SYSTEM.QUERYEXCEPTION: NUMBER OF RECORDS EXCEEDS LIMIT: 200

Sunday 1 July, 2018   Published in Troubleshooting Exceptions

Recently, when I was working with UserRecordAccess, I encountered an unusual QueryException. The Apex code that is listed below will help to reproduce the exception. To run it, you need to copy the apex code and execute it in the Developer Console, IDE, or ORGanizer. I want to notice that for successful exception reproduction, you need to have more than 200 Accounts on the salesforce organization. If there are not enough Accounts, you can use any other sObject.

```
Map<Id, Account> accountsMap = new Map<Id, Account>([SELECT ID FROM Account LIMIT 201]);
System.assert(accountsMap.size() > 200, 'Not enough records to reproduce the issue');

List<UserRecordAccess> result = [
    SELECT RecordId, HasReadAccess
    FROM UserRecordAccess
    WHERE UserId = :UserInfo.getUserId() AND RecordId IN :accountsMap.keySet()
];
```

By the text of the exception, we can make a conclusion that the problem is that the SOQL query returns more than 200 records. For a temporary solution, it occurred to me to limit the result to 200 records.

```
Map<Id, Account> accountsMap = new Map<Id, Account>([SELECT ID FROM Account LIMIT 201]);
System.assert(accountsMap.size() > 200, 'Not enough records to reproduce the issue');

List<UserRecordAccess> result = [
    SELECT RecordId, HasReadAccess
    FROM UserRecordAccess
    WHERE UserId = :UserInfo.getUserId() AND RecordId IN :accountsMap.keySet()
    LIMIT 200
];
```

To my surprise, the exception was repeated again. There is a problem in the construction of "RecordId IN: recordsIds". Exception works when the recordsIds collection contains more than 200 elements, regardless of the LIMIT value. The only solution is to split it into several SOQL queries. If you need to extract the list of records from the database, you can use SOQL for loop, and loads data in chunks of 200 records. Otherwise it will be necessary to implement the algorithm for dividing into parts of 200 elements.

```
List<UserRecordAccess> allRecordsAccess = new List<UserRecordAccess>();
for (List<Account> accountsList : [SELECT ID FROM Account LIMIT 201]) {
    // SOQL for loop loads data in chunks of 200 records
    Set<Id> accountsIds = (new Map<Id, Account>(accountsList)).keySet();
    allRecordsAccess.addAll([
        SELECT RecordId, HasReadAccess
        FROM UserRecordAccess
        WHERE UserId = :UserInfo.getUserId() AND RecordId IN :accountsIds
    ]);
}
```

I want to notice that the solution described above is costly for limits. It should be remembered that 100/200 SOQL queries are allocated to the execute context (depending on the Synchronous / Asynchronous context). That is to say in this solution we are trying to get UserRecordAccess for 201 records, as a result for limits we will have the following statistics:

Number of SOQL queries: 3 out of 100
Number of query rows: 402 out of 50000

Explanation of the number of spent limits:
1 SOQL - query 201 Accounts
1 SOQL - first chunk, query 200 UserRecordAccess records
1 SOQL - second chunk, query 1 UserRecordAccess