.
Canceled	The job was canceled before it could be run.
Running	The job is running successfully, and the org hasn't exceeded any limits.

State	Description
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.
Success	The job was successfully completed.

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

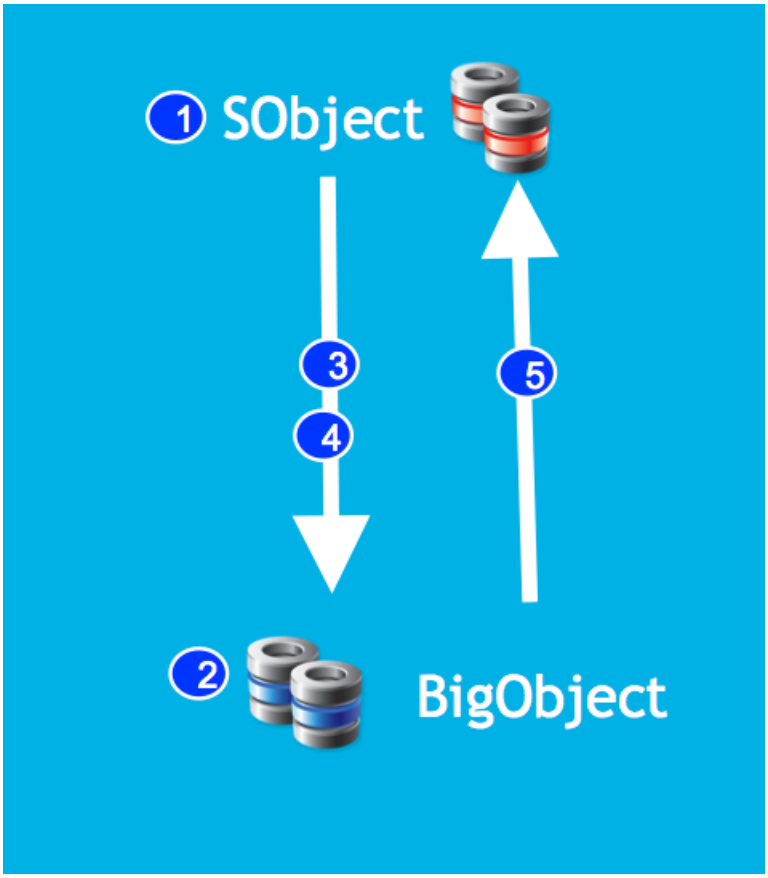
Pilot customers have already successfully implemented key use cases for Async SOQL. This section provides details of some common use cases.

Data Archiving

Data archiving allows companies to regularly archive snapshots of their CRM data in BigObjects for regulatory compliance while keeping the original data in Salesforce.

To archive records for long-term retention in a BigObject, Async SOQL can easily move any number of records from an sObject into a BigObject using this programmatic flow.

1. Define the SOQL query to extract the correct sObject records.
2. Define the target BigObjects.
3. Define the sObject to BigObject field mappings.
4. Use Async SOQL to copy records from the sObject to BigObject storage.
5. Use the same SOQL query to orchestrate the delete process.



Create a job by sending a POST request to the following URI. This example archives all cases with a closed status.

Example URI

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

Example request body

```
{
   "query": "SELECT id, CaseNumber, ClosedDate, ContactId, CreatedById, Description
             FROM Case WHERE Status = 'Closed'",

   "targetObject": "ArchivedClosedCases__b",

   "targetFieldMap": {"id": "originalCaseId__c",
                      "CaseNumber": "caseNumber__c",
                      "ClosedDate": "closedDate__c",
                      "ContactId": "contactId__c",
                      "CreatedById": "createdById__c",
                      "Description": "Description__c"
                     }
}
```

Example response body

```
{
   "jobId": "08PB000000003NS",
```

```
   "message": "",

   "query": "SELECT id, CaseNumber, ClosedDate, ContactId, CreatedById, Description
             FROM Case WHERE Status = 'Closed'",

   "status": "New",

   "targetFieldMap": {"id": "originalCaseId__c",
                      "CaseNumber": "caseNumber__c",
                      "ClosedDate": "closedDate__c",
                      "ContactId": "contactId__c",
                      "CreatedById": "createdById__c",
                      "Description": "Description__c"
                     },

   "targetObject": "ArchivedClosedCases__b"
}
```

Customer 360 Degree and Filtering

In this use case, administrators load a variety of customer engagement data from external sources into Salesforce BigObjects 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 BigObjects 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 BigObject, 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 BigObject 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"
}
```

Note: The metadata files for creating the BigObjects in this example are included in this zip file. Deploy the zip file into your org using Workbench or the Metadata API to try out the example.

Field Audit Trail

Field Audit Trail lets you define a policy to retain archived field history data up to ten 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 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 Data Leakage Detection, 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 particularly 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 organization.

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 her 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 need to 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}}'
"https://yourInstance.salesforce.com/services/data/v35.0/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 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. We plan to support more SOQL commands in future releases.

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

WHERE

- Comparison operators

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

- Logical operators

```
AND, OR
```

- Date formats, such as:

```
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.

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)
```

Example

```
SELECT COUNT(Id) FROM FieldHistoryArchive
```

HAVING

You can 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

You can use this option to avoid iterating through individual query results. You can 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. You can use the 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",
   "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"}
  }
  ```
  ```

CHAPTER 3 Bulk API for BigObjects

In this chapter ...

You can use Bulk API to import large volumes of external data (1 GB or more) into Salesforce BigObjects.

Note: This feature is available to select customers through a pilot program. To be nominated to join this pilot program, contact salesforce.com. Additional terms and conditions may apply to participate in the pilot program.

The two main options for importing data into standard or custom objects, SOAP API and Bulk API, both have limitations at high data volumes.

EDITIONS

Available in: Salesforce Classic

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

SOAP API

SOAP API is optimized for real-time client applications that update a few records at a time. It can be used for processing many records, but is less useful when the data set contains hundreds of thousands of records.

- The processing is synchronous, and can be slow depending on the size of the records and the number of associated triggers and workflow or validation rules.
- Importing such a large volume of data can push past the API limits quickly.

Bulk API

Bulk API is optimized to handle large data loads, from thousands to millions of records. It can import larger volumes more efficiently, in batches, as the processing is done asynchronously. However, it has the following limitations.

- The size of each data file cannot exceed 10 MB. This means 1 GB of data requires uploading 100+ files. Splitting high volumes of data (GB or TB) into such a large number of files is slow and inefficient.
- The number of batches cannot exceed 5,000 in 24 hours, and the processing time per batch is limited to 10 minutes. These limits can be quickly exceeded when importing GB or TB of data.

To overcome these limitations, Bulk API has been enhanced to import data into BigObjects, which are designed to store billions of records.

How Bulk API Works

You process a set of records by creating a *job* that contains one or more *batches*. The job specifies which object is being processed and what type of action is being used (query, insert, upsert, update, or delete). A batch is a set of records sent to the server in an HTTP POST request. Each batch is processed independently by the server, not necessarily in the order it is received. Batches may be processed in parallel. It's up to the client to decide how to divide the entire data set into a suitable number of batches.

A job is represented by the JobInfo resource. This resource is used to create a job, get status for an existing job, and change status for a job. A batch is created by submitting a representation of a set of records in an HTTP POST request.

Processing data typically consists of the following steps:

1. Create a new job that specifies the object and action.
2. Send data to the server in several batches.
3. Once all data has been submitted, close the job. Once closed, no more batches can be sent as part of the job.
4. Check status of all batches at a reasonable interval. Each status check returns the state of each batch.
5. When all batches have either completed or failed, retrieve the result for each batch.
6. Match the result sets with the original data set to determine which records failed and succeeded, and take appropriate action.

At any point in this process, you can abort the job. Aborting a job has the effect of preventing any unprocessed batches from being processed. It doesn't undo the effects of batches already processed.

Note: For details, see the Bulk API Developer's Guide.

How Is Bulk API for BigObjects Different?

The steps for using Bulk API are the same whether you're loading data into a BigObject or a standard or custom object. The key difference is that the size limit of the data file for any batch is increased to 100 MB (instead of 10 MB) for BigObjects. Bulk API automatically adjusts the processing flow, to handle larger data volumes, when the target object is a BigObject.

The following restrictions apply when using Bulk API for BigObjects.

1. A maximum of three batches can run concurrently.
2. All data must be in CSV file format. XML and binary attachments aren't supported.
3. The only supported operation is insert. You can't update, upsert, or delete data.
4. The only concurrency mode supported is parallel.
5. The following headers are not supported in the SOAP request.
   - Batch Retry
   - Line Ending
   - PK Chunking
6. You can only import data into the field types supported for BigObjects, that is, Text, Number, DateTime, and Lookup.
7. The status information is only updated after the job is completed.
8. You can't retrieve the entire data set uploaded, via the Request object. Only a summary of results is provided, listing the total number of records that succeeded or failed.

INDEX

B