Tips on handling Large Data on Salesforce

BEST PRACTICES

This section lists best practices for achieving good performance in deployments with large data volumes.

The main approaches to performance tuning in large Salesforce deployments rely on reducing the number of records that the system must process. If the number of retrieved records is sufficiently small, the platform might use standard database constructs like indexes or de-normalization to speed up the retrieval of data.

Approaches for reducing the number of records include:

- Reducing scope by writing queries that are narrow or selective.
  For example, if the Account object contains accounts distributed evenly across all states, then a report that summarizes accounts by cities in a single state is much broader—and takes longer to execute—than a report that summarizes accounts by a single city in a single state.
- Reducing the amount of data kept active

For example, if your volume of data is increasing, performance can degrade as time goes by. A policy of archiving or discarding data at the same rate at which it comes into the system can prevent this effect.

Scenario I:

Data Aggregation

Situation

The customer needed to aggregate monthly and yearly metrics using standard reports. The customer's monthly and yearly details were stored in custom objects with four million and nine million records, respectively. The reports were aggregating across millions of records across the two objects, and performance was less than optimal.

Solution

The solution was to create an aggregation custom object that summarized the monthly and yearly values into the required format for the required reports. The reports were then executed from the aggregated custom object. The summarization object was populated using batch Apex.

Scenario II:

Custom Search Functionality

Situation

The customer needed to search in large data volumes across multiple objects using specific values and wildcards. The customer created a custom Visualforce page that would allow the user to enter 1–20 different fields, and then search using SOQL on those combinations of fields.

Search optimization became difficult because:

- When many values were entered, the WHERE clause was large and difficult to tune. When wildcards were introduced, the queries took longer.
- Querying across multiple objects was sometimes required to satisfy the overall search query. This practice resulted in multiple queries occurring, which extended the search.
- SOQL is not always appropriate for all query types.

Solutions

The solutions were to:

- Use only essential search fields to reduce the number of fields that could be searched. Restricting the number of simultaneous fields that could be used during a single search to the common use cases allowed Salesforce to tune with indexes.
- De-normalize the data from the multiple objects into a single custom object to avoid having to make multiple querying calls.
- Dynamically determine the use of SOQL or SOSL to perform the search based on both the number of fields searched and the types of values entered. For example, very specific values (i.e., no wild cards) used SOQL to query, which allowed indexes to enhance performance.

Tip: Searches that contain a leading wildcard are better performed by SOSL than SOQL.

Scenario III:

Indexing with Nulls

Situation

The customer needed to allow nulls in a field and be able to query against them. Because single-column indexes for picklists and foreign key fields exclude rows in which the index column is equal to null, an index could not have been used for the null queries.

Solution

The best practice would have been to not use null values initially. If you find yourself in a similar situation, use some other string, such as N/A, in place of NULL. If you cannot do that, possibly because records already exist in the object with null values, create a formula field that displays text for nulls, and then index that formula field.

For example, assume the Status field is indexed and contains nulls.

Issuing a SOQL query similar to the following prevents the index from being used.

SELECT Name

FROM Object

WHERE Status__c = ”

Instead, you can create a formula called Status_Value.

Status_Value__c = IF(ISBLANK(Status__c), “blank”, Status__c)

This formula field can be indexed and used when you query for a null value.

SELECT Name

FROM Object

WHERE Status_Value__c = ‘blank’

This concept can be extended to encompass multiple fields.

SELECT Name

FROM Object

WHERE Status_Value__c = ” OR Email = ”