Sandbox Types / Developer Edition Usage

Developer Sandbox - Developer sandboxes are intended for coding and testing in an isolated environment. These environments include a copy of your production organization's configuration (metadata). Best for app development, unit testing (not integration or UAT)

Developer Pro Sandbox - (Formerly Config Only) Developer Pro sandboxes are intended for coding and testing in an isolated environment. These environments include a copy of your production organization's configuration (metadata). They have a larger storage limit than Developer sandboxes. The larger limit allows for more robust test data sets and enables this environment to handle more development and quality assurance tasks. Best for feature testing.

Partial Copy Sandbox - Partial Copy sandboxes are intended to be used as testing environments. These environments can be used for quality assurance tasks such as user acceptance testing, integration testing, and training. These environments include a copy of your production organization's configuration (metadata), and a subset of your production data as defined by a sandbox template. Best for UAT, external integration

Full Sandbox - Full sandboxes are intended to be used as testing environments. Only Full sandboxes support performance testing, load testing, and staging. These environments are a replica of your production organization, including all data—for example, object records and attachments—and metadata. The length of the refresh interval makes it difficult to use Full sandboxes for development. Best for production debugging.

Developer Edition - Use if 1) You have pro, group, or personal edition 2) Your app does not depend on data from any org 3) You have used all you SBs 4) ISV coding for AppExchange

Should I use IDE (Metadata)...	... or Migration (Package)
Has access to both orgs (source and destination)	1. Can target multiple orgs 2. Separate publication from installation 3. Dev projects needed test env with lots of data 4. Multistage release 5. Repetitive deployment with same parameters

Apex Language Notes

1. Case Insensitive, Strongly Typed
2. Classes can't be static, only outer classes can have static methods (static methods don't need initialized to run)
3. Data types: Primitive, Colleciton & Null (Non-standard: AnyType (used in .ValueOf()), Currency (Used in SOQL & SOSL to filter WHERE fields))
4. To define a constant, mark variable as static & final.
5. A class can implement multiple interfaces, but can only extend one.
6. Abstract classes cannot construct Apex Objects
7. Unit tests take no arguments, return void
8. Anonymous blocks: Use full permission of current user (respect field CRUD), cannot include static keyword, API .executeAnonymous('Apex Code');
9. Standard Controller run in user mode, even if VF page extended. Use "with sahring" on extension to maintain sharing
10. No DML in getter methods
11. It is possible to have more than one catch block for each try, one catch for each particular exception, with the general exception case last
12. Deploying to prod, every unit test in namespace runs, No trigger 0%, Class can have 0% if org average > 75%

13. “Transient” Keyword: Allows for one Controller to maintain state across pages, vars not stored in heap or view state
14. Heap Size Limits: Synchronous = 6mb, Asynchronous = 12mb, Email Service = 36mb, use SOQL for loops to process batches of 200

Triggers - Cannot be static, can call static or instance methods, Deleting a child record by deleting its master record will not fire the child object triggers

1. Loads the original record from the database or initializes the record for an upsert statement.
2. Loads the new record field values from the request and overwrites the old values.
   If the request came from a standard UI edit page, Salesforce runs system validation to check the record for:
   - Compliance with layout-specific rules
   - Required values at the layout level and field-definition level
   - Valid field formats
   - Maximum field length

   Salesforce doesn't perform system validation in this step when the request comes from other sources, such as an Apex application or a SOAP API call.
   Salesforce runs user-defined validation rules if multiline items were created, such as quote line items and opportunity line items.
3. Executes all before triggers.
4. Runs most system validation steps again, such as verifying that all required fields have a non-null value, and runs any user-defined validation rules. The only system validation that Salesforce doesn't run a second time (when the request comes from a standard UI edit page) is the enforcement of layout-specific rules.
5. Executes duplicate rules. If the duplicate rule identifies the record as a duplicate and uses the block action, the record is not saved and no further steps, such as after triggers and workflow rules, are taken.
6. Saves the record to the database, but doesn't commit yet.
7. Executes all after triggers.
8. Executes assignment rules.
9. Executes auto-response rules.
10. Executes workflow rules.
11. If there are workflow field updates, updates the record again.
12. If workflow field updates introduced new duplicate field values, executes duplicate rules again.
13. If the record was updated with workflow field updates, fires before update triggers and after update triggers one more time (and only one more time), in addition to standard validations. Custom validation rules are not run again.
14. Executes processes, If there are workflow flow triggers, executes the flows.
    Flow trigger workflow actions, formerly available in a pilot program, have been superseded by the Process Builder. Organizations that are using flow trigger workflow actions may continue to create and edit them, but flow trigger workflow actions aren’t available for new organizations. For information on enabling the Process Builder (beta) in your organization, contact Salesforce.
15. Executes escalation rules.
16. Executes entitlement rules.
17. If the record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the parent record. Parent record goes through save procedure.

18. If the parent record is updated, and a grandparent record contains a roll-up summary field or is part of a cross-object workflow, performs calculations and updates the roll-up summary field in the grandparent record. Grandparent record goes through save procedure.
19. Executes Criteria Based Sharing evaluation.
20. Commits all DML operations to the database.
21. Executes post-commit logic, such as sending email.

Apex Email

1. VF Email Template cannot be used for Mass Email
2. Methods implementing Messaging.InboundEmailHandler must be global (and therefore in a global class)
3. Messaging.sendEmail can only be called 10 times / transaction (within the daily liimit).
4. Email Services: Maximum Size of Email Message (Body and Attachments) 10 MB
5. On-Demand Email-to-Case: Maximum Email Attachment Size 25 MB
6. You can send mass email to a maximum of 1,000 external email addresses per day per, You can send an unlimited amount of email to your organization's internal users, which includes portal users.
7. Email Service & On Demand Email Processing: Number of user licenses multiplied by 1,000, up to a daily maximum of 1,000,000
8. The daily limit for emails sent through email alerts is 1,000 per standard Salesforce license per organization—except for free Developer Edition and trial organizations, where the daily workflow email limit is 15 per standard Salesforce license. The overall organization limit is 2,000,000. This limit applies to emails sent through email alerts in workflow rules, approval processes, flows, processes, or the REST API.

```
global class CreateTaskEmailExample implements Messaging.InboundEmailHandler {
  global Messaging.InboundEmailResult handleInboundEmail(Messaging.inboundEmail email,
                                                         Messaging.InboundEnvelope env){
    // Create an InboundEmailResult object for returning the result of the
    // Apex Email Service
    Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();
// If false, Salesforce rejects the inbound email and sends a reply email to the original sender containing the message specified in the Message field.
    result.success = true;
    // Return the result for the Apex Email Service
    return result;
  }
}
```

Web Services

1. Total Callouts / Transaction = 10, The size limit of a SOAP request or response is 3MB, HTTP Timeout max 120 seconds
2. Class and exposed methods, properties, and inner classes must be global to be used in WSDL. Must be static. Methods cannot be

overloaded.
3. Method signatures cannot include: Exceptions, Maps, Sets, Pattern Objects, Matcher Objects (no SOAP equivilants)
4. Runs as system by default, never uses object visibility, respects “with sharing” but may call class “without sharing” (reference schema for CRUD)
5. Send HTTP Requests through GET, POST & PUT, Request & Response count towards heap size

Asynchronous Apex (@future) Methods

1. Triggers cannot contain “webservice”, can execute a callout when invoked with @future method, Trigger does not wait for response
2. Use “@future (Callout=true)” to signify method will be making a callout (method can be public)
3. Methods marked @future must be static and can only return void AND @future starts new governor limit context
4. 50 @future methods / transaction, total in 24 hours = 250,000 or 200 * licenses (whichever is greater)
5. Parameters must be : primitives or arrays / collections of primitives
6. Cannot: be used in GET or SET or VF Controller Constructor Cannot call another @future method

Web Service Controller Example

```
global class AccountPlan {

  webservice String area;
  webservice String region;

  //Define an object in apex that is exposed in apex web service
  global class Plan {
    webservice String name;
    webservice Integer planNumber;
    webservice Date planningPeriod;
    webservice Id planId;
  }

  webservice static Plan createAccountPlan(Plan vPlan) {

     //A plan maps to the Account object in salesforce.com.
     //So need to map the Plan class object to Account standard object
     Account acct = new Account();
     acct.Name = vPlan.name;
     acct.AccountNumber = String.valueOf(vPlan.planNumber);
     insert acct;
```

```
    vPlan.planId=acct.Id;
    return vPlan;
  }
}
```

Dynamic DML

```
      //make a sObject b
      SObject b = Schema.getGlobalDescribe().get('Account').newSObject();
      //case it as Account
      Account a = (Account)b.getSobjectType().newSObject();
      a.Name = 'test';
      insert a;
      //get an sObject from Account a
      sObject sObjectA = a.getSobjectType().newSObject();
      //get the DescribeSObjectResult from sObject b's sObjectType
      Schema.DescribeSObjectResult sObjResult = b.getSobjectType().getDescribe();
      //get list of DescribeSObjectResult from Schema.describeSObjects(String List)
      Schema.DescribeSObjectResult[] sObjResults = Schema.describeSObjects(new String[]{sObjResult.Name});
      //cast Account c as a blank copy of Account a from Schema using first item in DescribeSObjectResult[]'s
list and the ID of a
      Account c = (Account) Schema.getGlobalDescribe().get(sObjResults[0].Name).newSObject(a.ID);
      //assert Account c is a blank copy of Account a
      system.AssertEquals(a.Id,c.Id);
      //clean up
      delete a;
```

Action Tags

apex:actionFunction - Component that provides support for invoking controller action methods directly from JavaScript code using an AJAX request. An <apex:actionFunction> component must be a child of an <apex:form> component. Can't be placed inside an iteration component.
Unlike <apex:actionSupport>, which only provides support for invoking controller action methods from other Visualforce components, <apex:actionFunction> defines a new JavaScript function which can then be called from within a block of JavaScript code.

apex:actionPoller - Timer that sends an AJAX request to the server according to a time interval that you specify. Each request can result in a full or partial page update.
An <apex:actionPoller> must be within the region it acts upon. For example, to use an <apex:actionPoller> with an <apex:actionRegion>, the <apex:actionPoller> must be within the <apex:actionRegion>.
If an <apex:actionPoller> is ever re-rendered as the result of another action, it resets itself.

apex:actionSupport - Component that adds AJAX support to another component, allowing the component to be refreshed asynchronously by the server when a particular event occurs, such as a button click or mouseover.

apex:actionRegion - An area of a Visualforce page that demarcates which components should be processed by the Force.com server when an AJAX request is generated. Only the components in the body of the <apex:actionRegion> are processed by the server, thereby increasing the performance of the page.
Note that an <apex:actionRegion> component only defines which components the server processes during a request—it does not define what area(s) of the page are re-rendered when the request completes.

VisualForce Component

Component named `recordDisplay`

```
<apex:component>
   <apex:attribute name="record" description="The type of record."
                   type="Object" required="true"/>

   <apex:pageBlock title="Viewing {!record}">
      <apex:detail />
   </apex:pageBlock>
</apex:component>
```

Next, create a page called `displayRecords` *and use the following code:*

```
<apex:page >
  <c:recordDisplay record="Account" />
</apex:page>
```

VisualForce Component Attributes
1. Types: Primitives, sObjects (including generic), One-dimentials lists with array notation (ex: String[]),"Map' (no type declarartion), Apex Classes

Shema			
Return	**=**	**Next Step**	**Do It In One Line**
// generate a Map of all sObject names (keys) to sObject tokens (values)			
Map<String,Schema.SObjectType> schemaMap	=	Schema.getGlobalDescribe();	Schema.getGlobalDescribe();
String objName = 'Account'; // get the Schema.DescribeSObjectResult using objName string			
Schema.DescribeSObjectResult aDSR	=	schemaMap.get(objName).getDescribe();	Schema.getGlobalDescribe().get(objName).getDescribe();
//generate Map of all field names (keys) to field tokens (values)			
Map<String,Schma.SObjectField> fieldMap	=	aDSR.fields.getMap();	Schema.getGlobalDescribe().get(objName).getDescribe().fields.getMap();
String fieldName = 'Industry'; // get the Schema.DescribeFieldResult of the fieldName string			
Schema.DesribeFieldResult iDFR	=	fieldMap.get(fieldName).getDescribe();	Schema.getGlobalDescribe().get(objName).getDescribe().fields.getMap().get(fieldName).getDescribe();
// get the Schema.PickListEntry list from the DescribeFieldResult iDFR (A runtime error is returned if the field is not a picklist.)			
List<Schema.PickListEntry> picks	=	iDFR.getPicklistValues();	Schema.getGlobalDescribe().get(objName).getDescribe().fields.getMap().get(fieldName).getDescribe().getPicklistValues();

```
for(Schema.PickListEntry pick : Schema.getGlobalDescribe().get(objName).getDescribe().fields.getMap().get(fieldName).getDescribe().getPicklistValues())
    if(pick.isDefaultValue())
        system.debug('Default value is: '+pick.getValue() + ' and the default label is: '+ pick.getLabel());
```

VisualForce Templates

apex:composition - An area of a page that includes content from a second template page. Template pages are Visualforce pages that include one or more <apex:insert> components. The <apex:composition> component names the associated template, and provides body for the template's <apex:insert>components with matching <apex:define> components. Any content outside of an <apex:composition> component is not rendered.

apex:insert - A template component that declares a named area that must be defined by an <apex:define> component in another Visualforce page. Use this component with the <apex:composition> and <apex:define> components to share data between multiple pages.

apex:define - A template component that provides content for an <apex:insert> component defined in a Visualforce template page.

SOSL

1. Searches field values across objects
2. Cannot be used in a trigger
3. Cannot search picklist values
4. Can be used inline (assignments, for loops)
5. Run dynamic: List<List<sObject>>> myQuery = search.query('SOSL_Search_String');
6. String search_string = 'FIND \'Joh*'\' IN EMAIL FIELDS RETURNING Account (id, name LIMIT 1), Contact LIMIT 2';
7. FIND{ MyProspect AND NOT ("Califor?ia" OR MyCompany)} - use { } in API but use '' in Apex
8. IN ... ALL, EMAIL, NAME, PHONE, SIDEBAR … FIELDS
9. WHERE (Includes / Excludes is only for multiselect picklists): RETURNING ACCOUNT(id WHERE BillingState IN ('California','NY''))