Big Objects Implementation Guide

Version 44.0, Winter '19

@salesforcedocs

Last updated: October 4, 2018

© Copyright 2000–2018 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

Chapter 1: Big Objects 1

Big Objects Best Practices 3
Define and Deploy Custom Big Objects 4
Populate a Custom Big Object 10
    Populate a Custom Big Object with Apex 10
Delete Data in a Custom Big Object 11
Big Objects Queueable Example 12
SOQL with Big Objects 14

Chapter 2: Async SOQL 16

Running Async SOQL Queries 19
Async SOQL Use Cases 24
Supported SOQL Commands 29

Index 32

CHAPTER 1 Big Objects

In this chapter ...

- Big Objects Best Practices
- Define and Deploy Custom Big Objects
- Populate a Custom Big Object
- Delete Data in a Custom Big Object
- Big Objects Queueable Example
- SOQL with Big Objects

Big objects store and manage massive amounts of data on the Salesforce platform.

Big objects capture data for use within Lightning Platform and are accessible via a standard set of APIs to clients and external systems. What differentiates big objects is that they are built to provide consistent performance whether there are 1 million records, 100 million, or even 1 billion. This scale is what gives big objects their power and what defines the features that are provided.

There are two types of big objects.

- **Standard big objects**—Objects defined by Salesforce and included in Salesforce products. `FieldHistoryArchive` is a standard big object that stores data as part of the Field Audit Trail product. Standard big objects are available out of the box and cannot be customized.
- **Custom big objects**—New objects that you create to store information unique to your org. Custom big objects extend the functionality that Lightning Platform provides. For example, if you're building an app to track product inventory, create a custom big object called `HistoricalInventoryLevels` to track historical inventory levels for analysis and future optimizations. This implementation guide is for configuring and deploying custom big objects.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

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

Custom Big Object Use Cases

- **360° view of the customer**—Extend your Salesforce data model to include detailed information from loyalty programs, feeds, clicks, billing and provisioning information, and more.
- **Auditing and tracking**—Track and maintain a long-term view of Salesforce or product usage for analysis or compliance purposes.
- **Historical archive**—Maintain access to historical data for analysis or compliance purposes while optimizing the performance of your core CRM or Lightning Platform applications.

Differences Between Big Objects and Other Objects

Because a big object can store data on an unlimited scale, it has different characteristics than other objects, like sObjects. Big objects are also stored in a different part of the Lightning Platform.

Big Objects	sObjects
Horizontally scalable distributed database	Relational database
Non-transactional database	Transactional database

Big Objects	sObjects
Hundreds of millions or even billions of records	Millions of records

These big object behaviors ensure a consistent and scalable experience.

- Big objects support only object and field permissions, not regular or standard sharing rules.
- Features like triggers, flows, processes, and the Salesforce app are not supported on big objects.
- When you insert an identical big object record with the same representation multiple times, only a single record is created so that writes can be idempotent. This behavior is different from an sObject, which creates a record for each request to create an object.

SEE ALSO:

Release Notes: Field History Tracking Data Deleted After 18 Months

Big Objects Best Practices

A big object is unique because of its ability to scale for massive amounts of data.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

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

Considerations When Using Big Objects

- You must use Metadata API to define a big object or add a field to a custom big object.

  Note: To deploy custom big objects through the Setup user interface, you can install a custom Lightning component. See Custom Big Object Creator for details.

  Important: The Custom Big Object Creator is a custom Lightning component available from Salesforce Labs. Salesforce Labs can modify or terminate access to this component at any time. As a result, Salesforce can't guarantee future availability of the Custom Big Object Creator or any of its functionality.

- Big objects support custom Lightning and Visualforce components rather than standard UI elements home pages, detail pages, list views, and so on.
- You can create up to 100 big objects per org. The limits for big object fields are similar to the limits on custom objects and depend on your org's license type.
- You can't use Salesforce Connect external objects to access big objects in another org.

Design with Resiliency in Mind

The big objects database stores billions of records and is a distributed system that favors consistency over availability. The database is designed to ensure row-level consistency.

When working with big data and writing batches of records using APIs or Apex, you can experience a partial batch failure while some records are written and others aren't. Because the database is highly responsive and consistent at scale, this type of behavior is expected. In these cases, simply retry until all records are written.

Keep these principles in mind when working with big objects.

- The best practice when writing to a big object is to have a retry mechanism in place. Retry the batch until you get a successful result from the API or Apex method.

  Tip: To add logging to a custom object and surface errors to users, use the `addError()` method. See An Introduction to Exception Handling.

  Tip: To verify that all records are saved, check the `Database.SaveResult` class. See SaveResult Class Reference.

- Don't try to figure out which records succeeded and which failed. Retry the entire batch.
- Big objects don't support transactions. If attempting to read or write to a big object using a trigger, process, or flow on a sObject, use asynchronous Apex. Asynchronous Apex has features like the `Queueable` interface that isolates DML operations on different sObject types to prevent the mixed DML error.
- Because your client code must retry, use asynchronous Apex to write to a big object. By writing asynchronously, you are better equipped to handle database lifecycle events.

Define and Deploy Custom Big Objects

You can define custom big objects with Metadata API. After you define and deploy a custom big object, you can view it in the Setup UI.

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

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

Define a Custom Big Object

Define a custom big object through the Metadata API by creating XML files that contain its definition, fields, and index.

- `object` files—Create a file for each object to define the custom big object, its fields, and its index.
- `permissionset/profile` files—Create a permissionSet or profile file to specify permissions for each field. These files are not required, but is required to grant access to users. By default, access to a custom big object is restricted.
- `package` file—Create a file for the metadata package to specify the contents.

Note: While custom big objects use the "CustomObject" metadata type, some parameters are unique to big objects and others are not applicable. The specific metadata parameters that apply to big objects are outlined in this document.

Naming Conventions for Custom Big Objects

Object names must be unique across all standard objects, custom objects, external objects, and big objects in the org. In the API, the names of custom big objects have a suffix of two underscores immediately followed by a lowercase "b" (__b). For example, an external object named "HistoricalInventoryLevels" is seen as HistoricalInventoryLevels__b in that organization's WSDL. We recommend that you make object labels unique across all objects in the org - standard, custom, external and big objects.

CustomObject Metadata

Field Name	Field Type	Description
`deploymentStatus`	DeploymentStatus (enumeration of type string)	Custom big object's deployment status (`Deployed` for all big objects)
`fields`	CustomField[]	Definition of a field in the object
`fullName`	string	Unique API name of a field
`indexes`	Index[]	Definition of the index
`label`	string	Object's name as displayed in the UI
`pluralLabel`	string	Field plural name as displayed in the UI

CustomField Metadata

Field Name	Field Type	Description
`fullName`	string	Unique API name of a field.
`label`	string	Field name as displayed in the UI.
`length`	int	Length of a field in characters (Text and LongTextArea fields only).
`pluralLabel`	string	Field plural name as displayed in the UI.
`precision`	int	Number of digits for a number value. For example, the number 256.99 has a precision of 5 (number fields only).
`referenceTo`	string	Related object type for a lookup field (lookup fields only).
`relationshipName`	string	Name of a relationship as displayed in the UI (lookup fields only).
`required`	boolean	Specifies whether the field is required. All fields that are part of the index must be marked as required.
`scale`	int	Number of digits to the right of the decimal point for a number value. For example, the number 256.99 has a scale of 2 (number fields only).
`type`	FieldType	Field type. Supports DateTime, Lookup, Number, Text, and LongTextArea.

Note: Uniqueness is not supported for custom fields.

Index Metadata

Represents an index defined within a custom big object. Use this metadata type to define the composite primary key (index) for a custom big object.

Field Name	Field Type	Description
`fields`	IndexField[]	The definition of the fields in the index.

IndexField Metadata

Defines which fields make up the index, their order, and sort direction. The order in which the fields are defined determines the order fields are listed in the index.

Field Name	Field Type	Description
`name`	string	Required. The API name for the field that's part of the index. This value must match the value of the `fullName` value for the field in the fields section and be marked as required.
`sortDirection`	string	Required. The sort direction of the field in the index. Valid values are `ASC` for ascending order and `DESC` for descending order.

Example: **Create Metadata Files for Deployment**

The following XML excerpts create metadata files that you can deploy as a package. Each Customer Interaction object represents customer data from a single session in an online video game. The `Account__c`, `Game_Platform__c`, and `Play_Date__c` fields define the index, and 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>true</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>
        <required>true</required>
        <type>Lookup</type>
    </fields>

    <fields>
        <fullName>Play_Date__c</fullName>
        <label>Date of Play</label>
        <required>true</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>

    <indexes>
        <fullName>CustomerInteractionsIndex</fullName>
        <label>Customer Interactions Index</label>
        <fields>
            <name>Account__c</name>
            <sortDirection>DESC</sortDirection>
        </fields>
        <fields>
            <name>Game_Platform__c</name>
            <sortDirection>ASC</sortDirection>
        </fields>
        <fields>
        <name>Play_Date__c</name>
        <sortDirection>DESC</sortDirection>
        </fields>
    </indexes>

    <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>PermissionSet</name>
    </types>
    <version>41.0</version>
</Package>
```

Customer_Interaction_BigObject.permissionset

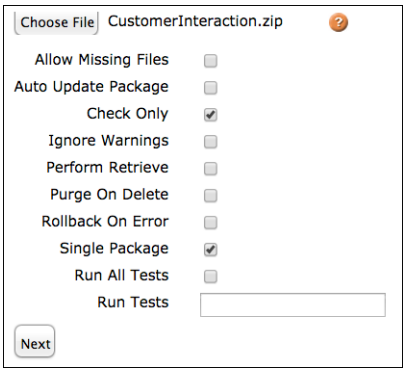
```
<?xml version="1.0" encoding="UTF-8"?>
<PermissionSet xmlns="http://soap.sforce.com/2006/04/metadata">

    <label>Customer Interaction Permission Set</label>

    <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.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>


</PermissionSet>
```

Deploy Custom Big Objects as a Metadata Package

Use the Metadata API to deploy a custom big object. You can use several different tools, like Workbench or the Ant Migration Tool, to deploy. When building a package to deploy a custom big object, make sure the `object` file is in a folder called "objects" and the

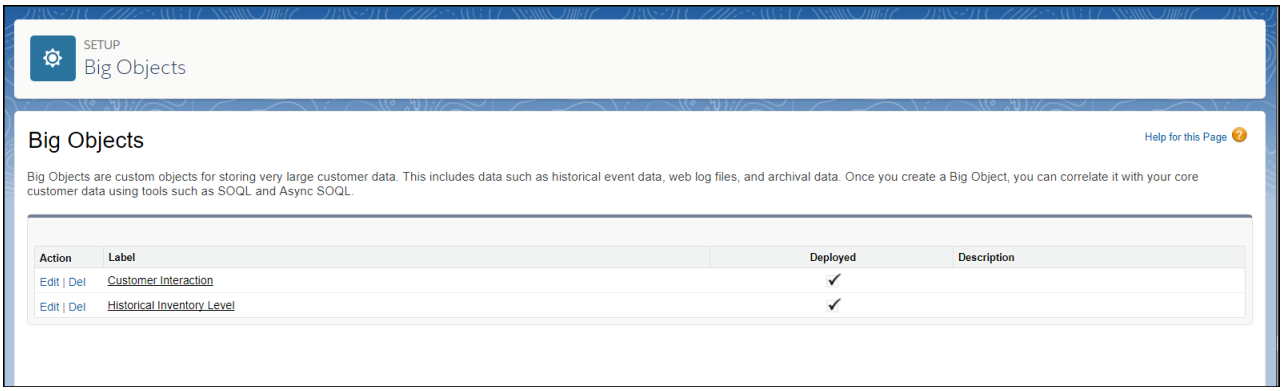
`permissionset` file is in a folder called "permissionsets". `package.xml` must be in the root directory, and not in a folder within the package.

Note: You can run a test deployment by using the `checkOnly` deployment option. In Workbench, select the **Check Only** option on the Deploy screen.

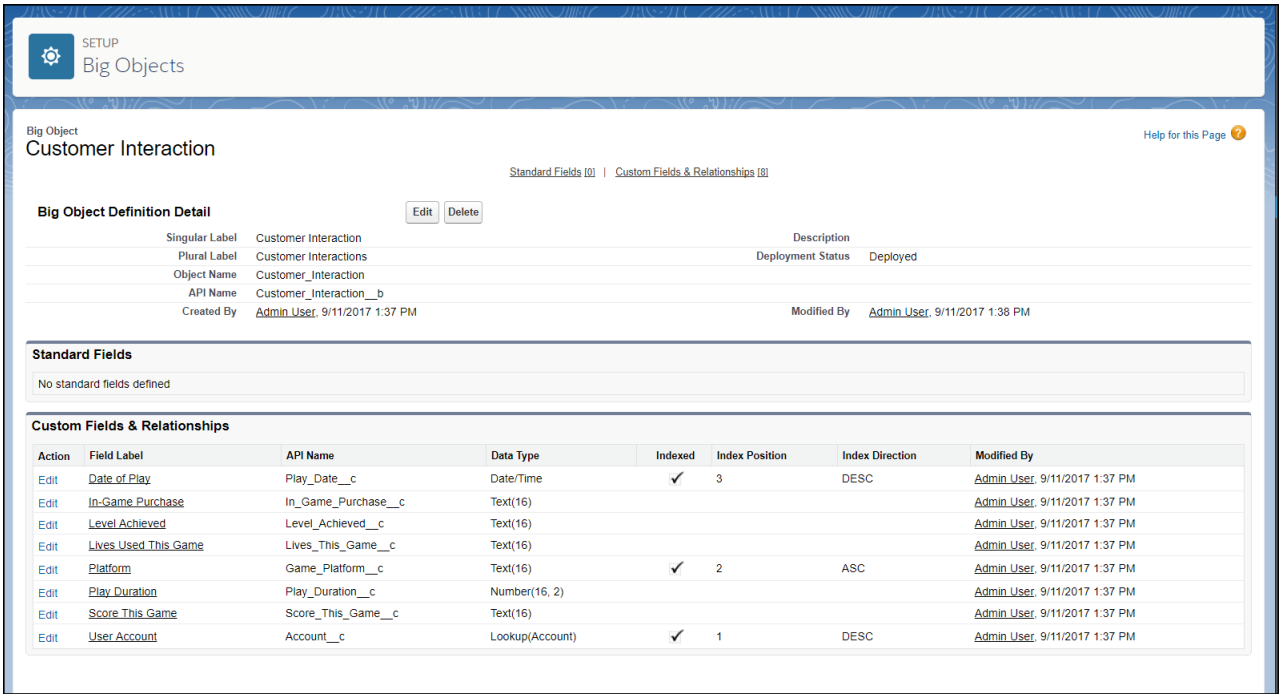


View a Custom Big Object in Setup

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



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



Populate a Custom Big Object

Use Salesforce APIs to populate a custom big object.

You can use a CSV file to load data into a custom big object via SOAP API Bulk API. The first row in the CSV file must contain the field labels used to map the CSV data to the fields in the custom big object during import.

Re-inserting a record with the same index but different data results in behavior similar to an upsert operation. If a record with the index exists, the insert overwrites the index values with the new data. Insertion is idempotent, so inserting data that already exists won't result in duplicates. Reinserting is helpful when uploading millions of records. If an error occurs, the reinsert reuploads the failed uploads without duplicate data. During the reinsertion, if no record exists for the provided index, a new record is inserted.

For example, this CSV file contains data for import into a Customer Interaction big 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
```

Populate a Custom Big Object with Apex

Use Apex to populate a custom big object.

You can create and update custom big object records in Apex using the `insertImmediate` method.

Warning: Apex tests that use mixed DML calls are not allowed and fail. If you write only to the Big Object, the test inserts bad data into the target big object that you have to delete manually. To contain test DML calls to the target big object, use a mocking framework with the batch Apex stub API instead.

Reinserting a record with the same index but different data results in behavior similar to an upsert operation. If a record with the index exists, the insert overwrites the index values with the new data. Insertion is idempotent, so inserting data that exists doesn't result in

duplicates. Reinserting is helpful when uploading millions of records. If an error occurs, the reinsertion reuploads the failed uploads without duplicate data. During the reinsertion, if no record exists for the provided index, a new record is inserted.

Here is an example of an insert operation in Apex that assumes a table in which the index consists of `FirstName__c`, `LastName__c`, and `Address__c`.

```
// Define the record.
PhoneBook__b pb = new PhoneBook__b();
pb.FirstName__c = 'John';
pb.LastName__c = 'Smith';
pb.Address__c = '1 Market St';
pb.PhoneNumber__c = '555-1212';
database.insertImmediate(pb);
// A single record will be created in the big object.
```

```
// Define the record with the same index values but different phone number.
PhoneBook__b pb = new PhoneBook__b();
pb.FirstName__c = 'John';
pb.LastName__c = 'Smith';
pb.Address__c = '1 Market St';
pb.PhoneNumber__c = '415-555-1212';
database.insertImmediate(pb);
// The existing records will be "re-inserted". Only a single record will remain in the big
 object.
```

```
// Define the record with the different index values and different phone number
PhoneBook__b pb = new PhoneBook__b();
pb.FirstName__c = 'John';
pb.LastName__c = 'Smith';
pb.Address__c = 'Salesforce Tower';
pb.PhoneNumber__c = '415-555-1212';
database.insertImmediate(pb);
// A new record will be created leaving two records in the big object.
```

Delete Data in a Custom Big Object

Use Apex or SOAP to delete data in a custom big object.

The Apex method `deleteImmediate()` deletes data in a custom big object. Declare an sObject that contains all the fields in the custom big object's index. The sObject acts like a template. All rows that match the sObject's fields and values are deleted. You can specify only fields that are part of the big object's index. You must specify all fields in the index. You can't include a partially specified index or non-indexed field, and wildcards aren't supported.

In this example, `Account__c`, `Game_Platform__c`, and `Play_Date__c` are part of the custom big object's index. When specifying specific values after the `WHERE` clause, fields must be listed in the order they appear in the index, without any gaps.

```
// Declare sObject using the index of the custom big object -->
List<Customer_Interaction__b> cBO = new List<Customer_Interaction__b>();
cBO.addAll([SELECT Account__c, Game_Platform__c, Play_Date__c FROM Customer_Interaction__b
 WHERE Account__c = '001d000000Ky3xIAB']);

Database.deleteImmediate(cBO);
```

To use the SOAP call `deleteByExample()`, declare an sObject that contains the fields and values to delete. The sObject acts like a template. All rows that match the sObject's fields and values are deleted. You can only specify fields that are part of the big object's

index. All fields in the index must be specified. You can't include a partially specified index or non-indexed field, and wildcards aren't supported. This example deletes all rows in which `Account__c` is 001d000000Ky3xIAB, `Game_Platform__c` is iOS, and `Play_Date__c` is 2017-11-28T19:13:36.000z.

Java example code:

```
public static void main(String[] args) {
  try{
       Customer_Interaction__b[] sObjectsToDelete = new Customer_Interaction__b[1];
       //Declare an sObject that has the values to delete
       Customer_Interaction__b customerBO = new Customer_Interaction__b();
       customerBO.setAccount__c ("001d000000Ky3xIAB");
       customerBO.setGame_Platform__c ("iOS");
       Calendar dt = new GregorianCalendar(2017, 11, 28, 19, 13, 36);
       customerBO.setPlay_Date__c(dt);
       sObjectsToDelete[0] = customerBO;
       DeleteByExampleResult[] result = connection.deleteByExample(sObjectsToDelete);
  }  catch (ConnectionException ce) {
       ce.printStackTrace();
  }
}
```

Note: Repeating a successful `deleteByExample()` operation produces a success result, even if the rows have already been deleted.

Big Objects Queueable Example

To read or write to a big object using a trigger, process, or flow from a sObject, use asynchronous Apex. This example uses the asynchronous Apex `Queueable` interface to isolate DML operations on different sObject types to prevent the mixed DML error.

Example: This trigger occurs when a case record is inserted. It calls a method to insert a batch of big object records and demonstrates a partial failure case in which some records succeed and some fail. To create metadata files for the `Customer_Interaction__b` object in this example, use the XML excerpts in the Create Metadata Files for Deployment on page 6 example.

Tip: To add logging to a custom object and surface errors to users, use the `addError()` method. See An Introduction to Exception Handling.

```
// CaseTrigger.apxt

trigger CaseTrigger on Case (before insert) {
    if (Trigger.operationType ==
TriggerOperation.BEFORE_INSERT){
        // Customer_Interaction__b has three required fields
 in its row key, in this order:
        // 1) Account__c - lookup to Account
        // 2) Game_Platform__c - Text(18)
        // 3) Play_Date__c - Date/Time
        List<Customer_Interaction__b> interactions = new
List<Customer_Interaction__b>();
```

EDITIONS

Available in: both Salesforce Classic and Lightning Experience

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

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

```
        // Assemble the list of big object records to be inserted
        for (Case c : Trigger.new) {
            Customer_Interaction__b ci = new Customer_Interaction__b(
                Account__c = c.AccountId,
                // In this example, the Case object has a custom field, also named
Game_Platform__c
                Game_Platform__c = c.Game_Platform__c,
                Play_Date__c = Date.today()
            );
            interactions.add(ci);
        }

        // CustomerInteractionHandler is an asynchronous queuable Apex class
        CustomerInteractionHandler handler = new
CustomerInteractionHandler(interactions);
        System.enqueueJob(handler);
    }
}
```

The trigger uses the `Queueable` Apex interface to asynchronously call a method to insert into a big object.

```
// CustomerInteractionHandler.apxc

public class CustomerInteractionHandler implements Queueable {

    private List<Customer_Interaction__b> interactions;

    public CustomerInteractionHandler(List<Customer_Interaction__b> interactions) {
        this.interactions = interactions;
    }

    /*
     * Here we insert the Customer Interaction big object records,
     * or log an error if insertion fails.
     */
    public void execute(QueueableContext context){

        List<ExceptionStorage__c> errors = new List<ExceptionStorage__c>();

        try {
            // We have to use insertImmediate() to insert big object records.
           List<Database.SaveResult> srList = Database.insertImmediate(interactions);


            // Check the save results from the bulk insert
            for (Database.SaveResult sr: srList) {
                if (sr.isSuccess()) {
                       System.debug('Successfully inserted Customer Interaction.');
                } else {
                       for (Database.Error err : sr.getErrors()) {
                        // Display an error message if the insert failed
                        System.debug(err.getStatusCode() + ': ' + err.getMessage() +
'; ' +
                                     'Error fields: ' + err.getFields());
```

```
                        // Write to a custom object, such as ExceptionStorage__c
                        // for a more durable record of the failure
                        ExceptionStorage__c es = new ExceptionStorage__c(
                               name = 'Error',
                            ExceptionMessage__c = (err.getMessage()).abbreviate(255),

                              ExceptionType__c = String.valueOf(err.getStatusCode()),

                            ExceptionFields__c =
(String.valueOf(err.getFields())).abbreviate(255)
                        );
                        errors.add(es);
                    }
                }
            }
        }
        catch (Exception e) {
            // Exception occurred, output the exception message
            System.debug('Exception: ' + e.getTypeName() + ', ' + e.getMessage());

            // Write any errors to a custom object as well
            ExceptionStorage__c es = new ExceptionStorage__c(
                   name = 'Exception',
                   ExceptionMessage__c = e.getMessage(),
                   ExceptionType__c = e.getTypeName()
            );
            errors.add(es);
        }

        // If any errors occurred, save the ExceptionStorage records
        if (errors.size() > 0) {
               insert errors;
        }
    }
}
```

SOQL with Big Objects

You can query the fields in a big object's index using a subset of standard SOQL commands.

Build an index query starting from the first field defined in the index, without gaps between the first and last field in the query. You can use =, <, >, <=, or >=, or `IN` on the last field in your query. Any prior fields in your query can only use the = operator. The `!=`, `LIKE`, `NOT IN`, `EXCLUDES`, and `INCLUDES` operators are not valid in any query.

The system fields `CreatedById`, `CreatedDate`, `SystemModstamp` can be included in queries. Do not use the `Id` field in a query. Including `Id` in a query returns only results that have an empty ID (000000000000000 or 000000000000000AAA).

The following queries assume that you have a table in which the index is defined by `LastName__c`, `FirstName__c`, and `PhoneNumber__c`.

This query specifies all three fields in the index. In this case, the filter on `PhoneNumber__c` can be a range.

```
SELECT LastName__c, FirstName__c, PhoneNumber__c
FROM Phone_Book__b
WHERE LastName__c='Kelly' AND FirstName__c='Charlie' AND PhoneNumber__c='2155555555'
```

This query specifies only the first two fields in the index. In this case, the filter on `FirstName__c` can be a range.

```
SELECT LastName__c, FirstName__c, PhoneNumber__c
FROM Phone_Book__b
WHERE LastName__c='Kelly' AND FirstName__c='Charlie'
```

This query specifies only the first field in the index. The filter on `LastName__c` can be a range.

```
SELECT LastName__c, FirstName__c, PhoneNumber__c
FROM Phone_Book__b
WHERE LastName__c='Kelly'
```

This query doesn't work because of a gap in the query where `FirstName__c` is required.

```
SELECT LastName__c, FirstName__c, PhoneNumber__c
FROM Phone_Book__b
WHERE LastName__c='Kelly' AND PhoneNumber__c='2155555555'
```

Note: These restrictions do not apply to Async SOQL queries against big objects.

CHAPTER 2 Async SOQL

In this chapter ...

- Running Async SOQL Queries
- Async SOQL Use Cases
- Supported SOQL Commands

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?

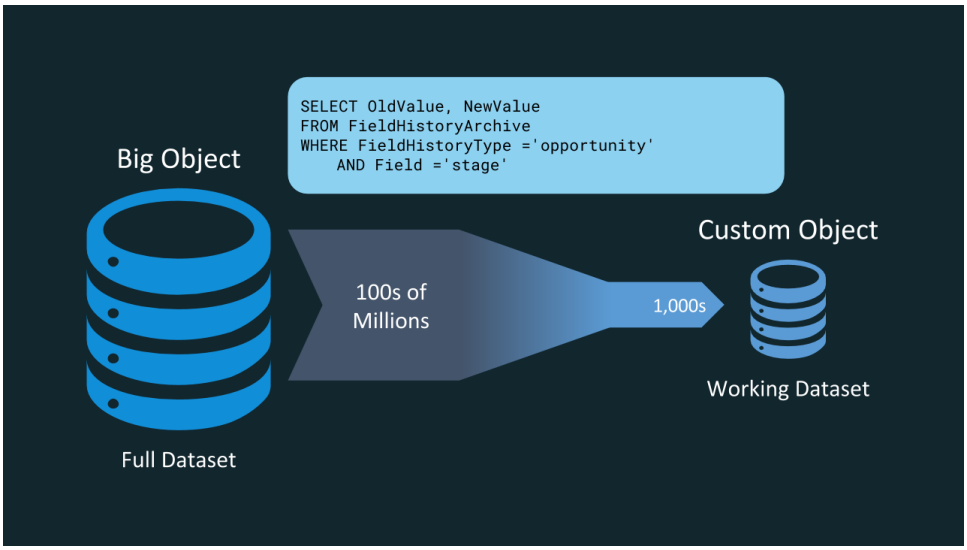
Use standard SOQL when:

- 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:

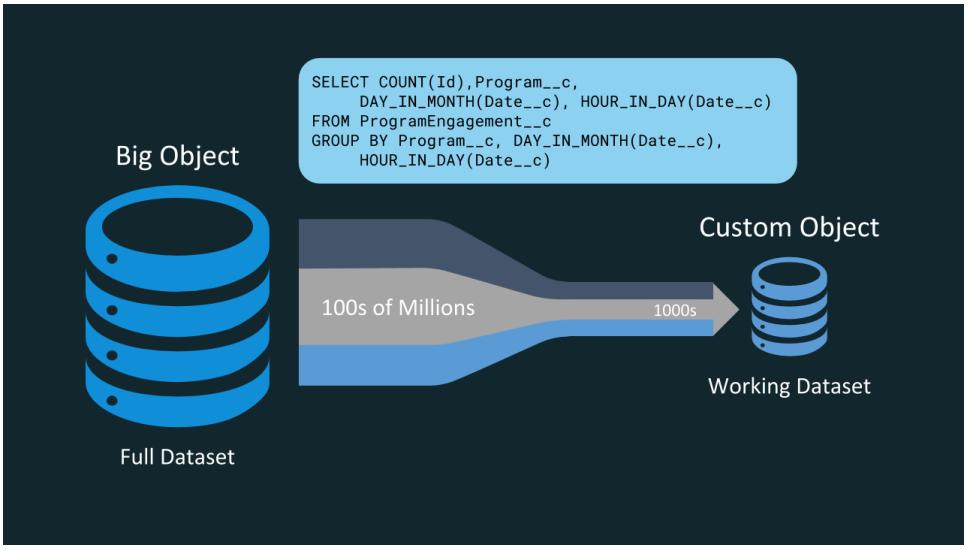
- You are querying against millions of records.
- You want to ensure that your query completes.
- You do not need to do aggregate queries or filtering outside of the index.

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.

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
`query`	String	Specifies the parameters for the SOQL query you want to execute.	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. 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. 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.	Required	35.0

Name	Type	Description	Required or Optional	Available Version
`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. 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. 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__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 `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. 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. • `Canceled`—The job was canceled before it could be run. • `Complete`—The job was successfully completed. • `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. • `Running`—The job is running successfully, and the org hasn't exceeded any limits. • `Scheduled`—The new job has been created and scheduled, but is not yet running. • `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. 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.	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. 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. 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__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" : "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.

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, as long as you specify them in that order.

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.

Event Monitoring

Login Forensics and Real-Time Events 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. These features are useful for compliance with regulatory and audit requirements.

Note: We provide Real-Time Events to selected customers through a pilot program that requires agreement to specific terms and conditions. To be nominated to participate in the program, contact Salesforce. Pilot programs are subject to change, and we can't guarantee acceptance. Feature isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions based only on the generally available products and features.

With Real-Time Events, 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, user ID, browser, and source IP address, 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/v43.0/async-queries/
```

Example POST request body

```
{
   "query": "SELECT Query, SourceIp, Username, EventDate FROM ApiEvent
            WHERE RecordData Like '%003D000000QYVZ5%'",

   "targetObject": "QueryEvents__c",

   "targetFieldMap": {"Query":"QueryString__c","SourceIp":"IPAddress__c",
                      "Username":"User__c", "EventDate":"EventDate__c",
                      "UserAgent":"UserAgent__c"
                      }
}
```

Example POST response body

```
{
   "jobId": "05PB000000001PQ",
```

```
   "message": "",

   "query": "SELECT Query, SourceIp, Username, EventDate
             FROM ApiEvent WHERE RecordData Like '%003D000000QYVZ5%'",

              "status": "Complete",
              "targetObject": "QueryEvents__c",
              "targetFieldMap": {"Query":"QueryString__c","SourceIp":"IPAddress__c",
            "Username":"User__c", "EventDate":"EventDate__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 Query, SourceIp, UserAgent, Username, EventDate FROM ApiEvent WHERE
RecordData Like'%003D000000QYVZ5%'","targetObject": "QueryEvents__c",
"targetFieldMap": {"Query":"QueryString__c", "SourceIp":"IPAddress__c", "Username":"User__c",
 "EventDate":"EventDate__c",UserAgent}}'
"https://yourInstance.salesforce.com/services/data/v43.0/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/v43.0/async-queries/
```

Example POST request body

```
{
   "query": "SELECT Query, Username, RecordData, EventDate FROM ApiEvent
             WHERE Query Like '%SSN__c%'",

   "targetObject": "QueryEvents__c",

   "targetFieldMap": {"Query":"QueryString__c", "Username":"User__c",
                      "EventDate":"EventDate__c", "RecordData":"Records_Seen__c"
                     }
}
```

Example POST response body

```
{
   "jobId": "08PB000000001RS",

   "message": "",

   "query": "SELECT Query, Username, RecordData, EventDate FROM ApiEvent
```

```
            WHERE Query Like '%SSN__c%'",

   "status": "Complete",

   "targetFieldMap": {"Query":"QueryString__c", "Username":"User__c",
                      "EventDate":"EventDate__c", "RecordData":"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",
 "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 29
- Aliases 29
- Commands 29
- Overview 16
- Queries 19
- Use cases 24

B

Big Objects
- @future 12

Big Objects *(continued)*
- Apex 10
- Composite primary key 4
- Considerations 3
- Custom Big Object 4
- Defining 4
- Deleting 11
- Deploying 4
- Example 12
- Index 4
- Overview 1
- Populating 10
- Querying 14