Salesforce Fundamentals: 7%

- Describe the considerations when developing in a multi-tenant environment.
- Understand design frameworks, such as MVC architecture and Lightning Component framework, and how to build applications using both declarative and programmatic tools.
- Given a scenario, identify common use cases for declarative versus programmatic customizations.

Data Modeling and Management: 13%

- Given a set of requirements, determine, create, and access the appropriate data model including objects, fields, and relationships.
- Describe the capabilities of the various relationship types and custom IDs and the implications of each on record access and development.
- Describe the options for and considerations when importing and exporting data into development environments.
- Describe the capabilities and use cases for formula fields and roll-up summary fields.

Process Automation and Logic: 38%

- Describe the capabilities of the declarative process automation features.
- Declare variables, constants, methods, and use modifiers and interfaces in Apex.
- Given a scenario, use and apply Apex control flow statements.
- Given a scenario, write Apex classes and use Apex interfaces.
- Given a scenario, write SOSL, SOQL, and DML statements in Apex.
- Given a use case, write Apex classes and triggers following best practices.
- Given a scenario, identify the implications of governor limits on Apex transactions.
- Describe the relationship between Apex transactions, the save order of execution, and the potential for recursion and/or cascading.
- Implement exception handling in Apex, including custom exceptions as needed.
- Use programmatic techniques to prevent security vulnerabilities.
- Given a scenario, use declarative functionality and Apex together to automate business logic.
- Given a scenario, identify the appropriate publish/subscribe logic for platform events.

Testing, Debugging, and Deployment: 17%

- Write and execute tests for triggers, controllers, classes, flows, and processes using various sources of test data.
- Describe the use cases for invoking anonymous code and the differences between invoking Apex in execute anonymous vs. unit tests.
- Describe the Salesforce Developer tools such as Salesforce DX, Salesforce CLI, and Developer Console, and when to use them.
- Describe how to approach debugging system issues and monitoring flows, processes, and asynchronous and batch jobs, etc.
- Describe the environments, requirements, and process for deploying code and associated configurations.

User Interface: 25%

- Given a scenario, display or modify Salesforce data using a Visualforce page and the appropriate controllers or extensions as needed.
- Describe the types of web content that can be incorporated into Visualforce pages.
- Incorporate Visualforce pages into Lightning Platform applications.
- Describe the Lightning Component framework and its benefits.
- Describe the types of content that can be contained in a Lightning web component.
- Given a scenario, prevent user interface and data access security vulnerabilities.
- Given a scenario, display and use a custom user interface components, including Lightning Components, Visual Flow, Next Best Actions, and Visualforce.
- Describe the use cases for Lightning component events and application events.

Section 1 - Salesforce Fundamentals - 7%

- Describe the considerations when developing in a multi-tenant environment.
- Understand design frameworks, such as MVC architecture and Lightning Component framework, and how to build applications using both declarative and programmatic tools.
- Given a scenario, identify common use cases for declarative versus programmatic customizations.

Describe the considerations when developing in a multi-tenant environment.

- **Tenant** - group of users/software apps that share hardware through an underlying piece of software
- Multiple Tenants on a server, share access to system resources
  - Memory, CPU, Network Controller, etc...
  - Dynamically allocated
- Single Tenant on a server, does not have another app to share resources with

Single Tenant Pros and Cons

- **Advantages** of Single-Tenant Hosting
  - More control because there is no sharing of resources.
  - Greater customization
  - Avoiding "noisy neighbor" Syndrome
- **Disadvantages** of Single-Tenant Hosting
  - No cost sharing between apps
  - Less efficient - don't run at full capacity

Multi-Tenant Pros and Cons

- **Pros**:
  - Shared cost between apps
  - Simplified hosting - hardware managed by Salesforce
- **Cons**:
  - Outage impacts all apps on that suite of hardware
    - Mitigated by redundancy by SFDC
  - Noisy neighbors: Other apps may dominate use of system resources
    - Mitigated by Governor Limits in SFDC

What this means for Salesforce Developers:

- No setup needed for: Network, OS, Databases, Servers, or any other hardware
- No worrying about: Missing JAR files, installing RedHat, wondering which load balancer to use
- Instead:
  - Design the data model,
  - Implement the business logic,
  - Test, Test, Test
  - and Deploy

Terminology

Organization/Environment - unique SFDC data and metadata

- Data, fields, code, processes, reports etc.

Instance - Server your org resides on.

App - Collection of interrelated functionality. Metadata in your org acting as one unit

Org vs Instance

So in other words:

- SFDC Org = House
- SFDC Instance = Neighborhood

We get a base model house from the Salesforce, customize them for our own needs (Architecture styles, Interior Designs) but they're all on plotted in the same neighborhood.

City of Salesforce builds our roads, sewage pipes, waterlines, network cables, electric cables

If the neighborhood gets hit by a Tornado all of the homeowners in that neighborhood are affected.

People in surrounding neighborhoods are untouched

Part 2
Understand design frameworks, such as MVC architecture and Lightning Component framework, and how to build applications using both declarative and programmatic tools.

MVC - A fundamental software design pattern that divides the related program logic into three interconnected elements and defines their relationships.

Model

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. Directly manages the data, logic and rules of the application.

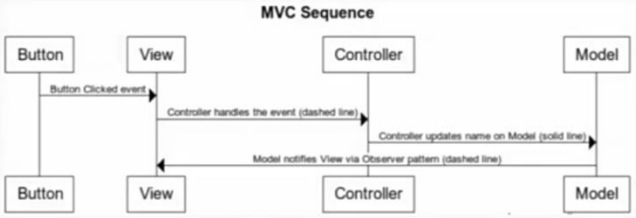
View

The representation of information.

Controller

Converts user input to commands for the model or view

MVC Sequence Diagram



MVC Benefits

Simultaneous Development – Multiple developers can work simultaneously

High Cohesion – Logical grouping of related actions on a controller together. The views for a specific model are also grouped together.

Loose Coupling – The very nature of the MVC framework is such that there is low coupling among models, views or controllers

Ease of Modification – Separation of responsibilities leads to simpler future development or modification

Multiple Views - Present different views for different users off the same model

Salesforce Lightning

- Easier to build responsive applications for any device.
- Lightning Component Framework
  - UI framework for developing SPAs for mobile and desktop devices.
- Two programming models:
  - Lightning Web Components
  - Aura Components

Lightning Web Components

Why switch from Aura?

- Easier to Learn - no Aura Framework abstraction
- Better performance
- More focus on web standards
- More transferable skills from other web app frameworks

Salesforce Lightning

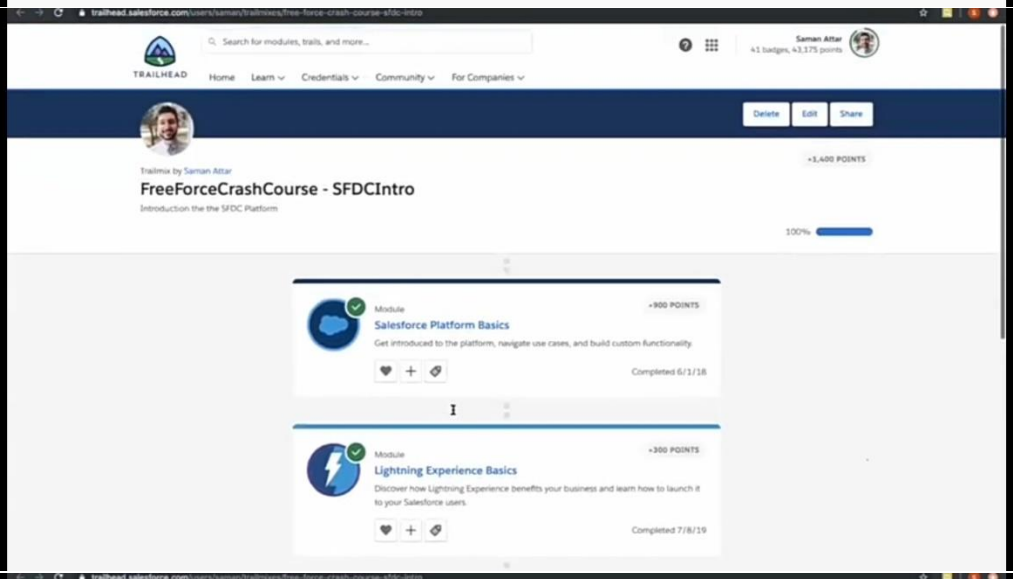
SPA - Single Page Application

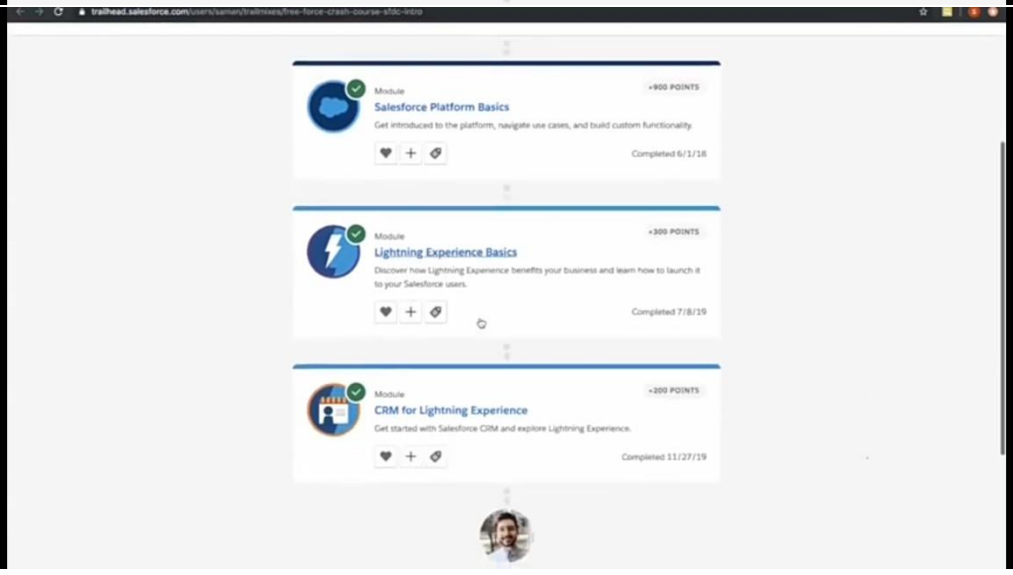
- Web Application that dynamically loads parts of the webpage, instead of loading the entire thing.
- Initial page load brings in all required HTML, CSS and Javascript
- Page does not reload nor does it transfer control to another webpage
- Faster and more seamless transitions through the web application

Given a scenario, identify common use cases for declarative versus programmatic customizations.

As we go through the course, this will become more clear.

- Process Builder vs Workflow Rules vs Trigger
- Process Builders vs Formula Fields vs Triggers
- Validation Rule vs Trigger
- Rollup Summary Field vs Trigger
- Flow vs LWC/Aura
- OOB Salesforce vs App Exchange





Topics For Today's Salesforce Training

i. **Salesforce** → Need & Rise
ii. Features Of Salesforce
iii. Demo: Salesforce In Action

What Is Cloud Computing?

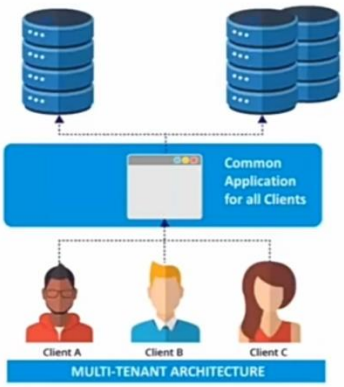
Its the use of remote servers to store, manage & process data, rather than on a local server/ personal computer

Working of Cloud Computing:

1. Software apps delivered on the cloud (vendor)
2. Software apps accessible via the cloud (client)

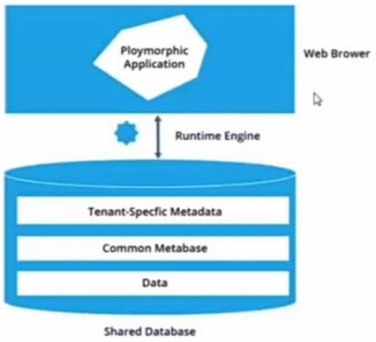
Benefits Of Cloud Computing & Salesforce

Multi-Tenant Architecture Of Salesforce



- One Server instance will be shared by multiple clients
- Economical; resources & maintenance cost is shared
- Vendor needs to update only one application, and changes will be reflected for all clients
- Single database stores data of multiple clients.
- Sharing rules enforced by Governor limits.

Meta-Data Architecture Of Salesforce

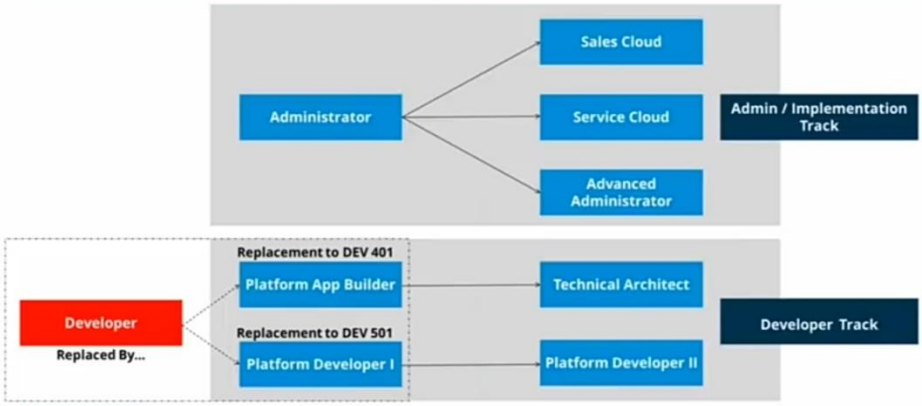


- Salesforce uses a metadata-driven development model.
- Functionalities of an app are defined as metadata in a database.
- Allows developers to concentrate only on building the application.
- Includes processes, assignment rules, sharing & security settings, Visualforce pages & Apex triggers.
- Stores page layouts for: Accounts, Contacts, Leads.

Products & Services Offered By Salesforce

Salesforce CRM Today

Salesforce Certification Roadmap



Course Objectives

As a candidate for the Platform Developer 1 credential, you need to be able to:

- Describe the structure of the exam and the exam's objectives.
- List the most heavily weighted exam objectives.
- Describe key features of the Lightning platform's programming languages, including: Apex, SOQL, SOSL, and Visualforce.
- Explain important platform programming concepts, such as the testing framework, governor limits, and the save order of execution.
- Explain the Application Lifecycle Management of a Salesforce development project.

Ryan Jackson
Lead Salesforce
Programmatic Developer

LIVE

Preparing For Your Platform Developer I Certification (CDW450)

LESSON 1:
Introduction to the Course and the Exam

LESSON 2:
Salesforce Fundamentals

LESSON 3:
Data Modeling and Management

LESSON 4:
Process Automation and Logic (Part 1)

LESSON 5:
Process Automation and Logic (Part 2)

LESSON 6:
User Interface

LESSON 7:
Testing, Debugging and Deployment

Who Should Take This Exam?

A programmer who has:

- One to two years of experience as a developer.
- At least six months of experience on the Salesforce platform.

Exam Objectives: A Visual Representation

CERTIFIED PROFESSIONAL

- Salesforce fundamentals (7%)
- Data modeling and management (13%)
- Process automation and logic (38%)
- User interface (25%)
- Testing, debugging and deployment (17%)

NOTE: The exam has 60 questions in total, and you need to get 65% correct to pass.

Exam Objectives

- Describe the considerations when developing in a multi-tenant environment.
- Understand design frameworks, such as MVC architecture and Lightning Component framework, and how to build applications using both declarative and programmatic tools.
- Given a scenario, identify common use cases for declarative versus programmatic customizations.

NOTE: The exam weighting for "Salesforce Fundamentals" is 7%.

Considerations of Multi-tenancy

DEFINITION: **Multi-tenancy** is the fundamental technology that clouds use to share IT resources cost-efficiently and securely.

Resources shared by Salesforce tenants include:

- The database, including data, metadata, indexes, field histories, and data relationships.
  - An org has a unique identifier that helps Salesforce identify which data belongs to that org.
- The run time engine.
  - To ensure that each tenant can access the run time engine's resources, each tenant is allocated a certain amount of resources. Exceeding these allocations causes exceptions.

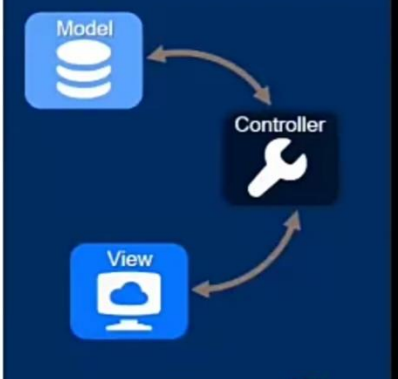
RESOURCE: Salesforce's Multitenant Architecture
https://developer.salesforce.com/page/Multi_Tenant_[illegible]

Describing the MVC Architecture

Model:
An entity representing data or activity.

View:
Visualization of the state of the model.

Controller:
A facility for changing the state of the model.



QUESTIONS:
1. Is a custom object part of the model, the view, or the controller?
2. Is a page layout a part of the model, the view, or the controller?

Choosing Among Customization Options

Configure Standard and custom solutions from the Setup menu	Buy Download apps and components from the AppExchange	Build Develop new solutions using programmatic tools
Declarative (Point & Click)	Package (Installed)	Programmatic (Code)
▪ No programming experience required. ▪ Integrated; updates automatically with Salesforce releases.	▪ Proven and reviewed solutions built for Salesforce. ▪ May include support and maintenance.	▪ Create apps that can do almost anything you want. ▪ Custom code builds on point and click functionality.
Simplicity & Speed	Already Built	Control & Flexibility
Salesforce Platform		

Knowledge Checkpoint: Declarative Process Automations

Salesforce provides declarative tools to automate your organization's repetitive business processes.

	Process Builder (Y/N)	Flow (Y/N)	Workflow (Y/N)	Approvals (Y/N)
Supports a time-based action	Y	Y	Y	N
Supports user-interaction	N	Y*	N	Y
Can call Apex code	Y	Y	N	N
Can delete a record	N	Y	N	N

RESOURCE:

Feel like you need a refresher on declarative automation? In a web brow use a search engine to search for the following terms: `Salesforce Which Automation Tool Do I Use?`

Lesson 3

Data Modeling and Management

Exam Objectives

- Given a set of requirements, determine, create, and access the appropriate data model including objects, fields, and relationships.
- Describe the capabilities of the various relationship types and custom IDs and the implications of each on record access and development.
- Describe the options for and considerations when importing and exporting data into development environments.
- Describe the capabilities and use cases for formula fields and roll-up summary fields.

NOTE: The exam weighting for "Data Modeling and Management" is 13

Lesson Agenda

- ▶ Data Modeling
- Data Management

A Review of Salesforce Objects

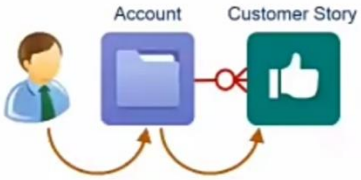
Objects on the Salesforce platform:

- Provide a predefined set of standard fields to capture common business information.
- Allow you to create custom fields to capture additional business information.
- Allow you to create custom relationships to link objects together.
- Allow you to create validation rules to verify that the data in one or more fields meets the specified criteria before the record is saved.
- Allow you to define page layouts and record types to control what a user sees when they view or edit a record.
- Allow you to automate business processes using workflow rules, processes, flows, and approval processes.
- Allow you to control record access and field-level security.

Relating Objects: a Master-Detail Relationship

27

- Sharing for the detail record is inherited from the master record.



- The detail record is automatically deleted when the parent is deleted.
- The parent reference is always required on the child record.

- You can add a lookup filter.
- You can choose whether or not the detail record can be reparented.

NOTE: The detail side of a master-detail relationship must be a custom object.

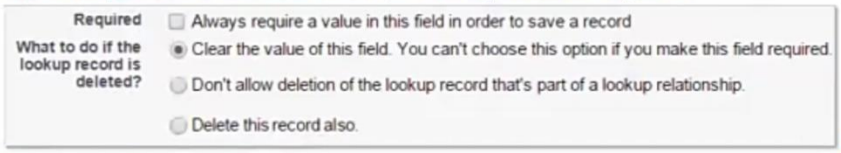
TRAILHEAD LIVE

Relating Objects: A Lookup Relationship

28

- The child record and parent record have independent sharing.

- The lookup field on the child record can be optional or required.

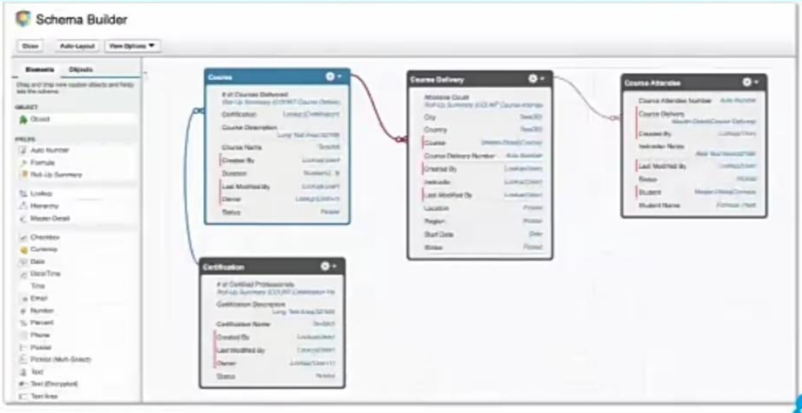


- You can add a lookup filter.

LIVE

How Can You Use Schema Builder to View Objects and Relationships?

29



CLICK PATH: Setup | Schema Builder

TRAILHEAD LIVE

Review: What Do You Remember about Data Loading Tools?

31

If you want to . . .	Data Import Wizard	API / Data Loader
Import fewer than 50,000 records.	Y	Y
Prevent duplicates when importing new records.	Y (see note)	Y (unique fields only)
Choose whether or not to trigger workflow rules.	Y	N
Load up to 5,000,000 records.	N	Y (API No limit)
Load objects such as products or opportunities.	N (see note)	Y
Schedule imports.	N	Y
Save mappings for later use.	N	Y
Export or delete data.	N	Y

Data Import Wizard: Account (Name and Site), Contact (Name or Email), Lead (Name or Email), Solution (Title), Campaign Member, Custom Objects (Name) and Salesforce ID and External-Unique Ids.

TRAILHEAD LIVE

Exam Objectives

33

- Describe the capabilities of the declarative process automation features.
- Declare variables, constants, methods, and use modifiers and interfaces in Apex.
- Given a scenario, use and apply Apex control flow statements.
- Given a scenario, write Apex classes and use Apex interfaces.
- Given a scenario, write SOSL, SOQL, and DML statements in Apex.
- Implement exception handling in Apex, including custom exceptions as needed.
- Use programmatic techniques to prevent security vulnerabilities.
- Given a scenario, use declarative functionality and Apex together to automate business logic.
- Given a scenario, identify the appropriate publish/subscribe logic for platform events.

NOTE: The exam weighting for "Process Automation and Logic" is 38%. These are half of the objectives.

TRAILHEAD LIVE

Lesson Agenda

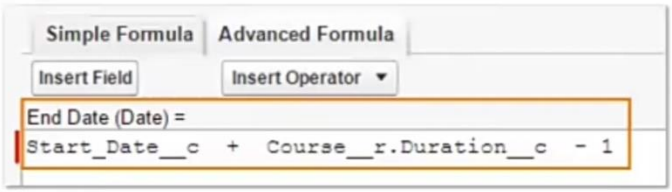
34

- Working with Formulas and Roll-up Summary Fields
- Working with Basic Constructs in Apex
- Working with SOQL
- Working with SOSL
- Working with DML
- Working with Exceptions and Governor Limits

LIVE

Working with a Formula Field

DEFINITION: A formula field is a field that derives its value from other fields, expressions, or values.



DEFINITION: You can create a cross-object formula on a child object to reference data from parent objects, up to 10 relationships away.

Considerations When Using Formula Fields

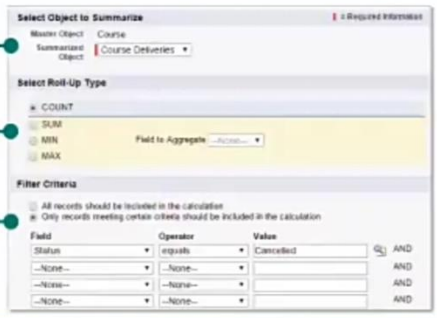
Formula Fields

- Formulas can reference other formulas
  - You may not create circular formulas
- Formulas have a maximum compile size
  - 3900 characters
- Formulas are calculated when the record is accessed
  - Value is not persisted in the database
  - Be careful when using in SOQL WHERE clauses or report filters
- Formulas are calculated in System Mode
  - User does not need access to the individual formula elements (objects or fields) to see the result

What is a Roll-Up Summary Field?

DEFINITION: A roll-up summary field is a field on a master record that summarizes date or numerical data from detail records.

- Select the detail object to summarize.
- Set the roll-up type to count, sum, min, or max.
- Determine which records to include in the calculation.



RESOURCE: For considerations and best practices, search for Roll-Up Summary Field in Help & Training.

Considerations When Using Roll-Up Summary Fields

Roll-up Summary Fields

Roll-Up summary fields are calculated and saved when the detail record is saved (assuming the value changes)
- Value is persisted in the database
- Will trigger any automation on the parent record

Roll-Up summary fields are calculated in System Mode
- User does not need access to the aggregated numerical or date fields

Lesson Agenda

Working with Formulas and Roll-up Summary Fields
▶ Working with Basic Constructs in Apex
Working with SOQL
Working with SOSL
Working with DML
Working with Exceptions and Governor Limits

What is Apex?

DEFINITION: **Apex** is Salesforce's cloud-based, object-oriented programming language, specifically designed for customizing and extending apps on the Salesforce platform.

- Tailored for data access and manipulation.
- Works in conjunction with declarative features.
- Has access to your org's metadata.
- Designed to work effectively and efficiently in a multi-tenant environment.

Apex and the Declarative Configuration

- The Salesforce platform tracks dependencies between sObjects used in Apex and declarative object definitions.
- Any metadata referred to in Apex cannot be changed or deleted.


```
1  public void updateLocation(List<Course_Delivery__c> deli
2      List<Course_Delivery__c> deliveriesToUpdate = new Li
3      for (Course_Delivery__c delivery: deliveriesToCheck)
4          if (delivery.location__c == 'Tokyo') {
5              delivery.region__c = 'JAPA';
6              deliveriesToUpdate.add(delivery);
7          }
8      }
9      update(deliveriesToUpdate);
10 }
```


Data Types: Primitives

Blob	Decimal	Long
Boolean	Double	String
Date	ID	Time
Datetime	Integer	

```
1  //Sample String method
2  Boolean validString = userString.isAlphanumeric();
3
4  //Sample Datetime method
5  Date curUserDate = curDateTime.date();
6
7  //Sample constant
8  static final DOUBLE PI = 3.14159;
```

Apex initializes all variables, regardless of type, to the special value `null`.

Constants are declared using the `static` + `final` modifiers.

NOTE: Although called primitives in Apex, these data types are similar to wrapper classes in Java. Variables declared to be of these data types are all objects.

Collections


```
1A  List<Account> accounts = new List<Account>();
2A  accounts.add(new Account(name='Account 1'));
3A  accounts.add(new Account(name='Account 2'));
4A  System.debug('First account is ' + accounts.get(0));
5A  System.debug('Second account is ' + accounts[1]);
```

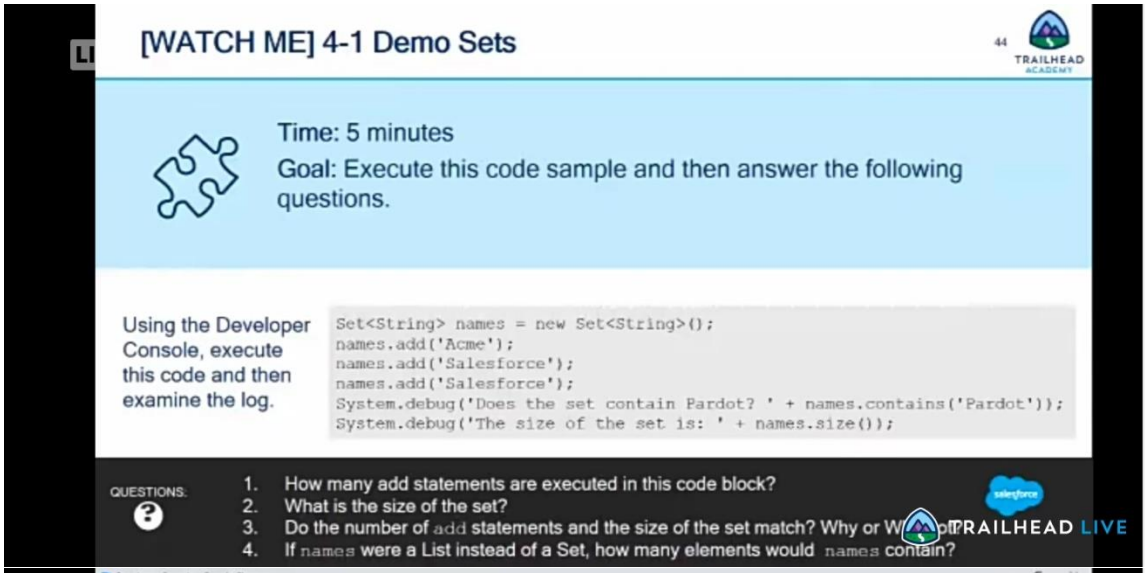
An ordered, indexed collection of elements.

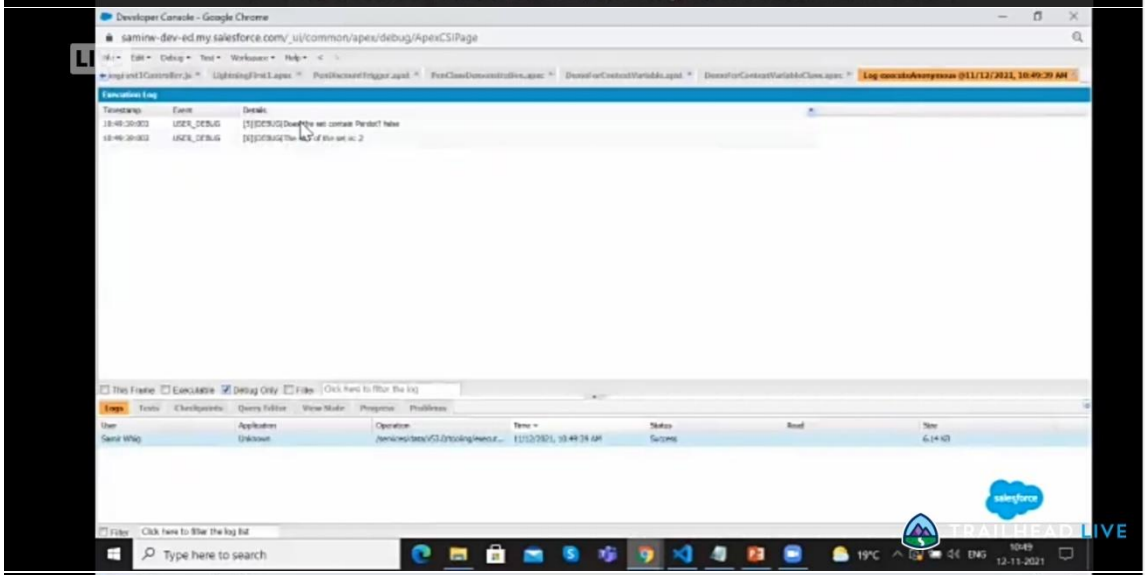
```
1B  Set<String> names = new Set<String>();
2B  names.add('Acme');
3B  names.add('Salesforce');
4B  names.add('Salesforce');
5B  System.debug('Does the set contain Pardot? ' + names.contains('Pardot'));
6B  System.debug('The size of the set is: ' + names.size());
```

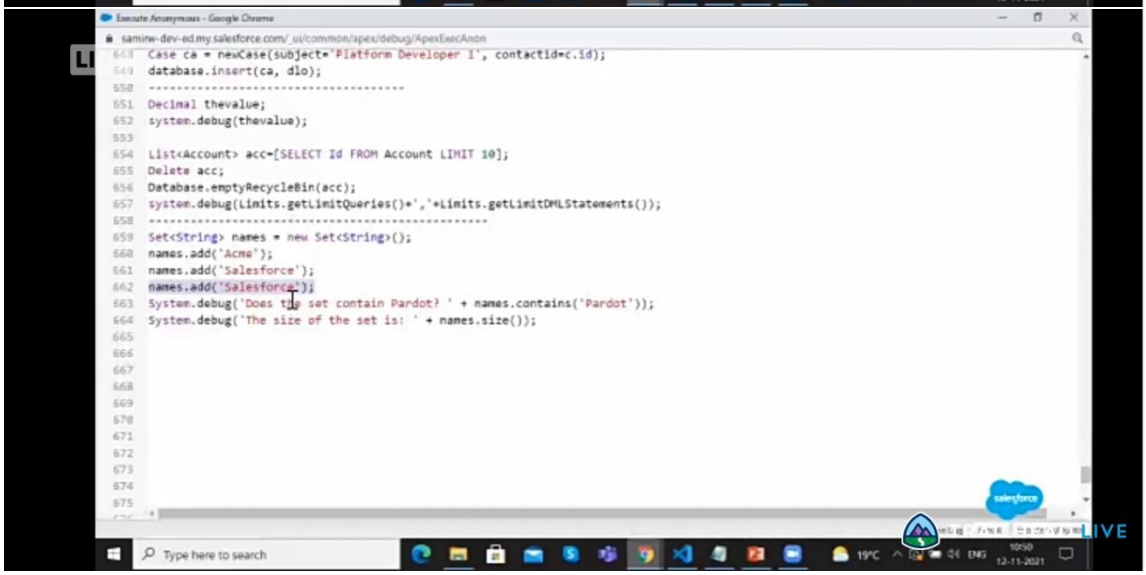
An unordered collection of elements that does not contain duplicates.

```
1C  Map<String, Integer> counts= new Map<String, Integer>();
2C  counts.put('Acme', 200);
3C  counts.put('Salesforce', 400);
4C  counts.put('NewCorp', 200);
5C  counts.put('Acme', 600);
6C  System.debug('The size of the map is: ' + counts.size());
7C  System.debug('The count for Acme is '
                + (counts.containsKey('Acme')? counts.get('Acme'):0));
```

A collection of key value pairs where each unique key maps to a single value.


[WATCH ME] 4-1 Demo Sets
44
TRAILHEAD ACADEMY
Time: 5 minutes
Goal: Execute this code sample and then answer the following questions.
Using the Developer Console, execute this code and then examine the log.
Set<String> names = new Set<String>();
names.add('Acme');
names.add('Salesforce');
names.add('Salesforce');
System.debug('Does the set contain Pardot? ' + names.contains('Pardot'));
System.debug('The size of the set is: ' + names.size());
QUESTIONS:
1. How many add statements are executed in this code block?
2. What is the size of the set?
3. Do the number of add statements and the size of the set match? Why or W
4. If names were a List instead of a Set, how many elements would names contain?
TRAILHEAD LIVE


Developer Console - Google Chrome
TRAILHEAD LIVE


Execute Anonymous - Google Chrome
Case ca = new Case(subject='Platform Developer 1', contactid=c.id);
database.insert(ca, dlo);
Decimal thevalue;
system.debug(thevalue);
List<Account> acc=[SELECT Id FROM Account LIMIT 10];
Delete acc;
Database.emptyRecycleBin(acc);
system.debug(Limits.getLimitQueries()+','+Limits.getLimitDMLStatements());
Set<String> names = new Set<String>();
names.add('Acme');
names.add('Salesforce');
names.add('Salesforce');
System.debug('Does the set contain Pardot? ' + names.contains('Pardot'));
System.debug('The size of the set is: ' + names.size());
TRAILHEAD LIVE

Conditionals and Loops

45

- If
- If-else
- Switch
- While
- Do-while
- For (traditional)
- For (list or set iteration)
- For (iterate over SOQL result)

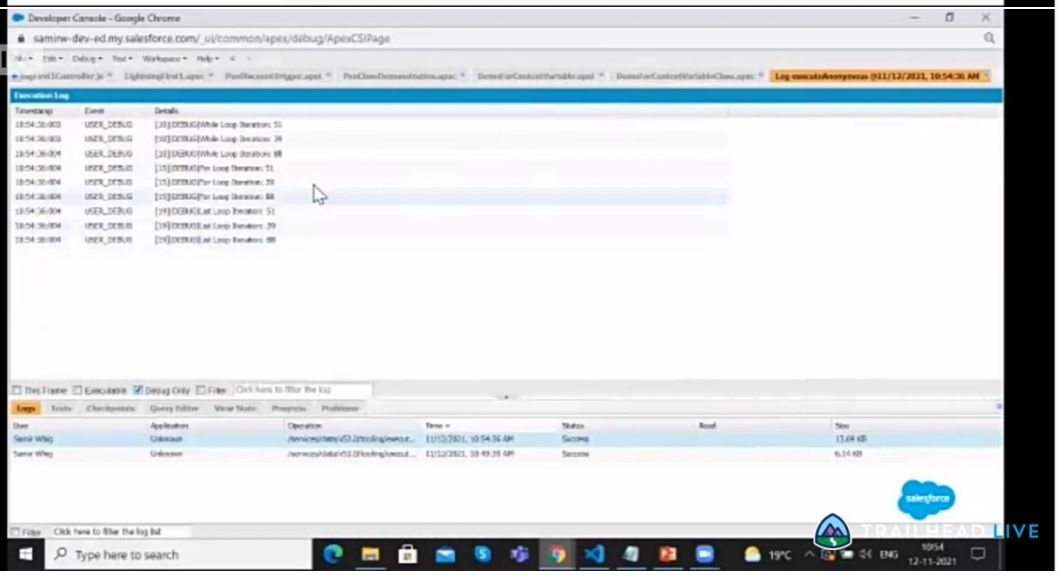
```
// A While Loop
while (iterator < threeNumbersSize) {
    System.debug('While Loop Iteration: ' + threeNumbers[iterator]);
    iterator++;
}

// A traditional For loop
for (iterator = 0; iterator < threeNumbersSize; iterator++) {
    System.debug('For Loop Iteration: ' + threeNumbers[iterator]);
}

// A For loop (list or set iteration)
for (Integer i :threeNumbers) {
    System.debug('List Loop Iteration: ' + i);
}
```

[WATCH ME] 4-2 Demo: Loops

46

1. Using the Developer Console, execute this anonymous block of code. Then examine the log.
2. Answer this question: do the 3 loops do the same thing?

```
List<Integer> threeNumbers = new List<Integer>();
threeNumbers.add(51);
threeNumbers.add(39);
threeNumbers.add(88);

Integer threeNumbersSize = threeNumbers.size();
Integer iterator = 0;

while (iterator < threeNumbersSize) {
    System.debug('While Loop Iteration: ' + threeNumbers[iterator]);
    iterator++;
}

for (iterator = 0; iterator < threeNumbersSize; iterator++) {
    System.debug('For Loop Iteration: ' + threeNumbers[iterator]);
}

for (Integer i :threeNumbers) {
    System.debug('List Loop Iteration: ' + i);
}
```



Lesson Agenda

47

- Working with Formulas and Roll-up Summary Fields
- Working with Basic Constructs in Apex
- ▶ Working with SOQL
- Working with SOSL
- Working with DML
- Working with Exceptions and Governor Limits

What is SOQL?

48

DEFINITION:

SOQL is the Salesforce Object Query Language.

SOQL allows developers to:

- Retrieve data, using user-defined selection criteria, from sObjects that reside in the Salesforce database.
- Integrate data retrieval into Apex and APIs.

SOQL is not SQL

Most SQL Variants	SOQL
Support statements for querying, CRED, transaction control, schema definition, and more	Only supports query statements
Support SELECT *	Does not support SELECT *
Support joins, which are written using "left" and "right" keywords	Supports "relationship queries," which are written using parent-child syntax
Do not support dot notation syntax to traverse foreign key relationships	Supports dot notation syntax to traverse object relationships
Are not governed by limits	Is multi-tenant aware (therefore, governed by limits)

SOQL SELECT Syntax

49

```
SELECT field1, field2, …
FROM sObject
[WHERE conditionExpression]
[LIMIT numberOfRows]
[Other options, such as GROUP BY]
```

SELECT Id, Name FROM Account WHERE CreatedDate = TODAY

WHERE Clause Operator

WHERE Clause Operator	Use
=	Equals
!=	Not equals
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
LIKE	Wildcard search in String fields; '%' matches 0+ characters. '_' matches 1+ characters
IN / NOT IN	Inclusion / Exclusion
INCLUDES / EXCLUDES	Inclusion and exclusion for multi-select picklists
AND	Logical AND
OR	Logical OR
NOT	Negation

RESOURCE: To find official documentation about these operators, use a search engine to search for the terms: `SOQL and SOSL Reference PDF`.

Type-Specific Considerations for the `Where` Clause

– Date values:

- The Date format is: YYYY-MM-DD.

```
... WHERE BirthDate = 1999-01-30
```

- In Apex, DateTime field values are in the Coordinated Universal Time (UTC). You may need to offset DateTime values to your local time zone (ex: -08:00).

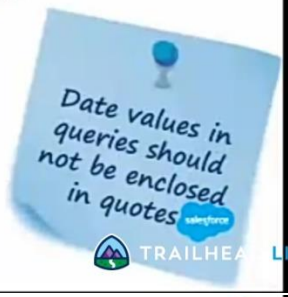
```
... WHERE ClosedDate > 2005-10-08T10:15:03-08:00
```

- You can write a query using date literals.

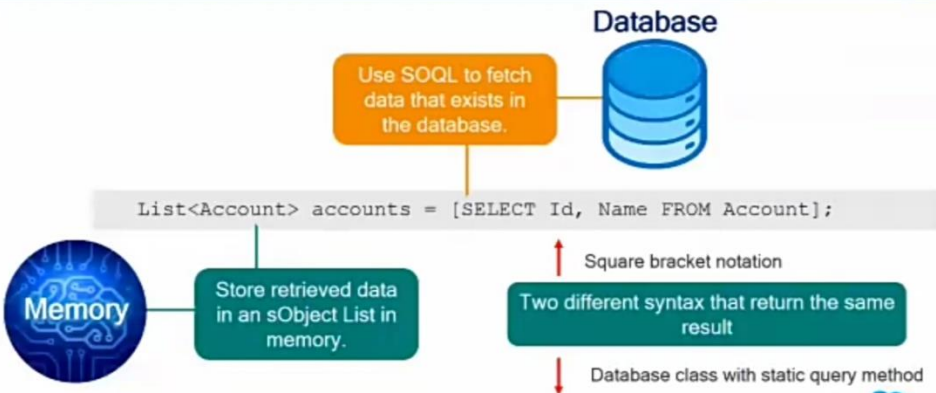
```
... WHERE ClosedDate != LAST_N_DAYS:365
```

– Boolean values can be used in SOQL.

```
... WHERE IsClosed = TRUE
```



Working with SOQL from Apex



Binding in the `Where` Clause

SOQL supports the binding operator (:) in the `WHERE` and `LIMIT` clauses.

- Both Variables, e.g. :myVar, and Expressions, e.g. :(aStatus + accountName), can be bound.

- Example of a bound Variable:

```
1B Set<String> caseSet = new Set<String>{'00001003', '00001005'};
2B List<Case> selectCases = [SELECT CaseNumber
                             FROM Case
                             WHERE CaseNumber IN :caseSet];
```

Use the colon operator to specify binding.

Return Types for SOQL Queries in Apex: sObject

You can assign the results of a query to a single sObject.

```
Account acc = [SELECT Id, Name FROM Account
               WHERE Name='Acme'];
```

But...

System.QueryException: List has no rows for assignment to SObject

or maybe...

System.QueryException: List has more than 1 row for assignment to SObject

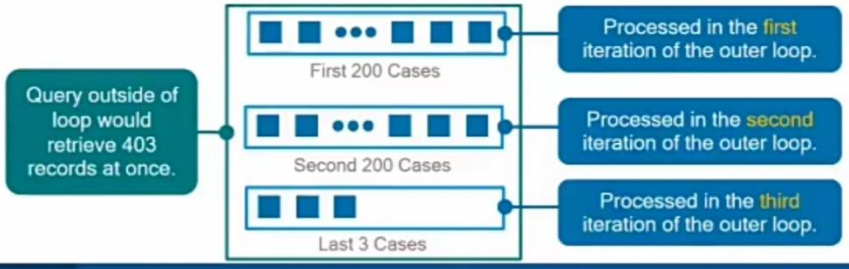
NOTE

If you cannot guarantee that a single sObject is returned *each* time the query is run, you should assign the query to an sObject list.

Using `List<sObject>` Iteration Variable


```
1 for (List<Case> cases : [SELECT CaseNumber FROM Case]) {
2     for (Case aCase : cases) {
3         //… process a single case
4     }
5 }
```

The outer loop executes once per 200 sObjects fetched by the SOQL query.



Query outside of loop would retrieve 403 records at once.

Processed in the first iteration of the outer loop.

Processed in the second iteration of the outer loop.

Processed in the third iteration of the outer loop.

NOTE:

A list iteration variable can help prevent you from exceeding the heap size limit when working with large volumes of data fetched by a query.

Relationship Naming Syntax

	Field Name
ID	accountId
Reference	account
Related List	contacts

Parent (one)

	Field Name
ID	course__c
Reference	course__r
Related List	course_deliveries__r

Child (many)

Referencing the Child-to-Parent Relationship

Account

ID	Name
123	Acc1

Account

Contact

ID	AccountId
456	123

SELECT Id, AccountId, Account.Name FROM Contact

NOTE:

You can access five levels of ancestors from a child using dot notation.

Referencing the Parent-to-Child Relationship

Account

ID	Name
123	Acc1

Course__c

ID	Name
789	Course1

Contacts Account

Course_Deliveries__r Course__r

Contact

ID	AccountId
456	123
111	123

SELECT Id, Name, (SELECT Id FROM Contacts) FROM Account

Course_Delivery__c

ID	Course__c
987	789
222	789

SELECT Id, Name, (SELECT Id FROM Course_Deliveries__ FROM Course__c

NOTE:

Only one level of nested queries is allowed in a SELECT claus

Lesson Agenda

- Working with Formulas and Roll-up Summary Fields
- Working with Basic Constructs in Apex
- Working with SOQL
- ▶ Working with SOSL
- Working with DML
- Working with Exceptions and Governor Limits

Anatomy of a Simple SOSL `FIND` Statement

DEFINITION: SOSL (Salesforce Object Search Language) allows developers to search text, email, and phone fields in multiple objects simultaneously.

Data type that holds the search results.

```
1 List<List<sObject>> acmes =
    [ FIND 'Acme' IN ALL FIELDS RETURNING Account, Opportunity ];
```

What string are we searching for?

Which type of field should be searched?

Which type of data should be returned?

What Does a SOSL Search Return?

SOSL has only one return data type: a List of Lists of sObjects.

The order the sObjects are listed in RETURNING clause determine their index in list.

```
1 List<List<SObject>> acmes = [ Find 'Acme' RETURNING Account(Name), Opportunity(Name) ];
```

The list contains one list for each sObject specified in the RETURNING clause.

SOSL Return Structure	Returned Records: Record Name
acmes[0] *(Accounts)*	acmes[0][0]: 'Acme'
acmes[1] *(Opportunities)*	acmes[1][0]: 'Acme - 1,200 Widgets'
	acmes[1][1]: 'Acme - 200 Widgets'
	acmes[1][2]: 'Acme - 600 Widgets'

Each sObject list contains those sObjects found mee... the search ...

Use Multiple `For` Loops to Process a `<List<List<sObject>>`


```
1 List<List<sObject>> acmes =
      [ Find 'Acme'
        RETURNING Account(Name), Opportunity(Name) ];
2
3 //The first sObject is the List of Accounts
4 List<Account> acmeAccounts = acmes[0];
5 for (Account acmeAccount : acmeAccounts) {
6     System.debug('Account: ' + acmeAccount.Name);
7 }
8
9 //The second sObject is the List of Opportunities
10 List<Opportunity> acmeOpportunities = acmes[1];
11 for (Opportunity acmeOpportunity : acmeOpportunities) {
12     System.debug('Opportunity: ' + acmeOpportunity.Name);
13 }
```

Find all Account and Opportunity records containing 'Acme.'

Parse the Accounts that have been returned.

Parse the Opportunities that have been returned.

Execution Log

Timestamp	Event	Details
14:34:49:113	USER_DEBUG	[8]|DEBUG|Account: Acme Inc (London)
14:34:49:113	USER_DEBUG	[8]|DEBUG|Account: Acme Inc
14:34:49:114	USER_DEBUG	[14]|DEBUG|Opportunity: Acme Inc - 6 Desktops
14:34:49:114	USER_DEBUG	[14]|DEBUG|Opportunity: Acme Inc - 20 Desktops
14:34:49:114	USER_DEBUG	[14]|DEBUG|Opportunity: Acme Inc - 600 Desktops
14:34:49:114	USER_DEBUG	[14]|DEBUG|Opportunity: Acme Inc - 700 Desktops

Where Can You Use SOSL?

You can execute SOSL searches inside:

- APIs.
- Apex statements, using:
  - Bracket notation:

```
[ Find 'Acme' RETURNING Account ]
```

  - `Search.query()`:

```
Search.query('Find {Acme} Returning Account')
```

Bound variable

```
1B String company = 'Acme';
2B List<List<sObject>> acmes2 = [ Find :company RETURNING Account ];
```


Knowledge Check: SOQL vs SOSL

QUESTIONS:

For each condition, determine whether SOSL or SOQL is a better option.

	SOQL	SOSL
You want to count the number of records meeting criteria.	✓	
You want data from one object or multiple related objects	✓	
You don't know which object or field the data resides in.		✓
You want to sort results as part of the query.	✓	✓
You want data from number, date, or checkbox fields.	✓	

Lesson Agenda

- Working with Formulas and Roll-up Summary Fields
- Working with Basic Constructs in Apex
- Working with SOQL
- Working with SOSL
- ▶ Working with DML
- Working with Exceptions and Governor Limits

Match the Programming Scenario to the DML Command

DEFINITION

DML: Apex's Data Manipulation Language allows you to persist the creation of or modifications to an instance of an sObject.

Match the scenario to the DML Command

1. Retrieve into memory a Contact whose LeadSource is 'Trade Show.' Modify its LeadSource to 'Other.' Persist this modification to your org.
2. Create a new instance of a Contact in memory. Persist this instance to your org.
3. Users were incorrectly entered as Contacts through the UI. Retrieve those Contacts into memory, and use their Id values to remove them from the org.
4. Create new Contacts in memory. Also modify existing Contacts that were called into memory using SOQL. Issue a single command to persist new and modified Contacts to the org.
5. Actually, the Contacts removed in Scenario 3 were created correctly. Use SOQL to retrieve those Contacts from the Recycle Bin into memory. Then, restore those Contact records.

DML Command

A. Insert
B. Update
C. Upsert
D. Delete
E. Undelete
F. Merge

Two Ways of Writing DML Commands

Standalone DML

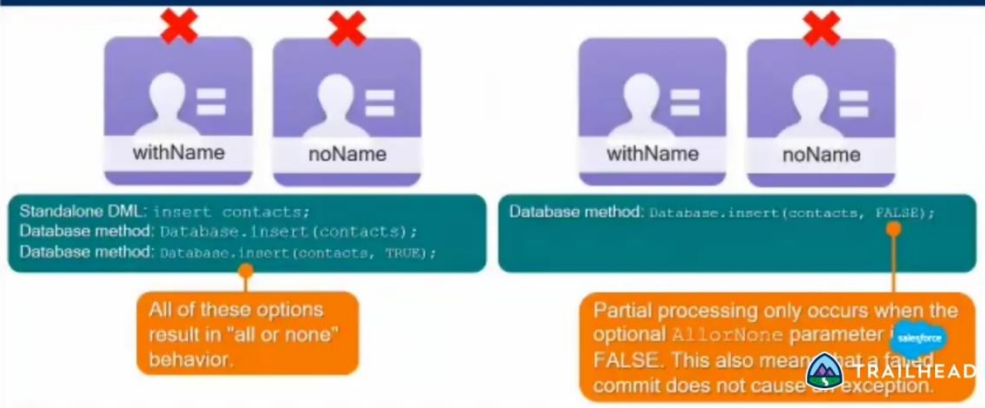
```
1A  Contact withName =
        new Contact(LastName = 'Rines');
2A  Contact noName = new Contact();
3A  List<Contact> contacts = new List<Contact>();
4A  contacts.add(withName);
5A  contacts.add(noName);
6A  insert contacts;
```

Database.method(sObject List)

```
1B  Contact withName =
        new Contact(LastName = 'Rines');
2B  Contact noName = new Contact();
3B  List<Contact> contacts = new List<Contact>();
4B  contacts.add(withName);
5B  contacts.add(noName);
6B  Database.insert(contacts);
```

Options for Partial Processing

What happens if inserting the Contact "noName" causes an exception, but inserting the Contact "withName" doesn't cause an exception?



Which Code Blocks Result in an Exception?


```
1A  List<Contact> contacts = new List<Contact>();
2A  insert contacts;
```

No exception.

```
1B  List<Contact> contacts = new List<Contact>();
2B  Contact noName = new Contact();
3B  Contacts.add(noName);
4B  insert contacts;
```

Ensure all the required fields are populated.

```
1C  List<Contact> contacts = new List<Contact>();
2C  Contact newContact = new Contact(LastName = 'Benett');
3C  Contacts.add(newContact);
4C  Contacts[0] = null;
5C  insert contacts;
```

Don't run DML operations on NULL elements.

```
1D  for (Integer i = 0; i<175; i++) {
2D      Contact testContact = new Contact(LastName = 'Test' + i);
3D      insert testContact; // LIMIT for DML commands in a single transaction = 150
4D  }
```

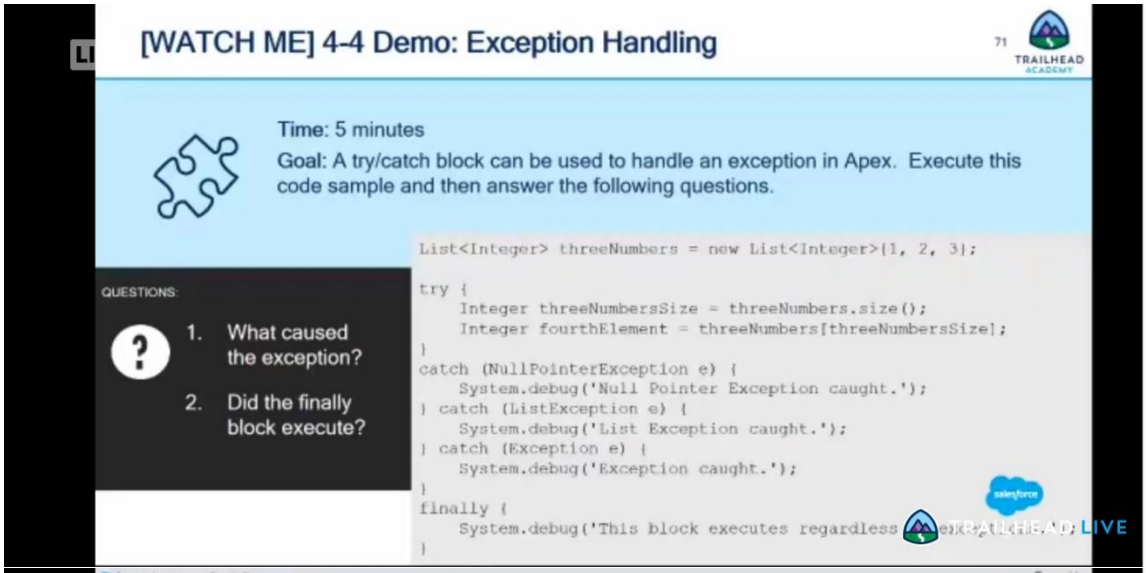
Stay within the DML Limits.

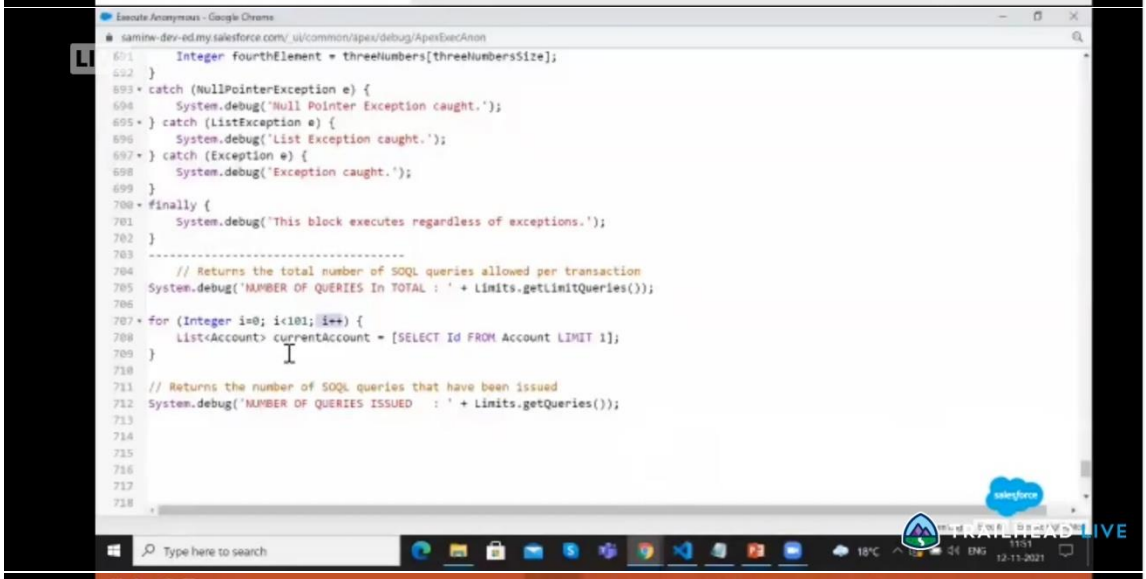
```
1E  List<Contact> contacts = new List<Contact>();
2E  Contact longName = new Contact(LastName =
        '00085chars00085chars00085chars00085chars00085chars00085chars00085chars00085chars00085chars00085');
3E  Contacts.add(longName);
4E  insert contacts; //LastName is a Text(80) field
```

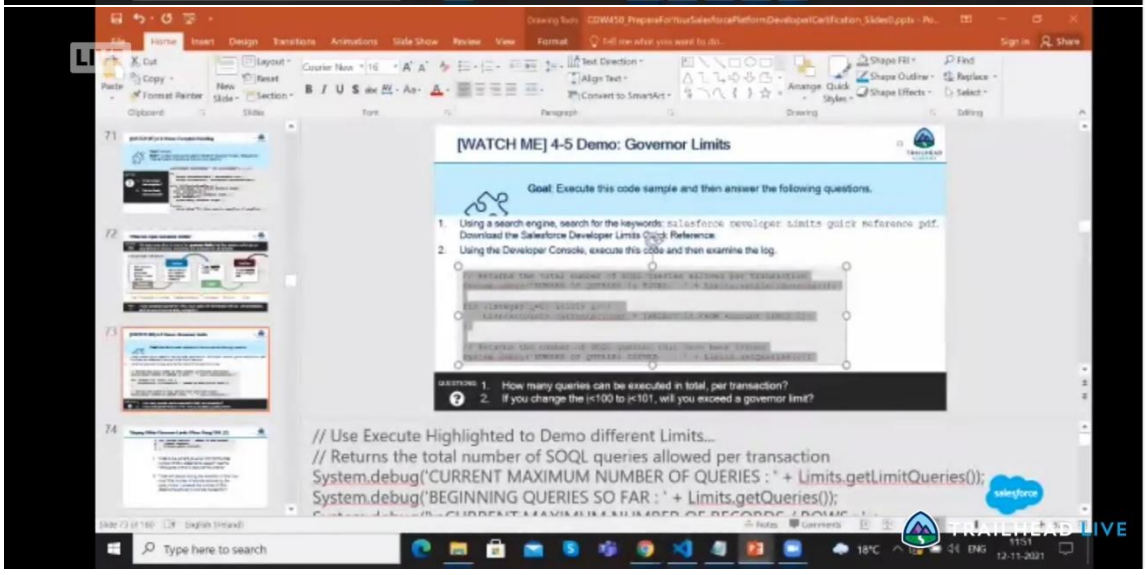
Ensure that field type restrictions are respected

Lesson Agenda

- Working with Formulas and Roll-up Summary Fields
- Working with Basic Constructs in Apex
- Working with SOQL
- Working with SOSL
- Working with DML
- ▶ Working with Exceptions and Governor Limits


[WATCH ME] 4-4 Demo: Exception Handling
71
TRAILHEAD ACADEMY
Time: 5 minutes
Goal: A try/catch block can be used to handle an exception in Apex. Execute this code sample and then answer the following questions.
QUESTIONS:
1. What caused the exception?
2. Did the finally block execute?
List<Integer> threeNumbers = new List<Integer>{1, 2, 3};
try {
Integer threeNumbersSize = threeNumbers.size();
Integer fourthElement = threeNumbers[threeNumbersSize];
}
catch (NullPointerException e) {
System.debug('Null Pointer Exception caught.');
} catch (ListException e) {
System.debug('List Exception caught.');
} catch (Exception e) {
System.debug('Exception caught.');
}
finally {
System.debug('This block executes regardless
TRAILHEAD LIVE

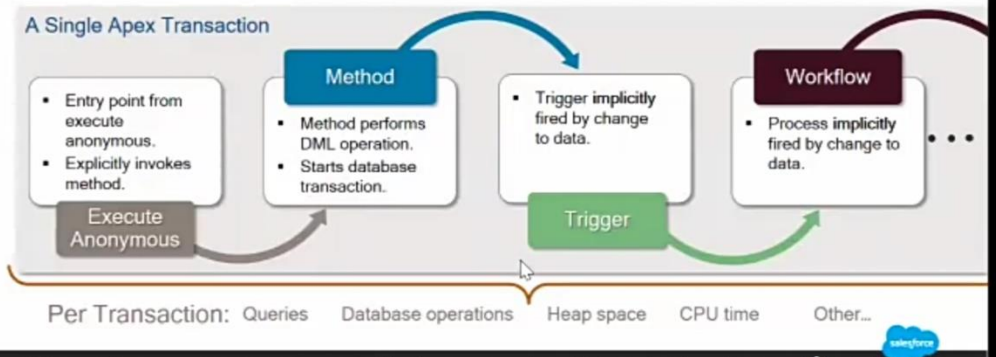

Execute Anonymous - Google Chrome
saminw-dev-ed.my.salesforce.com/_ui/common/apex/debug/ApexExecAnon
Integer fourthElement = threeNumbers[threeNumbersSize];
}
catch (NullPointerException e) {
System.debug('Null Pointer Exception caught.');
} catch (ListException e) {
System.debug('List Exception caught.');
} catch (Exception e) {
System.debug('Exception caught.');
}
finally {
System.debug('This block executes regardless of exceptions.');
}
// Returns the total number of SOQL queries allowed per transaction
System.debug('NUMBER OF QUERIES In TOTAL : ' + Limits.getLimitQueries());
for (Integer i=0; i<101; i++) {
List<Account> currentAccount = [SELECT Id FROM Account LIMIT 1];
}
// Returns the number of SOQL queries that have been issued
System.debug('NUMBER OF QUERIES ISSUED : ' + Limits.getQueries());
Type here to search
TRAILHEAD LIVE


[WATCH ME] 4-5 Demo: Governor Limits
Goal: Execute this code sample and then answer the following questions.
1. Using a search engine, search for the keywords: salesforce developer limits quick reference pdf. Download the Salesforce Developer Limits Quick Reference.
2. Using the Developer Console, execute this code and then examine the log.
QUESTIONS:
1. How many queries can be executed in total, per transaction?
2. If you change the i<100 to i<101, will you exceed a governor limit?
// Use Execute Highlighted to Demo different Limits...
// Returns the total number of SOQL queries allowed per transaction
System.debug('CURRENT MAXIMUM NUMBER OF QUERIES : ' + Limits.getLimitQueries());
System.debug('BEGINNING QUERIES SO FAR : ' + Limits.getQueries());
Type here to search
TRAILHEAD LIVE

What are Apex Governor Limits

72

DEFINITION: All Apex execution is bound by governor limits that the system enforces on operations to ensure resources are available for all tenants.



ALERT: If you exceed a governor limit, your code will terminate with an unhandleable, and hence unrecoverable, exception.

TRAILHEAD LIVE

[WATCH ME] 4-5 Demo: Governor Limits

73

Goal: Execute this code sample and then answer the following questions.

1. Using a search engine, search for the keywords: `Salesforce Developer Limits Quick Reference pdf`. Download the Salesforce Developer Limits Quick Reference.
2. Using the Developer Console, execute this code and then examine the log.

```
// Returns the total number of SOQL queries allowed per transaction
System.debug('NUMBER OF QUERIES In TOTAL : ' + Limits.getLimitQueries());

for (Integer i=0; i<101; i++) {
    List<Account> currentAccount = [SELECT Id FROM Account LIMIT 1];
}

// Returns the number of SOQL queries that have been issued
System.debug('NUMBER OF QUERIES ISSUED   : ' + Limits.getQueries());
```

QUESTIONS:
1. How many queries can be executed in total, per transaction?
2. If you change the i<100 to i<101, will you exceed a governor limit?

TRAILHEAD LIVE

Staying Within Governor Limits When Using DML (1)

74

```
1  for (Contact aContact : [SELECT Id FROM Contact]) {
2     //modify aContact
3     Database.update(aContact);
4  }
```

1. What is the current governor limit for the total number of DML statements issued? Use the limits guide online to discover the answer.
2. What will happen during the execution of this `for` loop if the number of records returned by the query in line 1 exceeds the number of DML statements allowed in a single transaction?

LIVE

Staying Within Governor Limits When Using DML (2)


```
1  List<Contact> contacts = new List<Contact>();
2  for(Contact aContact : [SELECT Id from Contact]) {
3      //modify aContact
4      contacts.add(aContact);
5  }
6  Database.update(contacts);
```

How does this code sample solve the issues we saw in the previous code sample?

Staying Within Governor Limits When Using DML (3)


```
1  for (List<Contact> contacts : [SELECT Id FROM Contact]) {
2      for (Contact aContact : contacts) {
3          //modify aContact
4      }
5      Database.update(contacts);
6  }
```

How does this code sample solve the issues we saw in the previous two code samples?

Lesson 5

Process Automation and Logic (Part 2)

Exam Objectives

78

- Given a use case, write Apex classes and triggers following best practices.
- Given a scenario, identify the implications of governor limits on Apex transactions.
- Describe the relationship between Apex transactions, the save order of execution, and the potential for recursion and/or cascading.

NOTE: The exam weighting for "Process Automation and Logic" is 38%. These are half of the objectives.

TRAILHEAD LIVE

Lesson Agenda

79

- Working with Apex Classes
- Working with Apex Triggers
- Describing the Save Order of Execution
- Platform Events

LIVE

Apex Class Use Cases

80

Model state and behavior (OOP)

```
1A public class GeneralClass {
1B     Boolean memberVariable;
1C     public void updateVariable() {}
1D }
```

Encapsulating business logic invoked by a trigger

```
2A public class sObjectTriggerHandler {
2B }
```

Testing

```
3A @isTest
3B private class AClass_Test {
3C }
```

Encapsulating reusable test methods

```
4A @isTest
4B public class TestDataFactory {
4C }
```

Controlling a Visualforce page: or Lightning Components

```
5A public class ACustomController {
5B }
```

And more...

- Custom SOAP and REST Web Services, for InBound and OutBound Integrations
- Bulk Apex, and Scheduled Processing
- In and OutBound custom eM... Processing

LIVE

More Uses Cases of an Apex Class

Implementing inheritance using an interface:

```
1A public interface CustomPaginator {
1B }
```

```
2A public class AccountPaginator implements CustomPaginator {
2B }
```

Implementing inheritance using a virtual class:

```
3A public virtual class CustomPaginator {
3B }
```

```
4A public class AccountPaginator extends CustomPaginator {
4B }
```

NOTE: In Apex, a data type of one class can be cast to and from a data type of another class, but only if the classes are related through inheritance.

Defining an Apex Class

Access modifier: who can see this class?

With/without/inherited sharing: which records can the class see?

```
1 public with sharing Class MyClass {
2     public DataType memberVariable;
3
4     public DataType memberProperty { get; set; }
5
6     public MyClass() {
7         // … Constructor logic
8     }
9
10    public void memberMethod() {
11        //… Method logic
12    }
13 }
```

A class can contain 0+ member variables.

A class can contain 0+ properties.

A class can contain 0+ constructors.

A class can contain 0+ methods.

Accessing an Apex Class or Class Member

Access Modifier Keyword	Applied to a Class	Applied to a Class Member
`global`	▪ Accessible to all Apex code everywhere. ▪ Used to define code for asynchronous Apex and services (email, web).	Accessible to all Apex code everywhere.
`public`	Accessible within your application or namespace.	Accessible within your application or namespace.
`protected`	Not available	Accessible to any: ▪ Inner classes in the defining Apex class. ▪ Classes that extend the defining Apex class.
`private`	▪ Applied to inner classes to make them accessible locally. ▪ Can be applied to test classes.	▪ *The default access modifier.* ▪ A private member is accessible only within the Apex class in which it is defined.

DEFINITION: Namespace prefixes are used in managed packages to differentiate classes, custom object and field names from those in use by other organizations.

Enforcing Sharing Rules


```
1A public with sharing class RespectsSharing {}
```

Respects the Sharing Model for the running user.

```
1B public without sharing class IgnoresSharing {}
```

Ignores the Sharing Model for the running user.

```
1C public inherited sharing class UsesCallerSharing {}
```

Uses the Sharing Model of the calling class.

Does this Class Respect the Running User's Sharing Model?


```
1A  public class NoKeywordPhraseClass {
1B  }
```

When this class is invoked by...	The Sharing Model is...
An Anonymous Block	Respected
A Test	Ignored
A Trigger	Ignored
A Web Service	Ignored
Another Class	▪ Respected, if the invoking class is "with sharing" ▪ Ignored otherwise

Simple Object-Level and Field-Level Security

WithSecurityDemo.cls

```
1  public with sharing class WithSecurityDemo {
2    @AuraEnabled(cacheable=true)
3    public static List<Contact> getContacts() {
4      return [
5        SELECT Name, Account.Name, SSN__c
6        FROM Contact
7        WITH SECURITY_ENFORCED
8        ORDER BY Name
9      ];
10   }
11 }
```

Enable field-level and object level security permissions checking.

- If any fields or objects referenced are inaccessible to the user, an exception is thrown, and no data is returned.
- Only available in Apex.

Sanitize Results and Prepare for DML Operations using `Security.stripInacessible()`


```
1  public void AccessDemo() {
2
3    List<Contact> myContacts = [SELECT LastName, SSN__c FROM Contact];
4
5    SObjectAccessDecision decision =
6      Security.stripInaccessible(AccessType.READABLE, myContacts);
7
8    for (Integer i = 0; i < myContacts.size(); i++) {
9      System.debug('Insecure access: ' + myContacts[i]);            //includes SSN__c
10     System.debug('Secure access: ' + decision.getRecords()[i]);  //doesn't include SSN__c
11   }
12
13   Contact c1 = new Contact(LastName='Smith', SSN__c='111-11-1111');
14   Contact c2 = new Contact(LastName='Williams');
15   List<Contact> newContacts = new List<Contact>{c1,c2};
16   SObjectAccessDecision securityDecision = Security.stripInaccessible(
17             AccessType.CREATABLE,
18             newContacts);
19   insert securityDecision.getRecords(); //inserted without SSN__c. No exceptions thrown!
20
21   System.debug(securityDecision.getRemovedFields().get('Contact')); // Prints "SSN__c"
22 }
```

Strip fields from SOQL results that fail FLS checks

Prepare for DML operations

Detailed Object and Field-Level Security using Apex Describe Information


```
1  public ID createContact( String firstName, String lastName, String phone, ID accountId ) {
2
3      Set<SObjectField> fieldsToCheck = new Set<SObjectField>{
4          Contact.FirstName, Contact.LastName, Contact.Phone, Contact.AccountId
5      };
6
7      NoAccessException ex = new NoAccessException();
8
9      if ( !Contact.sobjectType.getDescribe().isCreateable() ) {
10         ex.setMessage('User cannot create Contacts');
11         throw ex;
12     }
13
14     for ( SObjectField field : fieldsToCheck ) {
15         if ( !field.getDescribe().isCreateable() ) {
16             ex.setMessage('User cannot create Contact.' + field);
17             throw ex;
18         }
19     }
20
21     ... User has access, ok to insert new contact ...
22 }
```

Respects the Object Access for the running user.

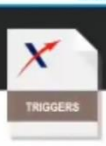
Respects the Field Access for the running user.

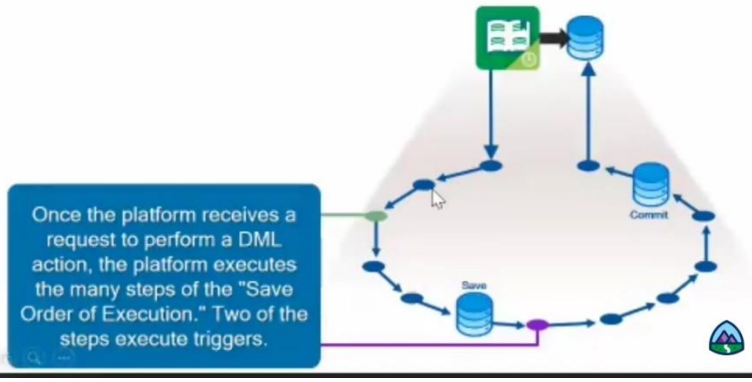
Lesson Agenda

- Working with Apex Classes
- ▶ Working with Apex Triggers
- Describing the Save Order of Execution
- Platform Events

A Trigger Provides a Code-Based Solution

DEFINITION: Trigger: Apex code that may execute because a DML event (insert, update, delete, or undelete) has occurred on an sObject.





Once the platform receives a request to perform a DML action, the platform executes the many steps of the "Save Order of Execution." Two of the steps execute triggers.

What Are the Two Types of Triggers Used For?

DEFINITION:

`before`: triggers are used to validate and potentially update record values.

`after`: triggers are used to access field values, such as Ids, that are set by the system and to effect changes in other records.

What type of trigger would you use to implement the following logic?

When a new Account has been created, automatically post a Chatter message with a link to the Account record to the associated account manager.

Syntax for Defining a Trigger

keyword

The name of the trigger.

keyword

This trigger is a part of the Save Order of Execution for DML events that occur on this sObject.

```
1 trigger TriggerName on sObject (before insert, before
      update, before delete, after insert, after update, after delete,
      after undelete) {
2     //Trigger logic …
3 }
```

Specify the DML events that fire this trigger.

Specify if this trigger contains logic for a `before` trigger and/or an `after` trigger.

Trigger Context Variables

Context Variable	What does it contain?	Where is it available?
`isInsert`, `isBefore`, `is…` (etc.)	Returns true, if the DML operation (e.g., `isInsert`) or timing (e.g., `isBefore`) is accurate for the event.	All triggers
`new`	A list of the new versions of the sObjects.	`insert`, `update` and `undelete` triggers
`newMap`	A map of IDs to the updated versions of the sObjects.	`before update`, `after insert`, `after update`, & `after undelete` triggers
`old`	A list of the previous versions of the sObjects.	`update` and `delete` triggers
`oldMap`	A map of IDs to the previous versions of the sObjects.	`update` and `delete` triggers
`size`	The number of records the Trigger is currently processing	All triggers
`isExecuting`	Returns true if the current context for the Apex code is a Trigger	All triggers
`operationType`	Returns a `System.TriggerOperation` enum response identifying the DML operation	All triggers

DEFINITION:

The run-time context which can be accessed using `System.Trigger` class attributes. It contains the current data values, DML event info, and more.

Using Context Variables to Determine What Logic Executes


```
1 trigger MyTrigger on
      MyObject__c (before insert, before update, after update) {
2     if (trigger.isBefore) {
3         if (trigger.isInsert) {
4             // Logic block 1
5         }
6         if (trigger.isUpdate) {
7             // Logic block 2
8         }
9     } else {
10        // Logic block 3
11    }
12 }
```


Using Context Variables to Determine What Logic Executes


```
1 trigger MyTrigger on
      MyObject__c (before insert, before update, after update) {
2
3       switch on trigger.operationType {
4             when BEFORE_INSERT {
5                   . . .
6             }
7             when BEFORE_UPDATE, AFTER_UPDATE {
8                   . . .
9             }
10            when else {
11                  . . .
12            }
13      }
14 }
```

Working With `Trigger.new` and `Trigger.oldMap`


```
1 trigger CourseDeliveryTrigger on Course_Delivery__c (before
      insert,  before update, before delete, after insert,
      after update) {
2     if (trigger.isAfter) {
3         if (trigger.isUpdate) {
4             for (Course_Delivery__c cd :trigger.new) {
5                 Date oldDate =
                      trigger.oldMap.get(cd.id).Start_Date__c;
6                 if (cd.Start_Date__c != oldDate) {
7                     // … Do some logic
8                 }
9             }
10        }
11    }
12 }
```

Iterate over `trigger.new` to perform logic on each sObject.

Use `trigger.oldmap` to determine changes between old and new records.

Triggers Execute on Implicitly Batched Data

How many times will this trigger run if the invoking DML action acted on a list of 200 Course Deliveries? 300?

```
1 trigger CourseDeliveryTrigger on Course_Delivery__c (before
      insert,  before update, before delete, after insert,
      after update) {
2     if (trigger.isBefore) {
3         if (trigger.isUpdate) {
4             for (Course_Delivery__c cd : trigger.new) {
5                 Date oldDate =
                      trigger.oldMap.get(cd.id).Start_Date__c;
6                 if (cd.Start_Date__c != oldDate) {
7                     // … Do some logic
8                 }
9             }
10        }
11    }
12 }
```

The `trigger.new` list is implicitly batched and contains at most 200 sObjects per iteration of the trigger.

Using `AddError` to Prevent a DML Action in a Trigger


```
1 trigger CourseDeliveryTrigger on Course_Delivery__c (before
      insert,  before update, before delete, after insert,
      after update) {
2
3     for (Course_Delivery__c cd : trigger.new) {
4         if (//… Some condition) {
5             //prevent the invoking DML action from completing
              cd.addError('This sObject cannot be saved.');
6         }
7     }
8 }
```

`sObject.addError` will prevent completion of the DML action.

NOTE:

`sObject.addError()` positions the error message at the top of the page in the UI.
`sObject.field.addError()` positions the error message next to the field itself.

Trigger Context Variables - Where are They Used?

100 TRAILHEAD ACADEMY

Event	Before	After
Insert	*Trigger.new (w)*	Trigger.new Trigger.newmap
Update	*Trigger.new (w)* Trigger.newmap *(w)* Trigger.old Trigger.oldmap	Trigger.new Trigger.newmap Trigger.old Trigger.oldmap
Delete	Trigger.old Trigger.oldmap	Trigger.old Trigger.oldmap
UnDelete		Trigger.new Trigger.newmap

(w) = Write-able

LIVE

Lesson Agenda

101 TRAILHEAD ACADEMY

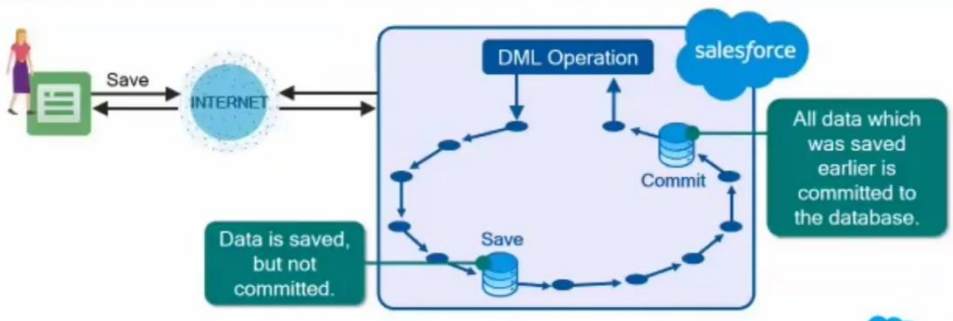
Working with Apex Classes
Working with Apex Triggers
▶ Describing the Save Order of Execution
Platform Events

LIVE

The Save Order of Execution

102 TRAILHEAD ACADEMY

DEFINITION: The Save Order of Execution describes the series of events that occur on the Lightning Platform when a record is saved.



NOTE: There is a similar (but fewer) series of events for delete and undelete operations.

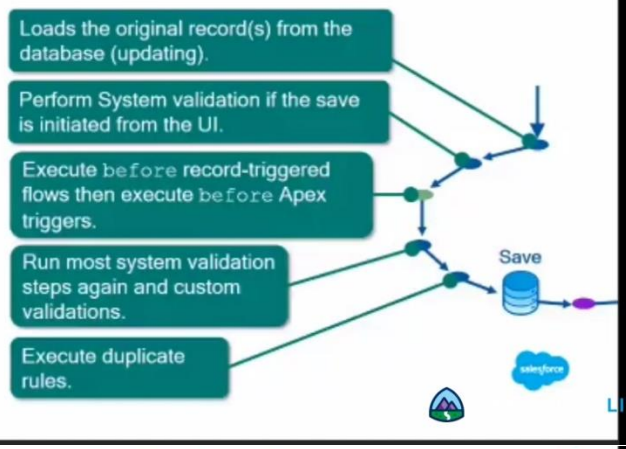
TRAILHEAD LIVE

What Happens Before the Save to the Database?

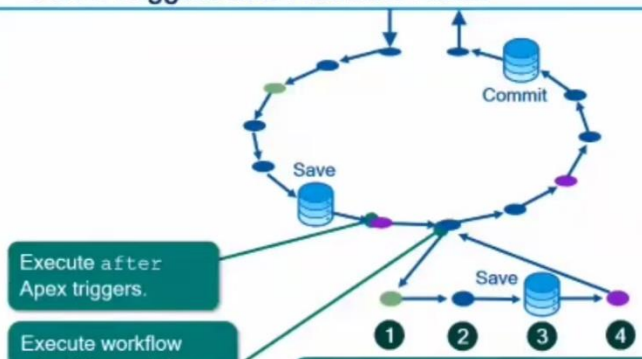
Consider the following scenario:

- The user saves a position record.
- The record passes all system validations.
- A before trigger is executed.
- System validation fails.

What event is likely to have caused the data to become invalid?



What Happens After the Save to the Database?
Part 1: Triggers and Workflow Rules



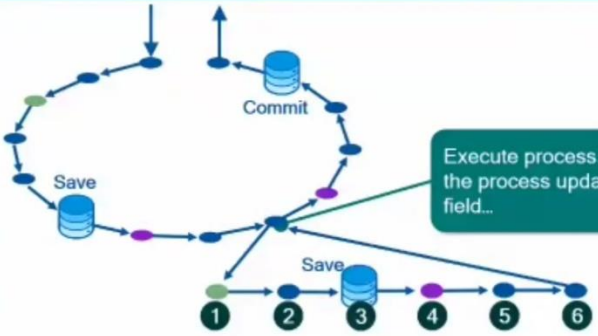
Execute after Apex triggers.

Execute workflow rule. If the workflow updates a field...

1. Execute before update Apex triggers
2. Execute most System validations (no custom).
3. Save the updated record to the database.
4. Execute after update Apex triggers.

- Executing these before and after Apex triggers can cause additional workflow rules to execute, causing these steps to begin again.
- Custom validation rules, duplicate rules and escalation rules are not run again.

What Happens After the Save to the Database?
Part 2: Apex Triggers and Processes



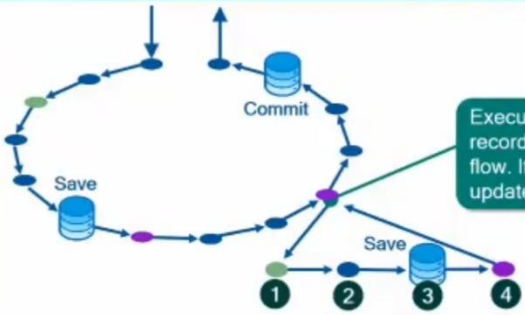
Execute process. If the process updates a field...

1. Execute before update Apex triggers.
2. Execute most System and all custom validations.
3. Save the updated record to the database.
4. Execute after update Apex triggers.
5. Execute workflow rules.
6. Execute processes again if the Recursion option is chosen in Process Builder.

The execution of these before and after Apex triggers can cause processes to execute against the same record in the same transaction if the Recursion option is chosen in Process Builder.

What Happens After the Save to the Database?
Part 3: After-Save Record-Triggered Flow

106



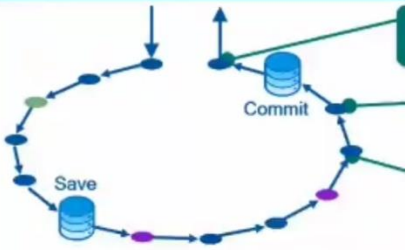
Execute after-save record-triggered flow. If the flow updates a field...

1. Execute `before update` Apex triggers
2. Execute most System and all custom validations.
3. Save the updated record to the database.
4. Execute `after update` Apex triggers.

The execution of these `before` and `after` Apex triggers can cause flow to execute against the same record in the same transaction.

What Happens After the Save to the Database? Part 4

107



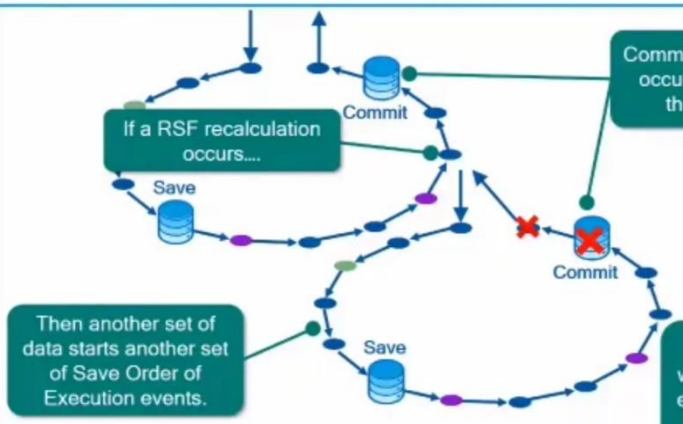
Execute post-commit logic.

Execute Criteria-Based Sharing evaluation.

Perform calculation for roll-up summary field or cross-object workflow on ancestors, working from parent upward.

What Happens When Another DML Save Operation is Performed?

108



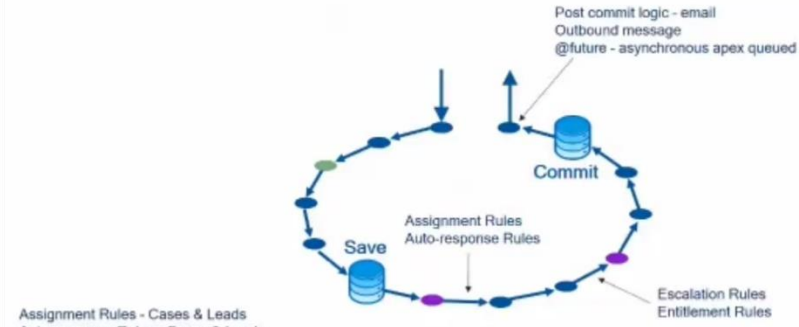
If a RSF recalculation occurs....

Commit and post-commit events occur just once, at the end of the current transaction.

Then another set of data starts another set of Save Order of Execution events.

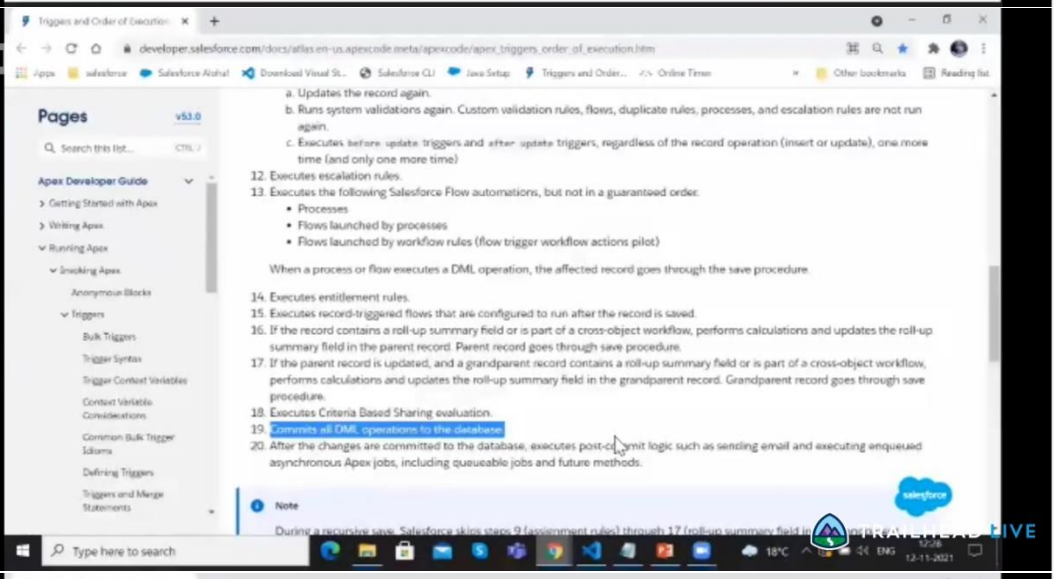
If a trigger updates a record, which causes another trigger to execute and update the original record, you can potentially get into a recursive situation.

Additional Steps



Assignment Rules - Cases & Leads
Auto-response Rules - Cases & Leads
Escalation Rules - Cases
Entitlement Rules = Cases & Work Orders

See **Apex Developer Guide - Triggers and Order of Execution**
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_triggers_order_of_execution.htm

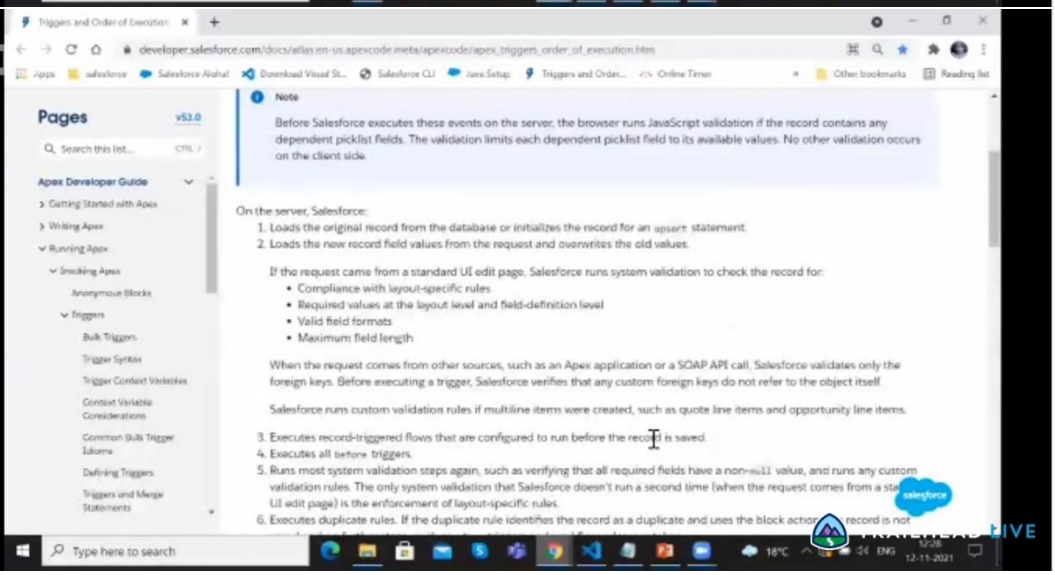


a. Updates the record again.
b. Runs system validations again. Custom validation rules, flows, duplicate rules, processes, and escalation rules are not run again.
c. Executes before update triggers and after update triggers, regardless of the record operation (insert or update), one more time (and only one more time)

12. Executes escalation rules.
13. Executes the following Salesforce Flow automations, but not in a guaranteed order.
    - Processes
    - Flows launched by processes
    - Flows launched by workflow rules (flow trigger workflow actions pilot)

    When a process or flow executes a DML operation, the affected record goes through the save procedure.
14. Executes entitlement rules.
15. Executes record-triggered flows that are configured to run after the record is saved.
16. 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.
17. 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.
18. Executes Criteria Based Sharing evaluation.
19. Commits all DML operations to the database.
20. After the changes are committed to the database, executes post-commit logic such as sending email and executing enqueued asynchronous Apex jobs, including queueable jobs and future methods.

Note



Note

Before Salesforce executes these events on the server, the browser runs JavaScript validation if the record contains any dependent picklist fields. The validation limits each dependent picklist field to its available values. No other validation occurs on the client side.

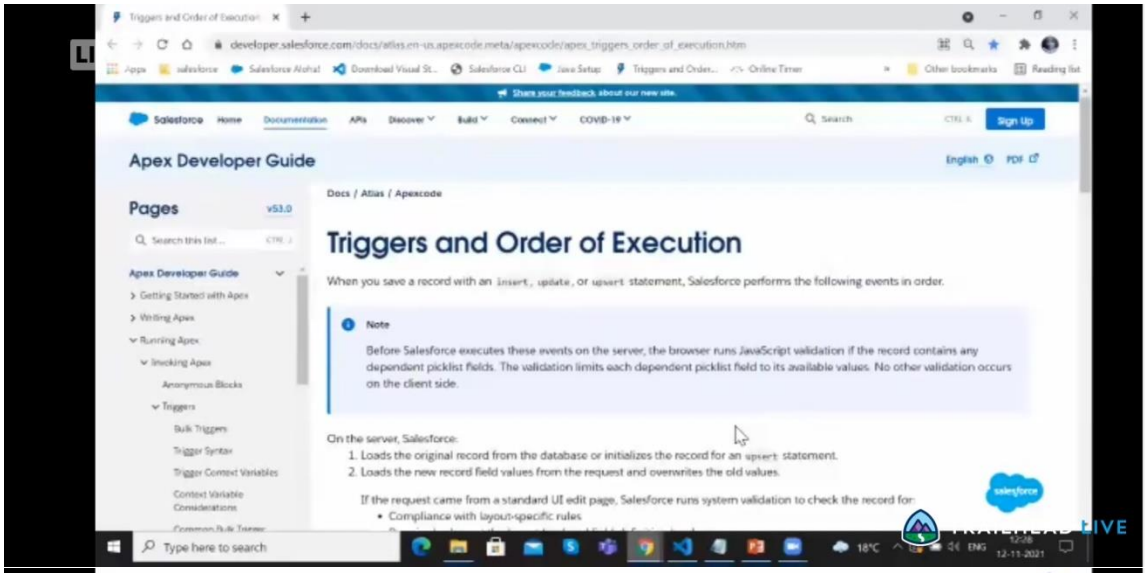
On the server, Salesforce:

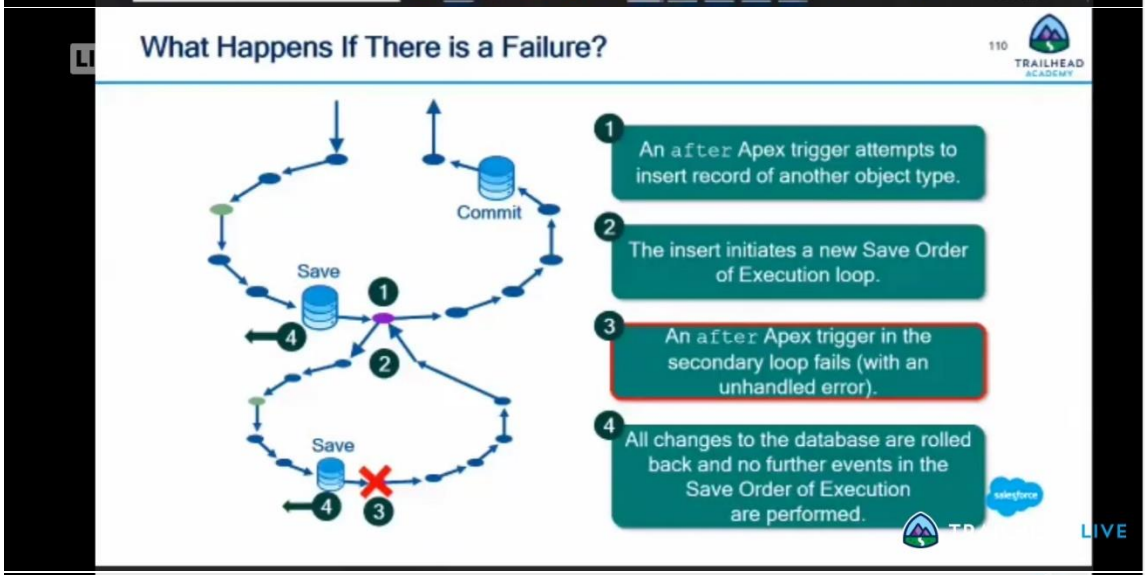
1. Loads the original record from the database or initializes the record for an `upsert` statement.
2. Loads the new record field values from the request and overwrites the old values.

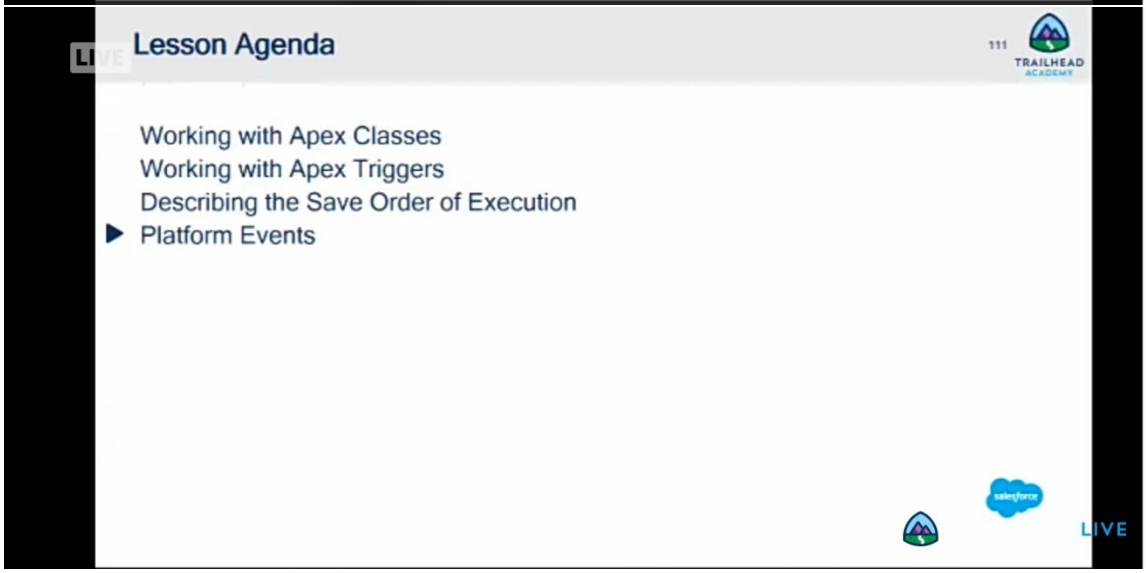
   If the request came from a standard UI edit page, Salesforce runs system validation to check the record for:
   - Compliance with layout-specific rules
   - Required values at the layout level and field-definition level
   - Valid field formats
   - Maximum field length

   When the request comes from other sources, such as an Apex application or a SOAP API call, Salesforce validates only the foreign keys. Before executing a trigger, Salesforce verifies that any custom foreign keys do not refer to the object itself.

   Salesforce runs custom validation rules if multiline items were created, such as quote line items and opportunity line items.
3. Executes record-triggered flows that are configured to run before the record is saved.
4. Executes all `before` triggers.
5. Runs most system validation steps again, such as verifying that all required fields have a non-`null` value, and runs any custom 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.
6. Executes duplicate rules. If the duplicate rule identifies the record as a duplicate and uses the block action, the record is not


Apex Developer Guide
Pages
Triggers and Order of Execution
When you save a record with an insert, update, or upsert statement, Salesforce performs the following events in order.
Note
Before Salesforce executes these events on the server, the browser runs JavaScript validation if the record contains any dependent picklist fields. The validation limits each dependent picklist field to its available values. No other validation occurs on the client side.
On the server, Salesforce:
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


What Happens If There is a Failure?
110
TRAILHEAD ACADEMY
Commit
Save
Save
1
2
3
4
An after Apex trigger attempts to insert record of another object type.
The insert initiates a new Save Order of Execution loop.
An after Apex trigger in the secondary loop fails (with an unhandled error).
All changes to the database are rolled back and no further events in the Save Order of Execution are performed.
LIVE


Lesson Agenda
111
TRAILHEAD ACADEMY
Working with Apex Classes
Working with Apex Triggers
Describing the Save Order of Execution
Platform Events
LIVE

What is a Platform Event?

112

TRAILHEAD ACADEMY

PLATFORM EVENTS

1. *Enable you to deliver secure, scalable, and customizable event notifications within Salesforce or from external sources.*
2. *Rely on a publish-subscribe architecture.*

TRAILHEAD LIVE

Event-Driven Software Architecture

113

TRAILHEAD ACADEMY

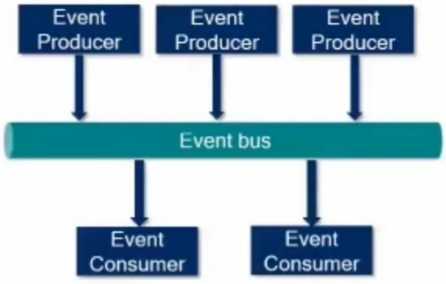
Event: a change in state that is meaningful in a business process.

Event Message: a message that contains data about the event.

Event Producer: the publisher of an event message over an event bus.

Event consumer: a subscriber to an event bus that receives messages from the event bus.

Event Bus (Channel): a stream of events on which an event producer sends event messages and event consumers read those messages. Also known as a channel.



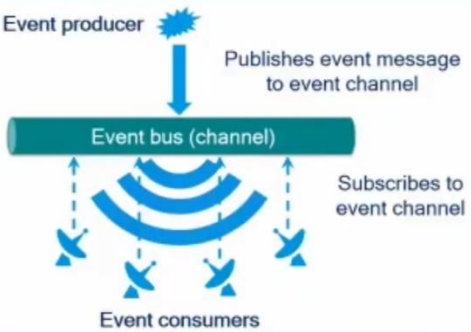
LIVE

Publisher/Subscriber Model

114

TRAILHEAD ACADEMY

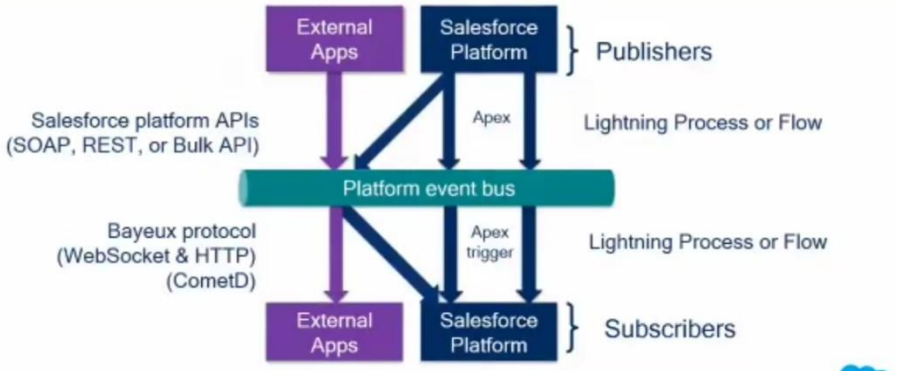
NOTE: A **publisher** categorizes messages into classes and sends them without knowledge of the subscriber that will receive it.



NOTE: A **subscriber** expresses interest in one or more classes and only receives messages that are of interest, without knowledge of the publisher that produced them.

TRAILHEAD LIVE

Platform Events Publisher & Subscribers



Platform Events and Objects

- Platform Events are similar to custom objects.
- Platform Events are defined in the same way you define a custom object.
- Unlike custom objects, Platform Events are not queryable using SOQL or SOSL.

	Platform Event	Custom Object
Instance	Event Message	Record
API name ends with:	__e	__c
Insert	☑	☑
Update	☒	☑
Delete	☒	☑
View in Salesforce UI	☒	☑
Set Read/Create Permissions	☑	☑

Publish a Platform Event Using Apex


```
// Method for publishing EventMessage__e events
public static void publishEvents(List<String> messages) {
      List<EventMessage__e> events = new List<EventMessage__e>();
      for (String message: messages) {
            events.add(new EventMessage__e(Message__c = message));
      }
      List<Database.SaveResult> results = EventBus.publish(events);

      // Inspect publishing results
      . . .
}
```

To publish event messages, call the `EventBus.publish` method.

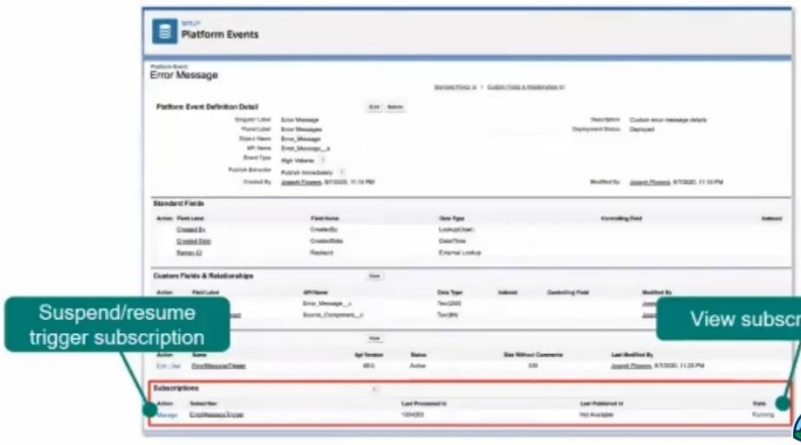
NOTE: Platform events can be configured to be published immediately or to be published as part of the post commit phase in the save order of execution

Subscribe to a Platform Event Using Apex

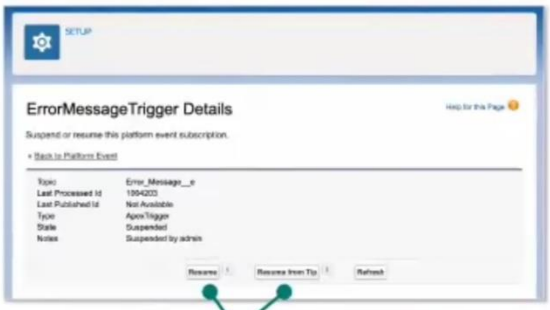

```
// Trigger for catching EventMessage__e events.
trigger EventMessageTrigger on EventMessage__e (after insert) {

      for (EventMessage__e evt : trigger.new) {

      ...

      }
}
```

Only after insert supported

Manage Your Platform Event Trigger Subscriptions from the User Interface



Resuming a Suspended Event Subscription



When resuming, choose whether or not to process events published during the suspension.

Lesson 6

121

User Interface

Exam Objectives

122

- Given a scenario, display or modify Salesforce data using a Visualforce page and the appropriate controllers or extensions as needed.
- Describe the types of web content that can be incorporated into Visualforce pages.
- Incorporate Visualforce pages into Lightning Platform applications.
- Describe the Lightning Component framework and its benefits.
- Describe the types of content that can be contained in a Lightning web component.
- Given a scenario, prevent user interface and data access security vulnerabilities.
- Given a scenario, display and use a custom user interface components, including Lightning Components, Visual Flow, and Visualforce.
- Describe the use cases for Lightning component events and application events.
- Given a user interface requirement, describe interactions between Apex and various types of page components, including Lightning Components, Visual Flow, Next Best Actions, etc.

NOTE: The exam weighting for "User Interface" is 25%.

Lesson Agenda

123

- Working with Visualforce Pages
- Working with Visualforce Controllers
- Working with the Lightning Component Framework

What is Visualforce?

- Visualforce markup: A tag-based markup language, similar to HTML.

```
<apex:page standardController="Account" lightningStylesheets="true">
    <apex:tabPanel >
        <apex:tab label="Details">
            <apex:detail relatedList="false"/>
        </apex:tab>
        <apex:tab label="Related Lists">
            <apex:relatedList list="Contacts"/>
            <apex:relatedList list="CombinedAttachments"/>
        </apex:tab>
    </apex:tabPanel>
</apex:page>
```

- Visualforce controllers:
  - Standard controllers (out of the box)
  - Custom controllers or Controller Extensions, written in Apex.

Binding Data to Visualforce Controllers

- The standardController attribute specifies the object on which the page and standard controller will operate, and the style of the tab that will display the page.

```
<apex:page standardController="Account">
```

- The Id parameter on the URL binds the page to a single record, giving it: data context.

```
https://na1.salesforce.com/apex/myPage?id=0013000000gzexd
```

NOTE: The sObject Id being passed in must be for an object of the same type as that specified by the standardController tag.

Expression Syntax

Tags use the same expression syntax as formula fields and other areas of the application. All content in `{!...}` is evaluated as an expression.

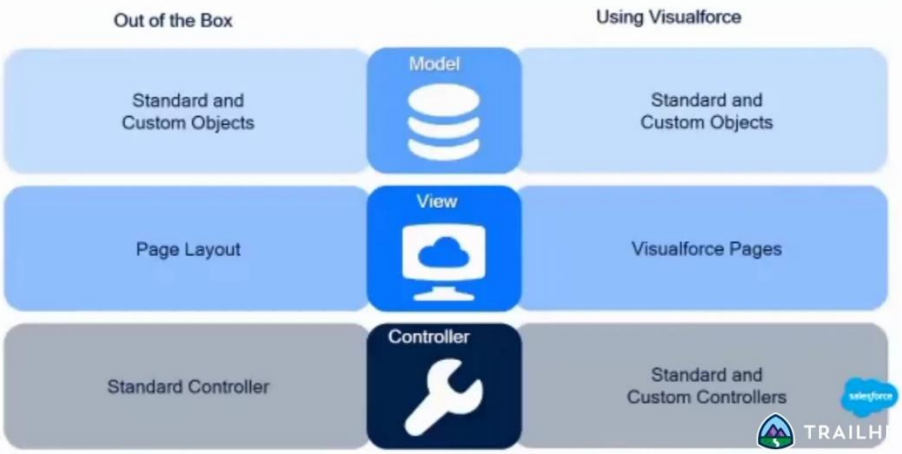
```
<apex:inputField value="{!Course__c.Duration__c}"/>
```

sObject Name: `Course__c`
Field Name: `Duration__c`
Contains the expression: `{!Course__c.Duration__c}`

Global variables, whose names start with $, can be accessed using the same syntax. For example:

- `{!$User.fieldName}`
- `{!$Page.otherVisualforcePage}`
- `{!$Resource.staticResource}`

Visualforce Pages and the MVC



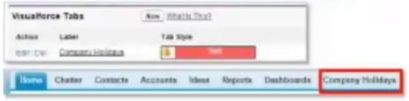
Launching Visualforce Pages

- Provide the URL for the page.

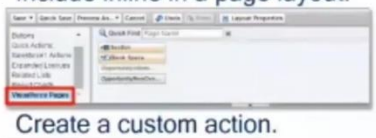
- Create a custom button or link.

- Override one of the standard actions.

- Create a custom Visualforce tab.

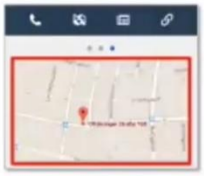


- Include inline in a page layout.



- Create a custom action.

- Create a mobile card.



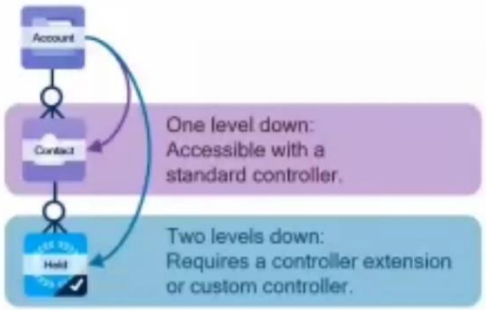
Lesson Agenda

Working with Visualforce Pages
▶ Working with Visualforce Controllers
Working with the Lightning Component Framework

When Can't You Use a Standard Controller?

132

You need a custom controller or controller extension when you must display data that is not accessible with a standard controller.



Additional reasons to use a custom controller or controller extension:

- Create custom behaviors
- Override existing functionality
- Customize the navigation

TRAILHEAD LIVE

When Can't You Use a Standard Controller?

140

- Controller Extensions are invoked using the extensions attribute of the opening `<apex:page>` tag:

```
<apex:page standardController = "sObject" extensions = "ApexExt1_CX, ApexExt2_CX">
```

- Custom Controllers are invoked using the controller attribute of the opening `<apex:page>` tag:

```
<apex:page controller = "CustomApexClass_CC" extensions = "ApexExt1_CX, ApexExt2_CX" >
```

- Order Matters: Extensions are evaluated from left to right.

  For Overrides, the first Extension having the called method is executed. If no Extension has the called method, e.g. "Save", the Save method of the Standard or Custom Controller is then executed.

NOTE: A single page can include only one main controller (standard or custom), but it may reference several controller extensions.

TRAILHEAD LIVE

Why Must a Page Use a Controller Extension?

134

Scenario	Controller
You want to use functionality already existing in the Standard or Custom Controller you are using for the page	Controller Extensions
You need to use your page with declarative Salesforce features that depend on a Standard Controller, such as creating a custom button or including your page within a page layout.	Controller Extension
You need a Controller for a Custom Visualforce Component	Custom Controller

NOTE: A Controller Extension can be used with either a Standard Controller or a Custom Controller class.

TRAILHEAD LIVE

What Can Controller Extensions and Custom Controllers Provide?

Getter methods that allow the view to retrieve data using the controller.

Setter methods that allow the view to set data in the controller.

Properties that can be used to get and set or store values.

Action methods to perform logic or navigation.

```
1  public class TheController {
2      String searchText;
3      List<Lead> results;
4      public String getSearchText() {
5          return searchText;
6      }
7
8      public void setSearchText(String s) {
9          searchText = s;
10     }
11
12     public List<Lead> Results { get; }
13
14     public PageReference doSearch() {
           results = (List<Lead>)[ FIND :searchText
                                   RETURNING Lead(Name, Email) ][0];
15
16         return null;
17     }
18 }
```


Controller Extension Constructors

A Controller Extension constructor takes an argument of type `ApexPages.StandardController` or a Custom Controller class.

- For Standard Controllers, the Extension can use this variable to call `getId()` or `getRecord()` to access the associated sObject.

```
1 public class MyControllerExtension {
2     private final Account acct;
3     public MyControllerExtension(ApexPages.StandardController stdController) {
4         this.acct = (Account)stdController.getRecord();
5     }
6     public String getGreeting() {
7         return 'Hello ' + acct.name + ' (' + acct.id + ')';
8     }
9 }
```

NOTE: Other than the required constructor, Controller Extensions can contain all the same functionality available in Custom Controllers.

Custom Controller Vs. Controller Extension.

Scenario	Answer
A quick-create page, allowing the user to create an account, contact, and opportunity, all from one page.	Custom Controller*
A page with custom functionality that can be launched using a button on a page layout.	Controller Extension
A multi-page wizard, which guides a user through the process of designing a sales plan for a new customer.	Custom Controller
A page that, on saving a new certification record, automatically redirects the user to a page where they can create a new related certification element.	Controller Extension

Standard List Controllers

DEFINITION

Standard List Controllers allow you to create Visualforce pages that display and act on a set of records. The collection of records is based on existing List Views defined for the sObject.

Including the `recordSetVar` attribute selects the Standard List Controller for the account object instead of the regular account Controller.

```
1 <apex:page standardController="Account" recordSetVar="accounts" >
2     <apex:pageBlock >
3         <apex:pageBlockTable value="{!accounts}" var="acc">
4             <apex:column value="{!acc.name}"/>
5         </apex:pageBlockTable>
6     </apex:pageBlock>
7 </apex:page>
```

The value of the recordSetVar attribute becomes the name of the property that enables page access to the resulting list of records.

Custom List Controllers

DEFINITION

Custom List Controllers use the ApexPages.StandardSetController object to enable the same powerful features of the standard List Controller but using the result of a query or a search

```
1  public class DisplayCertifications_CC {
2      // Standard Set Controller
3      public ApexPages.StandardSetController setCon {
4          get {
5              if(setCon == null) {
6                  setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                       [SELECT Name, Status__c FROM Certification__c]));
7              }
8              return setCon; }
9              set;
10     }
11     // Return a list of records
12     public List<Certification__c> getCertifications() {
13         return (List<Certification__c>) setCon.getRecords();
14     }
15 }
```

Instantiate the standardSetController to use built-in set controller methods.

Include your own custom meth

What is the Lightning Component Framework?

DEFINITION

The Lightning Component framework is a UI framework for developing responsive, dynamic web apps that run on the Salesforce platform and can easily access data in a Salesforce org.

Programmed using these languages:

- Client-side:
  - HTML
  - CSS
  - JavaScript
- Server-side:
  - Apex

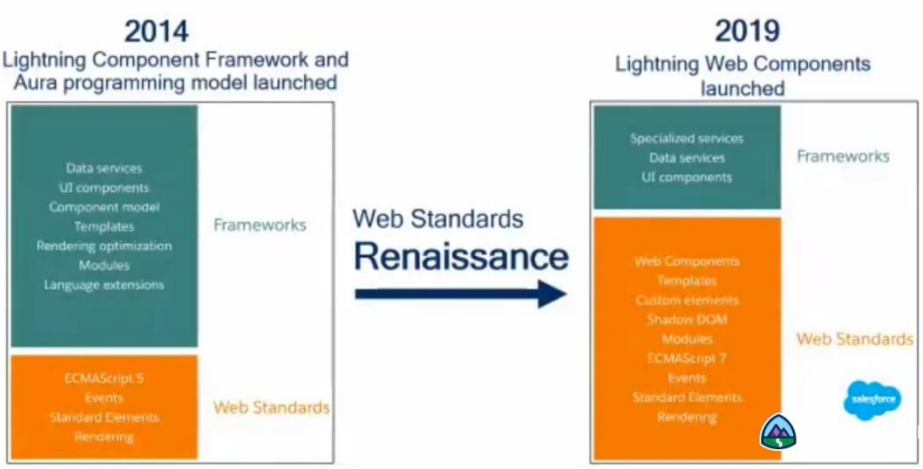
A framework is responsive when:

- It uses relative sizing.
- It allows for the application of different styles based on the context.
- It uses flexible scaling for images and media.

Two Programming Models

142

TRAILHEAD ACADEMY



LIVE

What is a Lightning Component? Application?

143

TRAILHEAD ACADEMY

DEFINITION:

A Lightning Component is a self-contained and reusable unit of the UI and can range in granularity from a single line of text to an entire app.

DEFINITION:

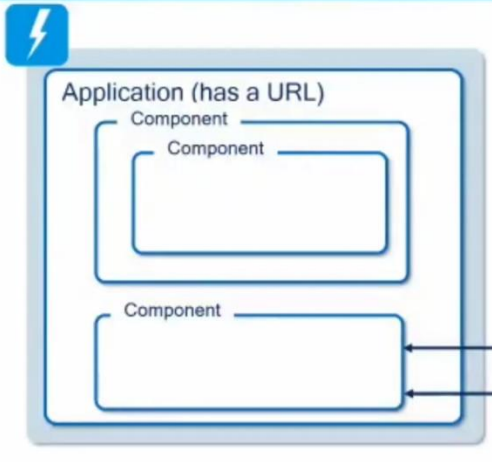
A Lightning Application is a top-level URL addressable container for one or more Lightning Components. Lightning Experience, Communities and the mobile app are Lightning Applications

LIVE

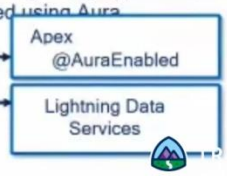
Lightning Components are Modular

144

TRAILHEAD ACADEMY



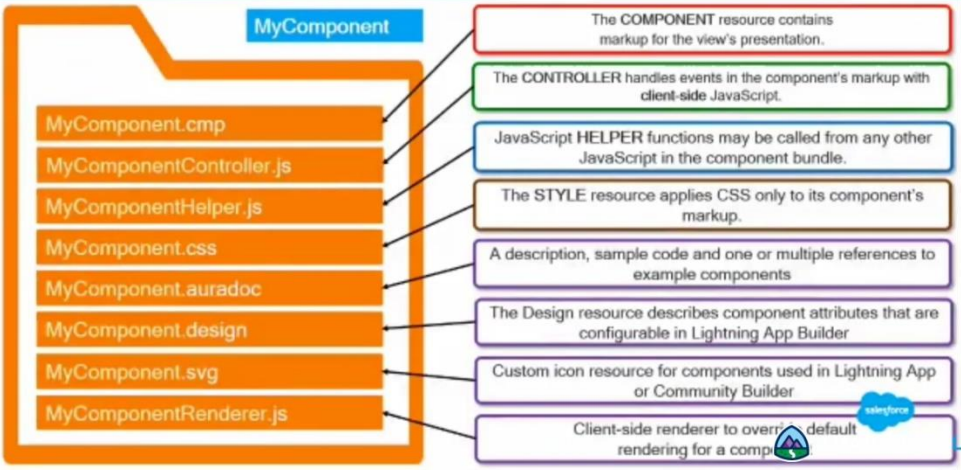
- A component can be either an Aura component or a Lightning web component
- An Aura component can contain an LWC, BUT an LWC cannot contain an Aura component
- An application cannot contain another application
- A custom Lightning Application can only be created using Aura



LIVE

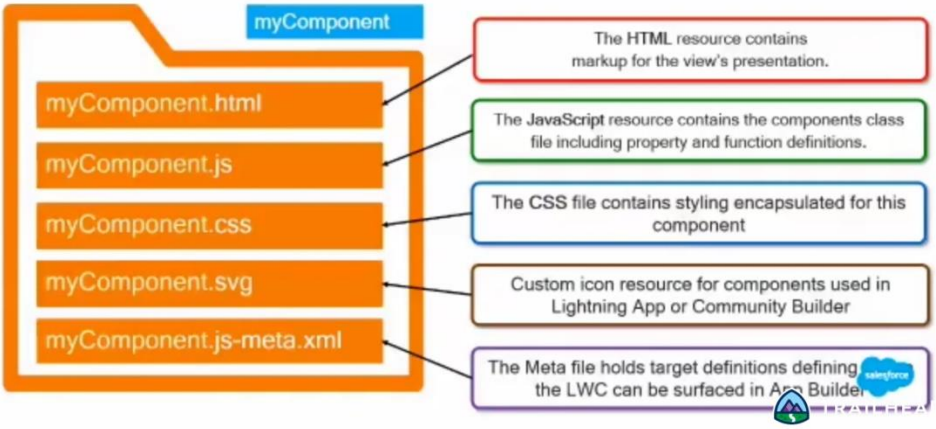
What Type of Resources are in an Aura Component Bundle

145



What Type of Resources are in a Lightning Web Component Bundle

146



Comparing a Visualforce Page and a Lightning Application

147

Visualforce:	Lightning:
▪ Page-centric	▪ App-centric (single-page apps)
▪ Stateful servers	▪ Stateless servers
▪ Stateless clients	▪ Stateful clients
▪ Not optimal for mobile	▪ Great for everything

Read on Trailhead: Develop for Lightning Experience | Lightning Experience Development | User Interface Development Considerations

Lesson 7

148

TRAILHEAD ACADEMY

Testing, Debugging, and Deployment

LIVE

Exam Objectives

149

TRAILHEAD ACADEMY

- Write and execute tests for triggers, controllers, classes, flows, and processes using various sources of test data.
- Describe the use cases for invoking anonymous code and the differences between invoking Apex in execute anonymous vs. unit tests.
- Describe the Salesforce Developer tools such as Salesforce DX, Salesforce CLI, and Developer Console, and when to use them.
- Describe how to approach debugging system issues and monitoring flows, processes, and asynchronous and batch jobs, etc.
- Describe the environments, requirements, and process for deploying code and associated configurations.

NOTE: The exam weighting for "Testing, Debugging, and Deployment" is 14%.

TRAILHEAD LIVE

Lesson Agenda

150

TRAILHEAD ACADEMY

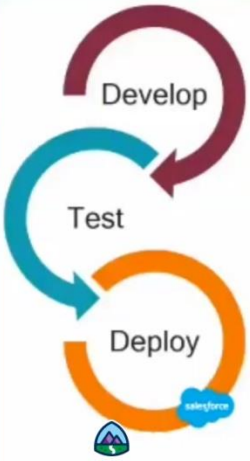
- ▶ Describing the Testing Framework and Requirements
- Creating Test Data and Tests
- Executing a Test
- Debugging
- Deployment

LIVE

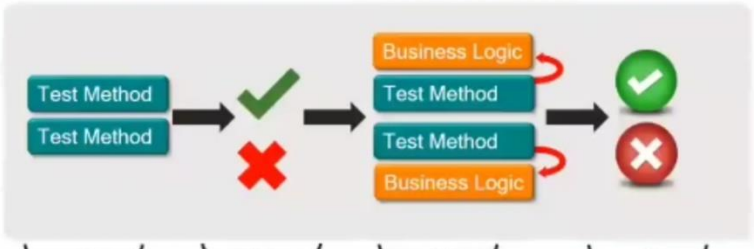
Testing Apex Code is a Requirement

In our sandbox, I've created a number of triggers and classes for the certification application. Before I can deploy this code to production, I need to ensure:

- 75% of Apex code must be executed successfully by test methods.
- Every Apex trigger must have some coverage in test methods, even though every line of every trigger does not have to be executed by a test.
- Every Apex test method must execute without throwing any uncaught exceptions or exceeding governors.



The Testing Framework



A developer writes test methods inside test classes.

Salesforce receives a specific command to run tests.

Salesforce runs test methods and any code they invoke.

Salesforce reports the success or failure of the tests, along with test coverage percentages.

DEFINITION: An Apex test method verifies whether a particular piece of Apex code is working as expected based on good, bad and bulk data.

Anatomy of an Apex Test Method

Tells the platform the following class will only be used for testing.

A test method must be static, return nothing and take no arguments.

Use `_Test` as the suffix for the name of a test class (recommended).

A test class should be `private`.

`@isTest` tells the testing framework that this method is a test. Not all methods in a test class need to be test methods.

```
@isTest
private class MyBusinessLogicClass_Test {

    @isTest
    private static void myBusinessLogicTest(){
        // Load test data
        …
        // Perform the test
        …
        // Verify result
        …
    }

    . . .
}
```

Lesson Agenda

- Describing the Testing Framework and Requirements
- ▶ Creating Test Data and Tests
- Executing a Test
- Debugging
- Deployment

Loading Test Data

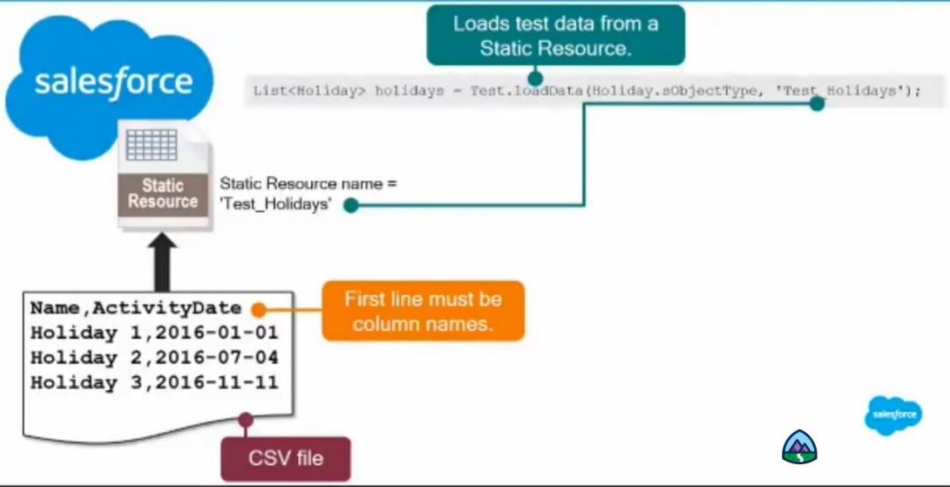
- Test data "factories" can be made accessible to all test methods within an org or just to test methods in a particular test class.
- A test data factory method annotated with `@testSetup` executes before any other method in the test class and provides data to all the test methods in the test class.

When would you use the test data factory class over the method, and vice versa?

```
1A @isTest
2A public class MyTestDataFactory {
3A     public static void insertTestAccounts() {
4A         //… create and insert Accounts
5A     }
6A     public static void insertTestContacts() {
7A         //… create and insert Contacts
8A     }
9A     public static void insertTestHolidays() {
10A        //… create and insert Holidays
11A    }
12A }
```

```
1B @isTest
2B private class MyBusinessLogicClass_Test {
3B     @testSetup
4B     private static void myTestDataSetupMethod() {
5B         //… create and insert Accounts
6B         //… create and insert Contacts
7B         //… create and insert Holidays
8B         //… Etc…
9B     }
10B    @isTest private static void myTestMethod1() {
11B        //… Use the test data created above
12B    }
13B    @isTest private static void myTestMeth
14B        //… Also uses the test data create
15B    }
16B }
```


Loading Test Data Declaratively Using a Static Resource



What Real Data Can a Test Method "See"?

158

```
1  @isTest
2  private class MyBusinessLogicClass_Test {
3
4      @isTest
5      private static void myBusinessLogicTest() {
6
7      }
8  }
```

Annotation can be applied at the class or individual test method level

```
1  @isTest(seeAllData=true)
2  private class MyBusinessLogicClass_Test {
3
4      @isTest(seeAllData=true)
5      private static void myBusinessLogicTest() {
6          List<User> users = [SELECT … ];
7      }
8  }
```

Standard Object Data	✗
Custom Object Data	✗
Setup Object Data	✓
Standard Object Data	✓
Custom Object Data	✓
Setup Object Data	✓

LIVE

Lesson Agenda

159

Describing the Testing Framework and Requirements
Creating Test Data and Tests
▶ Executing a Test
Debugging
Deployment

LIVE

Performing the Test

160

`startTest` and `stopTest` methods:

- Are used, in conjunction with methods of the Limits class, to validate how close your business logic code is to exceeding governor limits.
- Demarcate the boundaries of your actual test code within a test method.
- Provide a new context (i.e., an entirely new set of governor limits).

```
1 @isTest
2 private class AClass_Test {
3     @isTest
4     private static void myClassTestUser() {
5         // Load test data
6         Account anAccount = new Account (name = 'Salesforce');
7         insert anAccount;
8         System.debug(Limits.getDMLStatements());
9
10        // Perform the test
11        Test.startTest(); // Starts a brand new set of limits
12        System.debug(Limits.getDMLStatements());
13        Test.stopTest();  // Ends the set of limits started on line 7
14
15        // Verify result
16        System.debug(Limits.getDMLStatements());
17    }
18 }
```

1. What will line 8 print in the debug log?
2. What will line 12 print in the debug log?
3. What will line 16 print in the debug log?

LIVE

Verify the Test Result

Assertions

System.Assert(condition), System.AssertEquals(x, y), System.AssertNotEquals(x, y)

```
1  @isTest
2  private class MyBusinessLogicClass_Test {
3      @isTest
4      private static void myBusinessLogicTest() {
5
6          // Load test data - insert a course record
7
8          // Perform the test - insert a child Course Delivery record
9
10         // Verify result - query for a Course Delivery record and
11         // assert that we find exactly 1 record
12         System.AssertEquals(1, numberDeliveriesInserted, 'ERROR - Expected 1 Delivery record');
13     }
14 }
```

Optional error message

Expected Value

Actual value. If this value not equal to 1, a `System.AssertException` is raised and the test will fail

Testing Security Using `System.RunAs()`

`System.runAs(User u)` enables you to write test methods that allow you to specify the user context so that the user's security configuration is enforced.

```
1 @isTest
2 private class MyClass_Test {
3
4     @isTest
5     private static void myClassTestUser() {
6
7         // Load test data
8         User testUser = [SELECT Id, Name FROM User WHERE Profile.Name='Standard User'];
9         …
10        // Perform test
11        System.runAs(testUser) {
12            // Now, the rest of test runs as testUser.
13        }
14        // Verify result
15        …
16    }
17 }
```


How is Code Coverage Calculated?

Running tests causes the code coverage percentages to be calculated.

```
Code coverage percentage =
(number of covered lines) / (number of covered lines + uncovered lines)
```

Only executable lines of code are included.

You create a test method that only enters the first branch of the conditional.

1. What is the total number of lines in the `AccountTriggerHandler` class that need to be covered to achieve 100% code coverage?
2. How many lines did your test cover?
3. What is the code coverage percentage for that test?

```
1  // Logic for the AccountTrigger (before inser
2  public class AccountTriggerHandler {
3      public static void printPhone(List<Accoun
4          String debugString;
5          for (Account a :accounts){
6              debugString = a.Name + ': ';
7              if (a.Phone == NULL) {
8                  debugString += 'Null Phone';
9              } else {
10                 debugString += a.Phone;
11             }
12             System.debug(debugString);
13         }
14     }
15 }
```

Not Counted

Counted

When you test a condition that executes only one branch of a conditional, you will only "get credit" for testing that branch, and not the entire conditional.

Customizing the Debug Log

When using the Developer Console or monitoring a debug log, you can specify the level of information that gets included in the log.

- **Log category**: The type of information logged, such as information from Apex or workflow rules.
- **Log level**: The amount of information logged.
- **Event type**: The combination of log category and log level that specify which events get logged. Each event can log additional information, such as the line and character number where the event started, fields associated with the event, and duration of the event.

Remember Debug Logs are limited to 20mb per log

NOTE: The debug log does not include information from actions triggered by time-based workflows.

Customizing the Debug Log (Continued)

Change DebugLevel

Add | Remove

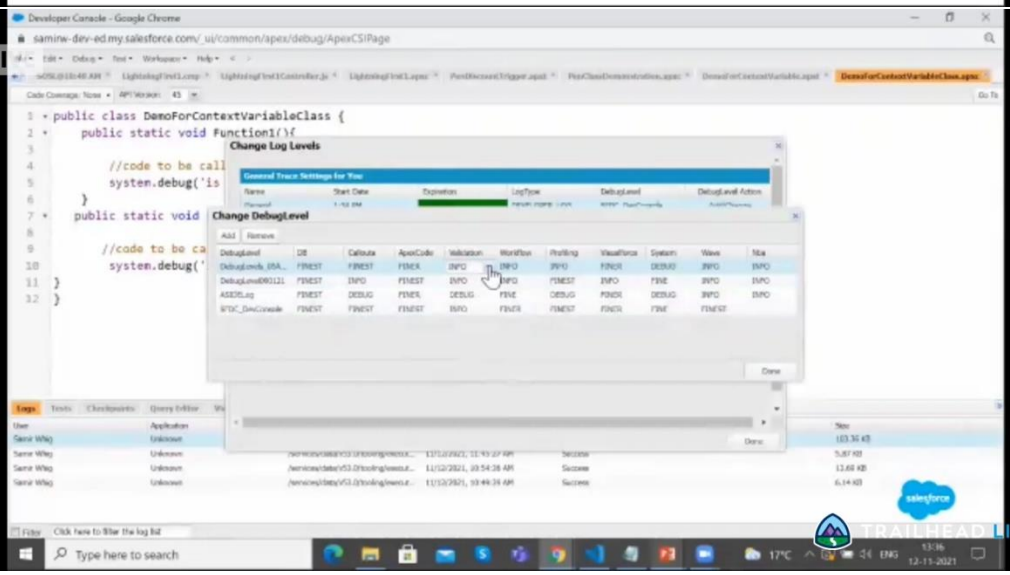
DebugLevel	DB	Callouts	ApexCode	Validation	Workflow	Profiling	Visualforce	System
SFDC_DevConsole	INFO	INFO	FINEST	INFO	INFO	INFO	INFO	DEBUG

Log categories:

- Database
- Workflow
- Callout
- Apex code
- Apex profiling
- Visualforce
- System

Log levels, from lowest to highest:

NONE
ERROR
WARN
INFO
DEBUG
FINE
FINER
FINEST



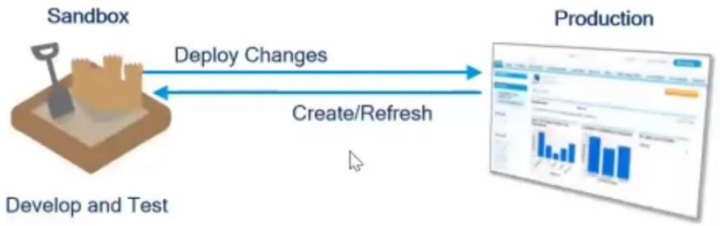
Lesson Agenda

- Describing the Testing Framework and Requirements
- Creating Test Data and Tests
- Executing a Test
- Testing Considerations
- Debugging
- ▶ Deployment

What is a Sandbox?

DEFINITION:

A sandbox is a replica of your production organization that allows you to develop and test in a separate environment without risking or compromising data.



Data Loader can be used to upload test data into sandboxes.

Salesforce Organization (Org) Types

- Production Orgs
- Trial Production Orgs
- Sandbox Orgs
- Developer Orgs
- Partner Developer Orgs
- Scratch Orgs

All Salesforce environments feature at least one Production org and usually one or more Sandbox orgs

Sandbox Types

Type	Features
Full Sandbox	▪ Copies the entire production organization, including the metadata and data. ▪ Has a storage limit based on its production organization. ▪ Can be refreshed every 29 days.
Partial Data Sandbox	▪ Copies the metadata and the data defined in a sandbox template from the production organization. ▪ Has a storage limit of 5GB of data and a maximum of 10,000 records per selected object. ▪ Can be refreshed every 5 days.
Developer Pro Sandbox	▪ Copies only the metadata from the production organization. ▪ Does not copy data, but you can load up to 1GB of data separately. ▪ Can be refreshed once per day.
Developer Sandbox	▪ Copies only the metadata from the production organization. ▪ Does not copy data, but you can load up to 200MB of data separately. ▪ Can be refreshed once per day.

Salesforce DX (SFDX)

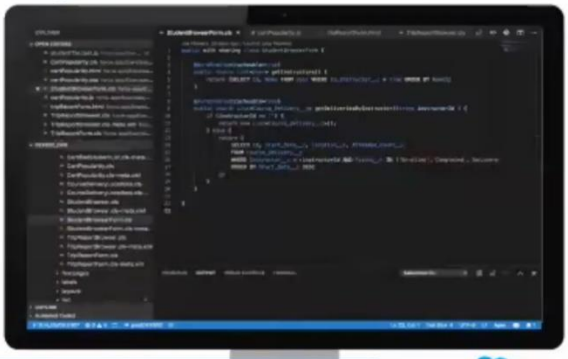
Build Together and Deliver Continuously

Source-Centric Development
Greater agility to test out features with confidence

Team Collaboration
Increased dev productivity, faster time to market

Continuous Integration and Delivery
Higher quality code, more automation

Open and Prescriptive
Build with the tools and processes you know and love; bring together Lightning, Force.com, and Heroku



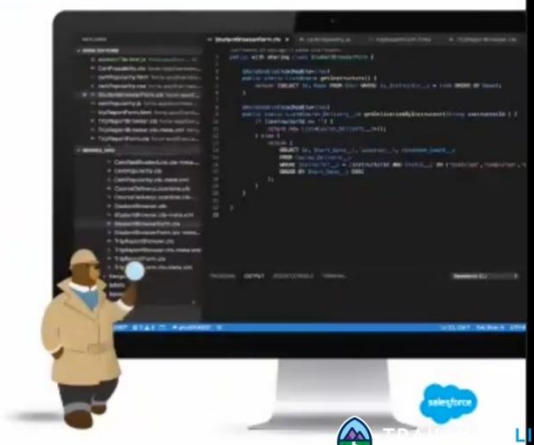
SFDX Provides you with an integrated, end-to-end lifecycle designed for high-performance agile development.

Open and Standards-Based Development Experience

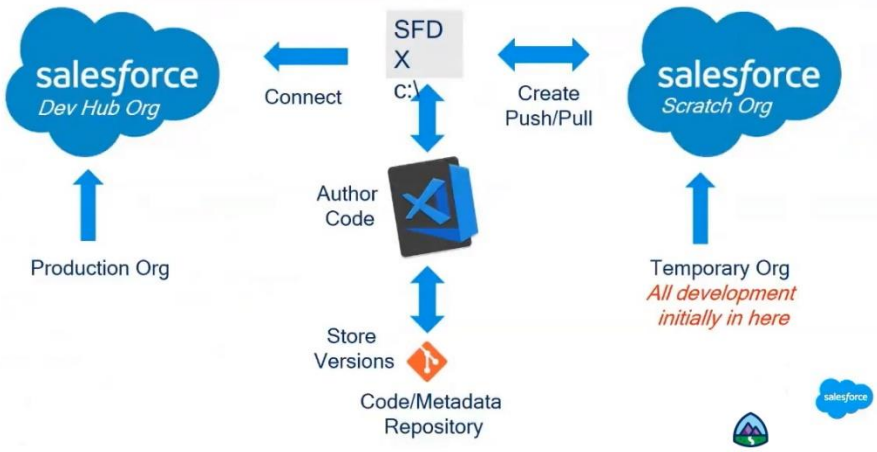
Development Options
Visual Studio Code, IntelliJ, Welkin, Cloud9, and more

Version Control Options
Git, CVS, Perforce, and more

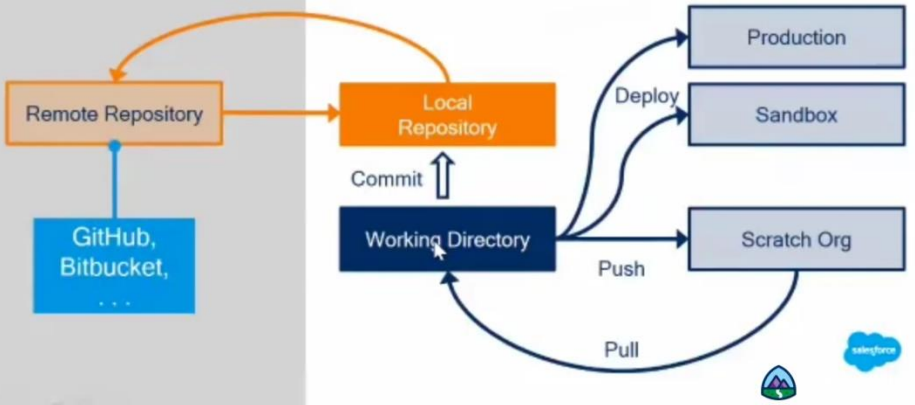
Build, Test, Deploy Options
Jenkins, Travis CI, Bamboo, JUnit, Selenium, Jest, Heroku Pipelines, and more



Development Environment



Working with Salesforce DX



What Can a Developer Tool Do?

Capability	Setup Menu	Developer Console	SFDX
Create and edit Apex classes and triggers.	✓	✓	✓
Execute anonymous Apex code.		✓	✓
Create and edit Visualforce pages.	✓	✓	✓
Create and edit Lightning Components.		✓*	✓
View debug logs.	✓	✓	✓
Run Apex tests.	✓	✓	✓
Run queries and view the results.		✓	✓
View the schema.	✓	✓	✓
Modify the schema.	✓		✓
Create a sandbox.	✓		Beta