System.QueryException: Aggregate query has too many rows for direct assignment, use FOR loop

System.QueryException: Aggregate query has too many rows for direct assignment, use FOR loop

My first thought when I saw the Salesforce Exception – *System.QueryException: Aggregate query has too many rows for direct assignment, use FOR loop* – was

> What is Salesforce talking about? I don't remember using any aggregate functions in my query!

Well, this exception is talking about an "aggregate query" not an "aggregate function." I had never heard the term "aggregate query" before, and I suspect you haven't either. After hunting around, I found the relevant Salesforce documentation – Salesforce Apex Developer Guide: SOQL For Loops.

The documentation explains that Salesforce *might* throw the System.QueryException when a query has a **sub-query** (this is the "aggregate query"), and our **sub-query returns more than 200 child records**.

For example, if we have an Account with more than 200 Contacts, we *might* get the exception when we run the following code with a query on Account with a sub-query for child Contacts.

Code that could throw an exception if there are more than 200 child records:

```
List<Account> accounts = [SELECT Name, (SELECT Name FROM Contacts) FROM Account WHERE ... ];
for( Account acc : accounts )
{
   // Either of the following lines may throw the QueryException exception
   // This is the 'direct assignment'
   List<Contact> contacts = acc.Contacts;

   // calling size() relies on the acc.Contacts list being available
   Integer count = acc.Contacts.size();
}
```

The way I conceptualize this is that the Salesforce database has not returned the complete list of child Contacts. So when I try to assign the child Contacts to the 'contacts' list, the child list is incomplete. Salesforce lets you know the child list is incomplete by throwing the QueryException.

Here's the fix:

Code that runs no matter how many child records:

```
// Note - we are using a 'SOQL for loop' here
for( Account acc : [SELECT Name, (SELECT Name FROM Contacts) FROM Account WHERE ... ] )
{
   List<Contact> contacts;
   try
   {
      contacts = acc.Contacts;
   }
   catch( QueryException e )
   {
      contacts = new List<Contact>();
      // Here's the 'FOR loop' the exception message says we should use
      // Within the outer SOQL for loop, this for loop can access the
      // complete list of child Contact records
      for( Contact con : acc.Contacts )
      {
         contacts.add( con );
      }
   }

   // Now we can use the 'contacts' List however we want.  Yay!
   System.debug( acc.Name + ' : ' + contacts.size() );
}
```

Using the try-catch as above is an **Apex best practice**. It will ensure our Apex runs efficiently, even when our sub-queries could possibly return more than 200 child records.