Four types of Standard fields will get created.

Name

Owner

Created by

Last modified by

ID - Object - 18 digit unique id…

15 digit

Standard Fields

Custom Fields

Create a custom object

Field Dependencies

Page layout specific sections

Will introduce to a Project

Define Objects, Fields

Relationships …

**13-March-2024**

Label API Name

Account Account

Position Position\_\_c

Scenario 1: Create a custom object called Position

Fields Columns Attributes

Rows Records Tuples

Name Value Type

Department IT,ENG,Sales PickList

Location Pick List

Start Date Date

Educa Quali Long Text

Scenario 2: Create below custom fields for Position object.

\*\*\*\*\* 1 Certfication

Label

Name

Description

Required

Help

Unique

External ID

Scenario 3: Create a List View called Cognizant Opportunities where Stage Name=’Closed Won’ and ‘Prospecting’

**Page Layout & Sections**

**Scenario 4:** Create two page sections on Position object and align other fields..

1. Job Description Section

Add fields Education Qualifications, Skills Req, Respon,, Job Desc

1. Compensation Details

Min Pay and Max Pay

**Field Dependencies**

Create a dependent relationship that causes the values in a picklist or multi-select picklist to be dynamically filtered based on the value selected by the user in another field.  
 • The field that drives filtering is called the "controlling field." Standard and custom checkboxes and picklists with at least one and less than 300 values can be controlling fields.  
 • The field that has its values filtered is called the "dependent field." Custom picklists and multi-select picklists can be dependent fields.  
  
**Step 1.** Select a controlling field and a dependent field. Click Continue when finished.  
  
**Step 2.** On the following page, edit the filter rules that control the values that appear in the dependent field for each value in the controlling field.

One field is dependent on another fields

Country - Picklist India Controlling Field

State - Picklist Telangana All 23/24 States Dependent Field

City - Picklist Hyderbad - Dependent Field

**Scenario 5:**

**Create a Field dependency on below fields for Position object**

**Status (Controlling) Sub Status (Dependent)**

New NA

Open Pending, Approved

Closed Filled, Rejected & Cancelled

**Department Pay Grade**

Eng Eng-100, Eng-200, Eng-300, Eng-400

IT IT-100,

Finance ACC-100

Sales S-100

Service C-100,

**Scenario 7:**

Create a Picklist field called Region on Account object

Picklist field values

Region

1) APAC

2) EMEA

3) LATAM

4) North America

Create another Picklist Field called Countries

Malaysia Mexico

Singapore Columbia

India Argentinia

China Chile USA Brazil

Canada Chille

Mexico

Germany

Finland

France

UK

Nigeria

Egypt

**Create Field dependencies on below two fields**

Region (Controlling Field)

Countries (Dependent Field)

**Case Study**

Hiring Manager will get the Job description.. complete position details..

Provide all the details to the Recruiter (Talent Acquisition Team)..

Publish these openings

1. Job Positing Sites (Linkedin, Naukri, Indeed,monster etc)
2. Company’s career page
3. Referrals
4. IJP (Internal Job Postings)

Candidates will start applying..

Filling cover letter, Job Application form

Surender - Skill Salesforce

Java

One Job Application Applying for one Position

One Position (Java) Multiple Job Applications

Shortlist of the candidates…

Interview is scheduled… Interviewer ( Rachana)

J001- Salesforce ----

One Job Application ----- Can undergo multiple Reviews

Interviewer (Rachna) -------- can conduct multiple reviews..

We need to document Scores / Ratings / Feedback …

4 Rounds -

Recruiter – Average scores of ratings..

Draft Offer Letter..

Sent an Offer Letter..

Derive Objects

Derive Fields for objects

Position / Job Positions

Job Application

Candidates

Interviewer

Reviews

Offer Letter

Job Posting Sites

Relationships

One to Many

Many to Many

Relationship field will always go on or should create on child object.

* These relationships are mainly categorized into the following types:
  1. Lookup
  2. Master-Detail
  3. Many to Many
  4. Self Relationship
  5. External Relationship

Indirect Lookup

External Lookup

6) Hierarchical Relationship

**14-March-2024**

**Look-up Relationship**

**\*\*\*\*\*\*\*\*\*1 or 2 Certfication**

Definition: *A lookup relationship can be used to link two objects together. It is the most basic type of relationship that creates a child-parent relationship between two objects.*

*A Lookup relationship involves finding value of a field based on the value in another field in another object. It is mostly used in the case of commonly shared data between two objects.*

* When you define a lookup relationship, you have the option to include a lookup field on the page layouts for that object as well as create a *related list* on the associated object's page layouts.

Characteristic of Lookup Relations

* Loosely coupled- Two objects have relation ship but they don’t have any dependencies. (When you delete the parent, child will not be deleted)
* The field reference can have null value
* No Cascade (deletion of records in one object will not effect the other object records)
* Sharing and Security will be independent - OWD
* Fields need not be kept in the UI alias Page Layout.
* There can be a maximum of 40 ***(38 Lookup*** +2 MD) relationships per child.



**Scenario 8:**

Create a lookup relationship on Job application

Position (Master) Job Application (Child)

\*\*\*\*\* Relation ship field should go on child object

Candidate (Master) Job Application (Child)

Master Detail Relationship

**Definition:**

A master-detail relationship can also be used to link two objects together. A master-detail relationship creates a tight relationship between the **parent and the child. T**he child record inherits security of the parent, and if the parent is deleted, all associated child records will also be deleted.

***or***

Master-detail relationship is like parent-child relationship where, master represents a parent and detail represents a child in which master object controls some behaviors of the detail object. Like whenever a Master object record is deleted then the detail object related to it also gets deleted.

Master-detail relationships created some extra functionality such as **Roll-up summary fields that allow you to calculate data on the parent from the child records**

Characteristic of MD Relations

**\*\*\*\*\*\*\*\*1 Certification Question**

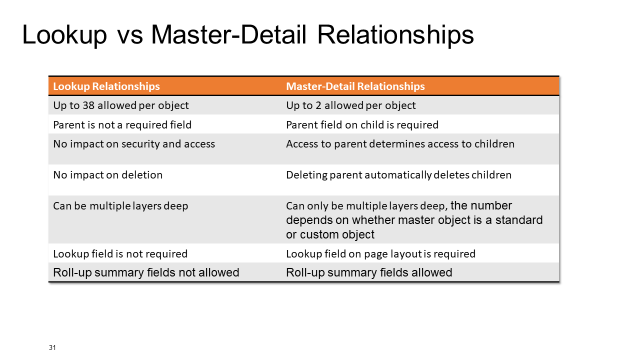
* Tightly coupled- Two objects have relationship but have dependencies on each. (When you delete the parent, child will also get deleted)
* The field reference cannot have null value
* Cascade happens(deletion of records in one object will effect the other object records (Child))
* Sharing and Security will be dependent on Master - OWD
* Fields should kept in the UI alias Page Layout.
* There can be a maximum of 40 (38 Lookup **+2 MD)** relationships per child.

**Scenario 7:**

**Create a MD Relationship between**

Position (Master) Interviewer(Child)

Job Application (Master) Review(Child)



**Many to Many Relationship**

Definition: *A many-to-many relationship allows each record of one object to be linked to multiple records from another object and vice versa*

Say you have a situation where it is required you have many of one record related to many of another. Duplicating that lookup field a number of times on each object is not best practice and will get very messy. This is where we should leverage Junction Objects.

*A many-to-many relationship (Also referred to as a junction object), allows you to create a relationship between two objects that need to model a many-to-many relationship. These are created with an object that has two master-detail relationships to two parent objects.*

* Allow for the relationship of two objects in a many-to-many fashion.
  + Candidates may apply for many positions by submitting a job application for each position.
  + A position may have many candidates.
  + Implementing a many-to-many relationship requires a third intermediate object called *Junction object.*
* A junction object is a custom object with two relationships.
* Used to link two other objects in order to build a many-to-many relationship between them.
* When creating a junction object, consider the following:
  + Name the object with a label that indicates its purpose.
  + Use the auto-number data type.
* Junction object records are deleted when either associated master record is deleted and placed in the Recycle Bin.
* However, junction object records are deleted permanently when both the associated master objects are deleted.

**Scenario 8:**

Create Many to Many relationships between

Position (M) Job Positing Sites (M)

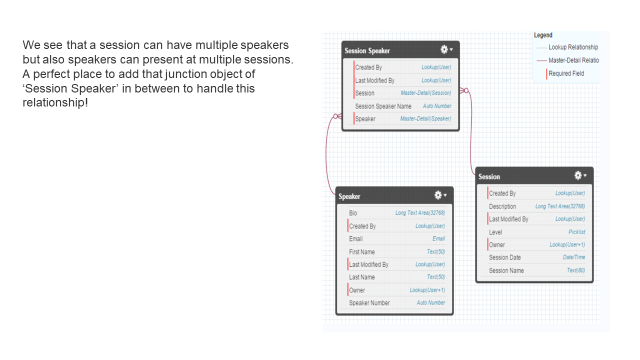
To Create M:M we need to create a junction object called Job Posting

2 MD Relation ship

Position (M) Job Posting (C)

Job Posting site ( M) Job Posting (C)

**Scenario 9: Create M:M relationship on below objects**

****

Self Relationship

**Scenario 10:** **Campaign Called**

Social Media Campaign (Parent Campaign)

Facebook Campaign

Instagram Campaign

Twitter

YouTube

WhatsApp

Schema Builder - **Scenario 11:**

Graphical representation of your objects and their relationships

\*\*\*\*\*you can create object, fields, relationships ..

**Create Schema builder by selecting all custom objects.**

**App Creation Scenario 12:**

Why can’t we create our own App and can incorporate Standard and Custom objects..

Custom App ---

How to add a logo? Documents folder/tab in Classic

Create a classic app and add a logo to your classic App

Migrate this classic App to Lightning App

\*\*\*\*\*\* Logo size should not be more than 20kb size in classic.

Administration, Customization and Security Model

Business Logic tools

Data Management

**15-March-2024**

Administration, Customization and Security Model

Company information

Licenses

User Management

Create an User

Activate

Deactivate

Freeze a user

Profiles

Types of Profiles

Standard & Custom Profiles

What Profiles determine?

Page Layouts

App / Org Level Access

Tab Level Access

Login Hrs.

IP Ranges

Administrative Permissions

General User Permissions

Standard & Custom Object Permissions

CRUD

C-Create

R-Read

U-Updated

D-Delete

Password Policies

Permission Sets

Permission Set Groups

Field Level Security

Record Types

Seven Stake holders

Recruiter (HR)

Hiring Manager(HM) / Project Manager

HR Manager -> Report on the recruitment process.

Interviewer.

CEO. SVP –HR, SVP-Development

**Things to consider when designing Apps** Consider the actors—who will be using the app?

Recruiter (HR)

Hiring Manager(HM) / Project Manager

HR Manager -> Report on the recruitment process.

Interviewer.

CEO, SVP –HR, SVP-Development

What will these users expect to see and do?  
 User will different level of Access

Based on their role / designation they will access to the application

What data is most important?

Candidates Data

Offer Data

Salary or Compensation details

What should these users be able to see—are there any data restrictions?  
 Yes .. based on the role they will have access to the data..

How can we make the user experience more streamlined and efficient?  
 Customization the UI as per the user roles

Creating profiles and users

Which users should be able to customize the app?

Sys Admin

Do we have seven users created?

Do we have specific user license and profile assigned to these users?

User :

As a Sys Admin you can create a user.. but you cannot delete a user

One user can have only one profile

One profile can have multiple users

Admin

Activate

Deactivate

Freeze the user

**Scenario 13:**

Create a user called Cynthia capibanoc

**Scenario 14:** Create below users in the user management and assign any user license and profile as of now. Later will change

Page Layouts

App / Org Level Access

Tab Level Access

Login Hrs.

IP Ranges

Administrative Permissions

General User Permissions

Standard & Custom Object Permissions

CRUD

C-Create

R-Read

U-Updated

D-Delete

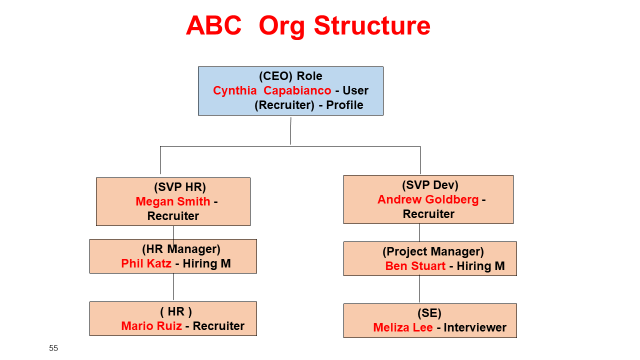
Password Policies

Permission Sets

Permission Set Groups

Field Level Security

Record Types



Profile determines what level of access does the user has..

One profile can have multiple users

Each profile uses Salesforce user license.

Divided into two types

1. Standard Profiles
2. Custom Profiles

\*\*\*\* You cannot create a profile but you can clone an existing profile.

One user can have only one profile.

Profile is going to determine below

Profiles

Types of Profiles

Standard & Custom Profiles

What Profiles determine?

Page Layouts

App / Org Level Access

Tab Level Access

Login Hrs.

IP Ranges

Administrative Permissions

General User Permissions

Standard & Custom Object Permissions

CRUD

C-Create

R-Read

U-Updated

D-Delete

Password Policies

Field Level Security

Record Types

Profiles will have two User Interface (UI)

Pagelayout, FLS and Record types can edit in the first UI page..

**Scenario 15: Create a Recruiter Profile and provide Object level access as per the table and assign this profile only to (Mario, Cynthia, Meghan and Andrew)**

**App Level Access - Org Level Access**

1) Recruitment App, Sales App (both Classic and Lightning)

**Tab Level Access**

2) Default On Only Recruitment Tabs.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\* Certification Question**

**Default On -** You can view the object on the Tab

**Default Off -** You cannot see an object on the tab but you can see the object in

All Items of App Launcher Page or + in classic

**Tab Hidden -** You cannot see the object either on Tab or in App Launcher. it is hidden.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3) Custom Object Level Access (Kindly refer the table)

4) Login Hrs (9:00 AM to 6:00 PM only on Weekdays)

5) IP Ranges (specific IP's which are provided by Network team)

6) Administrative Permissions … special permissions. Business will give.

7) General User Permissions

8) Standard & Custom Object Permissions.

10) Password Policies.

Login Access Policies – Login button will be enabled.

Administrators Can Log in as Any User Enable this option

**Session Settings**

Force relogin after Login-As-User - Uncheck this

\*\*\* you are in logged in use u cannot perform admin related activities

**Scenario 16:**

Create a Hiring Manager Profile and provide Object level access as per the table and provide access only to (Phil and Ben Stuart)

1) **App Level Access** Recruitment App, Sales App, Salesforce Chatter (both Classic and Lightning)

2) Only Recruitment Tab level Access.

Default On - Custom Tabs On

3) Object Level Access (Kindly refer the table)

Assign the users to the profiles...and Navigate the App and see the changes.

Hiring Managers**: Phil and Ben Stuart, Lavanya, Sushila**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 Login and check the Access level and create records as per object level permissions

Permission Set:

Phil doesn’t have Create, update and delete on all three objects –Position, Candidate and Job Application.

Sys Admin - to Grant additional permissions Create, Update and Delete on all three objects –Position, Candidate and Job Application. Assign these permissions only to me not to any users who are part of Hiring Manager Profile.

A permission set is a collection of settings and permissions that give users access to various tools and functions. Permission sets extend users’ functional access without changing their profiles and are the recommended way to manage your users’ permissions.

Users can have only one profile but, depending on the Salesforce edition, they can have multiple permission sets. You can assign permission sets to various types of users, regardless of their profiles.

Create permission sets to grant access for a specific job or task, regardless of the primary job function or title of the users they’re assigned to. For example, let’s say you have several users who must delete and transfer leads. You can create a permission set based on the tasks that these users must perform and include the permission set

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*We can create 1000 Permission sets per Org..\*\*\*\*\*\*\*\*\*\*\*\*

Permission sets- can be created on…..

1. Assigned Apps
2. Objects
3. Assigned Connected Apps
4. App Permissions
5. Apex Classes
6. Visual Force Pages
7. External Datasources

***Scenario 17:-***

Grant additional permissions set with CRUD access on Position, Candidate and Job Application objects only to Phil Khatz who belongs to Hiring Manager Profile..

And assign only to PhilKhtaz...

**Scenario 18:** **HW**

4) Create three users...

1) Donald Trump

2) Obama

3) Bush..

4) Cynthia.

Use – **Existing Custom: Sales Profile**

**App Level**

Sales Lightning App

Sales Classic App

Chatter..

**Tab Level.**

Standard Tabs..Default On

Enable / Default on Products and Price Book

**Standard Object Permissions**

Read Permission on Contacts

CRUD - Accounts, Leads, Opportunities.

CRUD- Products, Price Book.

Login as any of the user and you can see level of Access

**Permission Set Group**

A permission set is a collection of settings and permissions that give users access to various tools and functions. Permission sets extend users’ functional access without changing their profiles and are the recommended way to manage your users’ permissions.

Users can have only one profile but, depending on the Salesforce edition, they can have multiple permission sets. You can assign permission sets to various types of users, regardless of their profiles.

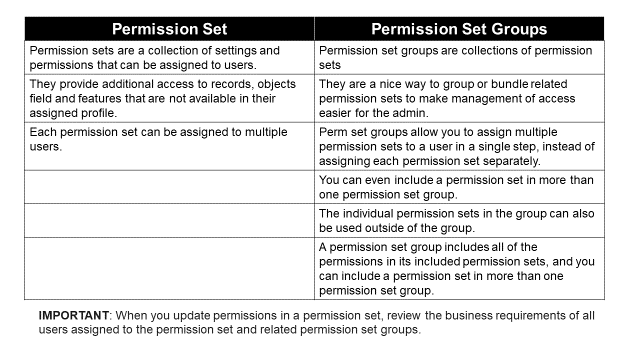
Create permission sets to grant access for a specific job or task, regardless of the primary job function or title of the users they’re assigned to. For example, let’s say you have several users who must delete and transfer leads. You can create a permission set based on the tasks that these users must perform and include the permission set

**Examples:**

1. We have many users in your organization with some fundamental job functions. We can assign all of then with one profile that grants them all access to do their job. But some set of people are working on special apps or some special functionality, for this type of special users we can create permission sets and can be assigned to them.

2. Some users need some temporary access to a specific set of fields and objects we can create a permission set with those object & field access and we can assign those specific users.

**Permission sets** are, as the name implies, a set of additional CRED permissions that can be applied to different profiles. Typically, they are task-based and related to different Objects and managed packages. For example, Sales users may be assigned a permission set giving them access to a CPQ app to generate quotes.



**Scenario 19:**

Create a user called Donald Trump and assign Custom Sales Profile – and at profile level assign App Access to Sales and Service App.

Now try to login as Donald Trump and try to access Custom Recruitment App.

*He cannot view custom App tabs or objects*

And he can access Sales App and Service App but try to access Order tab and try to create, edit or delete any records?

*You can view order tab but you cannot create, edit or delete any records?*

Now find a solution for above to scenarios

1) Create Permission Set called Access to Order Object and provide CRUD Access (Use Salesforce license)

2) Create Permission set called Access to Recruitment App and assign Recruitment App which is a custom App

Create a Permission set group called Access to Donald Trump and assign above two permission sets to this group.

Login as Donald and try to access above two features.

**Scenario 20 HW**

Create a user called Andrew and assign Salesforce Platform User licenses and login and check if he has any Access to Sales, Service Cloud and any Custom App?

\*\* You will not be able to see any Sales or Service App but you can see Custom App (Rec App)

1. Create Permission set to Access Recruitment App
2. Create Permission Set to Access Position and Candidate Objects

Create a permission set group called Additional Access to Andrew and assign above to Permission sets.

Login as Andrew and try to access App and Position and Candidate Objects. He will be able to access.

But try to access other objects he may not because we have not given any access at profile level. If you give at profile level all users who are part of Salesforce Platform User profile will have complete access

**18-March-2024**

**Scenario 21:**

Field Level Security – Is also part of Profiles

If you want any fields on a page layout to be hidden or read only and visible

Field level security can be implemented in four ways

**Scenario 22:**

1. Profiles

**Scenario:**  Ensure Min Pay and Max Pay fields are hidden for Interviewer Profile users.

Read Access

Edit Access

**Scenario 23:**

**Permission Sets**

**Scenario :** With Permission Sets.

Enable **Read only access** on **Interviewer Email ID / Employer Field** on Interviewer Object with Permission Set and assign these permission set to Mario, Cynthia users.

Record Types

Understanding record types in Salesforce is vital for gaining the most benefit from the customer relationship management (CRM) platform. Companies across all industries see an increase in sales revenue when properly using Salesforce.

Salesforce Record Types are a way of grouping many records of one type for that object. These can be applied to any standard or custom object, and allow you to have a different page layout, fields, required fields, and picklist values.

They should be used for records that have the same concept, but need to be different in execution.

Record types can be created on any standard or custom object in Salesforce, allowing you to configure different:

* Page layouts
* Fields
* Required fields
* Picklist values

**Key takeaways:**

1. Record types allow administrators to create a different page layout with custom picklist fields and values for the same business process and various business processes.
2. Benefits of record types include a single source of truth, multiple page layouts for the same object, and optimized worker efficiency.
3. Some factors to take into account when working with record types are that they don’t encompass all security levels, and the industry recommends having no more than 200 record types per Salesforce instance.
4. Reduce the number of fields on the page.
5. Improve data quality – sometimes users are forced to fill out record types which can lead to poor quality.
6. Specific page layout will be displayed for specific fields
7. Same as Record Type

**Scenario 24:**

**Scenario: On Position Object..**

Create two Record Types...

1) Technical Record Type

2) Non Technical Record Type..

i) Create three multi picklist field on Position Page layout

1) Operating System

2) Programming Languages

3) Domain Knowledge..

ii) Create two Page Layouts

1) Technical Page Layout

Add OS and Progm Language fields to this layout

2) Non-Technical Page Layout

Add or Assign Domain knowledge Field..

iii) Create two Record Types and assign only to **Recruiter and Sys Admin Profile only**

1) Technical Record Type

Assign this record type to Technical Page Layout

Edit the Dept and Paygrade Fields

2) Non Technical Record Type

Assign this record type to Non Technical Page Layout

Edit the Dept and Paygrade Fields

Home Work

**Scenario 25:**

HW- Create two Record Types.. on Candidate object –

Additional Fields on Candidate Page Layout - Email, Phone, Gender, Country

1) Non US Candidate

2) US Candidate

i) Create additional fields

**US Candidate**

SSNO

Unique ID No

Green Card Holder

H1B Candidate

**Non US Candidate**

Aadhar Card

PAN

Country Specific Identity

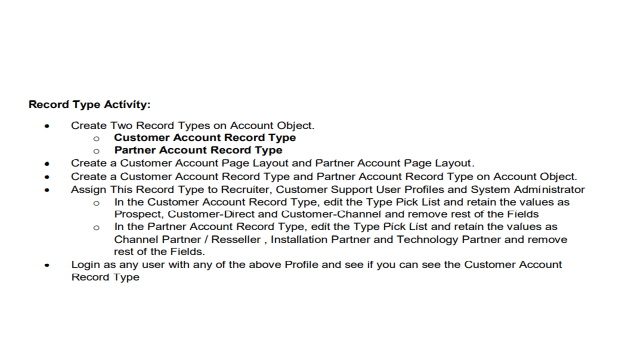
ii) Create two Page Layouts..

Assign those fields to Page Layouts

iii) Create Record type assign those page layouts with respective fields

iv) Assign it only to SysAdmin and Sales Profile or Recruiter profile

**Scenario 26:**



Record Level Security

Access to any records are divided into three types

1. Full Access

Create

Edit

Delete

Read

Share

Transfer the owner ship to other user.

1. Read and Write Access
2. Read only

Who will have full access to the records

Owner Field

Sys Admin

Owner

Queue Member

Based on the user above him Hierarchy \*\*\*\*\*\*\*\*\*\*\*\*\*\* When an option called

\*\*\*\*\* Grant Access using Hierarchies \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* If this check box is enabled…

\

Queue Member

\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 certification question

What is queue?

Queue is a group of resources who will manage workload effieclitly. …..once an record is shared to Queue

The users in the queue should accept that record..

Once he accepts the record….he will be the owner of the record and he can further can transfer to the sane queue or to other user

**Scenario 27:**

Transfer Owner ship

To which ever the record ownership that user should have Salesforce user license

Transfer a record which is owned by Shiva Kumar to Mario

**Scenario 28:**

Create a Queue called Recruiter Queue on Position, Job Application and Candidate.

Add users Cynthia, Meghan and MArio

U can have assigned to

Users

Roles

Public Groups

Roles and Subordinate.

**19-March-2024**

Who else will have full Access?

Let us create Role and Role Hierarchies and assign the users respectively.

**OWD – Organization Wide Default**

* Organization-wide defaults are a security setting that defines the baseline level of access to data/ records that you do not own.
* They are the only way to restrict access to data in the sharing model.

They can be defined for the custom as well as several standard objects.

**Private**

Only the record owner, and users above that role in the hierarchy, can view, edit, and report on those records.

**Public Read Only**

All users can view and report on records, but only the owner, and users above that role in the hierarchy, can edit them.

**Public Read/Write**

All users can view, edit, and report on all records.

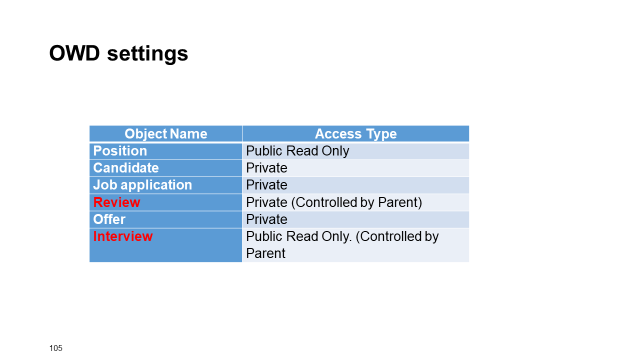
**Controlled by Parent**

A user can view, edit, or delete a record if she can perform that same action on the record it belongs to.

**Public Read/Write/Transfer** – only applicable to Lead and Case objects

**Scenario 29:**

Change the OWD of Position object to Private and login into any user and see the changes



Sharing the records ….

Three types

1. Manual Sharing
2. Sharing Rules

Criteria Based Sharing Rules

Owner Based Sharing Rules

1. Apex Sharing Rules

**Public Groups:** A public group is a set of users. It can contain individual users, other groups, the users in a particular role or territory, or the users in a role or territory plus all of the users below that role or territory in the hierarchy.

**Scenario 30:**

Create a Public Group called “Cognizant Executive Group” add uses – Cynthia, Meghan and Andrew and Role VP Marketing also.

**Manual Sharing** - If you want to share a single record to below any of the users or roles.

User

Role

Role and Subordinate

Public Groups

**Scenario 31:** Share any Position Record whose owner is Meghan to Andrew who is other part of the Org Structure.

In case if I want to share more than one record…

Two types of Sharing Rules

1. Criteria Based Sharing – sharing the records based on certain criteria or condition to below users

Use sharing rules to make automatic exceptions to your organization-wide sharing settings for defined sets of users.  
Note: "Roles and subordinates" includes all users in a role, and the roles below that role.  
You can use sharing rules only to grant wider access to data, not to restrict access.

User

Role

Role and Subordinate

Public Groups

**Scenario 32:** Share the records to whose position department =IT, Sales and Service to Roles (HR Recruiter, SVP Dev)

1. Owner Based Sharing – Sharing the records based on the Owner field….

**Scenario 33:** – Share the records which are owned by Cynthia to SVP HR and SVP-Dev

**View Set-up Audit Trail**

**Login History**

**Field History Tracking**

When you do some changes in the object fields, field history is going to track the old and new values, who changed, date and timings..

This page allows you to select the fields you want to track on the *Position History related list*. Whenever a user modifies any of the fields selected below, the old and new field values are added to the History related list as well as the date, time, nature of the change, and user making the change. Note that multi-select picklist and large text field values are tracked as edited; their old and new field values are not recorded.

For certain fields.. LongText… it will not track the old values, it will not track old and new field values..

While creating object enable field history tracking.

**Scenario 34:** Pls track old and new values of some of the position fields with Field History tracking.

**Login Flows**

**Business Process Logic tools**

Formula Fields

Simple

Advance Formula

Cross Object Formula Fields

Roll up Summary Fields

Validation Rules

Workflows Rules

Immediate

Time dependent

Email Templates Creation

Approval Process

Two types

Jump Start Approval Process

Standard Set-up Approval Process

Step by Step Approval Process

Skip Level Approval Process

Process Builder

Flows

Screen Flows

Record Triggered Flows

Scheduled Triggered Flows

Data Management

Data Import Wizard

Data Loader

Reports & Dashboards

Sales Cloud & Quote Process

Service Cloud

Experience Cloud

**20-March-2024**

Business Logic tools

**Formula fields –**

IT is a datatype…

Sometimes we may require the data stored in the field of an object to be the value obtained by performing few calculations on some other fields or by merging some fields and it is difficult to calculate field value for each record every time and also in few cases you may want to save the record only if it meets your requirement. So, here formula comes to the picture which will automatically determine the value of the field based on the given condition. Formulas are used in many areas in Salesforce such as Validation rules, workflow rules, process builder, etc. Users cannot change the value of a formula field manually.

1. Are part of each object details page fields section..
2. A read-only field that derives its value from a formula expression you define.
3. The formula field is updated when any of the source fields change.
4. You cannot view this field on the Page Layout of an Object
5. You can view this field on the details or record page layout of an objects..
6. Get the values from a formula expression which you define...

Math Operators (+, -, \*, /, ^, ())

Logical Operators (==, ! =, <, >, <=, >=, &&, ||)

Text Operators (&)

**Scenario 35:**

Calculate the Total Salary of an employee based on Basic, HRA and other allowances.

Before doing this exercise you need to have a record in Candidate and Job Application Object to perform this calculation.

Create formula field called as HRA with 30% of Basic Salary, other Allowances with 12% of Basic Salary, Special allowances with 12 % of Basic Salary and calculate the Total Salary in offer object by adding Basic+HRA+OA+SA+. You cannot see these fields on the page layout.

\*\*\*\*\* Add 10% of Percentage

**Cross Object Formula**

* + Can also be used to reference the fields of other objects – called as Cross – Object Formulas.
  + Enable you to incorporate merge fields from multiple objects for calculations and display
  + Create formulas that reference fields on parent or grandparent object (up to 10 levels)
  + Are limited to ten unique relationships per object across all formulas and rules for that object

\*\*\*\*\*\*Upto 10 Levels

Position Job Application Candidate

| |

Offer Review

Create a Cross Object Formula to display Candidate Name and Position Name on the Offer Page Layout.

More examples

**Scenario 36:**

Calculate the Total Pay Package for a Position

Man Pay + Max Pay..

**Scenario 37**

Create a formula field to calculate “No of days” when a particular position is opened. Consider (Start Date).

Start\_Date\_\_c Today()-Start\_Date\_\_c

Opened Date Today()-Datevalue(Open\_Date\_\_c)

**Roll up Summary Field :**

Roll-Up Summary is a type available in the Master object of an master-detail relationship.

Used to perform aggregate operations on the detail records.

It is a formula like read only field that can display the sum, min, or max value or record count of a field in a related list of child records.

For all custom master-detail relationships

For limited standard relationships (Account-Opportunity and Opportunity-Product)

There is an option to include all records in the roll-up or just records that meet certain criteria.

You can create upto 10 roll-up summary fields per object.

*If you see the field definition, opportunity to account is a lookup field. But in terms of business logic it behaves as Master-Detail one ....* e.g. you can create Rollup summary field in account, if an account is deleted then related opportunity will be deleted.

It is a also a datatype field enabled only on Master Object which has MD relationship.

This will help you to calculate the Aggregate Functions on Parent Object record of child records

Sum()

Avg()

Min()

Max()

Count()

Count(\*)

**Scenario 38**

Implement Roll-up Summary field on Job Application and Review object which we discussed in the class. Complete this activity.

Rev Cultfit Leade Exper Interv

J002 R001 9 9 9 Kajol

R002 8 9 9 Anil

R003 9 8 8

Sum of Cultural fit = 9+8+9 =26

Sum of Leadeship= 9+9+8=26

Sum of Exper =9+9+8=26

No of Interviewers Attended =Count()=3

Total Score = 26+26+26=78 (Formula Field)

**Validation Rule**

* Validation rules verify that the data a user enters in a record meets the standards you specify before the record is saved.
* A validation rule contains a formula or expression that evaluates the data in one or more fields and returns a value of True or False.
* Validation rules also include an error message to display to the user when the rule returns a value of True due to an invalid value.
* Error message can be displayed directly below field or at top of the page.
  + Multiple error messages may be displayed at one time.

**Scenario 39**

Create a validation rule where Min Pay cannot be greater than Max pay. If the user still enters it should through an error Message “Min Pay Cannot be greater than Max Pay” to be displayed near Min Pay field…

**Enforce data format..**

**Scenario 40**

Candidate Object: - Build Validation Rules to **Enforce Data Format using regex()-** Use Reg Exp to ensure State/Province field should have only two characters in Capital letters.

“Please provide State as Two Alphabets with Capital Letters” should display error of the user not entering as per format.

**21-March-2024**

**Scenario 41**

Position Object. -Build Validation Rules on Hiring Manager field **to Enforce Conditionally Required Fields** and display an error message.

Create a lookup relationship on User object

User(M) Position (Child)

**Scenario 42**

If a position record Status is closed or Rejected, Date Closed is a mandatory for that record. If the user is not entering Date Closed or blank and hitting the Save button it should throw any error stating that “Please enter Date Closed” on the Date Closed field.

Code:

ISBLANK( Date\_Closed\_\_c ) && (ISPICKVAL( Status\_\_c , "Closed") || ISPICKVAL( Status\_\_c , "Rejected"))

**Scenario 43**

Create a validation rule on Account object to enforce users to enter a **Support Plan Expiration** Date field when **Does This Account has Support Plan** field is selected which is a check box.

**Create two fields**

Support Plan Expiration Date

Does This Account has Support Plan Check Box

***Throw an error “You must enter an expiration date when Account Support Plan ” if Support Plan Expiration field is empty and Does This Account has Support Plan check box is selected.***

***Solution:***

***Does\_This\_Account\_Has\_Support\_Plan\_\_c =True && ISBLANK( Support\_Plan\_Expiration\_Date\_\_c)***

**Scenario 44**

Create a validation rule on Opportunity object to enforce users to choose “Reason for Lost” field when the Opportunity Stage is Closed Lost

**Create a Picklist field called “Reason for Lost” values below**

Lost to the Competitor

Unable to Handle Customer

Multiple Negotiations Failed

Lack of Leadership support

Customer Not Interested

Other Reasons

**Description:** A close reason is required when Opportunity is closed lost

**Error Message:** You must select a Reason for lost when Opportunity is Closed lost

ISPICKVAL( StageName , "Closed Lost") && ISPICKVAL( Reason\_for\_Closing\_\_c , "")

**Workflow Rules**

What do u mean by workflow

1. Automating some things
2. A series of steps or step by step to complete a process automatically
3. Process to complete a task automate some thing

* Actions: Entry criteria: which records
  + Object Type
  + Evaluation Criteria

Whenever a new record is created

Whenever a new record is created and updated

Whenever a new record is created, and any time it's edited to subsequently meet criteria

Rule Criteria

Not all records enter into Workflows / Approval process / PB/Flows

We need to specific criteria / condition

* Timing: when to execute actions
  + Immediately - immediate u want to execute
  + Time Dependent - at certain time, date..
* What to do- Actions 4 (types of workflow actions)
  + Assign Task
  + Send Email Alert
    - From current user’s email address
    - From organization-wide email
  + Update Field
  + Post Outbound SOAP Message

\*\* Father of Automation process..

**Scenario 45 :** Create a work flow to automatically assign only Open Positions to recruiter Queue. Whenever a record is created or edited the record owner is changed to Queue.

Object - Position

Evaluation Criteria - created, and any time it's edited to subsequently meet criteria

Rule Criteria – Status =Open

Immediate

Action – Owner Field = Recriuter Qeueue

**Scenario 46:** Create a time dependent workflow for notifying owner to follow on extended offer? Create WF on Offer object. Whose offer status=sent.

Create an Email Template and send it to the Candidate if he she doesn’t accept or reject the offer within seven days

Object - Offer

Evaluation Criteria - Whenever a new offer is created

Rule Criteria – Status =Sent

Time dependent – 7 days from offer created date

Action – Email alert is sent to the candidate

{!Contact.FirstName}

! - expression tag

Contact – Object

Pull / fetch the first name field or data from contact object

Create a time dependent workflow on candidate object when a new candidate is created where

Gender = male or female.

Action: Email should be sent to candidate (A Welcome email has to be sent)

**Approval Process**

* **Approvals consist of 6 parts:**
  + Process Definition (Global Characteristics)
  + Initial Submission Actions (Workflow Actions)
  + Step Definition (Decision Criteria & Approval Assignment)
  + Final Approval Actions (Workflow Actions)
  + Final Rejection Actions (Workflow Actions)
  + Recall Actions (Workflow Actions)

Not all records will enter into approval process

Once the record enters into approval process record is locked,

only Sys Admin and Approver can also edit … where it should be checked…

Two types

Jump Start Approval Process

Means it is a single step or single page approval process

**Jump Start Approval Process:**

**Scenario 47**

If the Opportunity Amount is greater than 1000000. Kindly get this record approved by CEO Cynthia. Update the record Status field has Approved.

Note: Create a status picklist field on Opportunity with below values Pending, Approved, Rejected, Recalled.

Object - Opportunity

Criteria – Amount > 1000000

Approver – Cynthia

Final Approver Actions - Status field Approved

Final Rejection Actions - Status field Rejected

**22-March-2024**

**Scenario 48**

**Approval Process.**

Step By Step

Entry Criteria – Status Equals to New

When a new Position is created by Mario Riuz,

If the Paygrade <300 it goes to Phil for approval.

If the Paygrade >300 and 400 it goes for Phil for initial approval-> and Meghan Smith for final approval

Final Approval Actions  
 Status- Open  
 Sub Status- Approved  
Final Rejection Actions  
 Status- Closed  
 Sub Status – Rejected / Not Approved

Homework Activities

**Scenario 49**

**JumpStart**

**Make sure that prospect accounts are approved before they’re converted to customers**

Due to local regulation, new customers must be approved by the legal team. When a user creates an account that has a type of **Prospect**, they manually submit it for approval by clicking **Submit for Approval** on the account. The record is evaluated only if its type is **Prospect** and there are more than 500 employees. When the account enters the approval process, Type gets set to **Pending** and the record is locked. If approved, Type gets set to **Customer** and the record is unlocked. If not approved, Type gets set back to **Prospect** and the record is unlocked.

**Before You Start**  
Verify that the Account object's Type field has the following picklist values: **Prospect**, **Customer**, **Pending**.  
  
**Challenge Requirements:**

Use the Jump Start Wizard to create a new approval process for the **Account** object

The approval process name must be **Approve New Account**

Pick any email template for approval assignment email template

For Entry Criteria, use the following criteria.

**Account: Type** equals **Prospect**

**Account: Employees** greater than **500**

For Approver, choose **Automatically assign to approver(s)** and assign yourself as the approver

Click **View Approval Process Detail Page** and finish off the remaining requirements.

In the Initial Submission section, add a Field Update action that updates the Account: Type field to **Pending**. The name of the action must be **Account Type To Pending**.

In the Final Approval Actions, add a Field Update action that updates the Account: Type field to **Customer**. The name of the action must be **Account Type To Customer**.

In the Final Approval Actions section, edit the Record Lock action to unlock the record

In the Final Rejection Actions section, add a Field Update action that updates the Account: Type field to **Prospect**. The name of the action must be **Account Type To Prospect**

Activate the approval process

**Scenario 50**

**Standard Setup Wizard**

Create an automated approval process for ABC Company and CEO Cynthia requested an automated system for processing discount requests from her reps using these parameters:

Create an Approval Process on Opportunity Object.

* + - If a discount is requested for 15% or less, it’s automatically approved.
    - If a discount is requested for more than 15%, it must be approved by a manager Phil Khatz
    - If a discount is requested for more than 40%, it must also be approved Megahn Smith

**Go to Opportunity Object**

* + - Create a number field "Discount Percentage"
    - Create a picklist field "Approval Status"

Pending

Approved

Not Approved

**Create an Email Template: Discount Request Responses**

Dear {!Opportunity.OwnerFullName},

Good news! Your recent discount request has been approved. Please log in to your org for details.

Best,

Megahn Smith

VP

* Use Standard Setup Wizard, Entry Criteria – greater than >0
* Create Initial Submission Actions Approval Status Pending

**Process Builder**

At Universal Containers (UC), company policy states that hiring managers must always take part in the interview process. UC wants a new interviewer record to be created automatically for the hiring manager whenever a new position is created.

You are working in the abc organization. And your organization want when a new Candidate is created. His/her email id created automatically by using the last name or Full Name and with the company name (Example : shiva@abc.com). Whenever we will enter the data of new Candidate with there Full Name/Last Name, an email id will be auto-generated and Introduction Email to be sent to the candidate..

or

**Scenario 51**

Whenever a new candidate is created automatically an email id should be created with candidate name.

Email Id Creation - Update on the same existing record

Send an email - A welcome email.

**Scenario 52**

When an Opportunity record is closed Won and Amount >2Million Dollar,  Closed Date= Today()+10 days

Create an Order record where Order Start Date equal to Today (), Status value should be equal to Draft, Account ID- Reference the Opportunity AccountID

Action- Create a Record

**Flows**

Flows in Salesforce can be implemented in two ways

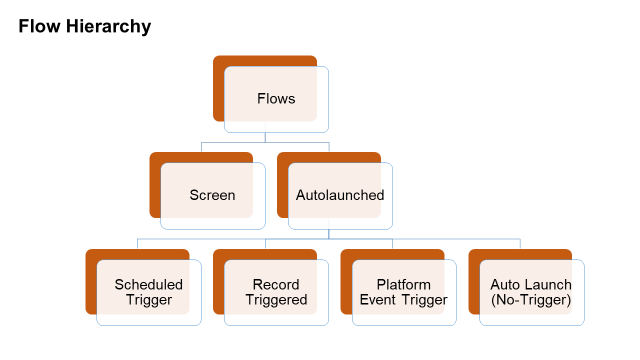
* Screen Flows
* Auto-launched Flow

**Screen Flows**

In this type of flow, there will be a series of screen elements to gather information from the user and perform some operation. Screen flows can be accessed from custom buttons, custom links, Visualforce Pages etc. This type of flow is implemented if a user interaction is needed in the process. Custom Screen has to be implemented.

**Auto-launched Flow**

Auto-screenshot runs in the background without any user interaction. Auto-launched flows can be accessed from custom buttons, custom links, Visualforce Pages, process builder and Apex etc. (Coding)

****

There are 3 main “building blocks” of any Flow:

1. Elements are the individual building blocks of the Flow. These perform logical actions such as assignments, decisions, or loops. There are also or commit record changes. data elements that will query the database

2. Connectors determine which element leads to which. Winter ‘21 enables Auto-Layout, and connects the Elements together automatically.

3. Resources are the individual variables of data that are to be used in a Flow – these can be strings of text, numbers, records, formulae, or collections.

**25-March-2024**

**Scenario 53**

Display a “HELLO WORLD” or “Hi – I am learning Salesforce Flows in Salesforce” on a screen flow and want this screen flow Message to be displayed /deployed on Home Page

**Scenario 54**

Display Logged In User Details on a Page Section with two columns

1) User First Name 2) User Last Name,

3) User Profile Name 4) User Email Id

5) User Orgaziation Name 6) User User Name

Use $Profile and $ User and $ Organization Global variables

I want to deploy this flow on Recruitment App- Home Page

**Scenario 55**

Universal Containers is seeing a number of candidate records missing information. They want a candidate wizard created that includes all the needed information to complete the record. Create a Custom Page where you can enter Candidate Details As Input Fields. Create variables where ever it is required

* 1. Candidate Name Text
  2. State & Province Text Validation Rule
  3. Mobile Number Phone
  4. Email Id Email ID Process Builder

Flow should take above fields as input values… and the record should be saved in candidate object from flow component that you created.

Input values in these fields..

Record should be created..

DML operation Create Records

Screen

Place screen components… variables will get created..

Create Record on Candidate Object

Map candidate object fields to screen component fields..

**Scenario 56: Create similar flow on Opportunity Object where..**

**Count all Mandatory Fields**

**Create an Opportunity Record through Flow Create Record (DML)**

**Scenario 57 :**

**Display a new Contact Page on Account Page. This should take First Name, Last Name, Email as Input from User. And Associate this contact record with an AccountID.**

1) Create a contact Flow and add that flow to a respective Account Page.

2) But you will not be able to save or insert the records. We need to do DML Operation

**Create a Records**

3) **But you will not be able to see in the Account Detail Page. We have to map with the respective Account ID.**

**But How to Associate to an Account?**

4) Create variable accountId / recordid variable and associate that variable to the Contact Record Page

5) Send an email alert to the newly created contact

Create an Email Template

Create an Email alert

**Scenario 58 Home Work**

**Display a new Opportunity Page on Account Page. This should take Opp Name, Close Date & Stage (Pick List) and Amount. And Associate this Opp Record record with an Account ID.**

1) Create a Opportunity Flow and add that flow to a respective Account Page.

2) But you will not be able to save or insert the records. We need to do DML Operation

**Create a Records**

3) **But you will not be able to see in the Account Detail Page. We have to map with the respective Account ID.**

**But How to Associate to a Account?**

4) Create variable accountId variable and associate that variable to the Record Page

**26-March-2024**

Data Management

Storing the records

Managing the data in database

Retrieving the data from Database

Performing DML operations (Insert, update, upsert, delete undelete etc etc)

In salesforce Data management is managed by below concepts

Point and Click / Declarative approach

1. Data Import Wizard
2. Data Loader
3. Data Loader.io
4. Workbench

Apex Programming

SOQL and SOSL

DML Statements

Database Class

Export, Export All, Insert, Update, Upsert

Delete, Hard Delete.

**Data Import Wizard**

1. This is an internal tool within the salesforce where you can perform certain DML operations
2. It supports

Insert

Update

Upsert

1. \*\*\*\*\*\*\*\* you can upload or insert upto 50K records only with Data Import Wizard
2. It will not support all Standard Objects but support all custom objects
3. \*\*\*\*\*\* Support only Account, Contact, Campaign member, Solution and Lead standard objects
4. Ensure that the data is in .csv file (Comma Separated value)
5. Once you are performing any dml operation ensure .. object fields should be mapped with .csv header columns…

**Scenario 59**

**Exercise: Activity for Salesforce Import Wizards**

1. **Create a Book Object with Fields**

**Book Name Text**

**BookID Text**

**Author Text**

**Book Type Pick List (Fiction, Classic, Drama, Biography)**

**Price Currency**

**Create a Tab**

1. **Add it to the Tab Section**

**Import or Insert - Data Import Wizard..**

**Scenario 60**

Perform Upsert Operation

Update some records in the .csv files

Insert three or four new records

INVALID\_OR\_NULL\_FOR\_RESTRICTED\_PICKLIST:Book Type: bad value for restricted picklist field: Emotion:Book\_Type\_\_c --

In case If I have more than > 50K records… where I want to insert bulk data…

**Data Loader**

**Scenario 61**

Data Loader is a client application for the bulk import or export of data. Use it to insert, update, delete, or export Salesforce records. When importing data, Data Loader reads, extracts, and loads data from comma-separated values (CSV) files or from a database connection. When exporting, Data Loader outputs CSV files.

1. This is an external tool you need to install this tool in your system
2. This requires Zulu JDK or JDK version >=17.0
3. It supports all custom and standard objects
4. It performs below DML operations

Export Export the data from objects

Export All Export the data from objects & from recycle bin

Insert / Import

Update

Upsert

Delete

Hard Delete - Permanently Delete

1. \*\*\*\*\*\*\*\*\*\*\* you can insert up to 5 million records at a time \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. Batch wise…2000 records …. Bulk update
3. We need to map salesforce object fields with .csv header columns
4. \*\*\*\*\* suggestive it whenever you are performing any DML operations… it is recommended that…

\*\*\*\* First you extract the data into a .csv file.. and use that file for any updating/ insertion or deletion etc etc..

* An easy-to-use wizard interface for interactive use
* An alternative command-line interface for automated batch operations (Windows only)
* Support for large files with up to 5 million records
* Drag-and-drop field mapping
* Support for all objects, including custom objects
* Process data in both Salesforce and Database.com
* Detailed success and error log files in CSV format
* A built-in CSV file viewer

About Data loader

<https://developer.salesforce.com/docs/atlas.en-us.dataLoader.meta/dataLoader/when_to_use_the_data_loader.htm>

Download: DataLoader

<https://developer.salesforce.com/tools/data-loader>

**Scenario 62**

Perform Export DML and export or extract all the records of book object into a .csv file.

**Scenario 63**

Perform Update, Upsert operations with (ID, Name) and \*\*\* External ID on Book object and see what happens?

Data Export:

\*\*\*\*\*\* 1 or two questions \*\*\*\*\*\*\*\*\*\*\*\*\*

Data Export lets you prepare a copy of all your data in salesforce.com. From this page you can start the export process manually or schedule it to run automatically. When an export is ready for download you will receive an email containing a link that allows you to download the file(s). The export files are also available on this page for 48 hours, after which time they are deleted.

**Reports & Dashboards**

**Reports**

1. Visualization of data in the form of charts
2. Collection of data
3. Easy to summarize the data
4. Get the outcomes of what we have did…
5. Analyze the data and take necessary or corrective actions
6. Based on the analyzed data you can do predictions
7. To make the data in understandable format
8. Representing the data clear and concise
9. Perform Trend analysis
10. Helps us for future projections / Forecasting

Reports in salesforce are dynamic nature…

Because data is stored on cloud

What salesforce .. they have already created Report templates for all Standard and Custom Objects

Reports in Lightning extraordinary

**Four types of Reports**

1. **Tabular Report**

Data is represented with Rows and Columns

1. **Summary Report**

The data is represented in the form of group by rows

1. **Matrix Report**

The data is represented in the form of Group by Rows and Group by Columns

1. **Joined Report**

Filtering the Reports

Implement Aggregate functions like Sum(), Min and Max

Reports in Classic vs Lightning

Printable Report only can be seen in Classic not in lightning

Download a Report in lightning and export a report in classic both are same

Add a Chart or Graph to a Report

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1 Certification

**Subscribe a Report**

When you subscribe to a report, you can define the set of conditions to meet before sending a notification, and choose how and when to be notified. Make sure to save any changes to your report before subscribing.

**Schedule a Report**

**Schedule a report that report will be scheduled at particular time.**

Add conditional formatting report

\*\*\*\* You cannot apply conditional formatting for Tabular Report. it can applied only for Summary and Matrix reports

Custom Formula Report

It can have applied only for Summary and Matrix reports

Custom Report Type

**What is a Custom Report Type?**

Custom report types allow you to build a framework in the report wizard, from which users can create and customize reports.

You build custom report types off of the relationships (master-detail and lookup) between objects so that you can:

Choose which objects to display to users creating and customizing reports

Define the relationships between objects displayed to users creating and customizing reports

Select which objects' fields can be used as columns in reports

Bucket Fields

**Scenario 64**

**\*\*\*\*\*\* HW – Create / Replicate all 16 reports on Recruitment Application objects…**

**Dashboard**

Without reports you cannot build or create dashboards

Dashboard is a combination of various reports representing the data in a visual format…

Dashboards not dynamic in nature but you need to refresh…

Dashboard can have upto 20 components…(20 different graph or chart components are there)

You can apply filters on dashboards

**Scenario 65- Create Dashboards on Recruitment and Sales Cloud applications**

**Sales Cloud**

Leads & Opportunities

Accounts and Contact Management

Automate the Sales Operations

Activity Management

Product from Salesforce

SAAS Product

Cloud based Platform

CRM

Vivek - CEO

Manufacturing Mobiles & Sports Motorcycles

Marketing Team

Marketing Cloud –

Conduct some Campaigns

WebtoLead

**Scenario 66**

**Create an App (Name it as some Sales App) and Add below objects.**

* Leads
* Organization/Accounts
* Contacts
* Opportunities
* Products
* Price Books
* Reports
* Dashboards
* Campaign (Most Important)

1. First Create an App (My Sales App) ...and add all the Standard Object-
2. Create a Campaign Web to Lead - (Generating Leads from a Web campaign)

Add in Type-Field - Add Web to Lead.. Campaign

**Leads-**

**You need to create as many as templates / Configure**

i) Email Template - Auto Response

ii) Auto-Response Rules

iii) Lead Assignment Rules Lead will get automatically assigned to a Sales user./ Sales Queue

Create an Lead Assignment Template..(Email Template)

***\*\*\*\*\* USer should have valid Salesforce User License..\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

**Go to Email templates and check if these templates are available**

iv) **Web Lead to Form**

Using pre-existing pages on your company's website, you can capture contact and profile information from users and automatically generate new leads in salesforce.com, enabling you to respond in real-time to customer requests.

3) Configure **Auto Response Rule - Create an Email Template**

**Three responses**

**Email. Message, thank you message over a web**

4) Configure **Lead Assignment Rules (Automatically assign to Respective SalesRep**)

\*\*\*\*Automatically assign leads to users or queues based on criteria you define. You can create multiple rules with different conditions, but only one rule can be active at a time. \*\*\*\***Email Template –Create**

Help for this Page

5) **Create an Auto Response Rule and** also create multiple email template as per your Sales process.

Automatically determine which email templates to use when sending auto-response messages to new leads from your website.

6) **Create Lead Assignment Rule** - Assign lead to a specific Sales USer..Lead will get automatically assigned to a Sales user./ APAC Sales Queue

Automatically assign leads to users or queues based on criteria you define. You can create multiple rules with different conditions, but only one rule can be active at a time.

Assignment Emails.. to be created..

1. Country = USA, India, Singapore
2. Lead Annual Revenue >=1000000

**Lead Generation Process**

7) **Web-to-Lead Setup**

Configure on Customer Application web

Using pre-existing pages on your company's website, you can capture contact and profile information from users and automatically generate new leads in salesforce.com, enabling you to respond in real-time to customer requests.

Where you can generate the leads from Web Application.

Code to generate web-to lead- which will automatically connect to lead object of SF..

Web-to-Lead (Built in code is there) – Configure.

**-------------------------------------------------------------------------------------------------------------------------**

**Configure Web to Lead**

8) Create Web to Lead (Create a Record) automatically Lead will get created in Leads Object

Sales user will start connecting with the lead to collect some more information and also will set up a meeting to give demo of your products.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9) Create an Email Template and send it to the correct lead.

10) Change the lead status as contacted on the Kanban Board

11) Before you convert that lead - Phone, Rating, Email- Mandatory fields should be updated. you cannot convert that lead

**12)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Lead Conversion Process \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Create a Validation Rule so that without updating Email, Phone and Rating, we cannot convert a lead into Accounts, Opportunities and Contacts. (Adding Functionality)

Business Requirement

AND(OR( ISPICKVAL( Rating , ""), ISNULL( Phone ) , ISNULL( Email )), ( IsConverted =true))

Once you convert the lead.. it will create Accounts, Opportunities and Contacts.

We will not see that lead record in the leads obect.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Every Opportunities has to purchase a Product or a Service. Hence I need to Create a Product. You can always edit and create extra fields and Product Family fields… Product is part of Opportunity Object. Here you will give the Product information but not price information.

**Account - Contact - Opportunity**

**Opportunity is related to Product, Price Book and Quote..**

Product is related to Price Book is related to Price Book Entries

Opportunity -🡪 Product - > Price Book ->Price Book entries

**Product - Price Book Process**

**Product Company - HUL – Products / FMCG – Maggi / Nestle /**

|

**Super Stockiest (SS)**

|

**Distributors**

**Wholesalers**   **Retailers Customers**

Two different Price Books 1) Standard 2) Standard Price book are created by Salesforce

13)Create and Add Products into the Product Object. (Product and Opportunities are related)

14)\*\*\*\*Go to Price Book and Add the Products first to the Standard Price Book and add price for each Product.

You have added the Price for each and very Product

SEZ – Special Economic Zone.. exempted GST … PB other PB

15)Go to the Product Choose the Standard Price Book and add the respective Products to your Opportunity.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Product and Price Book Process\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Scenario 67**

**Quote Process.**

16)You need to send a Quote to the customer on the total value. For this enable Quote object by typing Quote in the set-up menu.

Quote is related to Quote Items

17)Create a Quote and add all the products to the Quote and if require give discount to the customer. -

18)Sync that Quote and create the quote in an PDF file and share it to the Customer (Revised Quote) -

**Rejecting your quote..**

**Revising the Quote…**

19) Update the Opportunity and compare the total Amount values in the opportunity and quote and both should match.

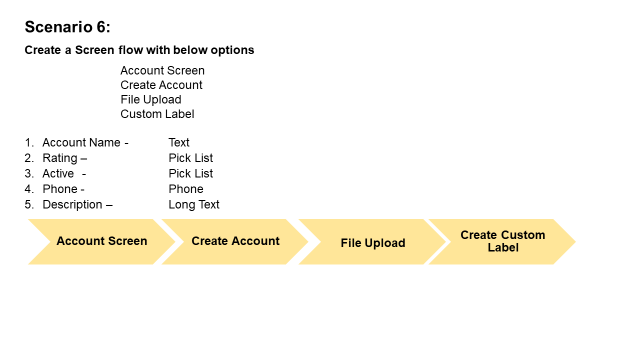
20)Update the Opportunity as Closed Won and update the fields like Lead Source and Primary Campaign Source.

21 You can see the updates in the Campaign object with the Opportunity Cost and Closed Won.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Please complete or practice Sales Cloud as per above steps given \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Additional use cases to solve Flows**

**Scenario 68**



**Scenario 69:**

**Create a Flow where you can Get the Account Records passing the ID by using Get Records DML Operation.**

1. Get Account Record
2. How to implement Decision

3) How to display Fault Message

1. Create a text variable called recordid
2. Create GetAccountRecord
3. Condition AccountId = recordid
4. Create a Decision wit hAccount Record Found & Account Record Not Found

For Account Record Found - {!Get\_Account\_Records.Id} does not equal to null

Account Record No Found just empty

5 . Create a Screen flow with Account Name and Phone Fields and map with {!Get\_Account\_Records.Name} & {!Get\_Account\_Records.Phone} in default values.

1. Create another Screen flow with Record Not Found and display the Fault Message.
2. Add a faultpath at GetAccountRecord Screen if the Account not found
3. Create a display text element and add $flow Error (**{!$Flow.FaultMessage}**

**Scenario 70:**

Create a flow where get an Account Record and fetch the related Contact Records for that specific Account Record.

1. Follow the steps of Scenario 8 till end ( 8th Step)
2. After Record Found (GetRecordScreen)

add GetContactRecord (DML)

Select Object – Contact

Conditions Met - AccountID equals – recordid (Which we created as a variable)

Select All Records

Select Automatically store all fields

1. To display multiple contact record
2. create Loop Element and give a name to the screen
3. Collection Variable - Select GetContactRecords
4. specify “First item to last item”.
5. Then Add Screen Element to fetch the Contact Records and select Loop Get Contact Details Values for each element components Add First Name, Last Name and Phone Fields
6. Edit the Footer label of Next to “Click Here for Contact Details”

**Scenario 71:**

Create a Screen Flow where the flow needs to fetch Account Record based on the user input which is an Account ID.

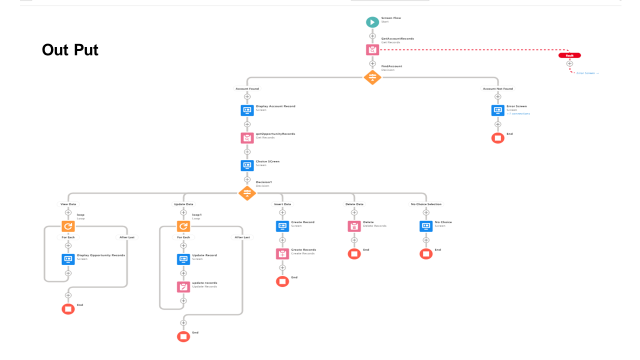
1. IF the Account Found use (Decision) Display Account details on a Screen
2. IF the Account Not Found use (Decision) Display Error Message on Screen

3) If the Account Found – Use Get Records and get the Opportunity Related Records of that Particular Record- Perform Four Choices and user has to choose Insert, Update, Delete and View – Create these choices

4) Perform or use Loop to iterate more than one Opportunities records are there

5)If the user select No Choice Selection. IT should display No Choice selected screen

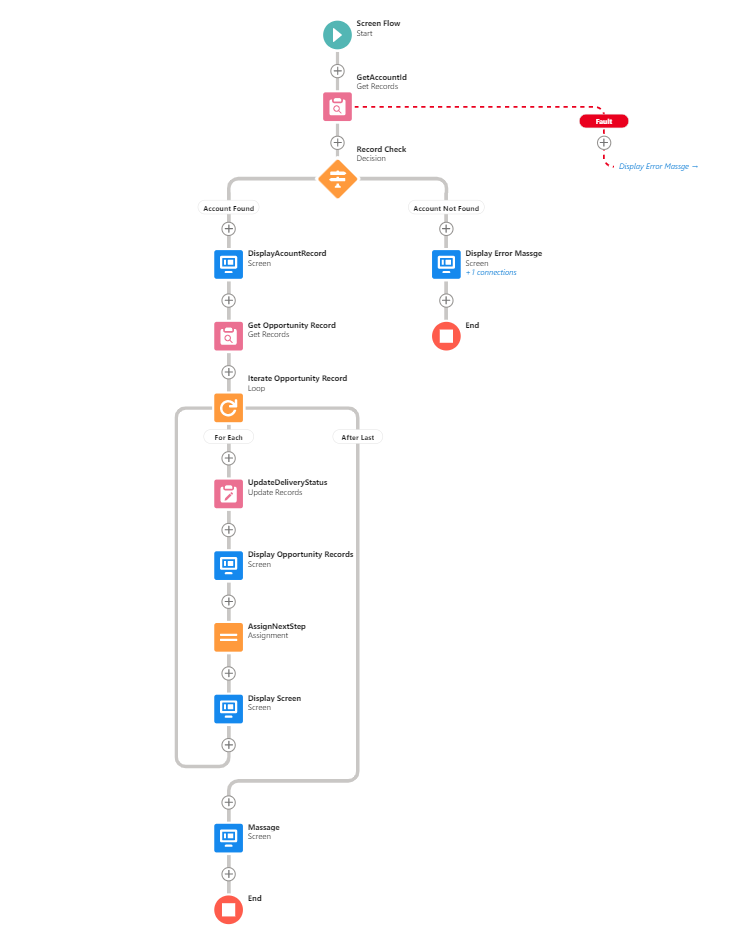
Output:



**Scenario 72:**

**Create a flow where get an Account Record and display that record and fetch the related Opportunity Records for that specific Account Record. And Update record for Delivery Status where value is "In Progress" To Completed. And Then Assign value for next step from user input.**

1. Get an Account Record and display Some fields on Flow with an AccountID.Use GetRecords Data.Create a variable “accountID” Create a Screen and add few field and Map these fields with “GetAccountRecord” Name Save it and Pass a Record ID. Out-Put You will be able to see a specific record.
2. IF the accountId is invalid will the flow will fail?
3. Add Decision Element “Account Not Found” and “Account Found”.
4. Display the Account Details from the Get Accounts – Create a Screen with fields and tag to GetAccountRecord variables.
5. Add Error Message screen for Account Not Found Decision Element with $Error Fault.
6. For a Specific Account get Opportunity Details add account ID variable More than one record.
7. To display multiple opportunity record create Loop Element and specify “First item to last item”.
8. Then Add Screen Element to fetch the Opportunity Records and select Loop Get Contact Details Values. Add Mandatory Fields.
9. Go to Get Account Records and add Fault and Link the Account Not Found Record screen.
10. Edit the Footer label of Next to “Click Here for Opportunity Details”.
11. Edit the Footer label of Next to “Next Opportunity Record”.
12. Now inside loop add “UpdateDeliveryStatus” which will update the delivery status to completed.
13. And add one assignment named as “AssignNextStep” to assign the next step value to as user input.
14. And add one more screen to display the assigned value.
15. At last add one more screen to display “Thank You” .



**Scenario 73:**

**Create a flow where get an account record and display that record and fetch the related Opportunity record for the specific account record. Also perform create, delete and update record where delete and update are connected in a loop.**

1. Get an Account Record and display Some fields on Flow with an AccountID Use GetRecords Data. Create a variable ‘accountID’.
2. Create a Screen and add few field and Map these fields with GetAccountRecord Name
3. Save it and Pass a Record ID. Out-Put You will be able to see a specific record.
4. IF the record ID is invalid will the flow will fail?
5. Add Decision Element Account Not Found and Account Found.
6. Display the Account Details from the Get Accounts – Create a Screen with fields and tag to Get AccountRecord variables.
7. Add Error Message screen for Account Not Found Decision Element with $Error Fault.
8. For a Specific Account get Opportunity Details add account ID variable (More than one record).
9. To display multiple opportunity record create Loop Element and specify “First item to last item”.
10. Then Add Screen Element to fetch the Opportunity Records and select Loop Get Contact Details Values. Add Mandatory Fields.
11. Go to Get Account Records and add Fault and Link the Account Not Found Record screen.
12. Edit the Footer label of Next to “Click Here for Opportunity Details”.
13. Edit the Footer label of Next to “Next Opportunity Record”.
14. After getOpportunity loop end, use create record to add record in Opportunity Record.
15. Create a loop to delete Opportunity record using Opportunity Name according to user wish.
16. After that create a loop to update Opportunity record using Opportunity Name according to user wish.
17. Connect update loop with delete loop. If update request is no, end.

**Service Cloud**

After Sales …….

It is a process or a framework once a case is generated and successfully closed.

Any issues facing by customer we will rectify those issues

Log the issues and get it resolved by the service team

Annual Maintenance contract.

Customer Service is outmost important gradient

Below are the varied channels to register a case

Email

Chat

Web to Case

Phone / Mobile Number

Walk-In to Customer care…

Messaging agents

Social Media

Framework , Tools available in the market

Cases, Service Request, Incidents, Complaints and issues

How we can resolve a case

**Solutions** & **Knowledge Articles**

**Scenario 78:**

**Step- No 1**

* Service Cloud Objects / or Service Cloud App

**Add Objects**

Accounts

Contacts

Opportunities

Cases

Solutions

Knowledge

**Step-2**

Go to Case Object and add “Hardware” and “Software” Values to the type Field picklist.

**Step-3**

Create Case Auto Response Rules – Configure & Create an Email Template - Auto Response Email Template

**Step-4**

Create Case Assignment Rules – Configure & Create an Email Template stating that the case is assigned to a Service Engineer

**Step 5**- Review the Case Object

Cases can be generated

Call

Walk to a Customer Care center

Email and Web

Messaging Event.

**Step No 6**

* Create Web-to-Case HTML Generator

**Step No 7**

Company Business Hours

Enable Business Hours Field on all Case Page layouts…

Business Hours

Select the days and hours that your support team is available. These hours, when associated with escalation rules, determine the times at which cases can escalate.  
If you enter blank business hours for a day, that means your organization does not operate on that day.

* Go to the field accessibility of Business hrs. field and enable as visible for Sys Admin and Recruiter Profile

**Step No 8**

* **Configure Case Escalation Rules -**

**Scenario**

Work Flow- If the case created > business hours, the status field should be changed to Escalated and email alert should go to the Manager

**Solutions and Knowledge will help the customer executive to provide solutions on the spot. Or immediately to the customer**

SME- he might have created some solution – for this kind of cases.

**Step No 9**

* Solutions is a standard object
* It is available only classic.
* **Solutions** are not upgraded in Lightning. It is available in Classic
* **A solution** is a detailed description of a customer issue / case and the resolution of that issue or case. ***Solution managers, administrators, and users with the appropriate permissions can create, review, and categorize solutions.*** They can also publish solutions to the Self-Service portal and make solutions public.
* The Solutions tab displays a home page that lets you quickly locate and manage solutions.

**Scenario 79**

**Additional Activity-** You can create a solution and send this solution for Approval to a Solution Manager User.

Case and solution are related.

How do you add a solution to a Case?

**Scenario 80: Additional Activity –** You can create a formula field called **Duration Taken to Close the Case**. which will calculate the total duration taken to close the case

**Date/Time Closed - Date/Time Opened**

**Knowledge Base and Articles**

**Experience Cloud**

Partner Portal

Customer Portal

Salesforce Experience Cloud lets you create branded digital experiences to share information and collaborate with people who are key to your business processes, such as customers, partners, or employees. Whether you call it a portal, help forum, support community, or something else, an Experience Cloud site is a great place to connect with the important folks in your life.

**What Is Experience Cloud?**

With Salesforce Experience Cloud you can build CRM-powered experiences quickly and easily. Experience Builder, Salesforce CMS, Mobile Publisher, and a suite of web apps join to help you create robust, customized sites for your every use case.

With Experience Cloud, customers and partners can build digital experience platforms to host marketing and corporate sites, partner portals, support forums, web applications, and more. You can start from a selection of prebuilt templates. Use the Customer Service template to build a self-service experience where your customers can help each other, which increases case deflection and user engagement. Create portals where customers can process insurance claims, access their healthcare information, schedule vaccine appointments, and purchase products from stores. Customers can process warranties and self-service claims on automobiles.

Companies can manage inventory and make purchases. Use the Salesforce platform to extend the power of CRM data to partners so that they can manage channel accounts and boost products with marketing campaigns.

**Build Your Site, Your Way**

Whether you’re an admin, designer, or developer, Experience Cloud makes it easy to work the way you want. You can:

Create customized sites using no-code, point-and-click tools and prebuilt templates, themes, and components.

Build custom components, apps, and processes using low-code tools for interactive development and automation.

Develop customized, enterprise-level sites and portals programmatically. Update sites quickly, using your preferred pro-code development tools such as Salesforce Extensions for VS Code, Salesforce CLI, or your favorite IDE or text editor.

**Experience Cloud Glossary**

Here are definitions to some key terms and concepts related to Experience Cloud.

**Experience Cloud**

Salesforce Experience Cloud is a digital experience platform that makes it easy to build beautifully designed, CRM-powered online sites. What was formerly known as Community Cloud is now called Experience Cloud, to better reflect the many types of digital experiences you can create and connect.

**Experience Cloud Site**

An Experience Cloud site is an online space for connecting with your customers, promoting collaboration, and streamlining key business processes.

**Experience Builder sites,** formerly called Lightning communities, are template-based sites that you customize in Experience Builder. These sites are built on one of two frameworks—Lightning Web Runtime (LWR) or Aura.

LWR sites are built with the latest LWR-based templates, such as the Build Your Own (LWR) template.

Aura sites are built with our original templates that run on Aura, such as Customer Service, Partner Central, and Customer Account Portal.

Salesforce Tabs + Visualforce sites are sites that you develop using Visualforce.

For simplicity, we use the terms LWR sites, Aura sites, and Visualforce sites where possible to help you identify which features apply to your site.

Communities / Community Cloud

Experience Cloud

LWC / Aura Components are built

Advance / Modern HTML

CSS

Javascript & ES6

Schema Builder

Process Builder

Flow Builder

Report Builder

Dashboard Builder

LWR- you can create your own template

\*\*\*\*\*\*\*\*\* In developer edition **Maximum number of published and unpublished sites: 100 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Record Triggered Flows**

Trigger Events

When an Opportunity is Created

Updated

Criteria

Amount = highvalues > 2000000

StageNAme=Closed Won

Action What is the action that you want to perform

Email Alert

Create a Record - Related Record (Contract Record is created)

Create an Product Record

Update a Field

When do u want to execute

Immediate or Record Changes

Time Specific - Time dependent (Scheduled Triggered Flows)

Record Trigger flow are two types

After Triggers

Before Triggers

After the record is saved

Before the record is saved

Apex Triggers

Two types

After Triggers

Before Triggers

**Scenario 81:**

**Create a Record Trigger Flow Whenever a New Account gets created a Welcome Email Need to be send to Account User**

Trigger

After Trigger

A new Account record is created

Criteria

No Criteria

Action

Send an Welcome email to the Account user..

Email Template and Email Alert

**Scenario 82: After the Record is Saved**

**After creation of an Account an opportunity record is created which will be associated to that Account**

**Object** – Account

Trigger is after Triggers.

**Configure Trigger – A Record is Created**

**Condition or Criteria Requirements** – None

**Actions:**

Create an Opportunity Record..

Category(Select Action & Related Records)

Closed Date Field Mandatory

Stage Name Field Mandatory

**Scenario 83:**

Create a record-triggered flow that creates a task for a high-value opportunity Amount > 1000000 whenever a new Opportunity is

**Created or edited**

**After Record Saved**

**Configure Trigger / Criteria / Condition**

All conditions are me

StageName = Negotiation/Review

Amount > 1000000

**Action :**

**Element: Create Records**

**Object: Task**

**Field values:**

**Subject = Follow up with the account owner by email (we won’t check the actual text)**

**Activity Date = enter today’s date (we’ll only check that you entered a date; it could be any date) Today()+5**

**OwnerId = $Record > OwnerId**

**WhatId = $Record > AccountId**

**Scenario 84:**

**Create a scheduled Triggered flow** on Account Object. When a new account is updated with Annual Revenue = 1500000 Update description as a Diamond Client.

Date : Today()

Timing: 20 minutes

Frequency-Once

Scheduled a Time

Condition- Annual Revenue - 1500000

Update - Description as - Diamond Client

**Scenario 85: Scheduled Record Triggered Flow**

Create a scheduled Triggered Flow when an Opportunity is stage is closed won send an email alert to Opportunity Owner. Scheduled this automation process on daily basis

1. Create an check box as Email Sent field on Opportunity
2. Create a Scheduled Trigger
3. Set Flow Data as Today and Frequency as Daily
4. Select Opportunity object and set the conditions as

Stage - Equals – Closed Won

Is Email Sent – Equals - False

1. Then Create an email template
2. Create an Email alert
3. Add Email action to your flow and select the email template - $Record ID- OpportunityId
4. Select Update Record DML action and add it to your flow and set the email status field

Email Sent Equals True

1. Connect the screens and send the time as 5 minutes > current or system time
2. Edit a Record in Opportunity i.e closed won stage records.
3. You can see your flow in Scheduled job section which is ready to execute at that time
4. Once the time reached automatically email alert is sent to the Record owner

**Scenario 86:**

Create a record-triggered flow that creates a Contract Record with Status as Draft for a high-value opportunity Amount > 1000000 and Stage Name equals Closed won

whenever a new Opportunity Record is

**Created or edited**

**After Record Saved**

**Configure Trigger / Criteria**

All conditions are met

StageName = Closed won

Amount > 1000000

Actions and Related Record

Create a Contract Record

Status = Draft

Account Name=$Record->AccountId

**Scenario 87:**

**Whenever an Account Parent Records get updated its related contacts records should get deleted**

1. Object – Trigger
2. Trigger – When a Record is Updated
3. No Conditions
4. Related Records
5. Get Contacts - Object- Contacts

Conditions which records to be deleted

AccountID Equals $Record.AccountID

First Name Equals Shiva

All Records – Click Done

6) Delete Contacts – Delete DML

Use the IDs stored in a record variable or record collection variable

Record Collections- Get Contacts

Click Done

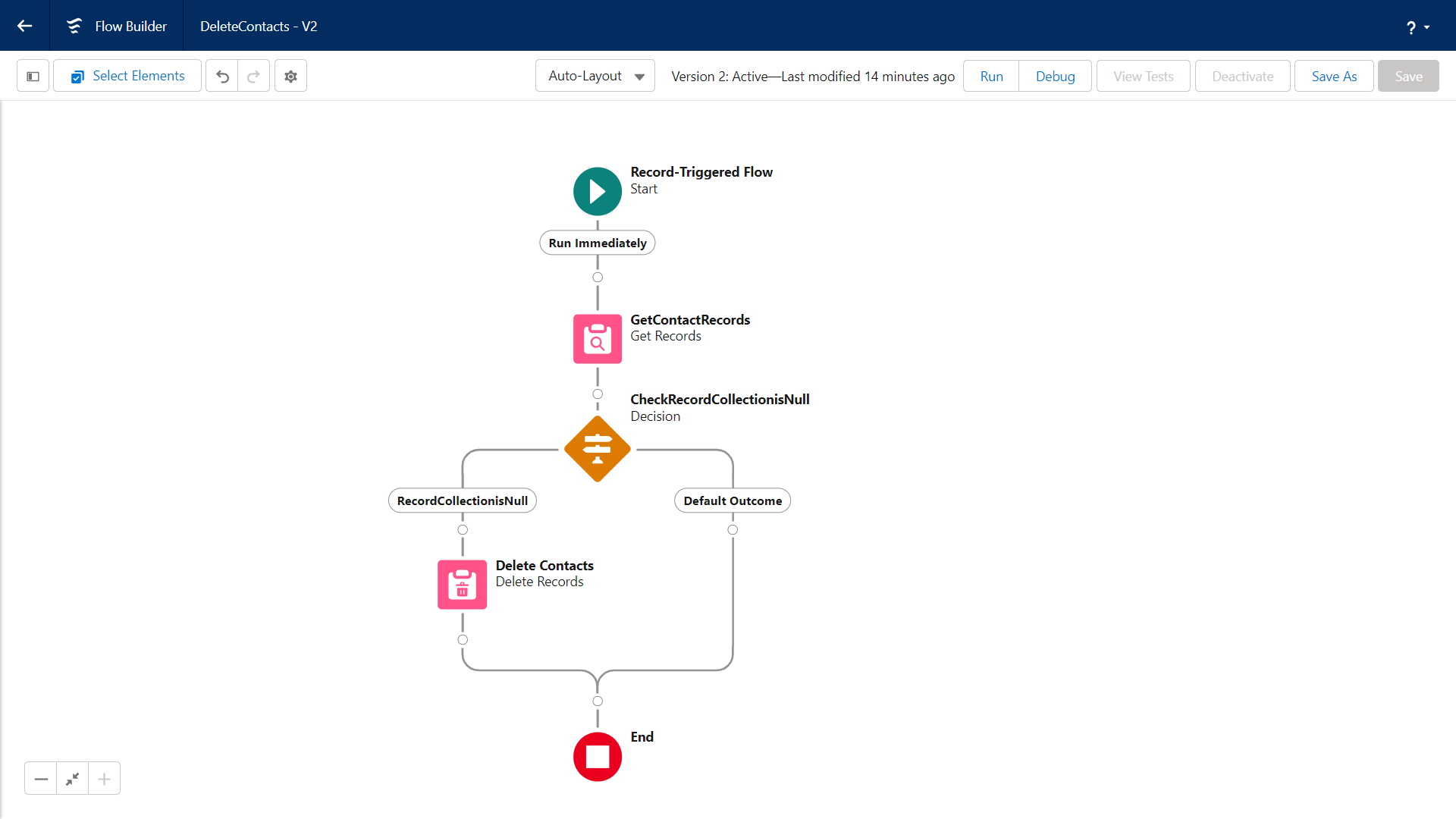
Activate

7) Go to Account and update an Account with contact as Shiva- corresponding records will get deleted

8) If you try to update a record and save it will through an error as it searches Get Contacts Record collections and it is empty .. for that

Add Decision Structure

Output:



**16-April-2024**

**Platform Developer-1 & 2**

**Programming Concepts - Apex**

**OOPs - Apex**

Class

Object

Inheritance

Encapsulation

Polymorphism

Abstraction

Interface

**Collections**

List

Set

Map

**Query Editor**

**SOQL – Salesforce Object Query Language**

Child to Parent Relationship

Parent to Child Relationship

**SOSL – Salesforce Object Search Language**

Will write & execute Apex and SOQL

Anonymous Window

Apex Programming

In declarative approach- Data Import Wizard, Data Loader, io and workbench.. point & click…

**Exception Handling**

**Here we will write Apex to perform DML operations**

**DML – Data Manipulation Language Statements**

Insert

Update

Upsert

Merge

Delete

UnDelete

**Database Class –**

**Visual Force pages**

Standard Controller

Standard List Controller

Custom Controller

Extension Controller

**Apex Triggers**

**Advance Apex**

@future method

Batchable Apex

Queueable Apex

Schedulable Apex

Invocable Apex

**Deployment Process**

Types of Sandboxes

Create an Account in Enterprise Edition

Will Create a Sandbox

**Two types of deployment**

Deploy a Built in Application from Sandbox to Production

Inbound Change sets

Outbound Change sets

**Apex Testing**

**Test classes**

**Deploy our code from Sandbox to Production environment**

**Customer con=new Customer();**

Customer- class

Con - Reference variable/ instance of a class / object

New – providing or allocating memory for that reference variable

Customer() – Constructor.. by default when a class is created a default constructor is created and invoked..

Int - primitive datatype

Integer - wrapper class

Decimal price=89.76;

String Name=”Venkat”

String Name=’Pavan’;

Name=98.87;

Price=’Pawan’

Let,

Var,

Const

Apex code - Class , Triggers,

Apex code in Anonymous window

\*\*\*\* SOQL - Query Editor

In Apex Class

Anonymous Window

<https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_date.htm#apex_System_Date_dayOfYear>

<https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_datetime.htm?q=Date>

Date myDate=Date.newInstance(2020,3,13);

System.debug('Old Date ' +myDate);

Date newDate=myDate.addDays(10);

System.debug('New Date ' +newDate);

Date newmonthDate=myDate.addMonths(4);

System.debug('New Date with new month ' +newmonthDate);

**Objects**

**Classes**

**Parameterization**

**Constructor**

**This Keyword**

What do you mean by object?

Real word entity

Car … - different types of car..

Prototype of car… / blue print…

**Object - Sates & behaviors**

States - fields / Properties / attributes / variables

Variables String Color = ‘Pink’

String Name

String Model

Decimal Price

Methods

Drive()

Speed()

Accelerating()

GPS()

Instantiate… class car.. objects / reference variables

BMW

Merck

Audi

Write an Class called Dog

States – Name

Type

Color

Behavior - Biting()

WagingTail()

Barking()

Create objects called - Labridor, GermanShephar, Pomerian

\*\*\*\*\*\*\*\* Null pointer exception



**Constructor** in Apex Programming is a code and is a special method that is invoked when an object is created from the class.

Constructor has the following properties.

1. Method name will be same as Class.
2. Access specifier will be public.
3. This method will have invoked only one that is at the time of creating an object.
4. This is used to instantiate the data members of the class.

Types of Constructor in Apex programming.

1. Default Constructor.
2. Non-parameterized Constructor.
3. Parameterized Constructor.

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**this KeyWord:**

You can use **this keyword** in dot notation, without parenthesis, to represent the current instance of the class in which it appears. Use this form of the  this keyword to access instance variables and methods

public class ThisDemo {

public String CompanyName;

public ThisDemo(String n)

{

this.CompanyName=n;

System.debug('The company name is ' + CompanyName);

}

public void accept()

{

System.debug('I am living in SipCot');

}

public void display()

{

System.debug('Which is in Sirsurei');

this.accept();

}

}

/\*ThisDemo td=new ThisDemo('Cognizant');

td.accept();

td.display();

\*/

///////////////////////////////////////////////////////////////////////////////////

Inheritance

Single

Multilevel

Hierachial

Encapsulation

Getter and Setter

Polymorphism

Method Overloading

Method Overriding

Abstraction

Abstract Method

Apex Properties

Interfaces

**Inheritance**

Inheritance is one of the key features of Object Oriented Programming. Inheritance provided mechanism that allowed **a class to inherit property of another class**. When a Class extends another class it inherits all **non-private members** including fields and methods.

**Extending a Class**

You can extend a class to provide more specialized behavior.

* A class that extends another class inherits all the methods and properties of the extended class. In addition, the extending class *can override the existing virtual methods by using the override keyword in the method* definition.
* Overriding a virtual method allows you to provide a *different implementation for an existing method*. This means that the behavior of a particular method is different based on the object you’re calling it on. This is referred to as polymorphism.

A class extends another class using the extends keyword in the class definition. A class can only extend one other class, but it can implement more than one interface.

**Encapsulation:**

The wrapping up of data and methods together is called encapsulation. For example, if we take a class we write the variables and methods inside the class. Thus, class is binding them together. So class is an example for encapsulation.

**Encapsulation**

The whole idea behind encapsulation is to hide **the implementation details from users**. If a data member is private, it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

**\*How to implement encapsulation in java/Apex:**

1) Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.

2) Have getter and setter methods in the class to set and get the values of the fields.

3) Private members cannot be accessed outside of the class. If you want to access private members outside of the class you must write getter and setter methods

Public String getName(String n) //// getter

{

Return Name;

}

Public void setName() ///// setter

{

Name=n;

System.debug(‘The value of Name is ‘ +Name);

}

**Automatic Properties**

**Using Automatic Properties**

Properties do not require additional code in their get or set accessor code blocks. Instead, you can leave get and set accessor code blocks empty to define an automatic property. **Automatic properties allow you to write more compact code that is easier to debug and maintain.** They can be declared as read-only, read-write, or write-only. The following example creates three automatic properties:

Public Integer y{set;get;}

**Abstraction :**

1. We cannot create an object for Abstract Class
2. If you use Keyword called Abstract .. that class become abstract class
3. Atleast one abstract method should be there..
4. You can declare an abstract method but you cannot define or implement that method..
5. Abstract class can have other concrete methods..

Abstract Class:

            1. It's a class which contains at least one abstract method.

            2. abstract class and method must be prefixed with 'abstract' keyword

            3. abstract class cannot be instantiated

            4. When a class is inherited from an abstract class, it must implement abstract methods of an abstract class, otherwise the derived class also becomes  abstract class

            5. By default abstract methods are virtual.

Example :

Cognizant – Company

Two types of employees

FullTime Employee ContractEmployee

Salary calculation will vary Salarayc calculation will vary..

Arrays

Collection of similar type of data…

Datatype [] variablename = new Datatype[];

Integer [] marks= new Integer[6]; /// Array

Integer [] marks=new Integer[6]{98,87,76,65,87};

A collection is a type of variable that can store multiple items. We have a sugar bowl with several sugar cubes inside it. The sugar bowl is considered a collection and the sugar cubes are items that are stored in that collection.

\*\*\*\*\*\*\*\* The collection is the type of ***variables that can store multiple numbers of records***\*\*\*\*\*\*. It can increase and decrease dynamically.

It is used to store, fetch and manipulate data. For example, list is used to collect elements and referred by a list object.

**Collections are of three types**

List

Set

Map

**List**

It creates a new instance of the List class. A list can hold elements of any data type. A list is an interface.

A list is an ordered collection of typed primitives, sObjects, user-defined objects, Apex objects or collections that are distinguished by their indices.

To declare a list, use the List keyword followed by the primitive data, sObject, nested list, map, or set type within <> characters.

List<Integer>marks=new List<Integer>();

List<Students>stud=new List<Students>()

List<Movies>mov=new List<Movies>();

List<Account>acc=new List<Account>();

Student<String>stud=new Student<string>();

Student is a class

String is a datatype

Stud is an reference variable or object

<string>() // type safety

**LISTS (Ordered Collection / Array[]) . Duplicates will be taken.** List can contain any number of records of primitive, collections**, sObjects**, user defined and built in Apex type. This is one of the most important type of collection and also, it has some system methods which have been tailored specifically to use with List. List index always starts with 0. This is synonymous to the array in Java. A list should be declared with the keyword 'List'.

**Add(), Size(), get(), clear(), set()**

List<Account>acc=[Select Id,Name from Account];

System.debug('Account Values ' +acc);

**List Example**

**public class ListDemo**

**{**

public static void main()

{

List<String>Student=new List<String>();

Student.add('Pooja'); // add the elements to the list

Student.add('Varsha');

Student.add('Rahul');

Student.add('Raj');

Student.add('Sayanthan');

System.debug('Student List' +Student);

List<String>NewName=new List<String>(); // one more list

NewName.addall(Student); // Addall() is a method adds all the elements in a specified

//listt..both lists should be same datatypes

System.debug('Newname List' +NewName);

system.debug('List Size is ' +NewName.size());

Newname.add('Sunny');

Newname.add('Shubham');

System.debug('Newname List' +NewName);

system.debug('List Size after adding is ' +NewName.size()); //size of the list

List<String>firstlist=new List<String>();

firstlist.add('Mumbai');

firstlist.add('Chennai');

firstlist.add('Hyderabad');

firstlist.add('Delhi');

for(String name1:firstList) // for each loop

{

System.debug('First List values ' +name1); // iterating the name1

}

System.debug('List value in 2 index ' +firstList[2]);

firstlist.remove(3); // remove the specific index value

System.debug('After remoing values from index 3' +firstlist);

firstlist.sort();

System.debug('Aftersorting the values ' +firstlist);

List<String>studplaces=new List<String>(); // empty List

studplaces=firstlist.clone(); // after cloning firstlist..

studplaces.add('Ranchi');

System.debug('After cloning the values in studplaces list ' +studplaces);

List<String>States=new List<String>();

// Statement is showin null value in States List

System.debug('The values in the List ' +States);

// Add elements to the list using add() metod

States.add('Telangana');

States.add('TN');

States.add('Maharastra');

States.add('Maharastra');

System.debug('The values in the List ' +States);

// get() method...to get a specific index

System.debug('Get the value in the index(0) ' +States[0]);

// set the index 0 value to Delhi

States.set(0,'Delhi');

System.debug('Get the value in the index(0) ' +States[0]);

System.debug('The values in the List ' +States);

States.clear(); // Clear the List

System.debug('The values in the List ' +States);

//Equals(List) - compares the list with the specific list and returns true if both are equals

Boolean checkequality;

checkequality=studplaces.equals(firstlist);

system.debug(checkequality);

}

}

///////////////////////////////////////////////////////////////////////////

**SET Collection:**

A set is an unordered collection of elements that do not contain any duplicates. Set elements can be of any data type—primitive types, collections, sObjects, user-defined types and built-in Apex types.

Set methods in Salesforce do not allow duplicate. The insertion order is not preserved in the set. It also grows dynamically in run time like a list. **You cannot perform DML with Set.** To declare a set, use the Set keyword followed by the primitive data type name within <> characters. Moving ahead, get set methods in Salesforce are generally used together as needed.

To declare a set, use the Set keyword followed by the primitive data type name within <> characters. For example:

new Set<String>()

public class SetDemo {

public static void main()

{

Set<String>name=new Set<String>();

name.add('Jack');

name.add('Jill');

name.add('Steve');

name.add('Smith');

System.debug('Set Values are ' +name);

System.debug('Set size is ' +name.size());

name.add('Smith'); // smith is a duplicate value which Set will not allow

name.add('Jim');

System.debug('Set Values are ' +name);

System.debug('Set size is ' +name.size());

//addAll() this method added all values of state to S2

Set<String>state=new Set<String>{'Alaska','New Jersey','Washinton DC','New York'};

System.debug('state Values are ' +state);

Set<String> s2=new Set<String>(state); // s2 is a new set..

System.debug('s2 Values are ' +s2);

//addAll() we have added List(myname) to Set(name1)

List<String>myname=new List<String>{'Chembur','Chikmangalore','KGF'};

Set<String>name1=new Set<String>();

name1.add('Rajkot');

name1.add('Banaras');

name1.addAll(myname);

System.debug('The values in name ' +name1);

//Contains Return true if the set contains the specified element

Boolean result=myname.contains('Chembhur');

System.debug(result); //false

Boolean result1=name1.contains('Banaras');

System.debug(result1);// true

//Contains All(Set to Compare): Returns true if a set contains all of the elements in the

//specified set

Set<String>myString=new Set<String>{'a','b','c'};

Set<String>yourString=new Set<String>{'B'};

Set<string>s3=new Set<String>{'a','b','c'};

Boolean result2,result3;

result2=myString.containsAll(yourString);

System.debug('ContainsAll() ' +result2);// case sensitive is considered hence it is false

result3=myString.containsAll(s3);

System.debug('ContainsAll() Result3 ' +result3); // true

//retain All - it will retain all the values that are existing in List and Set

boolean check;

check=name1.retainAll(myname);

System.debug('Check if all values are retained ' +check);

System.debug('The values in name ' +name1);

// Remove All ..

System.debug('\*\*\*\*\*\* Remove All \*\*\*\*\*\*\*\*\*');

Set<Integer>values=new Set<Integer>{10,20,30,40,50};

Set<Integer>ages=new Set<Integer>{10,20,30};

values.removeAll(ages);

System.debug('Removed values ' +values);

System.debug('Values List' +values);

System.debug('Ages List' +ages);

//Isempty()

System.debug('\*\*\*\*\*\* Is Empty \*\*\*\*\*\*\*\*\*');

Set<Integer>Marks=new Set<Integer>();

Boolean b1=Marks.Isempty();

System.debug('Is b1 is empty or not ' +b1);

// It will check if Marks set element is empty...

// It will display True because Marks set element is empty

Set<Integer>SemMarks=new Set<Integer>{10,20,30,40,50};

Boolean b2=SemMarks.Isempty();

System.debug('No It is not empty ' +b2);

// It will check if SemMarks set element is empty...

// It will display false because SemMarks set element is not empty

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Map – Collection

It is a key & value pair

Key Value

101 Bhushan

102 Vivek

103 Sai

104 Sushma

105 Kiran

106 Sai

Map<Integer,String>stud=new Map<Integer,String>();

Map<Integer,Integer>Marks=new Map<Integer,Integer>();

Roll No Marks

public class MapDemo {

public static void main()

{

Map<Integer,String>empmap=new Map<Integer,String>();

//Integer is a Key.. this value cannot be a duplicate it should be unique

//String is a Value ... this can be duplicate

empmap.put(101,'Vijay'); // Put() is to add values into the Map

empmap.put(102,'MSD');

empmap.put(103,'Surya');

empmap.put(104,'Vikram');

empmap.put(105,'Vijay');

System.debug('The Employees Names are ' + empmap);

System.debug('The Employees Names are ' + empmap.get(103));

// get() will get the specific Key value....

Map<String,String>chemicals =new Map<String,String>{'1000'=>'HCL','1002'=>'H2SO4','1003'=>'NH3'};

string value=chemicals.get('1002');

System.debug('Value of the specified Key using get function ' +value);

//to retreive the Keys

Set<String>setofkeys=new Set<String>(); // empty Set Collection

System.debug('Set values ' +setofkeys); // no values in set

setofkeys=chemicals.keySet(); // keyset() // chemical is a amp it will get the keys

System.debug('Values of set with Keys ' +setofkeys);

//containskey() it will check if the respective key is available or not

System.debug('Contains 1002 ? ' + chemicals.containskey('1002')); // display True

System.debug('Contains 1006 ? ' + chemicals.containskey('1006')); // display false

// map.size()

}

}

///////////////////////////////////////////////////////////////////////////////////////////////

A major difference between a Map and a List is traversal. In a List you will have to loop through each element to find out a specific element, whereas in a map you could fetch it directly using the key. This greatly reduces processing time.

**Map<Id,Account> testmap = new Map<Id,Account>([Select Id,Name from Account limit 5]);**

**system.debug('The size of the map is'+testmap.size());**

**system.debug('The keys are'+testmap.keySet());**

**system.debug('The values are'+testmap.values());**

///////////////////////////////////////////////////////////////////////////////

**To access elements in a map, use the Map methods provided by Apex. This example creates a map of integer keys and string values. It adds two entries, checks for the existence of the first key, retrieves the value for the second entry, and finally gets the set of all keys.**

Map<Integer, String> m = new Map<Integer, String>(); // Define a new map

m.put(1, 'First entry'); // Insert a new key-value pair in the map

m.put(2, 'Second entry');

System.debug(m);// Insert a new key-value pair in the map

System.assert(m.containsKey(1)); // Assert that the map contains a key

System.debug(m.containsKey(1));

String value = m.get(2); // Retrieve a value, given a particular key

System.assertEquals('Second entry', value); //assertEquals() is used to validate two values are equal.

Set<Integer> s = m.keySet(); // Return a set that contains all of the keys in the map

System.debug('Set Data ' +s);

When writing test classes in apex, System.assert() can be used to assert that a specific condition is true. If it is not , an error is returned that causes code execution to stop. System.assert() is for proving our code behaves as expected.

***assertEquals()*** is used to validate two values are equal. Basically it is used in test method. This method asserts that the first two arguments, x and y are the same, if they are not a runtime exception is thrown. System. AssertEquals accepts three parameters, the first two are mandatory. The first two parameters are compared to each other to check if they are equal. While the third parameter is optional, it is the message that will be displayed if the assertEquals results in false.

SOQL - Salesforce Object Query Language

It is proprietary language from Salesforce to interact with the database

\*\*\*\*\*\*\*\* 1 or 2 Certification Questions \*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SOQL in Salesforce.**

* [SOQL](https://www.tutorialkart.com/learn-apex-soql/)stands for "Salesforce Object Query Language”.
* It returns Records.
* Records are stores in collection. Records are pertaining to single sObject.
* SOQL is similar to the SELECT statement in the widely used Structured Query Language (SQL) but is designed specifically for Salesforce data.
* Search in single sObject and it retains records.
* SOQL retrieves the data from the database using “**SELECT**” keyword.
* It works on multiple objects at the same time.
* ***SOQL is not used in Triggers and can be used only in Apex classes and anonymous block.***
* SOQL against the same field will be slow.

**The following are SOQL keywords :**

AND, ASC, DESC, EXCLUDES, FIRST, FROM, GROUP, HAVING, IN, INCLUDES, LAST, LIKE, LIMIT, NOT, NULL, NULLS, OR, SELECT, WHERE, WITH

Relationship Queries

SOQL in Anonymous

Apex Classes with SOQL and SOSL

sObject

DML

Exceptions

SELECT Id,Name from Account

SELECT Name,Phone, AnnualRevenue from Account

SELECT LastName,FirstName,email,Birthdate from Contact

SELECT LastName,FirstName,email from contact where FIRSTNAME IN('Rose','Sean')

SELECT LastName,FirstName,email from contact where FIRSTNAME NOT IN('Rose','Sean')

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c where Department\_\_c='Engineering'

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c where Department\_\_c='Engineering' OR Department\_\_c='Information Technology'

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c where

Max\_Pay\_\_c >=9000 AND Min\_Pay\_\_c >=4000

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c where Department\_\_c IN('Engineering','Information Technology')

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c where Domain\_Knowledge\_\_c INCLUDES('Sales','Banking') *– Multipicklist*

SELECT Name, Phone,Email from Contact where Phone!=null

SELECT Name, Phone,Email from Contact where Phone!=null and Email !=null

SELECT Name, Phone,Email from Contact where (Phone!=null and Email !=null) or Email=null

Select Name, AccountId, StageName from Opportunity

Select Name,Opportunity.Account.Name,StageName from Opportunity

**Like Operator**

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like 'u%'

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like 'united%'

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like 'united oil I%'

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like 'E\_g%'

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like 'E%g%'

Select Name,Opportunity.Account.Name,StageName from Opportunity where Name like '%u%

% is a wild card that matches zero or more characters.

**Order By (Sorting)**

Select Name,Opportunity.Account.Name, StageName from Opportunity Order by Name

By default Asc..

Select Name,Opportunity.Account.Name, StageName from Opportunity Order by Name Desc

Select Name, Amount, StageName from Opportunity Order by Amount Desc NULLS FIRST

Select Name, Amount, StageName from Opportunity Order by Name Desc NULLS LAST

Select Name, Amount, StageName from Opportunity Order by Amount Desc NULLS LAST

SELECT Name, Phone,Email from Contact where Phone!=null ORDER BY NAME ASC

SELECT Name, Phone,Email from Contact where Phone=null ORDER BY NAME ASC

**LIMIT**

Select Name, Amount, StageName from Opportunity Order by Name ASC LIMIT 5

Select Name, Amount, StageName from Opportunity Order by Name DESC LIMIT 25

**OFFSET**

OFFSET means skipping of the records from 1 on..

Offset 10 means skipping from 10th Record

Select Name, Amount, StageName from Opportunity ORDER BY NAME LIMIT 10 OFFSET 2

Select Name, Amount, StageName from Opportunity ORDER BY NAME LIMIT 10 OFFSET 5

SELECT Name, Department\_\_c, Min\_Pay\_\_c, Max\_Pay\_\_c,Pay\_Grade\_\_c from Position\_\_c ORDER BY NAME LIMIT 10

OFFSET 5

Select Name, Amount, StageName from Opportunity where StageName='Closed Won' order by name OFFSET 10

Select Name, Amount, StageName from Opportunity where StageName='Closed Won' order by name LIMIT 10 OFFSET 4

**Fields ()**

SELECT FIELDS(ALL) FROM ACCOUNT LIMIT 10

SELECT FIELDS(CUSTOM) FROM ACCOUNT LIMIT 10

SELECT FIELDS(STANDARD) FROM ACCOUNT LIMIT 10

SELECT NAME, ID, FIELDS(CUSTOM) FROM ACCOUNT LIMIT 10

**Aggregate Functions**

**SOQL Group By Clause.**

In SOQL Group functions are used to group set of records by the values specified in the filed. This function will gather all the data in the specified fields and also allow to perform aggregated functions.

List of Aggregated functions in Group Function.

COUNT ( )

COUNT ( FIELD\_NAME )

COUNT\_DISTINCT ( )

SUM ( )

MIN ( )

MAX ( )

SELECT Department\_\_c, sUM(Min\_Pay\_\_c), sum(Max\_Pay\_\_c) from Position\_\_c group by Department\_\_c

SELECT Pay\_Grade\_\_c, sUM(Min\_Pay\_\_c), sum(Max\_Pay\_\_c) from Position\_\_c group by Pay\_Grade\_\_c

SELECT Department\_\_c, sUM(Min\_Pay\_\_c), sum(Max\_Pay\_\_c) from Position\_\_c group by Department\_\_c order by DEpartment\_\_c desc

SELECT Department\_\_c, sUM(Min\_Pay\_\_c), sum(Max\_Pay\_\_c) from Position\_\_c group by Department\_\_c order by DEpartment\_\_c desc LIMIT 3

SELECT Department\_\_c, sUM(Min\_Pay\_\_c), sum(Max\_Pay\_\_c) from Position\_\_c group by Department\_\_c order by DEpartment\_\_c desc LIMIT 3 oFFSET 2

sELECT LeadSource, Count(Name) from Lead Group By LeadSource

SELECT Department\_\_c, Count(Name) from Position\_\_c group by Department\_\_c order by DEpartment\_\_c ASC

Select Name, Max(BudgetedCost) from Campaign Group By Name

Select Name, Max(Amount), Min(Amount), Sum(Amount) from Opportunity Group By Name

Select StageName, Max(Amount), Min(Amount), Sum(Amount) from Opportunity Group By StageName

Select StageName, Max(Amount), Min(Amount), Sum(Amount) from Opportunity Group By StageName HAVING SUM(AMOUNT)>387654

Group By Roll-up clause returns the query with subtotals for aggregated data in query results

SELECT LeadSource, Count(Name) cnt from Lead Group By RollUp(LeadSource)

Select StageName, Max(Amount), Min(Amount), Sum(Amount) from Opportunity Group By RollUp(StageName)

Select StageName,Sum(Amount),Count(Name) from Opportunity Group By StageName Order by StageNAme

Select StageName,Sum(Amount),Count(Name) from Opportunity Group By StageName Having Count(Name)>=3 Order by StageName Asc

Select StageName,Sum(Amount),Count(Name) from Opportunity Group By StageName Having StageName LIKE 'P\_%'

Select Name,count(StageName) from Opportunity Group By Name Having Name LIKE 'U\_%'

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**SOSL**

**Salesforce Object Search Language**

* SOSL stands for “Salesforce Object Search language”.
* It returns fields.
* Records are pertaining to different sObjects.
* Search in entire Org and returns fields.
* SOSL retrieves the data from the database using the keyword “**FIND**“.
* It doesn’t work on multiple objects at the same time and need to write different SOQL queries for different objects.
* Searching data in your objects
* Search the data and retrieve the data..

**Scenario 119 – SOSL Queries to be Executed**

FIND {Indu} – anywhere in the System and Returning ID of the records where Indu name is found

Find out in the name field of a Opportunity

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName)

FIND {Indu} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName)

FIND {Indu} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName), contact(Name)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName),Contact(Name,FirstName),

Account(Name,Active\_\_c)

FIND {Indu} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName), contact(Name), User(UserName)

**Filtering / Where Clause**

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName where Amount>=354532)

FIND {United\*} In Name Fields Returning Account(id,Name,Active\_\_c)

FIND {United\*} In Name Fields Returning Account(id,Name),Opportunity(Name,Amount)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName order by StageName desc)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName where Amount>=354532 order by stageName desc)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName order by StageName desc LIMIT 6 Offset 3)

FIND {United Oil Gas} IN NAME Fields RETURNING Opportunity(Name,Amount,StageName LIMIT 6 OFFSET 3)

**\*\*\* toLabel() method to filter the records using a translated picklist values**

FIND {United Oil Gas} IN Name Fields RETURNING Opportunity(Name,Amount, toLabel(StageName) where toLabel(StageName)='Closed Won')

FIND {United Oil Gas Corp} IN NAME fields RETURNING Opportunity(Name,Amount,toLabel(STAGENAME) where toLabel(StageName)='Closed Won' AND Amount >=235000)

1. FIND {United Oil Gas Corp} IN NAME fields RETURNING Opportunity(Name,Amount,toLabel(STAGENAME) where toLabel(StageName)='Closed Won' OR Amount>=235678)

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Apex code in Annonymous window

Account[] accts=[SELECT Name,Phone,NumberOfEmployees FROM Account

WHERE (Industry = 'Energy' AND NumberOfEmployees>5)

ORDER BY Name

LIMIT 10];

System.debug(accts.size() + ' account(s) returned.'); // size of accts array

System.debug(accts);

for(Account a:accts) // Iterate the account details.

{

System.debug(a);

}

//////////////////////////////////////////////////////////////////////

public class FetchAccounts {

public static void main()

{

Account[] accts=[SELECT Name,Phone,NumberOfEmployees FROM Account

WHERE (Industry = 'Energy' AND NumberOfEmployees>5)

ORDER BY Name

LIMIT 10];

System.debug(accts.size() + ' account(s) returned.'); // size of accts array

System.debug(accts);

for(Account a:accts) // Iterate the account details.

{

System.debug(a);

}

}

}

//////////////////////////////////////////////////////////////////////

list<Account>listOfAccounts = [select id, name from account limit 5];

// execute the logic if the size of the account list is greater than zero

if(listOfAccounts.size()>0)

{

System.debug('The Acount Name is ' +listOfAccounts);

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

List<Position\_\_c> plist=[Select Name,Department\_\_c, Status\_\_c from Position\_\_c];

for(Position\_\_c p:plist)

{

System.debug('Position is ' + p.Name + ' Department ' + p.Department\_\_c + ' And Status ' +p.Status\_\_c);

}

///////////////////////////////////////////////////////////////////////////////////////////////////////

List<Position\_\_c>pos=[SELECT Name,Department\_\_c,Min\_Pay\_\_c,Max\_Pay\_\_c,Status\_\_c

from Position\_\_c where Status\_\_c='Open' order by Name asc];

for(Position\_\_c p:pos)

{

System.debug(' Name of Position ' + p.Name + ' Department ' + p.Department\_\_c +

' Min Pay ' + p.Min\_Pay\_\_c + ' and Max Pay ' + p.Max\_Pay\_\_c +

' Status of the position is ' + p.Status\_\_c);

}

////////////////////////////////////////////////////////////////////////////////////

Write an Apex Program to fetch Account records with Name, Annual Amount, Active\_\_c ..

///////////////////////////////////////////////////////////////////////////////////////////

**//variable binding**

public class PosVariableBind {

public static void main()

{

String pstat='Open'; // variable

List<Position\_\_c>pos=[SELECT Name,Department\_\_c,Min\_Pay\_\_c,Max\_Pay\_\_c,Status\_\_c

from Position\_\_c where Status\_\_c=:pstat order by Name asc]; // variable binding

for(Position\_\_c p:pos)

{

System.debug(' Name of Position ' + p.Name + ' Department ' + p.Department\_\_c +

' Min Pay ' + p.Min\_Pay\_\_c + ' and Max Pay ' + p.Max\_Pay\_\_c +

' Status of the position is ' + p.Status\_\_c);

}

}

}

/\*PosVariableBind.main();\*/

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Example for Value Binding:**

**SOQL queries support Apex variable binding. You can use an Apex variable and filter SOQL records against the value of that variable.**

String strName = ‘John Snow’;

List<Position\_\_c> positionList = [SELECT Name

FROM Position\_\_c WHERE Name =: strName];

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Fetch the no of records in an object**

Integer pcount=[SELECT count() from Position\_\_c];

System.debug('The no of positions are ' + pcount);

**To retrieve one record:**

Position\_\_c p=[select name,department\_\_c,status\_\_c from Position\_\_c where id='a066F0000148m1A'];

System.debug('Position Record '+p);

**SOSL Search**

Write an Apex program to retrieve and search Name & Industry fields from **Account** and Name and Department Fields from **Position\_\_c** whose Names must be started with **‘sa’;**

public class SearchDemo {

public static void main()

{

List<List<sObject>> dlist=[FIND 'sa' IN Name fields Returning Account(Name,Industry),

Position\_\_c(Name,Department\_\_c)];

List<Account>alist=dlist.get(0);

List<Position\_\_c>plist=dlist.get(1);

for(Account a:alist)

{

System.debug(' Account Name ' + a.Name + ' Industry ' + a.Industry);

}

for(Position\_\_c p:plist)

{

System.debug(' Position Name ' + p.Name + ' Department ' + p.Department\_\_c);

}

}

}

////////////////////////////////////////////////////////////////////////////////

**Relationship Query in Apex**

One to Many Relationship

Lookup

Master Detail

Account Contact

Dhoni Industries MSDhoni

MSK

Indu

Account Contacts

Saiket Solutions Saiket

Sourab

Ritik

Opportunities

1. Order – Cricket Bats - Saiket
2. Order – SmartPhones – Sourab
3. Order - Glouces and Pads - Ritik

Lookup or Master Details

Want to fetch the list of Child Records related to the Parent record in SOQL

I can achieve this with Relationship Queries

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* know the Name of Relationship between the two objects.. and you can get this relationship name by going to lookup or MD field on child object…

* **Child-to-Parent relationship**
* **Parent to Child Relation Ship**

**Child-to-Parent relationship**

**Child to Parent:**

**For Standard Objects:**

SELECT relationship\_field.parent\_obj\_field FROM child\_object;

List<Opportunity> oppList = [SELECT name, amount, closeDate, account.name

FROM Opportunity];

for(Opportunity o: oppList)

{

System.debug(‘Opp: ‘ + o.name + ‘ Acc: ‘+o.account.name);

}

**For Custom Objects:**

SELECT (relationship\_field)\_\_r.parent\_obj\_field FROM child\_object;

**Child to Parent Relationship Queries:**

**Scenario 111 –**

Write a query which returns the first names of all the contacts in the organization, and for each contact, the account name associated with (parent of) that contact.

**Select LastName,FirstName, Contact.Account.Name from contact**

**Scenario 112 –**

Write a query which returns the names of all the Opportunities in the organization, and for each Account, the account name associated with (parent of) that Opportunity.

**Select Opportunity.Name,Opportunity.Amount, Opportunity.Account.Name from Opportunity**

**Scenario 113 –**

Write a query which returns the case number of all the cases in the organization, and for each case, the account name associated with (parent of) that case. **HW**

**Scenario 114 –**

Write a query which returns AccountName, Contact Name and Contact ID for only the contacts whose account is related Industry=’Energy’

**SELECT ID, NAME, ACCOUNT.NAME,ACCOUNT.INDUSTRY FROM CONTACT WHERE ACCOUNT.INDUSTRY='ENERGY'**

**Scenario 115 –**

1. Contact c=[Select FirstName, LastName, Account.Name, Account.Industry from contact where id ='0032v00002oCXKY'];

System.debug('Account Name: ' + c.Account.Name);

System.debug('Industry: ' + c.Account.Industry);

**Parent-to-Child relationships**

The parent object has a name for the child relationship that is unique to the parent, the plural of

the child object name. For example, Account has child relationships to Assets, Cases, and Contacts among other objects, and has a relationshipName for each, Assets, Cases, and Contacts. These relationships can be traversed only in the SELECT clause, using a nested SOQL query.

***The Child Relationship Name for the lookup field on Contact is "Contacts" Therefore, you would use "Contacts" in Account queries to refer to Contact records whose Account field references a given Account.***

**Salesforce Object Query Language (SOQL) Understanding Relationship Names**

Write a query which will return all accounts, and for each account, fetch the first and last name of each contact associated with (the child of) that account.

**Scenario 115 –**

SELECT ACCOUNT.NAME,(SELECT FIRSTNAME,LASTNAME FROM Account.Contacts)FROM ACCOUNT

SELECT ACCOUNT.NAME,(SELECT Contact.FIRSTNAME,Contact.LASTNAME FROM Account.Contacts)FROM ACCOUNT

Select Account.Name,(SELECT LastName,email from Contacts)from Account

**Scenario 116 –** Write a query which will return all accounts, and for each account, fetch the Opportunity Name of each Opportunity associated with (the child of) that account. **HW**

1. **Select Account.Name, (select Opportunity.Name from Account.Opportunities) from Account**

**Scenario 117 – Execute the output for Scenario 115 in Apex Programme**

List<Account>accList=[Select Name,(Select Name,Email from Account.Contacts)from Account];

System.debug(accList);

for(Account acc:accList)

{

System.debug('Account Name' +acc.Name);

System.debug('Email ID ' +acc.Contacts);

}

**Scenario 118-** Write a Query which will return all Positions and for each Positon get the Job Applications of each Job Application associated with the child of Position\_\_c

|  |  |
| --- | --- |
| **Child Relationship Name** | **Job\_Apllications**  **Job\_Applications\_\_r** |

List<Position\_\_c>Plist=[Select Name,Department\_\_c,(SELECT NAME,ID from Job\_Applications\_\_r) from Position\_\_c];

for(Position\_\_c p:Plist)

{

System.debug(' Name ' + p.Name + 'Department' + p.Department\_\_c);

}

First one is for Account(Parent) – Contact (Child) relationship.

1. **List<Account> accList = [select id,name,(select name, id, email from contacts) from account];**

**for(Account a:accList)**

**{**

**System.debug('ID' +a.id+ 'Name' +a.name);**

**}**

1. ***Write query a Parent to Child Relationship from***
2. ***Candidate to Job Applications***
3. ***Positions to Interviews***
4. ***Job Application to Reviews - And try Aggregate functions in the Query..***

/////////////////////////////////////////////////////////////////

**Parent to Child (Apex Code)**

**public class RelationQueries {**

    public static void main(){  
        list <account> lst = [select id, name,(select firstname, lastname, email from contacts) from account  
       ];  
  
        for(account acc:lst)  
        {  
  
            system.debug(acc);       
  
        system.debug('\*\*\*\*\*\*\*Contact Names\*\*\*\*\*\*\*\*\*\*\*\*');  
      for(contact acc1 : acc.contacts)

{  
            system.debug(acc1);  
        }  
        }  
    }  
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////

**public class RelationQueries {**

    public static void main(string a)

{  
        list <account> lst = [select id, name,(select firstname, lastname, email from contacts) from account where name=:a ];  
  
        for(account acc:lst)  
        {  
  
            system.debug(acc);       
  
        system.debug('\*\*\*\*\*\*\*Contact Names\*\*\*\*\*\*\*\*\*\*\*\*');  
      for(contact acc1 : acc.contacts)

{  
            system.debug(acc1);  
        }  
        }  
    }  
}

RelationQueries.main('ashwin');

///////////////////////////////////////////////////////////////////////////////////////////////////////////////

**With ifelse**

public class RelationQueries {  
    public static void main()  
    {  
        list<account> li=[select name,(select id,firstname,lastname,email from account.contacts) from account];  
  
        for(Account a:li){  
            System.debug('\*\*\*\*\*\*\*\*\*\*\*Account with its contacts\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');  
            System.debug(a);  
  
            if(a.contacts.size()>0)  
            {  
                for(contact c:a.contacts)

{  
                    system.debug(c);  
                }  
            }  
            else{  
                System.debug('It does not have any contacts.');  
            }  
        }  
    }  
}

////////////////////////////////////////////////

**//Parent to Child Relation ship Query with Apex class**

public class ParentRelation {

public static void main()

{

List<account>lst=[Select Id,Name,(Select FirstName,LastName, email from Contacts)from Account where Name like 'E\_g%'];

for(Account acc:lst)

{

System.debug(acc);

System.debug('\*\*\*\*\*\*\*\*\*\*\*\* Contact NAmes\*\*\*\*\*\*\*\*\*\*\*');

for(Contact acc1:acc.Contacts)

{

System.debug(acc1);

}

}

}

}

**////////////////////////////////////////////////////////////**

**Points to be Remembered**

**Child to Parent**

\*\*\*\*\* Dot Notation

\*\*\*\*\* No Sub Query

**Parent to Child**

**Standard Objects**

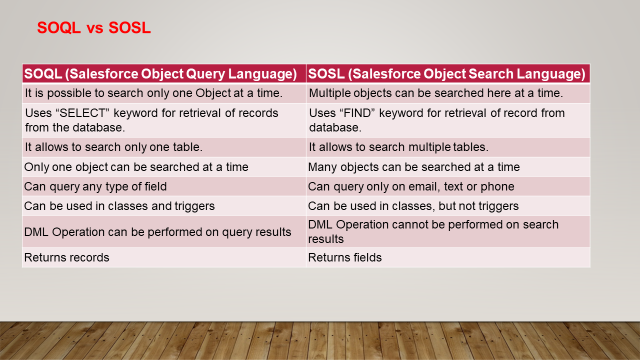
\*\*\*\*\* Sub Query

\*\*\*\*\* Plurals (Contacts, Assests, Cases, Products, etc)

**Custom Objects**

\*\*\*\*\* Sub Query

\*\*\*\*\* \_\_r (Suffix ) for Child Objects



Every Salesforce record is natively represented as an sObject in Apex. Before it is placed into Salesforce, every Salesforce record is a representation of an sObject.

**sObjects (Salesforce Object Type)**

Every Salesforce record is natively represented as an sObject in Apex. Before it is placed into Salesforce, every Salesforce record is a representation of an sObject.

Apex is completely connected with the database, unlike other programming languages like Python or C++. Therefore, in order to view the records or add new entries, we do not need to build any database connections. You can access Salesforce records and their fields directly from Apex

Sobjects are standard or custom objects that stores record data in the force.com database. There is also SObject datatype in apex that is the programmatic representation of these SObjects.

Developers referes to SObject and their fields by their API names.

Account acc=new Account(); acc is an sObject Standard

Position\_\_c pos=new Position\_\_c(); Custom sObject

sObject is nothing but a single record in your database or Object…

single record will have properties/fields/attributes…

Account acc=new Account();

Acc.Name

Datatype -----🡪 sObject ---------🡪 Account----🡪 acc is avariable

Account acc=new Account();

You can add fields to your subject –acct variable …. In two ways

***Through the constructor***

Student­­\_\_c st = new Student\_\_c( Name = ‘Rus’ age = ‘20’ email = ‘Rush@abc.com’);

or

***By using dot notation.***

Account acc=new Account();

acc.Name='Chennai Railways';

acc.AnnualRevenue=93939393;

acc.BillingCity='Hyderabad';

System.debug(acc);

**DML**

How many ways did we perform DML operations..

Below all are declarative or point click and approach

Workbench

DataLoader

Data Import Wizard

Data Loader.io

When it comes Programatically…

DML statements

Database Class

DML Statements

1. insert
2. update
3. upsert
4. delete
5. undelete
6. Merge

/\*Account acct = new Account(

Name='SFDC Computing',

Phone='(415)555-1212',

NumberOfEmployees=50,

BillingCity='San Francisco');

insert acct;

// acct record is created with Id

id getId=acct.Id;

// id is an datatype getId is a variable/type of Id

System.debug('The new record id is creaated ' + getId);\*/

Contact con = new Contact(

FirstName='Carol',

LastName='Ruiz',

Phone='(415)555-1212',

Department='Wingo',

AccountId='001IR00001rKi2yYAC');

insert con;

Delete DML Statement

Account[] Accts = [SELECT Id, Name FROM Account

WHERE Name like 'Testwood%'];

try {

delete Accts;

} catch (DmlException e) {

// Process exception here

}

**Updated with List and Insert after for loop**

List<Account> accountList = new List<Account>();

Account acc;

for (Integer i = 0; i < 150; i++)

{

acc = new Account();

acc.Name = ‘Welcome India ' + i;

accountList.add(acc);

}

insert accountList;

////////////////////////////////////

Update Statement

**// DML Update Statement Bulk Records**

Write an Apex prog where Position Education Qualifications need to be updated to ‘Btech’ for atleast five records with Array

public class UpdatePosition {

public static void updatePositons()

{

Position\_\_c [] plist=[Select Name, Job\_Description\_\_c from Position\_\_c where Department\_\_c='Engineering' LIMIT 5];

//Loop the records

for(Position\_\_c pos:plist)

{

pos.Education\_Qualifications\_\_c='BTech';

}

update plist; // Update the records

System.debug('Records Updated');

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////

**// DML Update Statement Bulk Records with Department\_del\_\_c is NULL with Array**

public class BulkUpdateDemo {

public static void updatePositons()

{

Position\_\_c [] plist=[Select Name, Job\_Description\_del\_\_c from Position\_\_c where Department\_del\_\_c=‘ ' LIMIT 5];

//Loop the records

for(Position\_\_c pos:plist)

{

pos.Department\_del\_\_c='Engineering';

pos.Education\_del\_\_c='BTech';

}

update plist; // Update the records

System.debug('Records Updated');

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**// DML Update Statement Bulk Records – List Collection**

Write an Apex prog where Position Education Qualifications need to be updated to ‘Btech’ for atleast five records

 public class UpdatePosition {

public static void updatePosRecords()

{

List<Position\_\_c>plist=[SELECT Name,Job\_Description\_\_c, Education\_\_c from Position\_\_c where Department\_\_c ='Engineering' LIMIT 5];

for(Position\_\_c p:plist)

{

p.Education\_\_c='B.Tech';

p.Hiring\_Manager\_\_c='005GB00000aZ0R8YAK';

p.Min\_Pay\_\_c=88484;

p.Max\_Pay\_\_c=98888;

}

try{

update plist;

System.debug('Records Updated');

}

catch(DMLException e)

{

System.debug(e.getMessage());

}

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Home Work and Extra use cases**

Write an Apex Prog where Account **Description** should be updated with “**Account Created with Update DML statement”** where Account Industry=’Energy’

**Write an Apex PRog DML Update Statement Bulk Records with Department\_\_c is NULL with Array**

pos.Department\_del\_\_c='Engineering';

pos.Education\_del\_\_c='BTech';

**Inserting and Updating Records :Value Binding:**

Using DML, you can insert new records and commit them to the database. Similarly, you can update the field values of existing records.

*This example inserts three account records and updates an existing account record. First, three Account sObjects are created and added to a list. An insert statement bulk inserts the list of accounts as an argument. Then, the second account record is updated, the billing city is updated, and the update statement is called to persist the change in the database.*

Account[] accts = new List<Account>();

for(Integer i=0;i<3;i++)

{

Account a = new Account(Name=‘Indian Account' + i,

BillingCity='San Francisco');

accts.add(a);

}

Account accountToUpdate; // sObject …

try

{

insert accts;

// Update account Acme2.

accountToUpdate =

[SELECT BillingCity FROM Account

WHERE Name='Acme2' AND BillingCity='San Francisco'

LIMIT 1];

// Update the billing city.

accountToUpdate.BillingCity = 'New York';

// Make the update call.

update accountToUpdate;

}

catch(DmlException e)

{

System.debug('An unexpected error has occurred: ' + e.getMessage());

}

// Verify that the billing city was updated to New York.

Account afterUpdate =

[SELECT BillingCity FROM Account WHERE Id=:accountToUpdate.Id];

System.assertEquals('New York', afterUpdate.BillingCity);

/////////////////////////////////////////////////////////////////////////////////////

**USecase Homework:** Replicate the above program on in Apex where Position or Candidate Object to **Inserting and Updating Records by using: Value Binding:**

**Upsert**

Insert + Update

**External ID \*\*\*\*\*\*\* 2 or 3 questions … in certification…**

**It** is field in Salesforce to say that this field is primary key in external Database which can be used to identify that external record exists in Salesforce or not.

If the record exists Salesforce will have an intelligence to update the record or it will create a record.

This field is usually references an ID from another (external) system. When a field is marked as external Id, the field will be automatically indexed.

You can use External IDs to ensure you have an ID that can be referenced both inside and outside of Salesforce.

Ex: If any application (Oracle) that have data of employees (like empId, name, designation etc.) is now linking with Salesforce so when they both linked together and application's data imported to Salesforce so at that time the empId of employees in existing application is considered as an external Id for Salesforce.

Note: External Id is not case-sensitive – for example, “XYZ” will be matched with “xyz”. if the custom field has enable the “Unique” attribute and the case-sensitive option for that attribute is selected, uppercase and lowercase letters will not be considered identical.

An object can have at most 7 External IDs’ fields. The field type should be any one of auto-number, email, number, or text. Custom fields marked as unique also count against an object’s limit of 7 External IDs’ fields.

**Considerations:**

* External Id is an attribute that can be added to a custom field to indicate that it should be indexed and treated as an Id.
* It is a user-defined cross-referenced field.
* It can be created for custom field type (Text, Number or Email ONLY).
* Each object can have up to 7 external Ids (as of Spring ’15).
* External IDs are indexed; meaning queries against them will run a bit faster (yay, large data volumes)`
* External IDs are indexed and thus searchable
* Configure UPSERT action to traverse object relationships defined in Salesforce, but use External Id’s from legacy system to discover Salesforce record ID’s.

**Benefits:**

* It helps improve Report and API SOQL performance.
* It can be used with UPSERT DML operation to seamlessly integrate apps with other systems.
* No need to know Salesforce record Id’s to load data.
* Very convenient for data integrations and migrations.
* The import wizard will detect existing records in Salesforce that have the same external ID.
* It becomes searchable in the sidebar search.

Upsert uses the sObject record's primary key (the ID), an idLookup field, or an external ID field to determine whether it should create a new record or update an existing one:

* If the key is not matched, a new object record is created.
* If the key is matched once, the existing object record is updated.
* If the key is matched multiple times, an error is generated and the object record is neither inserted or updated.

Book\_\_c b1=new Book\_\_c(Ext\_Book\_ID\_\_c='B003', Name='Veronica', Author\_\_c='Anupam Kher');

Insert b1;

// b1 record has been inserted

// so the variable b1 has now an ID

// which will be used to match the records by upsert

b1.Price\_\_c=987;

// Create a new book record b2, which doesn't have in the database

Book\_\_c b2=new Book\_\_c(Ext\_Book\_ID\_\_c='B062', Name='Valamai', Author\_\_c='Ajit Kumar');

Book\_\_c b3=new Book\_\_c(Ext\_Book\_ID\_\_c='B004', Name='Leo', Author\_\_c='Vijay');

//Create a List which will hold the new Books to Upsert

List<Book\_\_c> Books =new List<Book\_\_c> {b1,b2,b3};

// Call Upsert

upsert Books;

// Result: B1 is updated and b2 is created.

**Merge**

Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc' LIMIT 1];

Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];

try

{

merge masterAcct mergeAcct;

System.debug('Records Merged Successfully');

}

catch (DmlException e)

{

System.debug(e.getMessage());

// Process exception here

}

//////////////////////////////////////////////////////////////////////////////////////////

**Deleting Records**

After you persist records in the database, you can delete those records using the delete operation. Deleted records aren’t deleted permanently from Salesforce, but they are placed in the Recycle Bin for 15 days from where they can be restored. Restoring deleted records is covered in a later section.

Example

The following example deletes all accounts that are named ‘Welcome':

Account[] Accts = [SELECT Id, Name FROM Account

WHERE Name like ‘Welcome%’];

try {

delete Accts;

} catch (DmlException e) {

// Process exception here

}

////////////////////////////////////////////////////

**Restoring Deleted Records**

After you have deleted records, the records are placed in the Recycle Bin for 15 days, after which they are permanently deleted. While the records are still in the Recycle Bin, you can restore them using the undelete operation. If you accidentally deleted some records that you want to keep, restore them from the Recycle Bin.

**Example**

The following example undeletes an account named 'Universal Containers'. The ALL ROWS keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

Account a = new Account(Name='Universal Containers');

insert(a);

insert(new Contact(LastName='Carter',AccountId=a.Id));

delete a;

Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Universal Containers' ALL ROWS];

try {

undelete savedAccts;

} catch (DmlException e) {

// Process exception here

}

////////////////////////////////////////////////////////////////////

**Update Related Records**

Try

{

Contact queriedContact = [SELECT Account.Name

FROM Contact

WHERE FirstName = 'Archana' AND LastName='Radhakrishna'

LIMIT 1];

// Update the contact's phone number

queriedContact.Phone = '(415)555-1213';

// Update the related account industry

queriedContact.Account.Industry = 'Technology';

// Make two separate calls

// 1. This call is to update the contact's phone.

update queriedContact;

// 2. This call is to update the related account's Industry field.

update queriedContact.Account;

}

catch(Exception e) {

System.debug('An unexpected error has occurred: ' + e.getMessage());

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Delete Related Records**

The delete operation supports cascading deletions. If you delete a parent object, you delete its children automatically, as long as each child record can be deleted.

For example, deleting the account you created earlier (SFDC Account) will delete its related contact too.

Execute this snippet in the Anonymous Apex window of the Developer Console.

Account[] queriedAccounts = [SELECT Id FROM Account WHERE Name='SFDC Account'];

delete queriedAccounts;

You’ll see that both the account and its related contact were deleted.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**What is a Namespace**

A namespace can be said to be the prefixes used to differentiate custom object and field names in. managed Force.com AppExchange packages..

**Database class belongs to System Namespace**

Database class is contains methods for creating and manipulating data. Database class methods is another way of working with DML statements which are more flexible than DML Statements like insert, update, etc.

Database class methods return the results of the data operation. These result objects contain useful information about the data operation for each record, such as whether the operation was successful or not, and any error information. Each type of operation returns a specific result object type

These Database methods are static and are called on the class name.

Database.insert()

Database.update()

Database.upsert()

Database.delete()

Database.undelete()

Database.merge()

\*\*\*\*\*\*\*\* Important Questions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Difference Between DML Statement (insert) & Database.insert() method**

**If we use the DML statement (insert),**

1. In bulk operation if error occurs, the execution will stop and Apex code throws an error which can be handled in try catch block.
2. Partial insert is not supported.
3. Roll back is not supported.

Example: If we are inserting 10 records in an object, Where 5 records are correct and remaining 5 records are incorrect.

In DML statement (Insert) all the 10 records will be failed, because if one record is incorrect or error means all other remaining records will not be inserted. It will throw error.

**Database Class methods (Database.insert) used,**

1. In bulk operations if error occurs the remaining records will be inserted / updated means partial DML operation will be done.
2. Partial insert is supported.
3. Roll back is supported.
4. The only record throwing an error will not be inserted/updated.

Example: If we are inserting 10 records in an object, Where 5 records are correct and remaining 5 records are incorrect.

In Database.insert 5 records will be inserted, remaining 5 records will be failed.(i.e. Partial DML Operation).

An array of ***SaveResult objects*** is returned with the insert and update database methods. Each element in the ***SaveResult array*** corresponds to the sObject array passed as the sObject[] parameter in the Database method, that is, the first element in the SaveResult array matches the first element passed in the sObject array, the second element corresponds with the second element, and so on. If only one sObject is passed in, the SaveResult array contains a single element.

A SaveResult object is generated when a new or existing Salesforce record is saved.

**getErrors()**

If an error occurred, returns an array of one or more database error objects providing the error code and description. If no error occurred, returns an empty set.

**getId()**

Returns the ID of the sObject you were trying to insert or update.

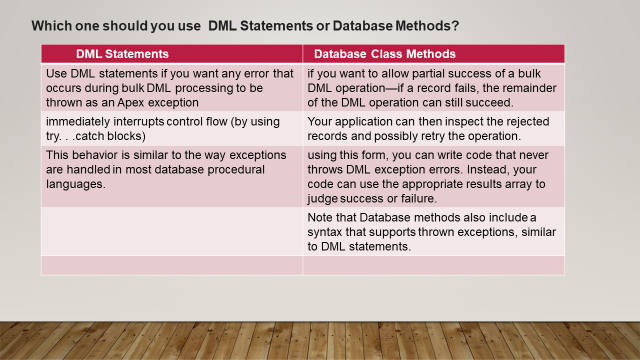
**isSuccess()**

Returns a Boolean that is set to true if the DML operation was successful for this object, false otherwise.

**getMessage()** - Will get the exception.

<https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_database.htm#apex_System_Database_delete_4>

<https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_dml_examples_merge.htm>



Exception Handling

**public class FirstExcepDemo** {

public static void main()

{

try{

integer a =0;

integer b=7/a;

System.debug('The value of b is ' +b);

}

catch(MathException me)

{

System.debug('Cannot divide by Zero');

}

}

}

///////////////////////////////////////////////////////////////////////////////

**public class SecondExcepDemo {**

public static void main()

{

try{

Integer [] Marks=new Integer[] {60,89,98,87,66}; // Array with defined size;

System.debug('The value in the marks [6] ' +Marks[6]);

}

catch(ListException le)

{

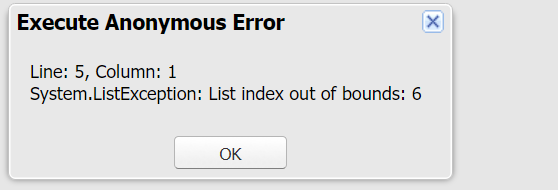
System.debug('There is no 6th Index in the above array');

System.debug(le.getMessage());

}

}

}



///////////////////////////////////////////////////////////////////////////////

**public class DMLExcepDemo** {

public static void main()

{

try{

Account acc=new Account(); // acc is an object or referense varaible

insert acc; // DML Operation - Inserting the acc record in Acconts

System.debug('Statement after inserting the values');

}

catch(DmlException dl)

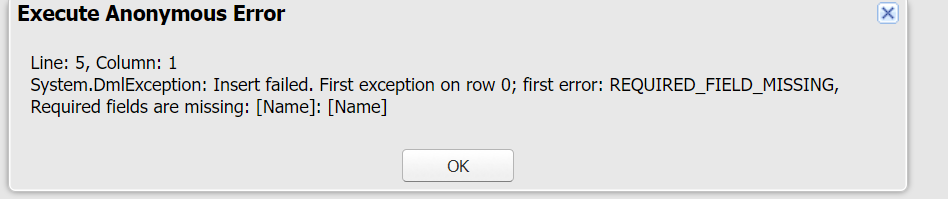
{

System.debug('The following exception has occured' +dl.getMessage());

}

}

}



///////////////////////////////////////////////////////////////////////////////

**public class ListExcepDemo**

{

public static void main()

{

List<Integer> lis=new List<Integer>();

lis.add(15); // Adding value called 15 to the list

Integer i1=lis[0];

System.debug('The value in the 0th index list ' +lis[0]);

lis.add(20);

Integer i2=lis[1];

System.debug('The value in the 1st index list ' +lis[1]);

try{

/// Exception Prone area...

//You are trying attempt to access the fourth element of List

Integer i3=lis[4];

System.debug('The value in the 1st index list ' +lis[4]);

}

catch(ListException le)

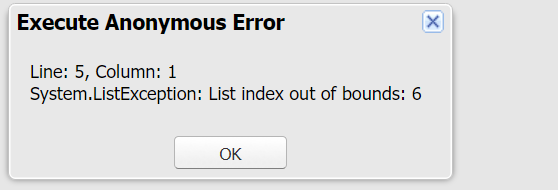
{

System.debug('The following exception has occured' +le.getMessage());

}

}

}



///////////////////////////////////////////////////////////////////////////////

// multi catch...& Null pointer exception

**public class NullExceptionDemo**

{

public static void main()

{

try{

String name; // not assgned any value to the String variable

Boolean bs=name.contains('MSDHONI');

// System.debug(bs);

}

catch(NullPointerException npe)

{

System.debug('The following exception has occured ' +npe.getMessage());

}

catch(MathException me)

{

System.debug('Cannot divide by Zero');

}

catch(ListException le)

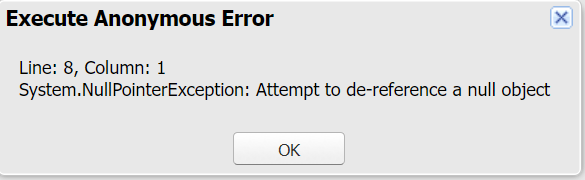
{

System.debug(le.getMessage());

}

}

}



///////////////////////////////////////////////////////////////////////////////

**DML Exception:**

try

{

Account acc=new Account();

insert acc;

System.debug('Statement after insert.');

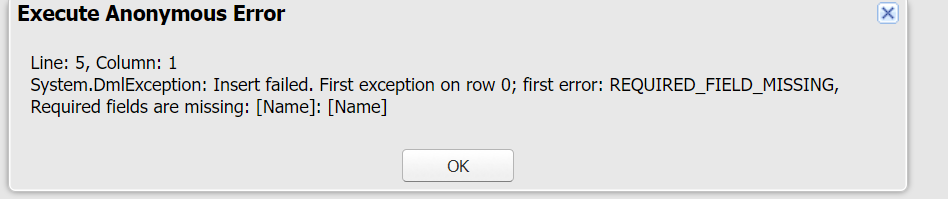
}

catch(DMLException dl)

{

System.debug('The following exception has occured ' +dl.getMessage());

}



///////////////////////////////////////////////////////////////////////////////

**List Exception**

try {

List<Integer> li = new List<Integer>();

li.add(15);

// This list contains only one element,

// but we're attempting to access the second element

// from this zero-based list.

Integer i1 = li[0];

Integer i2 = li[1]; // Causes a ListException

} catch(ListException le) {

System.debug('The following exception has occurred: ' + le.getMessage());

}

///////////////////////////////////////////////////////////////////////////////

**Null Pointer Exception**

This example creates a String variable named s but we don’t initialize it to a value, hence, it is null. Calling the contains method on our null variable causes a NullPointerException.

try {

String s;

Boolean b = s.contains('abc'); // Causes a NullPointerException

} catch(NullPointerException npe) {

System.debug('The following exception has occurred: ' + npe.getMessage());

}

///////////////////////////////////////////////////////////////////////////////

**QueryException**

Any problem with SOQL queries, such as assigning a query that returns no records or more than one record to a singleton sObject variable.

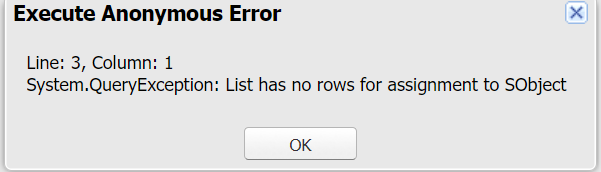
The second SOQL query in this example causes a QueryException. The example assigns a Merchandise object to what is returned from the query. Note the use of LIMIT 1 in the query. This ensures that at most one object is returned from the database so we can assign it to a single object and not a list. However, in this case, we don’t have a Merchandise named XYZ, so nothing is returned, and the attempt to assign the return value to a single object results in a QueryException. The exception is caught in our catch block and this is what you’ll see in the debug log: The following exception has occurred: List has no rows for assignment to SObject.

List<Account> lm = [SELECT Name FROM Account WHERE Name = 'XYZ'];

System.debug(lm.size());

Account m = [SELECT Name FROM Account WHERE Name = 'XYZ' LIMIT 1];

***The following exception has occurred: List has no rows for assignment to SObject.***



**After Handling Exception**

try {

// This statement doesn't cause an exception, even though

// we don't have a Account with name='XYZ'.

// The list will just be empty.

List<Account> lm = [SELECT Name FROM Account WHERE Name = 'XYZ'];

// lm.size() is 0

System.debug(lm.size());

// However, this statement causes a QueryException because

// we're assiging the return value to a Account object

// but no Account is returned.

Account m = [SELECT Name FROM Account WHERE Name = 'XYZ' LIMIT 1];

}

catch(QueryException qe)

{

System.debug('The following exception has occurred: ' + qe.getMessage());

}

///////////////////////////////////////////////////////////////////////////////

**SObjectException**

Any problem with sObject records, such as attempting to change a field in an update statement that can only be changed during insert.

This example results in an SObjectException in the try block, which is caught in the catch block. The example queries an Position\_\_c statement and selects only its Name field. It then attempts to get the Description\_\_c field on the queried sObject, which isn’t available because it isn’t in the list of fields queried in the SELECT statement. This results in an SObjectException. This exception is caught in our catch block and this is what you’ll see in the debug log: The following exception has occurred: SObject row was retrieved via SOQL without querying the requested field: Position\_\_c.Description\_\_c.

**SObjectException - Raised**

Position\_\_c pinv = new Position\_\_c(Description\_\_c='New Invoice');

insert pinv;

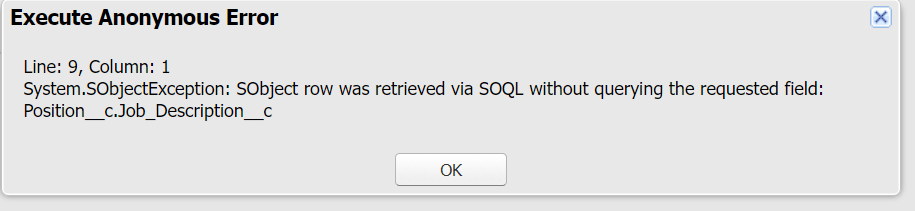
// Query the Position we just inserted

Position\_\_c v = [SELECT Name FROM Position\_\_c WHERE Id = :pinv\_\_c.Id];

// Causes an SObjectException because we didn't retrieve

// the Description\_\_c field.

String s = v.Description\_\_c;



**SObjectException - Handled**

try {

MSK\_Dev\_\_Position\_\_c pinv = new MSK\_Dev\_\_Position\_\_c(

MSK\_Dev\_\_Job\_Description\_\_c='New Invoice');

insert pinv;

// Query the invoice we just inserted

MSK\_Dev\_\_Position\_\_c v = [SELECT Name FROM MSK\_Dev\_\_Position\_\_c WHERE Id = :pinv.Id];

// Causes an SObjectException because we didn't retrieve

// the Description\_\_c field.

String s = v.MSK\_Dev\_\_Job\_Description\_\_c;

}

catch(SObjectException se)

{

System.debug('The following exception has occurred: ' + se.getMessage());

}

///////////////////////////////////////////////////////////////////////////////

try{

Position\_\_c pinv = new Position\_\_c(

Job\_Description\_\_c='Machine Learning');

insert pinv; // Record will get inserted...

// Query the Position we just inserted

Position\_\_c v = [SELECT Name FROM Position\_\_c WHERE Id = :pinv.Id];

// Causes an SObjectException because we didn't retrieve

// the Description\_\_c field.

String s = v.Job\_Description\_\_c;

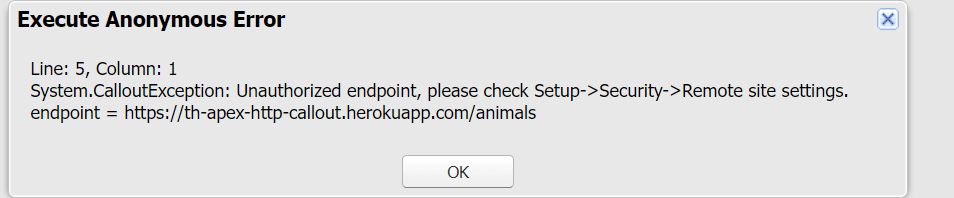
}

catch(SObjectException se)

{

System.debug('The following exception has occured' +se.getMessage());

}



**MathException**

**ListException (Array outbound except)**

**Null Pointer Exception**

**Query Exception**

**DML Exception**

**SObject Exception**

**System.CalloutException - This is thrown because unauthorized end point…**

**Common Exception Methods**

try {

MSK\_Dev\_\_Position\_\_c pinv = [SELECT Name FROM MSK\_Dev\_\_Position\_\_c LIMIT 1];

// Causes an SObjectException because we didn't retrieve

// the Description\_\_c field.

String s = pinv.MSK\_Dev\_\_Job\_Description\_\_c;

}

catch(Exception e)

{

System.debug('Exception type caught: ' + e.getTypeName());

System.debug('Message: ' + e.getMessage());

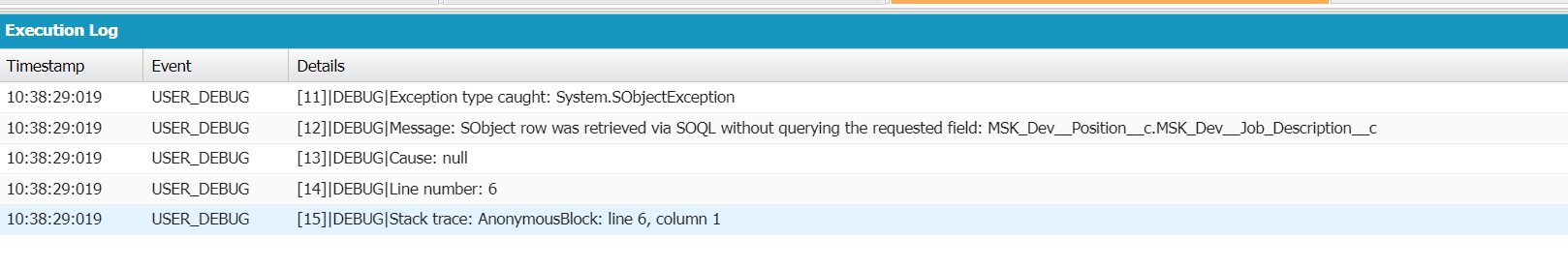
System.debug('Cause: ' + e.getCause()); // returns null

System.debug('Line number: ' + e.getLineNumber());

System.debug('Stack trace: ' + e.getStackTraceString());

}

***Out Put***

******

Account acc=[SELECT Id,Name from Account];

System.debug(acc);

**Apex Triggers**

It is a block of code..

Which will execute on table / or object… whenever a DML statement is performed

Block of code may executes before or After any DML statement

Triggers are divide into two types

**Apex Triggers can be classified into two types:**

**1. Before triggers can be used to update or validate record values before they are saved to the database.**

**2. After triggers: are used to access field values that are set by the database, and to affect changes in other records.**

Before /After Trigger

Event – before insert, before update, before delete

Event – after insert, after update, after delete, after undelete

**Events in triggers:**

* **Before Insert,**
* **Before Update,**
* **Before Delete,**
* **After Insert,**
* **After Update,**
* **After delete,**
* **After Undelete**

**Choose Context variables**

After choosing the event make a choice of **Context Variables:**

**Trigger.new**

– Returns or Hold a list of the new versions of the sObject records.

Note that this sObject list is only available in insert and update triggers, and the records can only be modified in before triggers.

**Trigger.old**

– Returns a list of the old versions of the sObject records. Note that this sObject list is only available in update and delete triggers.

1. Delete supports only Trigger.old
2. Insert supports only Trigger.new
3. Update supports both Trigger.old & Trigger.new

**newMap**

* A map of IDs to the new versions of the sObject records. Note that this map is only available in before update, after insert, and after update triggers.

**oldMap**

* A map of IDs to the old versions of the sObject records.Note that this map is only available in update and delete triggers.

**size** The total number of records in a trigger invocation, both old and new.

**Syntax**

Trigger <Name of the Tigger> on object / standard or custom (before / after)

Trigger updateAccount on Account(before insert, before update)

{

}

Whenever a new contact record is created without Account an error message should throw.

Whenever a new contact record is created or updated without an Account Name and Email Id as Null. It should throw an Error - HW

Whenever case is created with origin as email then set status as new and priority as Normal

Whenever lead is created with lead source as web then give rating as cold otherwise hot. It will be before insert

Whenever an existing opportunity stage name is modified to closed won then set closedate as today and type as customer.

**Scenario 11: Create a trigger to set rating field as HOT whenever an account is created or updated with Industry as Banking or Health care**

**Scenario 1:** Populate contact description ‘Contact record created successfully’ when user creates contact.

trigger ContactBeforeInsert on Contact (**before insert**)

{

// Trigger.New hold new version of Contacts

for(Contact contact: Trigger.new)

{

contact.Description = ‘Contact created successfully by using ContactBeforeInsert trigger’;

}

// No Need to write DML statement, trigger. New will be take care.

}

**Scenario 2:** Create a Trigger when a new Account record is created automatically ownership field should be updated with ‘Private’ when Industry field = ‘Banking’

**trigger IT1 on Account (before insert)**

**{**

**for(account a:trigger.new)**

**{ if(a.industry=='Banking')**

**{**

**a.ownership='Private';**

**}**

**}**

**}**

**Scenario 3 : Populate contact description with Modified User Name when user updates a contact.**

trigger ContactBeforeUpdate on Contact (**before update**)

{

// Trigger.New hold new version of Contacts

for(Contact contact: Trigger.new)

{

contact.Description = ‘Contact updated successfully by ‘+ userInfo.getUserName() ;

}

// No Need to write DML statement, trigger. New will be take care.

}

**Scenario 4 : How to write a trigger to inject the above 1 & 3 scenarios in one trigger**

trigger ContactBeforeInsertUpdate on Contact (**before insert, before update**) {

// Trigger.New hold new version of Contacts

for(Contact contact: Trigger.new)

{

if(trigger.isInsert)

{

contact.Description = ‘Contact created successfully by using ContactBeforeInsert trigger’;

}

If(trigger.isUpdate)

{

contact.Description = ‘Contact updated successfully by ‘+ userInfo.getUserName() ;

}

}

// No Need to write DML statement, trigger. New will be take care.

}

**Scenario 5 : Whenever a new contact record is created without Account an error message should throw.**

trigger IT\_contact on Contact (**before insert**)

{

for(contact c:trigger.new)

{

if(c.AccountId==null)

{

c.addError('Enter account name');

}

}

}

**Scenario 6 :** Whenever a new contact record is created or updated without an Account Name and Email Id as Null. It should throw an Error - HW

Create a trigger If Account==Null and Email fields ==Null through an error to “Enter Some value”

trigger IT\_Contact on Contact (**before insert, before update**) {

{

for (contact c:trigger.new)

{

if(c.Account==null && c.Email==null)

{ c.addError('Please Enter Some Value');

}

}

**Scenario 7 :** **Create a Trigger to Avoid using Candidate First Name as ‘M’**

trigger AvoidCandidatesWithStartingWithM on Candidate\_\_c (**before insert, before update**)

{

List<Candidate\_\_c> myCandidateList = Trigger.new;

for(Candidate\_\_C can:myCandidateList)

{

if(can.First\_Name\_\_c.startsWith('M'))

{

can.adderror('Cannot Create the Candidate as its name starts with M:- '+can.First\_Name\_\_c);

}

}

}

**Scenario 8 : Whenever case is created with origin as email then set status as new and priority as Normal ..**

**\*\*\*\* can implement isInsert and Isupdate context variables in this example..**

Before insert has Trigger.new

Trigger.New Returns a list of the new versions of the sObject records

Trigger emailCase on Case(**before insert**){

for(case c:trigger.new)

{

if(c.origin =='Email')

{

c.status ='New';

c.priority ='Normal';

}

}

}

**Scenario 9: Whenever lead is created with lead source as web then give rating as cold otherwise hot. It will be before insert**

Before insert has trigger.new.The trigger. new provides the records that are about to be inserted, or updated.

Trigger leadsource on lead(**before insert**){

for(lead l:trigger.new)

{

if(l.leadsource =='Web')

{

l.Rating ='Hot';

}

else {

l.Rating='Cold';

}

}

}

**Scenario 10: Whenever an existing opportunity stage name is modified to closed won then set closedate as today and type as customer.**

As it is on same object so we will use before update

Before update has trigger.new, trigger.old, trigger.newmap, trigger.oldmap

To get curent date use system.today();

Trigger stageNameOpp on Opportunity(before update)

{

for(Opportunity opp:trigger.new){

if(opp.Stagename == 'Closed Won')

{

opp.CloseDate= System.today();

opp.Type ='Customer';

}

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////

**Usage of IsBefore –(isInsert & isUpdate)**

**Scenario 11: Create a trigger to set rating field as HOT whenever an account is created or updated with Industry as Banking or Health care**

trigger AccountUp on Account (before insert,before update)

{

if((Trigger.isBefore) && ((Trigger.isInsert) || (Trigger.isUpdate)))

{

for(Account acc : Trigger.New)

{

if(acc.Industry=='Banking' || acc.Industry=='Healthcare')

{

acc.Rating='Hot';

}

}

}

}

**Scenario 12: If a new Position is created, Min Pay should not be less than 2000 (IsInsert) and if an existing record is updated Max Pay is should not be less than 7900. (Isupdate)**

trigger PositionIsinsertIsUpdate on Position\_\_c (before insert, before update)

{

for(Position\_\_c P:Trigger.New)

{

if(Trigger.isInsert && P.Min\_Pay\_\_c<2000)

P.Min\_Pay\_\_c.addError('Minimum Pay Should be greater than 2000');

else if(Trigger.isUpdate && P.Max\_Pay\_\_c<7900)

{

P.Max\_Pay\_\_c.addError('Maximum Pay Should be less than 50000');

}

}

}

**Solution: IsInsert is used before Inserting and isUpdate on the old records when updated**

**Scenario 13 Create a Trigger on Position when user enters Min Pay greater than Maxpay. It should through an error**

**trigger PositionTriggerPay on Position\_\_c (before insert, before update)**

**{**

**if(Trigger.isBefore) // Before Trigger Type operation**

**{**

**if(Trigger.isInsert || Trigger.isUpdate)**

**{**

**for(Position\_\_c p:Trigger.New)**

**{**

**if(p.Min\_Pay\_\_c>p.Max\_Pay\_\_c)**

**{**

**p.Min\_Pay\_\_c.addError('Min Pay cannot be greater than Max Pay in record');**

**}**

**}**

**}**

**}**

**}**

Or This Code

trigger PositionMinAndMaxPay on Position\_\_c (before insert, before update) {

for(Position\_\_c Pos:Trigger.New)

{

if(Trigger.IsInsert || Trigger.IsUpdate)

{

if(Pos.Min\_Pay\_\_c>Pos.Max\_Pay\_\_c)

{

Pos.Min\_Pay\_\_c.addError('Minimum Pay cannot be greater than Max Pay');

}

}

}

}

**Extra Activity 14:** Whenever a contact is created with Department=cse,EEE, Cardioloy an emaid has to be created(your email id)

trigger Deptt on Contact (before insert) {

for(contact c:trigger.new)

{

if(c.Department=='CSE')

{

c.Email='shiva@gmail.com';

}

}

}

**Extra Activity 15**

**If the user is entering same contact records with the same email id which is already existing in contact display error message**

**( Create a record with the same email id)**

trigger exists on Contact (before insert) {

list<string>st=new list<string>();

for(contact c:trigger.new)

{

list<contact>a=[select id,name,Email,lastname from contact where Email=:c.Email];

if(a.size()>0)

{

c.Email.adderror('Email already existing');

}

}

}

**After Triggers**

**Scenario 16: // Automatically create an Opp when an Account is created**

trigger AutoOpp on Account(after insert)

{

List<Opportunity> newOpps = new List<Opportunity>();

for (Account acc : Trigger.new)

{

Opportunity opp = new Opportunity();

opp.Name = acc.Name + ' Opportunity Deal';

opp.StageName = 'Prospecting';

opp.CloseDate = Date.today() + 90;

opp.AccountId = acc.Id; // Use the trigger record's ID

newOpps.add(opp);

}

insert newOpps;

}

**Scenario 17:**

*When an Account is created it should create a contact under that account with account name*

trigger AccountActions on Account (before insert, before update)

{

List<Contact> contactlist=new List<Contact>();

for (Account accobj:trigger.New)

{

Contact contobj=new Contact();

contobj.Accountid=accobj.id;

contobj.LastName=accobj.Name;

contactlist.add(contobj);

}

if(!contactlist.isEmpty())

{

insert contactlist;

}

}

**Apex Handlers.**

It is always a best practice to write a single trigger on object and process all operation in apex class (we called it as handler) instead of writing logic in trigger. In handler you can specify the order of event based on different context variables. If you write many triggers in on object,  
then you cannot control the order in which triggers gets executed.

**DML Statements / SOQL queries cannot be written in for loop. It should be written after the loops**

**As best practice, you should not write whole logic in Trigger. You should write logic in an handler or Apex Class and call that class with a method in a trigger.**

**More Logic/lines of code use handler.**

Whenever a new Account is created if Industry==null update acc Industry as banking and if Description == null update acc Description 'This record is created by Trigger Handler'

**Scenario 18**

public class AccTriggerHandler1 {

public static void beforeInsert(List<Account>accList)

{

for(Account acc:accList)

{

if(acc.Industry==NULL)

{

acc.Industry='Banking';

}

if(acc.Description==NULL)

{

acc.Description='This record is created by Trigger Handler';

}

}

}

}

trigger Handler1 on Account (before insert,before update) {

AccTriggerHandler1.beforeInsert(Trigger.New);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Scenario 19:**

*When an Account is created it should create a contact under that account with account name*

Apex Trigger:

trigger AccountActions on Account (before insert, before update)

{

List<Contact> contactlist=new List<Contact>();

for (Account accobj:trigger.New)

{

Contact contobj=new Contact();

contobj.Accountid=accobj.id;

contobj.LastName=accobj.Name;

contactlist.add(contobj);

}

if(!contactlist.isEmpty())

{

insert contactlist;

}

}

**ApexHandler.**

public class AccountActionsHandler {

public static void CreateContacts(List<Account> accountslist)

{

List<Contact> contactlist=new List<Contact>();

for (Account accobj: accountslist)

{

Contact contobj=new Contact();

contobj.Accountid=accobj.id;

contobj.LastName=accobj.Name;

contactlist.add(contobj);

}

if(!contactlist.isEmpty())

{

insert contactlist;

}

}

}

**Apex Trigger- Calling CreateContacts() from Trigger**

trigger AccounActionsHand on Account (after insert)

{  
AccountActionsHandler.CreateContacts(trigger.new);  
  
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Visualforce Pages**

HTML design the UI

<html>

<head>

<title>

<body>

Welcome to Cognizant Chennai

</body>

</title>

</head>

</html>

VF Page it is a component framework design to built custom pages..

It is a framework includes tag based markup language similar to / like HTML

MVC

Model (Database) Metadate API’s (Standard and Custom Objects)

View User Interface - Visualforce Pages / Aura / LWC

Controller Logic / Business Process Apex Classes / Controllers

Apex Triggers

VFCode- you can add- HTML and CSS and JS as optional…

Server – Salesforce database is your server it will render – refresh the page…

* + What is Visualforce?
  + Component-based user interface framework for the Force.com platform. The framework includes a tag-based markup language, similar to HTML.
  + Visualforce uses the traditional model-view-controller (MVC) paradigm.
  + A developer creates Visualforce pages by composing components, HTML and optional styling elements on the Force.com platform. Each page is then accessible by a unique URL. When someone accesses a page, the server renders the page.

Mark-up - UI

Controller - Logic

U can use or add components

<apex:Page>

<apex:CommandButton>

<apex:TextBox>

<apex:Label>

<apex:dataTable>

Tied to a controller… what is this controller (either Standard or Custom Objects)

Once you create an object a standard controller gets created..

Object Name / API StdController

Account Account Account

Position Position\_\_c Position\_\_c

Ties these components to your controller and execute the logic..

***Visuaforce is an interesting tool. “Visualforce is a markup language that allows you to describe the user interface components that live on your Force.com pages.”***

***Visualforce pages consist of two elements: Visualforce markup, and a controller. That is, you use markup language to define the components that will live on your page and tie it to a controller (either standard or custom) to execute the logic behind those components. Visualforce lives on the server. So any code you write will be generated and run on the server.***

**Scenario 1**

<apex:Page stdController=”Account” Title=”AccountPage”

<apex:page><br/><br/>

<h1>

Welcome to VisualForce Pages..

We just started learning

</h1> <br/><br/>

First Name :{!$User.FirstName}<br/>

Last NAme :{!$User.LastName}<br/>

User Name :{!$User.UserName}<br/>

Today's Date : {!Today()}<br/>

Year :{!Year(Today())}<br/>

Month :{!Month(Today())}<br/><br/><br/>

<p>

Next week date will be : {!Today()+7}

</p>

<p>

Tomorrow's date will be : {!Today()+1}

</p>

<p>

The Squareroot of 49 is {!SQRT(49)}

</p>

</apex:page>

<apex:Page>

<apex:form>

<apex:PageBlock>

<apex:PageBlockSection>

Text box

List box

<apex:CommandButton>

There are four types of Controllers

Standard Controllers

Standard List Controllers

Custom Controllers - Apex Code

Extension Controllers / Custom Extension Controller - Apex code

Standard / Custom - Standard Controller… these are inbuilt by Salesforce

I will use these Standard Controllers to fetch, read,create data on these Standard / Custom objects . Account, Contact, Position\_\_c, Candidate\_\_c, Blood\_Bag\_\_c

**Scenario 2:** **Create a VFPage of StandardController of Position object and display or bring a record into the current context.**

<!—

-->

You need to pass a record ID in the url..

<https://nalandatech-dev-ed--c.visualforce.com/apex/StandardDemo?id=a005j00000AxTAoAAN>

**HW- Scenario 3:** Create a VFPage of StandardController of Account object and

display some of the fields or bring a record into the current context.

**Scenario 4:** Create a VFPage where user can input the values into an Account object. / Create a VFPage to insert the Account Records

Std Controller - Account

<apex:Input Field>

To Save or Insert the data … Buttons or Labels

Save

Cancel

Quick Save

**HW-Scenario 5:** Create a VFPage where user can input the values into an Position\_\_c object. / Create a VFPage to insert the Position\_\_c Records

Std Controller - Account

<apex:Input Field>

To Save or Insert the data … Buttons or Labels

**Scenario 6:** Create a VFPage where user can input the values into an Account object <apex:outputLabel> & <apex:inputText>--> . / Create a VFPage to insert the Account Records

Std Controller - Account

<apex:Input Field>

To Save or Insert the data … Buttons or Labels

**HW Scenario 7:** Create a VFPage where user can input the values into an Position object <apex:outputLabel> & <apex:inputText>--> . / Create a VFPage to insert the Position Records

Std Controller - Account

<apex:Input Field>

To Save or Insert the data … Buttons or Labels

**Scenario 8 :** Please display the associated contact information (Name, Mailing city, Phone, Title) for a Particular Account. ***Use a Page Block Table as a component*** … and pass the account Record Id into the current context

**HW Scenario 9 :** Please display the associated Opportunity information (Name, Mailing city, Phone, Title) for a Particular Account. ***Use a Page Block Table as a component*** … and pass the account Record Id into the current context

**Scenario 10:** Create a VFPage on Review object where all the Score parameters like (Experience, Cultural Fit, Leadership skills) should display in the Radio button, and option values or Ratings should be Excellent, Very Good, Good, Fair, Bad…for all the three reviews.

<apex:Page>

<Aura:CommandButton value=”Save” action=”{!Save}”>

<Lightning:Button>

**List Controller**

**Standard list controllers** allow you to create **Visualforce** pages that can display or act on a set of records. Examples of existing **Salesforce** pages that work with a set of records include **list** pages, related **lists**, and mass action pages

Displaying a **list** of records is a fundamental behavior of almost all web apps. **Visualforce** makes it extremely easy to display a **list** of records of the same type, using just markup, no back-end code.

***A standard list controller can return up to 10,000 records. Custom controllers can work with larger results sets.***

***Using a standard list controller is very similar to using a standard controller. First you set the standardController attribute on the <apex:page> component, then you set the recordSetVar attribute on the same component.***

***The recordSetVar attribute not only indicates that the page uses a list controller, it sets the variable name of the record collection. This variable can be used to access data in the record collection.***

Earlier.. how many records we got only single record… record is we have passed..

\*\*\*\*\* I want to display all the opportunities record..

**RecordSetVar** – collection of records displayed in the form of datatable

**Renderas =**”pdf”

Datat to be displayed in tabular format. and I want in a **pdf format..**

**Scenario 11:** Please display the Opportunity information Name,StageName,Closedate in a Table structure and ensure you can download that data in a PDF format for Opportunity Object

**Convert VF Page to Lightning Experience**

*The Lightning Design System is collection of design patterns, components, and guidelines for creating unified UI in the Salesforce ecosystem. The Design System makes it easy for you to build applications that comply with the new Salesforce Lightning look and feel without reverse engineering the UI as custom CSS.*

Icons

### Categories

* [**Utility**](https://www.lightningdesignsystem.com/icons/#utility)
* [**Doctype**](https://www.lightningdesignsystem.com/icons/#doctype)
* [**Standard**](https://www.lightningdesignsystem.com/icons/#standard)
* [**Custom**](https://www.lightningdesignsystem.com/icons/#custom)
* [**Action**](https://www.lightningdesignsystem.com/icons/#action)

Design Patterns

Grids

**Custom Controller**

Controllers contain Apex that defines how pages interact with each other, and the rest of the platform

Ability to reuse any standard Salesforce UI behavior like new, edit, save, etc (standard controller) and have access to Salesforce data.

Ability to define new UI behaviors and navigation using Apex (custom controller).

Developers can use Standard Controllers (generated automatically for all Standard and Custom Objects) to implement standard Salesforce behavior (Save, Delete etc) on Visualforce Pages.

Custom Controllers to implement custom behavior.

A ***custom controller*** is an Apex class that implements all of the logic for a page without leveraging a standard controller. Use custom controllers when you want your Visualforce page to run entirely in system mode, which does not enforce the permissions and field-level security of the current user.

Scenario 12: Input Employee Name, Employee Address on a VFPage and display these details by clicking on a button

Scenario 13: Create a VFPage where to calculate and perform Arithmetic Operations

Scenario 14: Create a List of Account Records in an Apex and fetch those records using SOQL statement and display all these records on a VFPage

Scenario 15: Create an Apex Class where you can insert Account, Opportunity and Contact records.. and insert these details on a VFPage

Custom Extension Controller

Standard Controller = Candidate\_\_c extensions =” CandidateExtCnt”

@Future Method

* ***When to use Future Method:***  
    
  If you want to make the execution of the apex program to run asynchronously then we make use of future method.
* When you specify future , the method executes when Salesforce has available resources.
* Future methods are used to run the process in a separate thread, at later time when system resources are available. We can use future methods for any operation we would like to run asynchronously in its own thread.
* Each future method is queued and executes when system resources become available.
* A future method runs in the background, asynchronously. You can call a future method for executing long-running operations, such as callouts to external Web services or any operation you’d like to run in its own thread, on its own time.
* You can also use future methods to isolate DML operations on different sObject types to prevent the mixed DML error.
* Each future method is queued and executes when system resources become available. That way, the execution of your code doesn’t have to wait for the completion of a long-running operation.
* A benefit of using future methods is that some governor limits are higher, such as SOQL query limits and heap size limits.
* Future methods provide the benefits of not blocking the users from performing other operations and providing higher governor and execution limits for the processes.

**Real-World Scenarios and Use Cases of Future Method**

Here are three common use cases where the Future Method shines:

1. **Background Processing:**When you have tasks that don’t require immediate attention or user interaction, such as data deletion or cleanup operations, you can utilize the Future Method to move them to the background. By doing so, you optimize resource utilization and improve overall efficiency.

**2.DML Operations and Mixed DML Exception:** In Salesforce, you might encounter a “Mixed DML” exception when trying to perform DML operations on both setup and non-setup objects in a single transaction. By encapsulating the setup object operations within a Future Method, you can separate them into a different transaction, avoiding the exception and ensuring smooth execution.

**3.Trigger Callouts:** Salesforce imposes limitations on making callouts directly from triggers. However, by invoking a Future Method from a trigger(trigger handler), you can perform callouts asynchronously, mitigating the restriction and ensuring smooth integration with external systems.

**Key Points:**

1. The method annoted with @future is known as **future method.**
2. Future method :
   1. can only **return void type.**
   2. must be **static method.**
   3. cannot take objects and sOjects as parameter.
   4. allow callout when callout = true is specified.(Default is callout = false).

 3. Specified parameter must be **primitive data types, arrays of primitive data types, or collections of primitive data types.**

4.  *Method may or may not be executing in same order as it is called.*

 It cannot be used in visualforce controller either in **getMethodName()** or in **setMethodName()** nor in constructor.

5. Nested future calling is not allowed means you cannot call a future method in another future method neither in trigger.

**Rules to define a future method:**

1. All the future methods should have “@future” annotation.
2. All future methods should be defined as static.
3. All the future method should have void as a return type
4. Only primitive variables can be passed as parameters.
5. When we call the future method they will be added in Queue and from Queue they will be executed.
6. If you want to invoke web services from the future method then define

@future(callout=true)

1. Any asynchronous job that is running in Salesforce will be registered with AsyncApexJob Object.

How to track the status of future methods?

Write a SOQL query on Async Apex Job.

Declarative way to check the status.

Setup -> Monitor-> Jobs->Apex Jobs

Future methods can be used to increase the Governing Limits

@future(limits=dml\*2)

**Drawbacks:**

1. Future Method will not return JobId in the apex code.
2. The future method can not be called from another future method.
3. The future method will not support sObjects as parameters. (The future method will not support Subjects as parameters.)
4. Limits:

**Within the transaction, we can call 50 future methods.**

Mixed DML Exception:

If you make a DML operation on setup objects and non-setup objects, in a single transaction then it throws Error as “mixed DML Exception”

5. Future Methods have their own set of limits, such as the number of method invocations, CPU time, heap size, and more. It’s important to monitor and optimize your code to avoid hitting these limits.

**6. Future Methods do not guarantee order of execution**. *Since they run asynchronously, the order in which they are invoked may not necessarily reflect the order in which they are completed. If the order of execution is critical, you should consider alternative solutions or additional logic to handle sequencing.*

**7. Future Methods do not support transaction control.** *They operate in their own separate transactions, meaning they cannot be rolled back if an error occurs. If you require transactional consistency, you might need to explore other mechanisms, such as Queueable Apex.*

**Batchable Apex**

It is used to run large jobs (think thousands or millions of records!) that would exceed normal processing limits. Each time you invoke a batch class, the job is placed on the Apex job queue and is executed as a discrete transaction.

A developer can now employ batch Apex to build complex, long-running processes that run on thousands of records on the Lightning Platform. Batch Apex operates over small batches of records, covering your entire record set and breaking the processing down to manageable chunks. For example, a developer could build an archiving solution that runs on a nightly basis, looking for records past a certain date and adding them to an archive. Or a developer could build a data cleansing operation that goes through all Accounts and Opportunities on a nightly basis and updates them if necessary, based on custom criteria.

For example, if you want to make a field update of all records in any object which is having more number of records, then governor limits restricts us to process that operation. **Because in a single transaction we can only process 10,000 records.**

To use batch Apex, write an Apex class that implements the Salesforce-provided interface **Database.Batchable** and then invoke the class programmatically.

To monitor or stop the execution of the batch Apex job, from Setup, enter Apex Jobs in the Quick Find box, then select Apex Jobs.

To write a Batch Apex class, your class must implement the **Database.Batchable** interface and include the following three methods:

**Start ()** method is automatically called at the beginning of the apex job. This method will collect record or objects on which the operation should be performed. These records are divided into subtasks and pass those to execute method.

**Execute()**  Method performs an operation which we want to perform on the records fetched from start method. sObject records or a list of sObject records), up to 200 records at a time

**Finish ()** method executes after all batches are processed. Use this method to send confirmation email notifications.

**Using Database.BatchableContext**

All the methods in the Database.Batchable interface require a reference to a Database.BatchableContext object. Use this object to track the progress of the batch job.

**Using Database.QueryLocator to Define Scope**

The start method can return either a Database.QueryLocator object that contains the records to use in the batch job or **an iterable**. ***With the QueryLocator object***, the governor limit for the total number of records retrieved by SOQL queries is bypassed and you can query up to 50 million records. However, with an Iterable, the governor limit for the total number of records retrieved by SOQL queries is still enforced.

**Using an Iterable in Batch Apex to Define Scope**

The start method can return either a Database.QueryLocator object that contains the records to use in the batch job or an iterable. Use an iterable to step through the returned items more easily.

Example-1

global class BatchPositionDemo implements Database.Batchable<sObject>  
{  
    global Database.QueryLocator start(Database.BatchableContext bc)  
    {  
        String query='Select id,Name from Position\_\_c';  
        return Database.getQueryLocator(query);   
    }  
    global void execute(Database.BatchableContext bc, List<Position\_\_c> pslist)  
    {  
        // Process each batch records  
  
        for(Position\_\_c plist:pslist)  
        {  
            plist.Name=plist.Name + 'Cognizant';                  
        }  
        try  
        {  
        update pslist;  
        }  
        catch(DMLException e)  
        {  
            System.debug(e);  
        }  
    }  
  
    global void finish(Database.BatchableContext bc)  
    {  
        System.debug('Updated Records Successfully');  
    }  
}

**Example-2**

public class BooksBatchUdt implements Database.Batchable<sObject>

{  
    public Database.queryLocator start(Database.BatchableContext bc){  
        String query = 'SELECT Id,Price\_\_c,Book\_Type\_\_c FROM Book\_\_c';  
        return Database.getqueryLocator(query);  
    }  
  
    public void execute(Database.BatchableContext bc, List<Book\_\_c> bookList){  
        for(Book\_\_c book : bookList){  
            if(book.Price\_\_c>=500){  
                book.Book\_Type\_\_c='Fiction';  
            }  
        }  
        try{  
            update bookList;  
        }catch(Exception e){  
            System.debug(e);  
        }  
    }  
    public void finish(Database.BatchableContext bc){  
        System.debug('Batch processed done!');  
  
    }  
}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Queueable Apex -**

**What is Queueable?**

Released in Winter '15, Queueable Apex is essentially a superset of future methods with some extra #awesomesauce.

Take control of your asynchronous Apex processes by using the Queueable interface. This interface enables you to add jobs to the queue and monitor them. Using the interface is an enhanced way of running your asynchronous Apex code compared to using future methods.

Apex processes that run for a long time, such as extensive database operations or external web service callouts, can be run asynchronously by implementing the Queueable interface and adding a job to the Apex job queue. In this way, your asynchronous Apex job runs in the background in its own thread and doesn’t delay the execution of your main Apex logic. Each queued job runs when system resources become available. A benefit of using the Queueable interface methods is that some governor limits are higher than for synchronous Apex, such as heap size limits.

**Why should we learn Queueable? When we have a future method**

Queueable is similar to @future method, which is a set of code that runs in background which is asynchronous.

IF it is similar to @future method why should we learn Queueable?

Queueable jobs are similar to future methods in that they’re both queued for execution, but they provide you with these additional benefits.

**Getting an ID for your job:** When you submit your job by invoking the System.enqueueJob method, the method returns the ID of the new job. This ID corresponds to the ID of the AsyncApexJob record. Use this ID to identify your job and monitor its progress, either through the Salesforce user interface in the Apex Jobs page, or programmatically by querying your record from AsyncApexJob.

**Using non-primitive types:** Your queueable class can contain member variables of non-primitive data types, such as sObjects or custom Apex types. Those objects can be accessed when the job executes.

**Chaining jobs:** You can chain one job to another job by starting a second job from a running job. Chaining jobs is useful if your process depends on another process to have run first.

Schedulable Apex

**Apex Testing**

Deployment Process

Create Sandbox

Types of Sandbox

Create Changesets (Inbound and Outbound)

The url which I have given is for creating an Enterprise edition (46 days trail version)

Created Sandbox..

**Apex Testing**

Testing - 1+1=2 Expected Outcome

1+1=4 Actual Outcome

Test is failed…because Actual outcome is not matching with Expected Outcome

ATM - Insert the card…

Insert \*\*\*\* (four digit pin number)

8888 True

0000 True

9999 True

8283 True

%%76 False

\_ ( 78 False

Expected Outcome - four digit number only..

Actual Outcome - other than four digit number rest all failed test cases..

White Box - Developers - Test the code

Black Box - Tester – functionality

Are you developers or Testers

Developers

It to test the code… not execute

100-125 Programs - Are your programs is tested?

You need to test your code..

Programmer or Developer is going to do Unit Testing or Module Testing.

In Salesforce

Salesforce requires at least 75% of your code to be covered by our test classes. Salesforce has done that to make sure that our code doesn’t break in any situation in Production.

**What to Test in Apex ?**

Salesforce recommends that you write tests for the following:

**Single action**

Test to verify that a single record produces the correct, expected result.

**Bulk actions**

Any Apex code, whether a trigger, a class or an extension, may be invoked for 1 to 200 records. You must test not only the single record case, but the bulk cases as well.

**Positive behavior**

Test to verify that the expected behavior occurs through every expected permutation, that is, that the user filled out everything correctly and did not go past the limits.

**Negative behavior**

There are likely limits to your applications, such as not being able to add a future date, not being able to specify a negative amount, and so on. You must test for the negative case and verify that the error messages are correctly produced as well as for the positive, within the limits cases.

**Restricted user**

Test whether a user with restricted access to the sObjects used in your code sees the expected behavior. That is, whether they can run the code or receive error messages.

**WHY UNIT TESTS ARE NEEDED**

* Validate Desired Behavior
* Reduce the bug Cost
* Perform Bulk tests
* 75% Code coverage for Production deployment

**WHAT TO TEST**

* Apex Trigger
* Apex Class – Handler/Helper, WebService, Apex REST, SOAP
* VF Page
* Apex Batch/Queueable/Future Method
* Custom Controller

**The following are the benefits of Apex unit tests**.

1. Ensuring that your Apex classes and triggers work as expected
2. Having a suite of regression tests that can be rerun every time classes and triggers are updated to ensure that future updates you make to your app don’t break existing functionality
3. Meeting the code coverage requirements for deploying Apex to production or distributing Apex to customers via packages
4. High-quality apps delivered to the production org, which makes production users more productive
5. High-quality apps delivered to package subscribers, which increase your customers trust

Important Points

* At least 75% of your Apex code must be covered by unit tests, and all of those tests must complete successfully. But this should not be our focus. We should aim for **100% code coverage**, which ensures that you cover each positive and negative use case of your code to cover and test each and every branch of your code.
* Calls to System.debug are not counted as part of Apex code coverage.
* Test methods and test classes are not counted as part of Apex code limit. So, no worries about writing long test class with more methods just to make sure that all your code branches are covered.
* Every trigger you are trying to deploy should have at least 1% coverage, but yes overall coverage of your production org after getting your code deployed should be 75%, otherwise Salesforce won’t let you deploy your code.
* ***Class can be deployed on 0% coverage as well, but as I told in last point, that overall coverage of your production org after getting your code deployed should be 75%, otherwise Salesforce won’t let you deploy your code.***

**The key points while writing a test class are:**

1. You have to start your class with ***@isTest*** annotation, then only Salesforce will consider this class as test class.
2. Keep your class as Private, and the best practice is to name your test class as your original Class or trigger Name + ‘Test’.
3. Methods of your test class have to be ***static, void and testMethod*** keyword has to be used.
4. Prepare your test data which needs to be existing before your actual test runs. There are multiple techniques of creating test data now a days, for example, setup method, static resources etc. For more details about ***@testsetup method check below link @testSetup*** method in apex test class
5. Use Test.startTest() and Test.stopTest() to make sure that the actual testing of your code happens with the fresh set of governer limits. These methods help you to reset your governor limits just before your actual code of testing get executed.
6. Once your test code runs between Test.startTest() and Test.stopTest(), you must use assert statements to test whether your actual code is executing correctly and giving the results as expected. In our case, we are test whether book’s price has been set to 90 or not. If this assert statement returns false, then your test class will fail, and will let you know, that something is not correct in your code, and you need to fix your original code.
7. Because we are testing a simple trigger, we could not show the testing using negative use cases, but in an ideal world, you should write multiple methods in your test class, few should test your positive use cases, and other should test your negative test cases.

**Deployment Process**

1. **Create an App in Sandbox and deploy that app in Production environment.**

**Sandbox**

Create an App in Sandbox and add some objects to that app and deploy this App in Production environment by creating Change Sets

Create an App in Sandbox and will deploy that App in Production Environment

Change sets - Inbound and Outbound

Will deploy change sets into Production Environment?

Change Set –

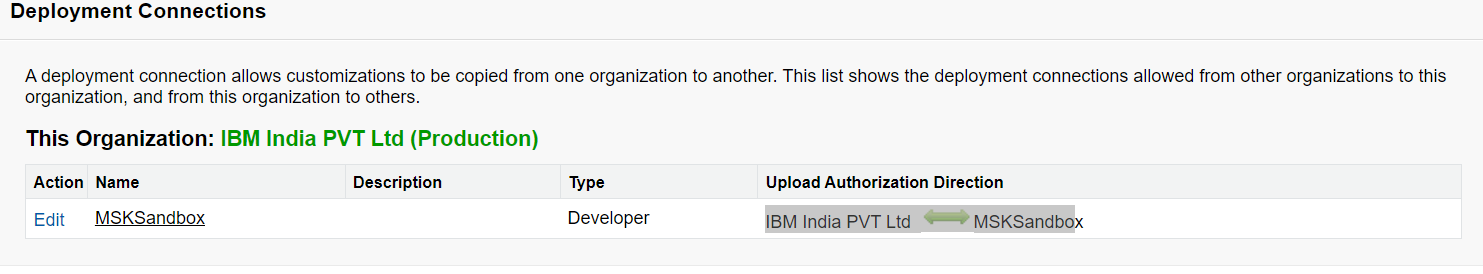
**How to Create an App in Sandbox and how to deploy this App in Production**?

**Change sets - Deployment Process**

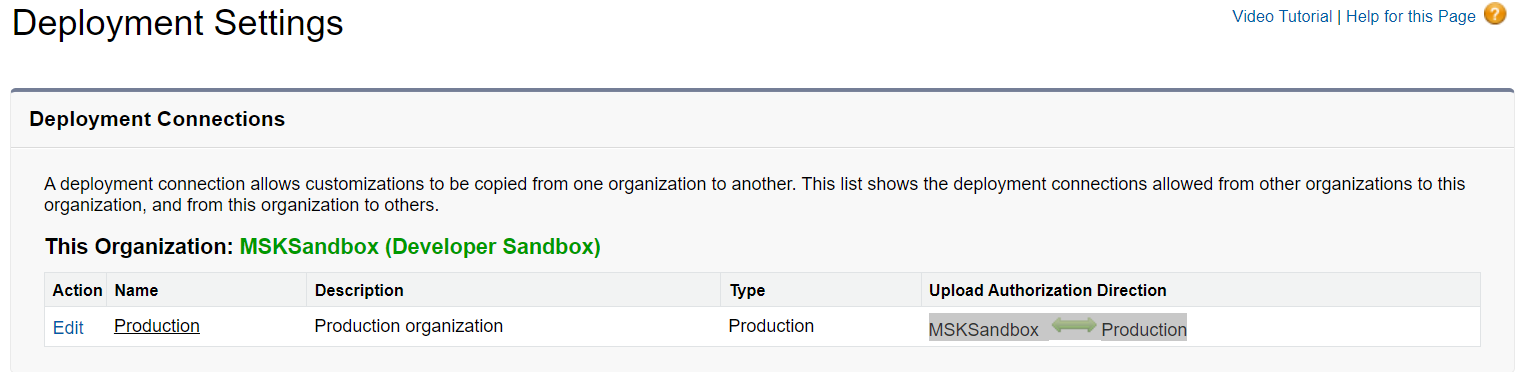
1. Create an App in Sandbox..

Setup a communication process between Production Env and with Sand Box

Production Org – Deployment Settings 🡪 Allow inbound changes



Sandbox Org – Deployment Settings 🡪 Allow inbound changes



**Production 🡨---🡪 Sandbox**

**Tab->object -> fields (Metadata) Data about data..**

**What is Change Sets**

**Collection of Components**

Objects

Metadata

Records

Tabs

Apps

Apex Classes

Apex Triggers

Aura

LWC

**Two types of Change sets**

**Inbound Change set**

**Outbound Change set**

An outbound change set contains customizations that you want to send from this organization to another organization. These customizations can include new components or modifications to existing components, such as apps, objects, reports, or Apex classes and triggers. An outbound change set can't be used to delete or rename components in another organization.

**Out bound Change set in Sandbox**

Add the components

Added the dependencies

Add the Profile

**Upload the change set**

Choose the organization that will receive the change set. Once the upload completes, the administrator responsible for authorizing deployments in the target organization will be notified.

**Push – to Prod Org**

**Inbound Change Sets**

Validated to check if any errors are there in change set

Deployed the Change sets in Prod Org

Successful.

You can view the App in Production Org..

What are the other Deployment methods are there?

1. Change sets
2. Eclipse with Force.com IDE (Deprecated)

Eclipse IDE deployment is easy to use where you can migrate code with easy steps but this deployment method is recommended only if you have less number of test classes. If you have more than 100 classes eclipse generally hang and doesn’t show any progress.

1. VS Code
2. Code Builder
3. Salesforce Package
4. Force.com Migration Tool – ANT/Java based -

\*\*\*\*\*\*\* How do you test your code \*\*\*\*\*\*\*\*\*\*\*\*\*\*

You can test

Apex Trigger

* Apex Class – Handler/Helper, WebService, Apex REST, SOAP
* VF Page
* Apex Batch/Queueable/Future Method
* Custom Controller

Code Coverage -75%

Scenario 1: Write an Apex class which will calculate the Arithmetic operations of a two numbers and test that code…(Write an Apex Test Class to test it and get 75% of Code coverage)

**Step1: Create an Apex Class:**

public class Calculations

{

public Integer printSum(Integer x, Integer y)

{

return x+y;

}

public Integer printProduct(Integer x, Integer y)

{

return x\*y;

}

}

**Step2: Create an Apex Class to test the methods written in the above class:**

**Test Class**

@isTest (if you want to test a class if not the class will not test)

public class TestCalculations

{

public static testmethod void testPrintSum() test method is compulsory of u want to test a method.

{

Integer x=20;

Integer y=10;

Integer expected=30;

Calculations c1=new Calculations();

Integer actual=c1.printSum(x, y);

System.assertEquals(expected, actual); ->

}

public static testmethod void printProduct()

{

Integer x=20;

Integer y=30;

Integer expected=600;

Calculations c1=new Calculations();

Integer actual=c1.printProduct(x, y);

System.assertEquals(expected, actual);

}

}

To run the above program click on “Run Test” button which is placed at the top right corner. Then you can find the result

**Unit Testing Scenario 3**

Create a simple Apex trigger which blocks inserts and updates to any contact with a last name of ' SUPERSTAR'. Then write unit tests that achieve 100% code coverage.

trigger RestrictContactByName on Contact (before insert, before update)

{

//check contacts prior to insert or update for invalid data

For (Contact c : Trigger.New) {

if(c.LastName == ‘SUPERSTAR')

{ //invalidname is invalid

c.AddError('The Last Name " '+c.LastName+' " is not allowed for DML');

}

}

}

**Test Class**

@isTest

public class TestContact {

public static testMethod void Test()

{

List<Contact> listContact= new List<Contact>();

Contact c1 = new Contact(FirstName='Vipul', LastName='Kumar', email='vipul@test.com');

Contact c2 = new Contact(FirstName='Newron', LastName ='SUPERSTAR’,email='newron@test.com');

listContact.add(c1);

listContact.add(c2);

Test.startTest();

try

{

insert listContact; // DML Command

}

catch(Exception ee)

{

}

Test.stopTest();

}

}

**Example 4:** Create a simple Apex trigger which blocks inserts and updates to any contact with a last name of ' SUPERSTAR'. Then write unit tests that achieve 100% code coverage.

public class CreateContact {  
    public static void createcon(list<contact> conlist){  
        if(!conlist.isEmpty()){  
            for(contact c:conlist){  
                if(c.lastname=='SUPERSTAR'){  
                    c.addError('The Last name "'+c.LastName+'" is not allowed for DML');  
                }  
            }  
        }  
    }  
}

///////////////////////////////////////////////////////////////////

trigger RestrictContactByName on Contact (before insert,before update) {  
    if(trigger.isbefore){  
        if(trigger.isInsert){  
            CreateContact.createcon(Trigger.New);  
        }

/\*   if(trigger.isInsert || Trigger.isUpdate){  
            CreateContact.createcon(Trigger.New);\*/ // you can use this or below code

  if(trigger.isUpdate){  
            CreateContact.createcon(Trigger.New);  
        }  
    }  
}

////////////////////////////////////////////////////////////////////////////

@isTest  
public class TestRestrictContactByName {  
    public static testmethod void Testcreatecon(){  
        list<Contact> listcon=new list<Contact>();  
        contact c1=new contact(Firstname='vipul',lastname='kumar',email='vipulkumar@gmail.com');  
        contact c2=new contact(Firstname='Newron',lastname='SUPERSTAR',email='Newron@gmail.com');  
        listcon.add(c1);  
        listcon.add(c2);  
        Test.startTest();  
        try{  
            insert listcon;  
  
        }catch(exception e){  
  
        }  
        Test.stopTest();  
    }  
}

//////////////////////////////////////////////////

@isTest

public class TestRestrictContactByName {

public static testMethod void Test()

{

List<Contact>listcontact=new List<Contact>();

//take 2 sobjects

Contact c1= new Contact(FirstName='Raamm', LastName='Leela', email='ramleela@test.com');

Contact c2= new Contact(FirstName='Ravan', LastName='INVALIDNAME', email='ravaa@test.com');

listcontact.add(c1);

listcontact.add(c2);

//add sobjects to the list

Test.startTest();

System.assertEquals('Leela',c1.LastName);

System.assertEquals('INVALIDNAME',c2.LastName);

Test.stopTest();

}

}

// System.assertEquals('INVALIDNAMES',c2.LastName);

Go to contacts create or update a contact record with INVALIDNAME it will through the error written in Trigger

Book Example

public class Books

{

public static void applydiscount(Book\_\_c [] book1)

{

for(Book\_\_c b:book1)

{

b.Price\_\_c \*=0.9;

}

}

}

You add an Apex class that contains a method for updating the book price. This method is called by the trigger that you will be adding in the next step.

You now have a class that contains some code that iterates over a list of books and updates the Price field for each book. This code is part of the applyDiscount static method called by the trigger that you will create in the next step.

**you create a trigger for the Book\_\_c custom object that calls the applyDiscount method of the Books class that you created in the previous step.**

// Trigger Handler

trigger BooksTriger on Book\_\_c (before insert) {

Book\_\_c [] books1=Trigger.New;

books.applydiscount(books1);

}

You now have all the code that is needed to update the price of all books that get inserted. However, there is still one piece of the puzzle missing. Unit tests are an important part of writing code and are required. In the next step, you will see why this is so and you will be able to add a test class.

you add a test class with one test method. You also run the test and verify code coverage. The test method exercises and validates the code in the trigger and class. Also, it enables you to reach 100% code coverage for the trigger and class.

@isTest

private class HelloWorldTestClass {

static testMethod void validateHelloWorld()

{

Book\_\_c b = new Book\_\_c(Name='Behind the Cloud', Price\_\_c=100);

System.debug('Price before inserting new book: ' + b.Price\_\_c);

// Insert book

insert b;

// Retrieve the new book

b = [SELECT Price\_\_c FROM Book\_\_c WHERE Id =:b.Id];

System.debug('Price after trigger fired: ' + b.Price\_\_c);

// Test that the trigger correctly updated the price

System.assertEquals(90, b.Price\_\_c);

}

}

you have completed all the steps necessary for writing some Apex code with a test that runs in your development environment. In the real world, after you’ve tested your code and are satisfied with it, you want to deploy the code and any prerequisite components to a production org. The next step shows you how to do this deployment for the code and custom object you’ve created.

1. From Setup, enter Outbound Changesets in the Quick Find box, then select Outbound Changesets.
2. If a splash page appears, click Continue.
3. In the Change Sets list, click New.
4. Enter a name for your change set, for example, HelloWorldChangeSet, and optionally a description. Click Save.
5. In the Change Set Components section, click Add.
6. Select Apex Class from the component type drop-down list, then select the MyHelloWorld and the HelloWorldTestClass classes from the list and click Add to Change Set.
7. Click View/Add Dependencies to add the dependent components.
8. Select the top checkbox to select all components. Click Add To Change Set.
9. In the Change Set Detail section of the change set page, click Upload.
10. Select the target organization, in this case production, and click Upload.
11. After the change set upload completes, deploy it in your production organization.
12. Log into your production organization.
13. From Setup, enter Inbound Change Sets in the Quick Find box, then select Inbound Change Sets.
14. If a splash page appears, click Continue.
15. In the change sets awaiting deployment list, click your change set's name.
16. Click Deploy.
17. In this tutorial, you learned how to create a custom object, how to add an Apex trigger, class, and test class. Finally, you also learned how to test your code, and how to upload the code and the custom object using Change Sets.

@Invocable Method

public class AccountQueryAction {

@InvocableMethod(label='Get Account Names' description='Returns the list of account names corresponding to the specified account IDs.' category='Account')

public static List<String> getAccountNames(List<ID> ids)

{

List<String> accountNames = new List<String>();

List<Account> accounts = [SELECT Name FROM Account WHERE Id in :ids];

for (Account account : accounts)

{

accountNames.add(account.Name);

}

return accountNames;

}

}

**An Apex annotation** modifies the way a method or class is used similar to annotations in Java. Annotations are defined with an initial @ symbol, followed by the appropriate keyword. To add an annotation to a method, specify it immediately before the method or class definition.

The main use of annotation is to instruct the compiler that the class is to be used as the specific class as defined by the annotation. Annotations are defined with an initial @ symbol, followed by the appropriate keyword.

**Apex supports the following annotations.**

* + 1. @AuraEnabled
    2. @Deprecated
    3. @Future
    4. @InvocableMethod
    5. @InvocableVariable
    6. @IsTest
    7. @ReadOnly
    8. @RemoteAction
    9. @SuppressWarnings
    10. @TestSetup
    11. @TestVisible

The **@AuraEnabled** annotation enables client- and server-side access to an Apex controller method. Providing this annotation makes your methods available to your Lightning components (both Lightning web components and Aura components). Only methods with this annotation are exposed.

**@Deprecated:**  
Use the deprecated annotation to identify methods, classes, exceptions, enums, interfaces, or variables that can no longer be referenced in subsequent releases of the managed package in which they reside. This is useful when you are refactoring code in managed packages as the requirements evolve. New subscribers cannot see the deprecated elements, while the elements continue to function for existing subscribers and API integrations.

**@Future:**  
Use the future annotation to identify methods that are executed asynchronously. When you specify future, the method executes when Salesforce has available resources.  
To test methods defined with the future annotation, call the class containing the method in a startTest, stopTest code block. All asynchronous calls made after the startTest method are collected by the system. When stopTest is executed, all asynchronous processes are run synchronously.

**@IsTest:**  
Use the isTest annotation to define classes or individual methods that only contain code used for testing your application. The isTest annotation is similar to creating methods declared as testMethod.

@SuppressWarnings

This annotation does nothing in Apex but can be used to provide information to third party tools.

**@ReadOnly:**  
The @ReadOnly annotation allows you to perform unrestricted queries against the Force.com database. All other limits still apply. It’s important to note that this annotation, while removing the limit of the number of returned rows for a request, blocks you from performing the following operations within the request: DML operations, calls to System.schedule, calls to methods annotated with @future, and sending emails.

**@RemoteAction:**  
The RemoteAction annotation provides support for Apex methods used in Visualforce to be called via JavaScript. This process is often referred to as JavaScript remoting.

**@TestVisible:**  
Use the TestVisible annotation to allow test methods to access private or protected members of another class outside the test class. These members include methods, member variables, and inner classes. This annotation enables a more permissive access level for running tests only.

@testSetup Test Setup Methods

Use test setup methods (methods that are annotated with @testSetup) to create test records once and then access them in every test method in the test class. Test setup methods can be time-saving when you need to create reference or prerequisite data for all test methods, or a common set of records that all test methods operate on. Is used to create common test data and use this method in test classes where you want same data. Method marked with @TestSetUp annotation executes before any testMethod. You just need to wrap the test data inside this testsetup method.

**@InvocableMethod**

Use the **InvocableMethod** annotation to identify methods that can be run as invocable actions.

How to use apex action?

Action encapsulate a piece of logic that allows a user to perform some work, such as sending the email. Use actions to add more functionality to your applications. Choose from standard actions, such as posting to Chatter or sending email, or create actions based on your company’s needs.

*If a flow invokes Apex, the running user must have the corresponding Apex class security set in their user profile or permission set.*

Invocable methods are called natively from REST API’s, Apex, Flow, or Einstein bot that interacts with the external API source. Invocable methods have dynamic input and output values and support describe calls.

**Calling as an Apex Actions**

Now you can able to call the apex class from the apex action using the rest API resource. use the following rest API resource.

**InvocableMethod Considerations**

* 1. The invocable method must be static and public or global, and its class must be an outer class.
  2. Only one method in a class can have the InvocableMethod annotation.
  3. Other annotations can’t be used with the InvocableMethod annotation.

**Below code sample shows an invocable method with primitive data types.**

public class AccountQueryAction

{

***@InvocableMethod(label='Get Account Names' description='Returns the list of account names corresponding to the specified account IDs.' category='Account')***

public static List<String> getAccountNames(List<ID> ids)

{

List<String> accountNames = new List<String>();

List<Account> accounts = [SELECT Name FROM Account WHERE Id in :ids];

for (Account account : accounts) {

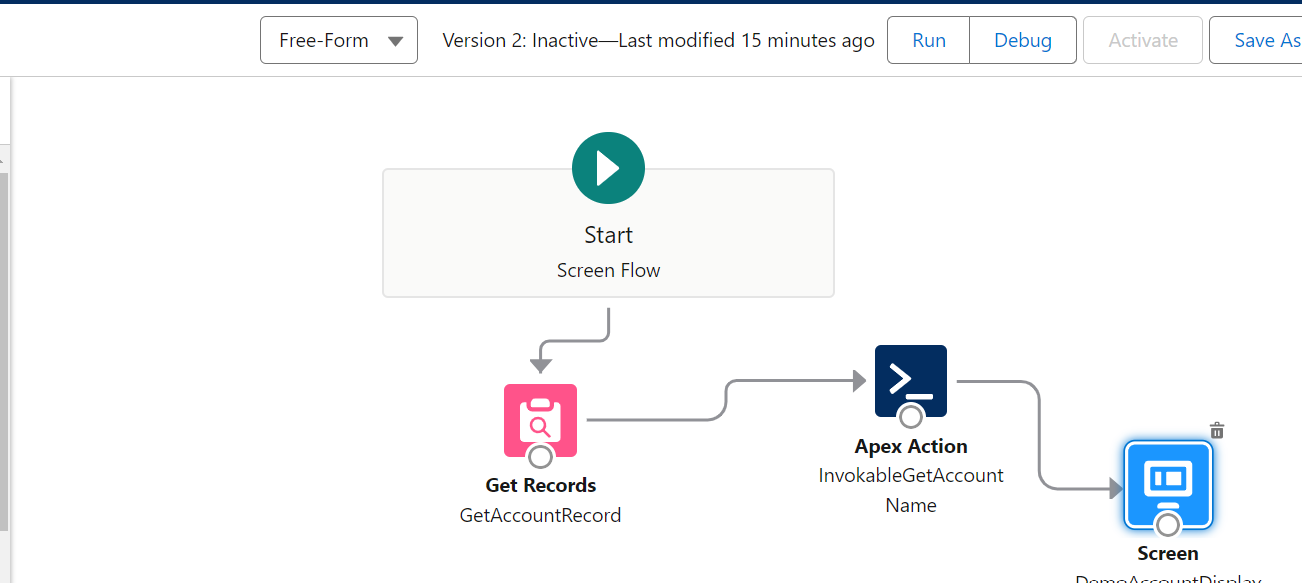
accountNames.add(account.Name);

}

return accountNames;

}

}



///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**Custom Settings**

Custom settings are similar to custom objects. The concept of custom setting in Salesforce is similar to that of custom objects. Just like a developer or operator could create custom objects if the existing Salesforce objects are not enough for the requirements of the business, custom settings can be created to facilitate application developers in building custom data sets associate all necessary custom data according to customized and specific requirements of a profile, a user, an account or the organization as a whole. Application developers can create custom sets of data and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. Formula fields, validation rules, flows, Apex, and SOAP API can then use this data.

There is a lot of similarity between custom objects and settings. Both these components allow Salesforce developers to build custom sets of data and associate the same to the relevant entity according to the client’s requirements. Also, users can define custom fields in both custom objects and settings.

However, custom settings have relatively fewer types of fields, which include:

1. Checkbox

2. Date/time

3. Number

4. Percent

5. Text

6. Text area

7. Phone

8. Currency

9. URL

Also, there is no provision for page layouts or record types available to the developers for custom settings. Due to this, Salesforce Developers and operators are handicapped when it comes to re-order necessary fields in a concerning page layout. In order to do so, developers will have to create Visualforce pages according to the requirements.

In custom settings, there is no clear definition of workflow rules, validation rules, or triggers that are available in the custom objects.

In spite of these shortcomings, the custom settings are not useless components and have some very critical benefits.

1. The data pertaining to custom settings are stored in the application cache.
2. This allows sufficient access to the users, saving them the cost of making several queries to access the database.
3. Also, using custom settings helps the users in getting rid of Governor Limits almost completely.
4. Provided the user is employing GET methods and not SOQL to query the settings, they will be able to extract all the necessary values with absolutely no impact on the count kept by the governor regarding the number of queries performed by the user.

**There are two types of custom settings.**

**List Custom Settings**

A type of custom setting that provides a reusable set of static data that can be accessed across your organization. If you use a particular set of data frequently within your application, putting that data in a list custom setting streamlines access to it. Data in list settings does not vary with profile or user, but is available organization-wide. Examples of list data include two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. Because the data is cached, access is low-cost and efficient: you don't have to use SOQL queries that count against your governor limits.

public class DemoCustomSetting

{

public List<Country\_\_c>getData {get;set;}

public DemoCustomSetting ()

{

//The following example returns a map of custom settings data. The getAll method returns values

// for all custom fields associated with the list setting.

Map<String,Country\_\_c>alldata= Country\_\_c.getAll();

//the getValues method to return all the field values associated with the specified data set.

// This method can be used with both list and hierarchy custom settings, using different parameters.

getData = alldata.values();

}

}

<apex:page controller="DemoCustomSetting" sidebar="false" >

<apex:form >

<apex:pageblock title="Currency Details">

<apex:pageblockTable value="{!getData }" var="A">

<apex:column value="{!A.Name}"/>

<apex:column value="{!A.Currency\_\_c}"/>

</apex:pageblockTable>

</apex:pageblock>

</apex:form>

</apex:page>

**Hierarchy Custom Settings**

A type of custom setting that uses a built-in hierarchical logic that lets you “personalize” settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or “lowest,” value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

**JavaScript**

Chrome

Edge

Internet Explorer

Safari

Brave

Opera

Firefox

Netscape Navigator

Security…

Integer num1=100; ////////// Strongly typed

num1;

num1=100;

num1=’Shiva’

Class

Object

Math

Date

String

Var

Let

Const

Constructor Fun1()

Objects

Asynchronous

* **Understand basic programming constructs**
  + Comments
  + Variables
  + Datatypes
  + Conditional statements
  + Iteration statements

Three types of Variables

We can declare a variable in three ways. Each keyword holds some specific reason or feature in JavaScript. Basically, we can declare variables in three different ways by using

1. var
2. let
3. const keywords.

Each keyword is used in some specific conditions

Var Variable

var str1='Hello'; // scope of str1 variable is global

function myFunction()

{

    var str1='Pradeep'; // scope of str1 variable is local to myFunction()

    console.log(str1);

}

myFunction();

console.log(str1);

**Let**

***is used to declare variable locally. If you used this keyword to declare a variable then the variable can accessible locally and it is changeable as well. It is good if the code gets huge.***

**The let keyword was introduced in ES6 (2015).**

* Variables defined with let cannot be Redeclared.
* Variables defined with let must be Declared before use.
* Variables defined with let have Block Scope.
* Let is the new var in modern javascript which will solve the block scoping issue here.

// global scope

let age=87;

{    // block scope

    let city='Hyderabad'; // local or block scope

    let state='TS';

    console.log(city);

console.log(state);

}

console.log(age);

let length=78;

let breadth=88;

const result=length\*breadth;

console.log(result);

let length=99;

function AreaofTriangle()

{

    // function scope let variables

    let length=78;

    let breadth=98;

    let result=length\*breadth;

    console.log(result);

}

AreaofTriangle();

console.log(length);

<!doctype html>

<head>

<style>

/\* CSS comes here \*/

</style>

<title>Let Statement</title>

</head>

<body>

<h2>Understand use of let in JavaScript</h2>

<script>

// global scope

let gx = 200

// block scope

{

let x = 10

//let x='jd'

document.write("<br>Inside the block: "+x)

// document.write("<br>Inside the block: "+x)

}

// function scope

**function** show()

{

let x = 20;

let y='je'

document.write("<br>Inside the function: "+x)

document.write("<br>Inside the function: "+y)

alert(x);

}

show();

document.write("<br>Global gx: "+gx)

</script>

</body>

</html>

**Const**

const pi=31.4;

console.log(pi);

pi=pi+10; //TypeError: Assignment to constant variable.

console.log(pi);

// const with an Array

const city=["Hyderabad","Chennai","Jaipur"];

console.log(city);

city.push("Bangalore");

console.log(city);

city.push("Mumbai");

console.log(city);

const fruits=new Map([["apples",600],

                     ["Mangoes",300],

                     ["Oranges",400],

                    ["Straw berry",800]]);

console.log(fruits);

const city=["Hyderabad","Chennai","Jaipur"]; // Array

const person={ …………… } // object with properties

const fruits=new Map([["”]]) /// const with map

const pi=3.14 /// const variable…

// declaring const as an Object

const person={  // person is an object

    firstname:"Shiva", // properties

    lastname:"Kumar",  // properties

    city:"Hyderabad",  // properties

}

console.log(person);

console.log("First Name " +person.firstname);

console.log("Last Name " +person.lastname);

console.log("City " +person.city);

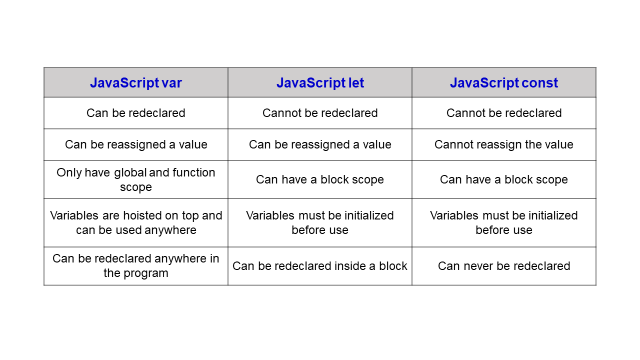
// Assigning anothe values to a const object called person...

person.firstname='Shirisha';

person.lastname='M';

person.city='Vijaywada';

console.log(person);



// Switch Case Statement

let dayOfWeek = 'Monday';  
switch (dayOfWeek)

{  
 case 'Monday':  
 console.log('It is the start of the week.');  
 break;

case 'Tuesday':  
 case 'Wednesday':  
 case 'Thursday':

console.log('It is a working day.');  
 break;

case 'Friday':  
 console.log('It is Friday! Almost weekend.');  
 break;

case 'Saturday':  
 case 'Sunday':  
 console.log('It is the weekend!');  
 break;

default:  
 console.log('Invalid day of the week.');  
}

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Iterations : JavaScript</title>

<script>

//for loop

**function** showForLoopData() //function

{

**for**(i=1;i<=10;i++){

document.write(i+"</br>");

}

}

// while loop

**function** showWhileLoopData() //function

{

i=1;

**while**(i<=10){

document.write(i+"</br>");

i++;

}

}

// Do While //

**function** showDoWhileLoopData() // function

{

i=1;

**do**{

document.write(i+"</br>");

i++;

}**while**(i<=10);

}

// Difference between While and Do While Loops //

**function** whileAndDoWhileDiff()

{

i=11;

**while**(i<=10){

document.write('This is from while loop <br/>');

document.write(i+"</br>");

i++;

}

i=11;

**do**{

document.write('This is from do while loop <br/>');

document.write(i+"</br>");

i++;

}**while**(i<=10);

// The loop condition does not satisfy With the value of i but still the loop will execute at least once. Since the condition

//is checked in the end

}

</script>

</head>

<body>

<button onclick="showForLoopData()">For</button><br/>

<button onclick="showWhileLoopData()">While</button><br/>

<button onclick="showDoWhileLoopData()">Do While</button><br/>

<button onclick="whileAndDoWhileDiff()">Difference</button><br/>

</body>

</html>

Arrays

Math

Regular Expression

String Methods

Date Object

Functions

Arrow Functions

Objects

**JavaScript Array** is also like a data type that is used to store **one or more than one value of a similar type together**. It's like a container which is used to store values in a single variable. We can use an array to store values of string type, or integer type, or an object or any other valid data type in JavaScript.

JavaScript does not have a specific array datatype for defining arrays. But, there is a predefined Array object and its methods to work with arrays. Arrays also have properties in JavaScript like length which is not a function but a property.

We can create JavaScript array either by using the new keyword or using the array literal notation i.e. [ ] (square brackets).

Regular Expression,

String, Array or Object Math or Date…objects

Will have Properties and Methods

Double quotes: "Hello".

Single quotes: 'Hello'.

Backticks: `Hello`.

Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.

Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example

let names = "John";

// embed a variable

console.log(`Hello, ${names}!`); // Hello, John!

// embed an expression

console.log( `the result is ${1 + 2}` ); // the result is 3

//The expression inside ${…} is evaluated and the result becomes a part of the string.

**Arrow Function**

This was the standard way of calling a function. Now let me show you modern way for the same. There’s another very simple and concise syntax for creating functions, that’s often better than Function Expressions.

Modern JS introduced a shorter way to define functions using what is called arrow functions. ***Using the fat arrow symbol ( => ),*** you can now create the same function using code like this:

let func = (arg1, arg2, ..., argN) => expression;

This creates a function func that accepts arguments arg1..argN, then evaluates the expression on the right side with their use and returns its result.

Let func=()=>expression

() => expression

param => expression // Single Parameter

(param) => expression // Single parameter within ()

(param1, paramN) => expression // more than single parameter

() =>

{

statements

}

param =>

{

statements

}

(param1, paramN) =>

{

statements

}

/// Traditional function

function check(a,b)

{

  return a+b+200;

}

console.log(check(222,333));

// above function replacing with => Arrow

let check1=(a,b)=> a+b+200;

console.log(check1(222,333));

Different ways of using Arrow function

// Traditional anonymous function

(function (a, b) {

  const chuck = 42;

  return a + b + chuck;

});

// Arrow function

let result=(a,b) =>

  {

  const chuck = 42;

  return a + b + chuck;

};

console.log(result(100,200));

/// Traditional function

function check(a,b)

{

  return a+b+200;

}

console.log(check(222,333));

// above function replacing with => Arrow

let check1=(a,b)=> a+b+200;

console.log(check1(222,333));

let sayHi=(a,b) =>

  {

    console.log("Hello! Girls");

    return a+b;

  }

console.log(sayHi(67,89));

**Function in Javascript**

1. Function message();
2. Function message(a,b);
3. Function message(a,b)

{

Return a+b;

}

1. let userName = 'Ashwin'; // outer variable

function showMessage1()

using the outer variable in the function or modifying

1. let userName = 'Ashwin';  
   function showMessage1()

{

    userName='Uday'; //Changed the value from outer variable

}

1. Let func=()=>expression => arrow function
2. Function as Objects

let x = **function** someFunction(y)

{

document.write("Function called with value: " + y);

}

// we can call the function using the variable now

x(10);

Account acc=new Account();

Account acc=new Account(Name=’Dell Laptops’, AnnualAmount=202020);

Insert acc;

Account acc=new Account();

acc.Name=’Chennai City’;

acc.AnnualAmount=03939393;

insert acc;

Name ChennaiCity

Key Value

In JS.

Const user=new user(); // object with constructor

Let movies={……} // object literal syntax

Key Value pair

Let movies ={

Name:”Puspa”,

Actor:”AlluArjun”,

Price:892

};

ECMA 6+

1. var vs let
2. Functions
3. Arrow Functions
4. Objects
5. Constructor Functions
6. ES6 Classes
7. Callback Functions
8. Promise
9. Async/Await
10. Modules
11. **Import/Export Modules**
12. Destructuring
13. Array Methods
14. Spread Operator
15. JSON

Constructor Function

**What is a Constructor in JavaScript?**

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the ***new*** keyword.

The purpose of a constructor is to create a new object and set values for any existing object properties.

The way to create an "object type", is to use an object constructor function.

Objects of the same type are created by calling the constructor function with the new keyword: What Happens

**When A Constructor Gets Called?**

When a constructor gets invoked in JavaScript, the following sequence of operations take place:

1. A new empty object gets created.
2. The ***this*** keyword begins to refer to the new object and it becomes the current instance object.
3. The new object is then returned as the return value of the constructor. In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.
4. It is considered good practice to name constructor functions with an upper-case first letter.

**Example 8: JavaScript Built-In Constructors**

// use Object() constructor to create object

const person = new Object({ name: "John", age: 30 });

// use String() constructor to create string object

const name = new String ("John");

// use Number() constructor to create number object

const number = new Number (57);

// use Boolean() constructor to create boolean object

const count = new Boolean(true);

console.log(person);

console.log(name);

console.log(number);

console.log(count);

**Aura Component Framework**

Introduce and brief about Aura and will teach few examples… or scenarios…

Once that is done…

LWC – Lightning Web Components

Lightning Component Framework ------ also called as Aura

Component driven model

**Why Salesforce Introduced Aura Framework ?**

At the beginning of the Salesforce Development times, before Aura Framework, developers used Visual Force, which is an HTML Tag-based mark-up language to develop their Visual Force web pages and Apex to control the internal logic.

But this HTML based Visual Force standards were not compatible to build large scale enterprise solutions and complex applications. So, developers needed something that could support the ongoing huge demand for web-based applications with complicated business logic.

**In 2014,** Salesforce launched the Lightning Component Framework and it changed the Salesforce Programming way forever. Lightning Component Framework brought Aura Framework which used a component-driven model that was brilliant in developing large-scale enterprise applications.

Since the time the web technology is being changed every day, seeing an unprecedented change in JavaScript Frameworks, redefined way of creation of a web page, and more. Lightning Component also has to become Lightning web components to accommodate the newest technologies in Salesforce Development.

Built UI developing single page for mobile devices

Devices

Mobile

Laptop

Tab

Desktop

Created a component called

Combo box..

Reuse component – in any application on any webpage

Validates – Some process…

App Builder

Home Page

Record Page

App Page

Experience Builder

As of Spring ’19 (API version 45.0), you can build Lightning components using two programming models: the ***Lightning Web Components model***, and the original ***Aura Components model***. Lightning web components are custom HTML elements built using ***HTML and modern JavaScrip***t. Lightning web components and Aura components can coexist and interoperate on a page.

Configure Lightning web components and Aura components to work in Lightning App Builder and Experience Builder. Admins and end users don’t know which programming model was used to develop the components. To them, they’re simply Lightning component

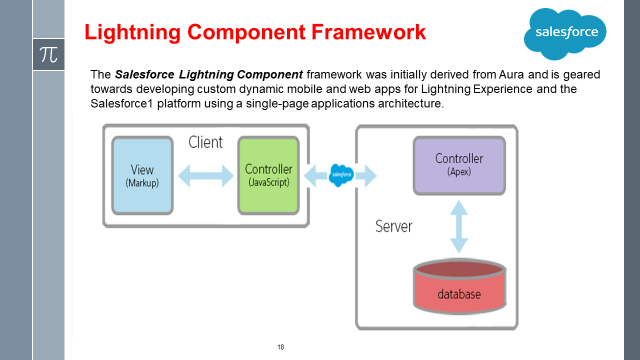
LWC are lighting custom HTML elements…

How do we create these components…

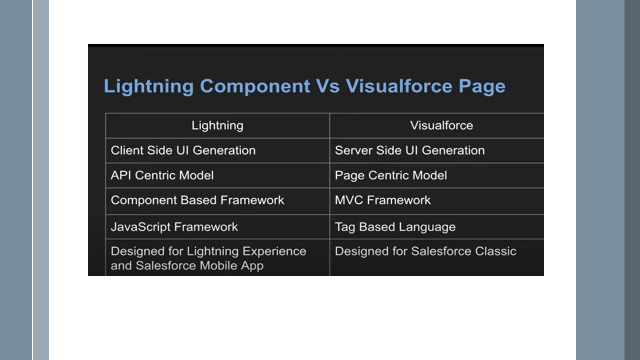
Lightning components

UI - HTML / Custom HTML

Logic - Javascript at client side / Apex at server side



Server- Salesforce Database



ChenFirstComp

This component bundle is embedded with 8 files / Resources

1. Componet .cmp

UI use HTML components or Tags and also you can use or Add Aura components tags..

build a User interface by using component as well collection of Lightning input components to collect user input. Once the components are created, there are several

<aura:component>

What VFPage - <apex:page>

1. Controller (js)- Contains the javascript code for your component to call the action events declared in the lightning component. It calls the doInit method that runs before your component Html content is loaded, doInit is not required in all components. This file contains the client-side JavaScript controller methods to handle events fired and handled by the components.
2. Style…

U want to add some coloring or graphics or texture…

CSS – Cascading Style Sheets

1. Helper - is also a kind of controller… which is the extension of existing Controller (Js)

I want to execute this component.. I need to have app.. I will add this component to an App..

Scenario 2: Create an Aura component that will displays a list of items

We have retrieved the source code from Org…

Did some changes and deploy the code..

You need to deploy this code to your org

Two Aura Components that we created…

<c:ChenFirstComp/>

<c:ChenListComp/>

Component Attributes

First, we need to learn how to enable a component to accept input when it’s created. That is, we need to set values on the component. We do this using ***attributes***.

The second thing we need to learn is how to actually *use* these values to change a component’s behavior and output.

Attributes on components are like instance variables in objects. They’re a way to save values that change, and a way to name those value placeholders.

<body bgcolor=’Green’ > attributes of Body Tag element

<font size=20 color=’red’> attributes of font Element

<aura:attribute>

Attributes

Name

Type

Message

Default

Required

Description

Access

Scenario 3: Create an aura component which has all <aura:attribute>component attributes

{!Account.Name}

{!$User.FirstName}

! --- expression tag

$ - Global variable

User- User object

First Name… you are fetching the first name value of an user….

{!v.message}

Scenario 4: Create an Lightning Component called CalculatorComp..

Define 2 attributes and perform arithmetic operations….

**Scenario 5:** Create a lightning component (Aura) where user can input values and can perform Arithmetic operators and the calculations should happen once the user clicks on Arithmetic Operators Buttons…

Write even handlers in the Client side controller (Javascript) where your logic is performed…..

**Scenario 6:** Please add an calculator image to the above scenario component..

**Scenario 7:** **Create LWC with various Buttons like Neutral,Brand, Destructive, Destructive text,success and also write an event handler while you click on each button to display a Name of the button**

**Scenario 8:** **Create a LightningCard as LWC..Create a title, footer and create button called new**

<template>

  <!--  <div class="foo">

    <h1>Added Title, Footer and Button to Lightning-card</h1>

    </div>-->

<lightning-card variant="narrow" title="Header Footer" icon-name="utility:contract\_payment">

<h1 slot="Title">Hi Welcome to the Component</h1>

<p class="slds-var-p-horizontal\_small">

    This is the body.. you can write your content here

</p>

<div slot="actions">

    <lightning-button label="Click Here" slot="actions"></lightning-button>

</div>

<!-- slot attribute is passed to the lightning-button to align with the lightning-card.-->

<!--Slots are placeholders where a parent component can insert content directly into a child component.-->

<div slot="footer">

    <div class="cont">

        <p>Contact us  : @Virukshatech   @@ Chennai T-Nagar, Nugumbakam</p>

    </div>

</div>

</lightning-card>

</template>

**Scenario 9:**

**Create LWC Check box and it should enable you to select only one or two option. Use handlechange event handler to capture what you have selected**

<https://www.lightningdesignsystem.com/utilities/text/#Color-Weak>

<https://developer.salesforce.com/docs/component-library/bundle/lightning-badge/example>

<https://developer.salesforce.com/docs/component-library/bundle/lightning-card/example>

Iterators

**Scenario 10:**

**Create LWC where you can display the list of the contact list with ID, Name and Title Fields with for:each directive, use for:item="currentItem"**

**renderList.HTML**

**Scenario 11:**

Scenario11: Create LWC where you can display the list of the Movies list with Movie Name,   Movie Hero, and VillanFields with for:eachdirective, use for:item="currentItem" …. Add   appropriate CSS wherever it is required

  renderList.HTML

**Scenario 12: Implement lwc If:else**

<template>

    <lightning-card title="HelloConditionalRendering with LWC:if" icon-name="custom:custom14">

        <div class="slds-m-around\_medium">

          <lightning-input

            type="checkbox"

            label="Show details"

            onchange={handleChange}

          ></lightning-input>

          <template lwc:if={areDetailsVisible}>

            <div class="slds-m-vertical\_medium">These are the details!</div>

          </template>

        </div>

      </lightning-card>

</template>

import { LightningElement } from 'lwc';

export default class Sixrenderifelse extends LightningElement {

    areDetailsVisible = true;

  handleChange(event) {

    this.areDetailsVisible = event.target.checked;

  }

}

**Scenario 13:**

**Implement lwc:else if whenever the time changes the DOM templates will render based on the current system time.**

<template>

    <lightning-card title="Greeting Based on Time of Day with multiple lwc:elseif">

        <div class="slds-p-around\_medium">

            <template lwc:if={isMorning}>

                <h1>Good morning! Rise and shine.</h1>

            </template>

            <template lwc:elseif={isAfternoon}>

                <h1>Good afternoon! Hope you're having a great day.</h1>

            </template>

            <template lwc:elseif={isEvening}>

                <h1>Good evening! Relax and enjoy your evening.</h1>

            </template>

            <template lwc:else>

                <h1>Hello! Welcome anytime.</h1>

            </template>

        </div>

    </lightning-card>

</template>

**JS**

import { LightningElement } from 'lwc';

export default class Sixrenderifelse2 extends LightningElement {

    isMorning = false;

    isAfternoon = false;

    isEvening = false;

    isDefault = true;

    connectedCallback() {

        this.calculateTimeOfDay();

    }

//The connectedCallback() method is one of our lifecycle hooks.

// know that the method is triggered when a component is inserted in the document object model (DOM).

// In this case, it starts the timer.

    calculateTimeOfDay() {

        const currentHour = new Date().getHours();

        if (currentHour >= 6 && currentHour <=9) {

            this.isMorning = true;

        } else if (currentHour >=10 && currentHour < 13) {

            this.isAfternoon = true;

        } else if (currentHour >= 13 && currentHour < 24) {

            this.isEvening = true;

        }

    }

**Scenario 14:**

### How to get current user id in lightning web component | Access logged in user ID in LWC

//https://developer.salesforce.com/docs/platform/lwc/guide/reference-salesforce-modules.html#salesforceuser

{!$User.FirstName} VFPage

{!v.name} Aura

{name} LWC

Onclick() ={!c.getadd} --- Aura c is a component getadd is event handler name

Getadd (component,event,helper) //// handling that event…

{

/////

}

Onclick={handlechange}

Handlechange(event)

{

/////////////

}

**Use Getters Instead of Expressions**

To compute a value for a property, use a JavaScript getter. For example, to convert the name to all uppercase letters, use a getter function in the JavaScript class, not an expression in the template.

Getters are much more powerful than expressions because they’re JavaScript functions. Getters also enable unit testing, which reduces bugs and increases fun.

Define a getter that computes the value in your JavaScript class.

Access the getter from the template.

get propertyName() { ... }

Access the getter from the template.

{propertyName}

In this example, a user enters their first and last name. A JavaScript getter computes a new value, uppercasedFullName, and the template renders it: DEANNA LI.

In .js file

Componenet-1 Navallur

// public properties

// decorated with @api

@api Name = ‘A2B Bhavan’;

@api City = ‘Navallur’;

Component-2 Chennai

Name = ‘Chennai Central Railway’;

City= ‘Chennai’;

Private Reactive Property

Component -1

@track DOB =11-Jan-2023;

@track Age=23;

Component-1

DOB=06-Jan-2024

Age=24;

apiChild

Js

@api ProductName = ‘Laptops’;

@track City =’Hyderabad’

apiParent

{fruits} // attribute

apiParent.js

<c-api-child product-name

<c-apichild product-name=

@wire(getAccountList)accounts

@wire(getAccountList)wiredAccounts({data,error})

LWC – Communication between components…

Parent to Child

Child to Parent

Sixparenttochild Child Component

This component is having

@api changeBarColor()