RELATIONSHIP QUERIES

By using relationship queries, we can fetch the records from multiple Salesforce related objects. i.e., instead of using a separate SOQL query to fetch the records from each object, we can use a single SOQL Query to fetch the records from both Parent and Child Objects, hence the number of SOQL queries used in a transaction are reduced and the application performance can be improved.

To use Relationship Queries, both the objects should be associated with either Lookup/ Master-Detail association.

We have the following scenarios in Relationship queries.

1. Parent to Child (Standard Association)

2. Parent to Child (Custom Association)

3. Child to Parent (Standard Association)

4. Child to Parent (Custom Association)

Parent to Child (Standard Association)

In this scenario, we can fetch the parent record along with the related child records from the child objects, which are associated with the "Standard Associations" (i.e. Relations are mapped by Salesforce by default).

We can implement this feature with the help of "Sub-Queries / Inner Queries".

While preparing the Sub-Query, we have to represent the "Child Object Name" in "Plural Format".

Syntax

```
[ Select <Field1>, <Field2>, <Field3>,..., <FieldN>,
  (Select <field1>,<field2>,<field3>,...,<fieldN> from <ChildObject1PluralFormat>),
  (Select <field1>,<field2>,<field3>,...,<fieldN> from <ChildObject2PluralFormat>),
  (Select <field1>,<field2>,<field3>,...,<fieldN> from <ChildObject3PluralFormat>),
        .....
        .....
from <ParentObjectName>]
```

Salesforce supports only one level of Sub-Queries. We can't write a Sub-Query inside another Sub-Query.

Use Case: Write an apex program to fetch Account Record data based on the specified Account Name at runtime and fetch the related contacts, opportunities and case records and represent them on the Debug Log File.

Class Code:

```
public class RelatedRecordsHelper
{
Public static void GetAccountAndRelatedRecords(String accountRecordName)
  {
          if(accountRecordName != Null && accountRecordName != '')
        {
List<Account> lstAccounts = [Select id, name, rating, industry, annualrevenue, active__C,
                              (Select id, firstName, lastName, Email, Phone, fax, title from Contacts),
                               (Select id, name, amount, stagename, closedate from Opportunities),
                               (Select id, caseNumber, status, priority, origin, reason from Cases)
                                              from Account  Where name =: accountRecordName];
          if(! lstAccounts.isEmpty())
           {
              for(Account accRecord : lstAccounts)
              {
                  System.debug('Account Record Id is....: '+ accRecord.Id);
                  System.debug('Account Record Name is....: '+ accRecord.Name);
                  System.debug('Rating Value is.....: '+ accRecord.Rating);
                  System.debug('Industry Name is.....: '+ accrecord.Industry);
                  System.debug('Annual Revenue is....: '+ accRecord.AnnualRevenue);
                  System.debug('Active Status is.....: '+ accRecord.Active__c);
                  System.debug('-----------------------------------------');
                    List<Contact> lstContacts =   accRecord.Contacts;
                  System.debug('Contact Records Count.....: '+ lstContacts.size());
                     if(! lstContacts.isEmpty())
                     {
                         for(Contact con : lstContacts)
                         {
                             System.debug('Contact Record.....: '+ con);
                         }
                     }
                  System.debug('-----------------------------------------');
                  List<Opportunity> lstOppty = accRecord.Opportunities;
                  System.debug('Number of Opportunity Records.....: '+ lstOppty.Size());
                    if(! lstOppty.isEmpty())
                     {
                         for(Opportunity oppty : lstOppty)
                         {
```

```
                        System.debug('Opportunity Record is....: '+ oppty);
                    }
                }
         System.debug('----------------------------------------');
          List<Case> lstCases = accRecord.Cases;
          System.debug('Number of Case Records........: '+ lstCases.Size());
                if(! lstCases.isEmpty())
                {
                    for(Case cs : lstCases)
                    {
                        System.debug('Case Record is....: '+ cs);
                    }
                }
            }
        }
    }
    System.debug('Number of SOQL Queries Used.....: '+ System.Limits.getQueries());
    System.debug('Number of Records Retrieved.....: '+ System.Limits.getQueryRows());
  }
}
```

Execution

RelatedRecordsHelper.GetAccountAndRelatedRecords('United Oil & Gas Corp.');

Parent to Child (Custom Association)

In this Scenario, we can fetch the parent record along with the related child records from the child objects which are associated with the Custom Association (i.e. Relations are mapped by Salesforce Developer / Administrator).

We can implement this feature with the help of "Sub-Queries / Inner Queries".

While preparing the Sub-Query, we have to represent the Child Object Name in Plural Format and post-fixed with __r.

Syntax

```
        [ Select <Field1>, <Field2>, <Field3>,..., <FieldN>,
          (Select <field1>,<field2>,<field3>,...,<fieldN> from <Child1PluralFormat>__r),
          (Select <field1>,<field2>,<field3>,...,<fieldN> from <Child2PluralFormat>__r),
          (Select <field1>,<field2>,<field3>,...,<fieldN> from <Child3PluralFormat>__r),
                .....
                .....
        from <ParentObjectName>]
```

Salesforce supports only one level of Sub-Queries. We can't write a "Sub-Query" inside another Sub-Query.

Use Case: Write an apex program, to fetch the Hiring Manager Record based on the specified Name and fetch the Related Position Records from the Child Object.

Child to Parent (Standard Relation):

In this scenario, we can fetch the Child Record Data along with associated Parent Records by traversing up to 10 Levels.

Child ---> Parent ---> Grand Parent ---> Grand Grand Parent.....10 Levels

While referencing the Parent Records use the below syntax.

Syntax:
```
[ Select <field1>, <field2>, <field3>,....,<fieldN>,
        <parentName>.<fieldName>, <parentName>.<fieldName>,
        <parentName>.<grandParent>.<fieldName>,
        <parentName>.<grandParent>.<grandGrandParent>.<fieldName>
                ...... upto 1o levels
        from <ChildObjectName>]
```

Use Case: Write an Apex program, to fetch the Case Records based on the specified case number at runtime and fetch the associated Parent Record details.

Class Code:

```
public class RelatedRecordsHelper
{
   Public static void GetCaseDetails(String caseRecordNumber)
   {
      if(caseRecordNumber != Null && caseRecordNumber != '')
      {
         List<Case> lstCases = [Select id, caseNumber, status, priority, origin, reason, type, subject,
                  Contact.FirstName, Contact.LastName, Contact.Email,
                  Contact.Account.Name, Contact.Account.Rating, Contact.Account.Industry,
                  Contact.Account.Owner.FirstName, Contact.Account.Owner.LastName,
                 Contact.Account.Owner.Profile.Name,
                  Contact.Account.Owner.Profile.UserLicense.Name
                        from Case       Where caseNumber =: caseRecordNumber];
         System.debug('Number of Case Records Returned....: '+ lstCases.Size());
         if(! lstCases.isEmpty())
```

```
            {
                for(Case csRecord : lstCases)
                {
                    System.debug('Case Record Id is....: '+ csRecord.Id);
                    System.debug('Case Status is........: '+ csRecord.Status);
                    System.debug('Case Priority is......: '+ csRecord.Priority);
                      System.debug('Case Type is..........: '+ csRecord.Type);
                    System.debug('Case Origin is........: '+ csRecord.Origin);
                    System.debug('Case Reason is.........: '+ csRecord.Reason);
                    System.debug('Case Subject is........: '+ csRecord.Subject);
   System.debug('Contact Person Name is.....: '+ csRecord.Contact.FirstName + ' '+ csRecord.Contact.LastName);
        System.debug('Contact Person Email is....: '+ csRecord.Contact.Email);
         System.debug('Account Record Name is.....: '+ csRecord.Contact.Account.Name);
        System.debug('Account Rating is.......: '+ csRecord.Contact.Account.Rating);
        System.debug('Account Industry is.....: '+ csRecord.Contact.Account.Industry);
 System.debug('Account Owner Name is......: '+ csRecord.Contact.Account.Owner.FirstName + ' '+ csRecord.Contact.Account.Owner.LastName);
 System.debug('Owner Profile Name is.....: '+ csRecord.Contact.Account.Owner.Profile.Name);
 System.debug('Owner License Name is: '+ csRecord.Contact.Account.Owner.Profile.UserLicense.Name);
                }
            }
        }
    }
}
```

Execution: RelatedRecordsHelper.GetCaseDetails('00001016');

Child to Parent (Custom Relation)

In this scenario, we can fetch the Child Record data along with associated Parent Records by traversing up to levels.

Child ---> Parent ---> Grand Parent ---> Grand Grand Paren.....10 Levels

While referencing the Parent Records, we have to use the below syntax.

Syntax:

```
[ Select <field1>, <field2>, <field3>....,<fieldN>,
          <parentName>__r.<fieldName>, <parentName>__r.<fieldName>,
          <parentName>__r.<grandParent>__r.<fieldName>,
          <parentName>__r.<grandParent>__r.<grandGrandParent>__r.<fieldName>

          ...... up to 10 levels

          from <ChildObjectName>]
```

Use Case: Write an apex program to fetch the Candidate Record based on the Candidate Name and fetch the associated Parent Records.

Class Code

```
public class RelatedRecordsHelper

{

   Public static void GetCandidateDetails(String candidateName)

   {

      if(candidateName != Null && candidateName != '')

      {

         List<Candidate__c>  lstCandidates  =  [Select  id,  name,  location__C,
contact_number__c, current_ctc__c, notice_period__C,

                        Position__r.Name, Position__r.Location__c,

                        Position__r.Hiring_Manager__r.Name,
Position__r.Hiring_Manager__r.Contact_Number__C

                  from Candidate__C    Where name =: candidateName];

         System.debug('Number of Candidate Records.....: '+ lstCandidates.Size());

         if(! lstCandidates.isEmpty())

         {

            for(Candidate__c cnd : lstCandidates)

            {

    System.debug('Candidate Record is.....: '+ cnd);

   System.debug('Position Applied for is....: '+ cnd.Position__r.Name);

   System.debug('Position Location is.......: '+ cnd.Position__r.Location__c);

   System.debug('Hiring Manager Name is....: '+ cnd.Position__r.Hiring_Manager__r.Name);

    System.debug('Hiring        Manager        Contact        Number.....:        '+
cnd.Position__r.Hiring_Manager__r.Contact_Number__c);

            }

         }

      }

   }

}
```

Execution: RelatedRecordsHelper.GetCandidateDetails('Praveen Kumar');