Map with Object as key:

```
Map<Account, List<Contact>> items =new Map<Account, List<Contact>>();
Account a = new Account(Name = 'test');
items.put(a, new List<Contact>());
System.assert(items.size() == 1);
//present in map
System.assert(items.containsKey(a));
//changed name
a.Name = 'Name2';
System.assert(items.size() == 1);
//not found in map
System.assert(!items.containsKey(a));
//able to insert
items.put(a, new List<Contact>());
//now 2 items
System.assert(items.size() == 2);
```

Apex uses a hash of the field values as the internal value to use when searching for the object in the map or set. Changing a field on an object changes this hash value, causing the same object to appear as two distinct objects when used as keys. Given that one of the main purposes of maps and sets when used with objects is to hold them while they are being modified, using objects as keys or in sets is a sure way to create subtle and hard to find bugs.

Contains Method

1. use Set rather the list, if list items are more, let say 100 an above
2. Set uses a hash table index to speed up searches. It is literally designed for that purpose.

SOQL

- **Like** operator
  - *where name like 'tes'* – matches names with 'tes' values
  - *where name like 'tes_'* – matches names with ('tes'+a char) values, only 4 char names
  - *where name like 'tes%'* – matches names with 'tes….' Values, at least 3 and more char
  - *where name like 'tes_%* - at least 4 char and more
  - *where name like 'tes_\%%'* – 5 char or more as '_' and '\%' is same

- **Reserved** characters = The single quote (') and backlash (\) characters are reserved in SOQL queries and must be preceded by a backslash to be properly interpreted.
  SELECT Id FROM Account WHERE Name LIKE 'Bob\'s BBQ'

- **Alias** :
  *SELECT count() **total** FROM Contact c, c.Account a WHERE a.name = 'MyriadPubs'*
  select count(id) from account k where k.name like 'test%'

- NULL value :
  - *If you run a query on a boolean field, null matches FALSE.*
  - *!= null -> true*

- *INCLUDES- EXCLUDES : used only for multiselect picklist*

- *Date and time format*

Format	Format Syntax	Example
Date only	YYYY-MM-DD	1999-01-01
Date, time, and time zone offset	• YYYY-MM-DDThh:mm:ss+hh:mm • YYYY-MM-DDThh:mm:ss-hh:mm • YYYY-MM-DDThh:mm:ssZ	• 1999-01-01T23:01:01+01:00 • 1999-01-01T23:01:01-08:00 • 1999-01-01T23:01:01Z

- **ORDER BY :**
  - in case of duplicate values, order by doesn't guarantee order.
  - **NULLS LAST**: SELECT Name FROM Account ORDER BY Name DESC NULLS LAST
  - **NULLS FIRST**: SELECT Name FROM Account ORDER BY Name DESC NULLS FIRST

- ***OFFSET:***
  - *Max value is 2000.*
  - ***In subquery with no where clause, it can be used only if parent query has limit 1 clause***

    SELECT Name, Id
    (SELECT Name FROM Opportunity LIMIT 10 OFFSET 2 // No where clause)
    FROM Account ORDER BY Name LIMIT 1 //valid limit

- ***FIELDS****() – can be used with query returning at most 200 records. You can either use LIMIT 200/where clause satisfying 200 records*

  - *SELECT FIELDS(ALL) FROM Account*
  - *SELECT FIELDS(CUSTOM) FROM Account LIMIT 200*
  - *SELECT FIELDS(STANDARD) FROM Account*

- *GROUP BY :*
  - *must use, in case of aggregate function*
  - *Can't be used in __r fields*

- ***GROUP BY ROLLUP : Group by in a composite key****:*
    - *Select count(Id), name, rating from Account group by rollup(name,rating)*
    - *If one field is used, Group by and group by rollup are same.*

- *HAVING :* ***Followed by GROUP BY,***
    - ***can filter aggregate functions too.***
        - *select count(id), rating from account group by rating having count(id) > 10*
    - ***can filter fields that are in GROUP BY clause***
        - *select count(id), rating from account group by rating having rating = 'HOT'*

- *FOR VIEW / FOR REFERECNED : Salesforce* generates a list of recently viewed and referenced records.
    - Select Id from Account For View
        - The LastViewedDate field for the retrieved record is updated.
    - Select Id from Account For *REFERECNED*
        - The LastReferencedDate field is updated for any retrieved records.

- **Count_Distinct() : aggregate function – ignores null values**

SOSL

- *FIND {text} IN {Search Group} RETURNING objectType(fieldList ORDER BY fieldOrderByList LIMIT number_of_rows_to_return OFFSET number_of_rows_to_skip)*
- *Search Group :* If unspecified, the default is `ALL FIELDS`
    - `ALL FIELDS`
    - `NAME FIELDS`
    - `EMAIL FIELDS`
    - `PHONE FIELDS`
    - `SIDEBAR FIELDS`: means basically indexed fields. So, if you want to make any field searchable through SIDEBAR fields (its faster), just make it ExternalId in field definition.

- *Find {text}*
    - *It will return all subject having text in any of the Name,text,email,sidebar,phone etc.*
- *Find {text} Returning Object1, object2… so on.*

- *Return objects with ID fields that matches*

- *Find {Text} in Name Fields Returning Object1(a,b), object2(c,d)… so on*
  - *Check in Name fields only*
- *Find {Text} in Name Fields Returning Object1(ID,Name,type where type = 'XYZ' limit 20), object2… so on*
  - *Returns only 20 Object 1 records having type = xyz , object2 all matched records*

In Apex , {} is replaced by ''.
List<list<sobject>> records = Search.query('FIND \'test\' RETURNING Account(Id, Name, name_phone__c)');

SOSL Filters

1. *WITH SPELL_CORRECTION: default true*
   a. FIND {Tets} RETURNING Account(Id, Name, name_phone__c) WITH SPELL_CORRECTION = true -> return all accounts with all permutations of text given.

Working with SOQL Aggregate Functions

```
AggregateResult[] groupedResults
  = [SELECT CampaignId, AVG(Amount) aver
     FROM Opportunity
     GROUP BY CampaignId];
for (AggregateResult ar : groupedResults)  {
   System.debug('Campaign ID' + ar.get('CampaignId'));
   System.debug('Average amount' + ar.get('expr0')); //can use "aver" instead of expr0
}
```

Queries that include aggregate functions are still subject to the limit on total number of query rows.

For `COUNT()` or `COUNT(fieldname)` queries, limits are counted as one query row, unless the query contains a GROUP BY clause, in which case one query row per grouping is consumed.

Result Classes

1. *SaveResult – insert and update*
2. *UpsertResult*
3. *UndeleteResult*
4. *DeleteResult*

- *Common methods,*
  1. *Boolean : isSuccess()*
  2. *Id: getId – record ID -> even in delete too.*
  3. *Database.Error[] : getErrors()*
     - *getFields*
     - *getMessage*

```
// Create two accounts, one of which is missing a required field
Account[] accts = new List<Account>{
    new Account(Name='Account1'),
    new Account()};
Database.SaveResult[] srList = Database.insert(accts, false);

// Iterate through each returned result
for (Database.SaveResult sr : srList) {
    if (sr.isSuccess()) {
        // Operation was successful, so get the ID of the record that was processed
        System.debug('Successfully inserted account. Account ID: ' + sr.getId());
    }
    else {
        // Operation failed, so get all errors
        for(Database.Error err : sr.getErrors()) {
            System.debug('The following error has occurred.');
            System.debug(err.getStatusCode() + ': ' + err.getMessage());
            System.debug('Account fields that affected this error: ' + err.getFields());
        }
    }
}
```

Schema & GlobalDescribe

1. **Get Sobject type using record Id?**

   ```
   String sobjectType = sId.getSObjectType().getDescribe().getName();
   ```

2. **Get all objects**

   ```
   Map<String, SObjectType> sObjects = Schema.getGlobalDescribe();
   ```

3. **fields of any object : returns map <API _NAME, Label>**

   ```
   Map<String,String> fieldsMap = sObjects.get('Account').getDescribe().fields.getMap();
   ```

4. **Get Prefix : starting 3 char of ID**

   ```
   String prefix = Schema.getGlobalDescribe().get(Objectname).gtDescribe().getKeyPrefix();
   ```

5. **Compare subject types using record Id?**

   Id.getSobjectType() == Schema.Object.SObjectType

Apex Triggers

1. The `after undelete` trigger events only run on top-level objects. For example, if you delete an Account, an Opportunity may also be deleted. When you recover the Account from the Recycle Bin, the Opportunity is also recovered. If there is an `after undelete` trigger event associated with both the Account and the Opportunity, only the Account `after undelete` trigger event executes.