Query Plan Tool from Salesforce - A Hidden Gem

Sudipta Deb Saturday, April 28, 2018

Query Plan in the developer console can help to speed up SOQL queries done over large volumes of data. This is such a powerful tool when dealing with bad performance. Developer can use this tool to optimize and speed up SOQL queries.

We can enable Query Plan in the Developer Console by -

Before we go into details, it's better to understand how Salesforce uses indexes while running the SOQL. As we all know it is always good to go for selective query instead of full table scan. But even with selective query, when we are using different filter criteria, it is not always the best SOQL query we are writing. The reason behind is that just writing where clause in the query does not guarantee that the filter is selective. So it is very important to understand which are the fields are good to be used in filter. Let's understand that first in the below section -

All we need to use is the indexes in the filter.

- If we are using filter on standard fields, then we are using index if -
  - it is primary key (Id, Name, OwnerId)
  - it is a foreign key (CreatedById, LastModifiedById, Lookup, Master-Detail)
  - it is an audit field (CreatedDate, SystemModstamp).
- If we are using filter on custom fields, then we are using index if that field is marked as Unique or External Id.

Without indexed filter, the query will not be considered for optimization.

So does that mean if I use filter which is indexed, I am safe and my query is optimized? Answer is Yes and No.

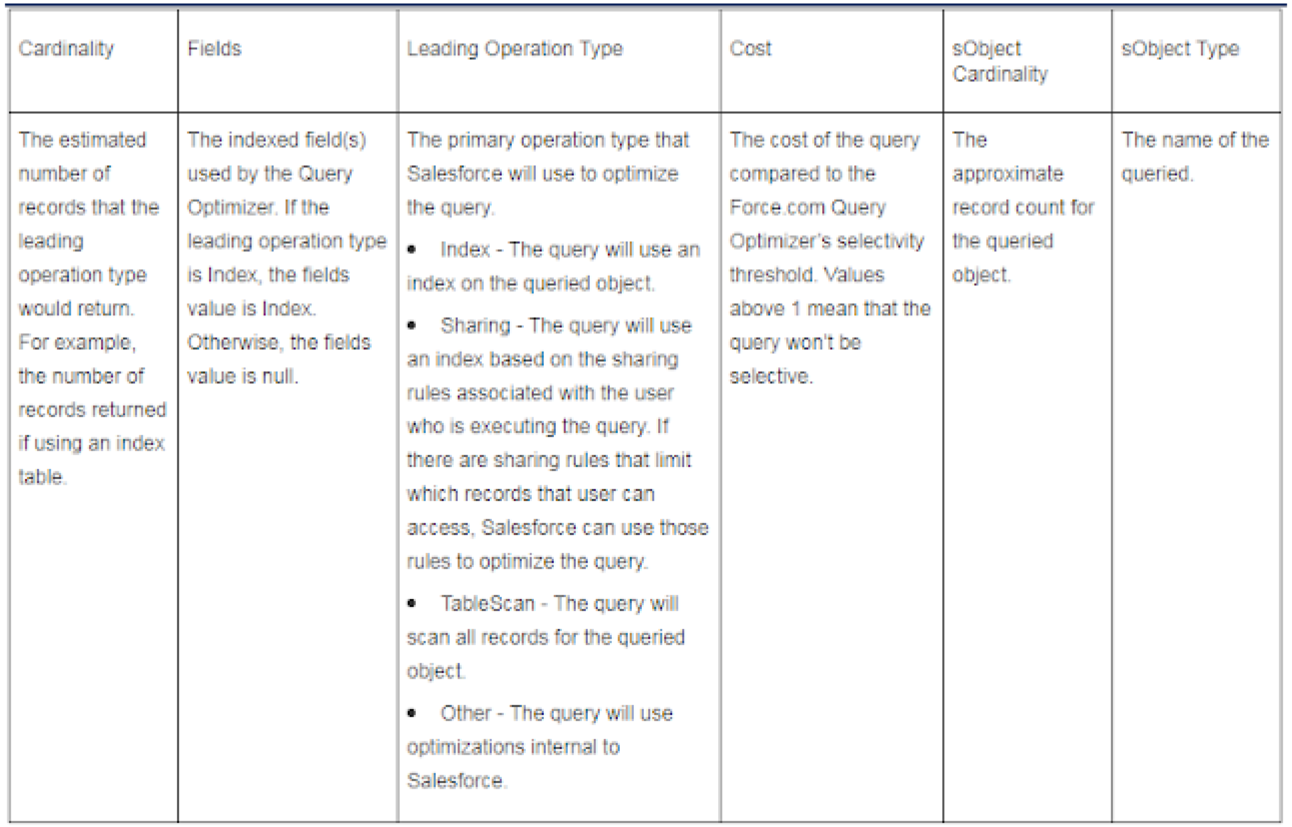
Why Yes, because definitely using indexed field in filter clause will make the query optimized, but there is a catch which I am going to explain now -

When SOQL query is using indexed field, Salesforce will determine how many records it would return. For example -

- For a Standard index, the threshold is 30% of the first million targeted record and 15% of all records after that first million. In addition to that, selective threshold for a standard index maxes out at 1 million (which is only possible if the total number of records is 5.6 million).
- For a Custom index, the threshold is 10% of the first million targeted record and 5% of all records after that first million. In addition to that, selective threshold for a standard index maxes out at 333,333 (which is only possible if the total number of records is 5.6 million).

So if the filter exceeds the threshold, then query will not be considered for optimization.

Query Plan will provide us the below information about the SOQL query -

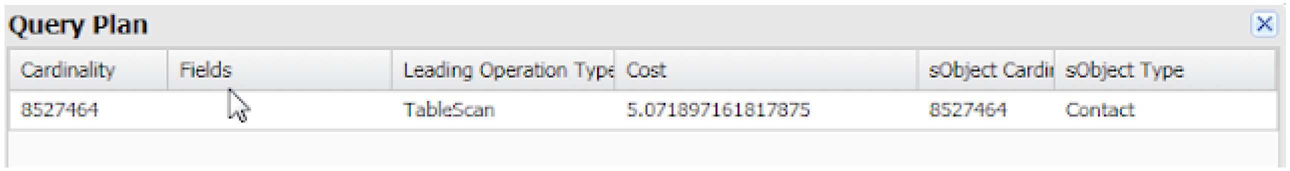


Cardinality	Fields	Leading Operation Type	Cost	sObject Cardinality	sObject Type
The estimated number of records that the leading operation type would return. For example, the number of records returned if using an index table.	The indexed field(s) used by the Query Optimizer. If the leading operation type is Index, the fields value is Index. Otherwise, the fields value is null.	The primary operation type that Salesforce will use to optimize the query. • Index - The query will use an index on the queried object. • Sharing - The query will use an index based on the sharing rules associated with the user who is executing the query. If there are sharing rules that limit which records that user can access, Salesforce can use those rules to optimize the query. • TableScan - The query will scan all records for the queried object. • Other - The query will use optimizations internal to Salesforce.	The cost of the query compared to the Force.com Query Optimizer's selectivity threshold. Values above 1 mean that the query won't be selective.	The approximate record count for the queried object.	The name of the queried.

Now with the above information, let's execute few queries and analyze the result -

Query 1: Select count() from Contact

Result:



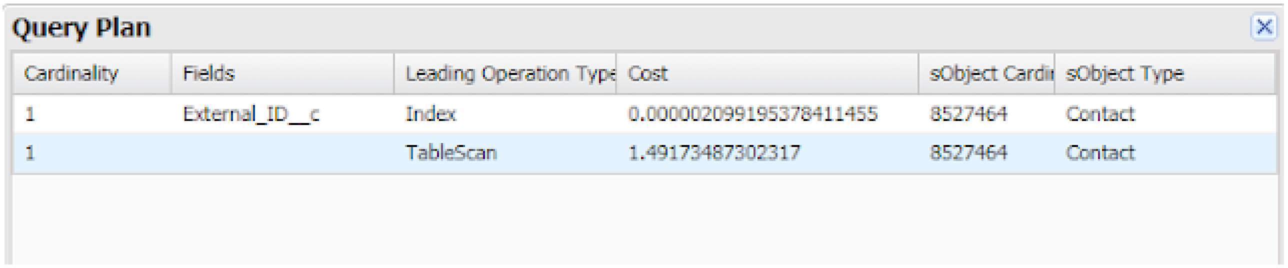
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
8527464		TableScan	5.071897161817875	8527464	Contact

Here I am not using any index field, Field column is blank, Operation Type is TableScan and cost is > 1. Anytime the cost of a SOQL query is greater than 1, it means it is not considered for Optimization.

Query 2: Select count() from Contact Where External_Id__c = '0xP1500100KXFU4512'

Result:



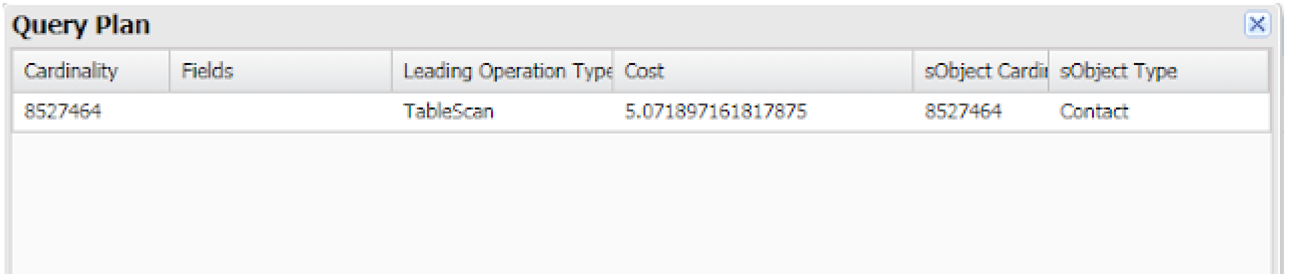
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
1	External_ID__c	Index	0.000002099195378411455	8527464	Contact
1		TableScan	1.49173487302317	8527464	Contact

Here I am using custom index field (External_ID__c) which is there in the Field column, Operation Type is Index and cost is < 1. Since we have one cost which is less than 1, that will be used to optimized the query.

Query 3: Select count() from Contact Where External_Id__c != '0xP1500100KXFU4512'

Result:



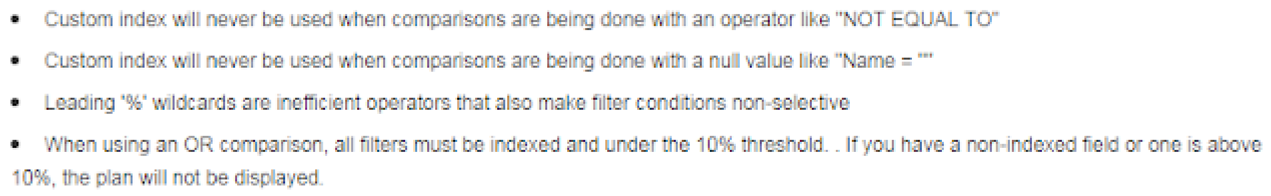
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
8527464		TableScan	5.071897161817875	8527464	Contact

Here I am using custom index field (External_ID__c) but still the field column in blank and Operation Type is TableScan and cost is > 1. So definitely this query is not optimized. But Why??

The reason is -

There are few unsupported operations which will make your query not good enough for optimization. They are as mentioned below -

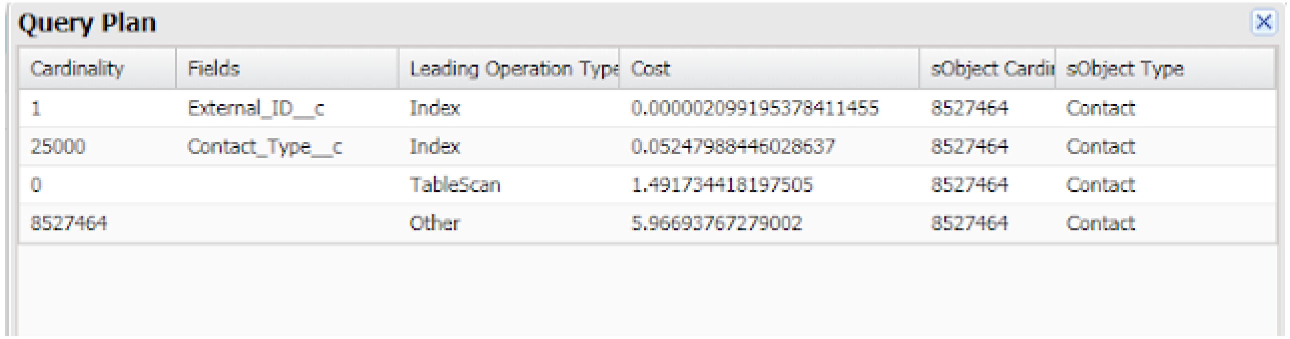


- Custom index will never be used when comparisons are being done with an operator like "NOT EQUAL TO"
- Custom index will never be used when comparisons are being done with a null value like "Name = ''"
- Leading '%' wildcards are inefficient operators that also make filter conditions non-selective
- When using an OR comparison, all filters must be indexed and under the 10% threshold. . If you have a non-indexed field or one is above 10%, the plan will not be displayed.

Query 4: Select count() from Contact Where External_Id__c = '0xP1500100KXFU4512' and

Contact_Type__c = 'Customer'

Result:



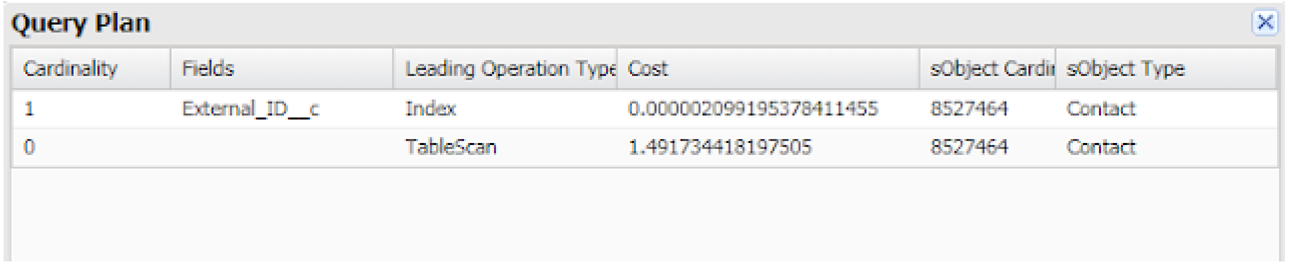
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
1	External_ID__c	Index	0.00000209195378411455	8527464	Contact
25000	Contact_Type__c	Index	0.05247988446028637	8527464	Contact
0		TableScan	1.491734418197505	8527464	Contact
8527464		Other	5.96693767279002	8527464	Contact

Here I have used second indexed field(Contact_Type__c) in the query. In this scenario, Salesforce will select the plan with lowest cost.

Query 5: Select count() from Contact Where External_Id__c = '0xP1500100KXFU4512' And Client_Status__c = 'Active'

Result:



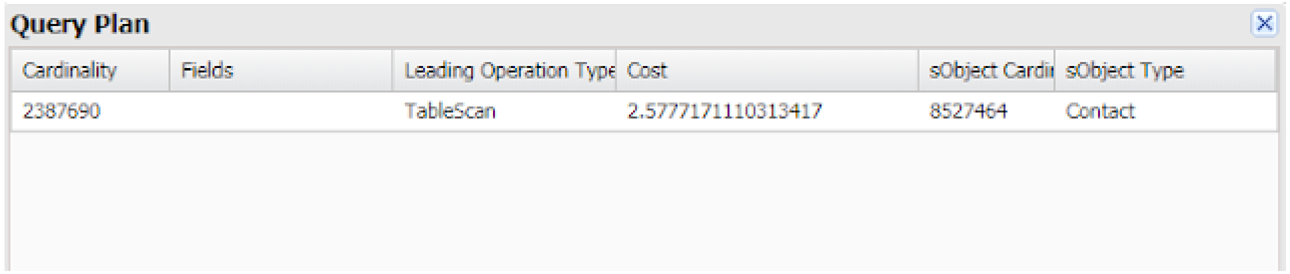
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
1	External_ID__c	Index	0.00000209195378411455	8527464	Contact
0		TableScan	1.491734418197505	8527464	Contact

Here I have used second non-indexed field(Client_Status__c) in the query. That is why only one indexed field in the result.

Query 6: Select count() from Contact Where External_Id__c = '0xP1500100KXFU4512' Or Client_Status__c = 'Active'

Result:



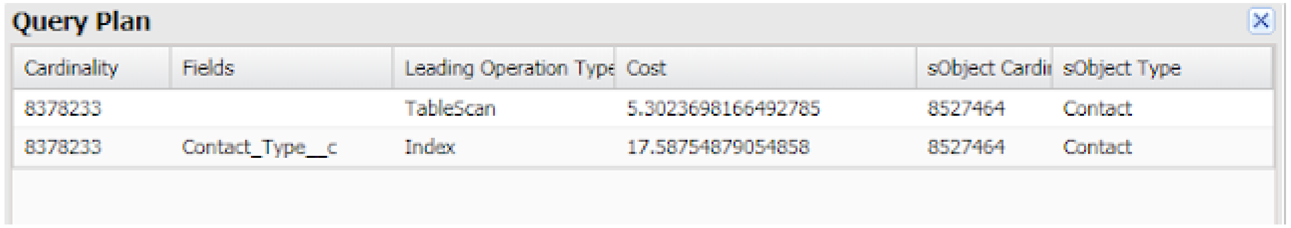
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
2387690		TableScan	2.5777171110313417	8527464	Contact

Here I have used second non-indexed field(Client_Status__c) in the query with or condition.When using OR condition, all filters must be indexed and under the 10% threshold.

Query 7: Select count() from Contact Where Contact_Type__c = 'Customer'

Result:



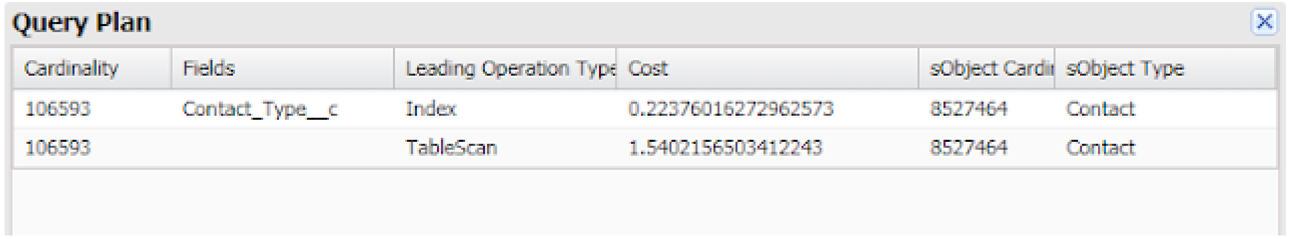
Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
8378233		TableScan	5.3023698166492785	8527464	Contact
8378233	Contact_Type__c	Index	17.58754879054858	8527464	Contact

Here even though I have used indexed field(contact_type__c), but still the query is not optimized as you can see the lowest cost is also greater than 1. The reason is that here the finding variable 'Customer' is resulting more than 10% of the full table.

Query 8: Select count() from Contact Where Contact_Type__c = 'Advisor'

Result:



Query Plan

Cardinality	Fields	Leading Operation Type	Cost	sObject Cardi	sObject Type
106593	Contact_Type__c	Index	0.22376016272962573	8527464	Contact
106593		TableScan	1.5402156503412243	8527464	Contact

Here I have changed the binding variable to 'Advisor' and resulting data count is less than 10% of the full table. That is why this query is optimized.