Apex Basics and Database

Monday, August 08, 2016 10:27 AM

Apex Collections: List

Lists hold an ordered collection of objects. Lists in Apex are synonymous with arrays and the two can be used interchangeably.

The following two declarations are equivalent. The colors variable is declared using the List syntax.

```
1  List<String> colors = new List<String>();
```

Alternatively, the colors variable can be declared as an array but assigned to a list rather than an array.

```
1  String[] colors = new List<String>();
```

Generally, it's easier to create a list rather than an array because lists don't require you to determine ahead of time how many elements you need to allocate.

Intro to Apex Code for Programmers Webinar

sObject Types

sObject Methods

Manipulate Records with DML

DML Statements accept a single sObject or a list. Lists are more efficient for processing records.

Upsert - Using a specified field to determine the presence of existing objects, this DML statement creates new records or updates existing records.

Merge - Merge up to 3 sObject records of the same type into one of the records. It deletes the others and re-parents related records to that merged record.

An Id for a newly inserted account is available directly after insertion.

```
01  // Create the account sObject
02  Account acct = new Account(Name='Acme', Phone='(415)555-1212',
    NumberOfEmployees=100);
03  // Insert the account by using DML
04  insert acct;
05
06  // Get the new ID on the inserted sObject argument
```

```
ID acctID = acct.Id;
// Display this ID in the debug log
System.debug('ID = ' + acctID);

// Debug log result (the ID will be different in your case)
// DEBUG|ID = 001D000000JmKkeIAF
```

You can also perform the other DML operations on this same record.

- insert
- update
- upsert
- delete
- undelete
- merge

Use **DML statements** if you want any error that occurs during bulk DML processing to be thrown as an Apex exception that immediately interrupts control flow (by using try. . .catch blocks). This behavior is similar to the way exceptions are handled in most database procedural languages.

Use **Database class methods** if you want to allow partial success of a bulk DML operation—if a record fails, the remainder of the DML operation can still succeed. Your application can then inspect the rejected records and possibly retry the operation. When using this form, you can write code that never throws DML exception errors. Instead, your code can use the appropriate results array to judge success or failure. Note that Database methods also include a syntax that supports thrown exceptions, similar to DML statements.

Database Methods

Apex contains the built-in Database class, which provides methods that perform DML operations and mirror the DML statement counterparts.
These Database methods are static and are called on the class name.

- Database.insert()
- Database.update()
- Database.upsert()
- Database.delete()
- Database.undelete()
- Database.merge()

Unlike DML statements, Database methods have an optional allOrNone parameter that allows you to specify whether the operation should partially succeed. When this parameter is set to false, if errors occur on a partial set of records, the successful records will be committed and errors will be returned for the failed records. Also, no exceptions are thrown with the partial success option.
This is how you call the insert method with the allOrNone set to false.

```
Database.insert(recordList, false);
```

The Database methods return result objects containing success or failure information for each record. For example, insert and update operations each return an array of Database.SaveResult objects.

```
1       Database.SaveResult[] results = Database.insert(recordList, false);
```

About Transactions

DML operations execute within a transaction. All DML operations in a transaction either complete successfully, or if an error occurs in one operation, the entire transaction is rolled back and no data is committed to the database. The boundary of a transaction can be a trigger, a class method, an anonymous block of code, an Apex page, or a custom Web service method. For example, if a trigger or class creates two accounts and updates one contact, and the contact update fails because of a validation rule failure, the entire transaction rolls back and none of the accounts are persisted in Salesforce.

Write SOQL Queries

When SOQL is embedded in Apex, it is referred to as inline SOQL.

To include SOQL queries within your Apex code, wrap the SOQL statement within square brackets and assign the return value to an array of sObjects. For example, the following retrieves all account records with two fields, Name and Phone, and returns an array of Account sObjects.

You don't need to specify the Id field in the query as it is always returned in Apex queries, whether it is specified in the query or not. For example: SELECT Id,Phone FROM Account and SELECT Phone FROM Account are equivalent statements. The only time you may want to specify the Id field if it is the only field you're retrieving because you have to list at least one field: SELECT Id FROM Account. You may want to specify the Id field also when running a query in the Query Editor as the ID field won't be displayed unless specified.

Instead of using the equal operator (=) for comparison, you can perform fuzzy matches by using the LIKE operator. For example, you can retrieve all accounts whose names start with SFDC by using this condition: WHERE Name LIKE 'SFDC%'. The % wildcard character matches any or no character. The _ character in contrast can be used to match just one character.

SOQL statements in Apex can reference Apex code variables and expressions if they are preceded by a colon (:). The use of a local variable within a SOQL statement is called a bind.

To get child records related to a parent record, add an inner query for the child records. The FROM clause of the inner query runs against the relationship name, rather than a Salesforce object name.

To access the related object in Apex, grab an element from the parent sObject then use dot notation to access the child object.

```
Account[] acctsWithContacts = [SELECT Name, (SELECT FirstName,LastName FROM Contacts)
                               FROM Account
                               WHERE Name = 'SFDC Computing'];
// Get child records
Contact[] cts = acctsWithContacts[0].Contacts;
System.debug('Name of first associated contact: '
```

```
        + cts[0].FirstName + ', ' + cts[0].LastName);
```

You are allowed to traverse a relationship from a child object to a field on its parent object using dot notation.

For standard objects, the __r is not necessary.

Write SOSL Queries

SOSL allows you to perform text searches (Case-Insensitive) in records across multiple standard and custom object records in Salesforce.

Force.com allows you to embed SOSL in Apex.

Return a list of lists of objects that contain SFDC in any of its fields - specified to Account Name or Contact First or Last Name:

```
1  List<List<SObject>> searchList = [FIND 'SFDC' IN ALL FIELDS
2                                    RETURNING Account(Name),
Contact(FirstName,LastName)];
```

Why use SOSL? When you want to be more generic with your search results across multiple objects

This is the syntax of a basic SOSL query:

```
1  FIND 'SearchQuery' [IN SearchGroup] [RETURNING ObjectsAndFields]
```

SearchQuery is the text to search for (a single word '' or a phrase ""). Search terms can be grouped with logical operators (AND, OR) and parentheses. Also, search terms can include wildcard characters (*, ?). The * wildcard matches zero or more characters at the middle or end of the search term. The ? wildcard matches only one character at the middle or end of the search term.

SearchGroup Criteria is optional and will search all fields if not specified. Can be one of the following:
ALL FIELDS
NAME FIELDS
EMAIL FIELDS
PHONE FIELDS
SIDEBAR FIELDS

If ObjectsAndFields is not specified, the search result contains the IDs of all the objects found.