Platform Developer - I Study Guide

Aashna Budhiraja (abudhiraja@salesforce.com)
October 2020

Exam Outline:

- **Salesforce Fundamentals: 7%**
- **Data Modeling and Management: 13%**
- **Process Automation and Logic: 38%**
- **User Interface: 25%**
- **Testing, Debugging, and Deployment: 17%**

Useful Resources:

- *Trailheads:*
  - Platform Developer 1 Trail
  - Visualforce & Lightning Experience
  - Study for your Platform Developer 1 Exam *(Good set of practice questions, attempt right before your exam)*
- *Super Badges:*
  - Process Automation Specialist
  - Apex Specialist
- Trello Board with topic-wise resources
- Video Series for Basics (2018)
- *Documentation/Developer Guide:*
  - Which Automation Tool Do I Use?
  - Using the with sharing/without sharing keywords
  - Junction Object
  - Apex sharing model
  - Apex Governor Limits
- *Blogs:*
  - When to Click instead of Write Code
  - Tips for Preparation
- *Pluralsight courses:*
  - Knowing When to Code in Salesforce
  - Taming the Salesforce Order of Execution

(P.S. I did not go through all of these resources in detail. Focus mainly on the trails and get the basics clear first.)

PD1 Trail - Synopsis

Index

Salesforce Fundamentals

Platform Development Basics

- Multitenant architecture
- Declarative vs. Programmatic development - When to Click instead of Write Code
- Metadata-driven development model
  - App: set of objects, fields, other functionality that supports a business process
  - Objects: tables in the Salesforce database
  - Records: rows in object database tables
  - Fields: columns in object database tables

- Salesforce Languages:
  - Lightning Component Framework: UI development framework similar to AngularJS or React.
  - Apex: Salesforce's proprietary programming language with Java-like syntax.
  - Visualforce: Markup language to create custom Salesforce pages with HTML like code, and can use a combination of Apex and JavaScript.
  - ➢ *Difference: Lightning components - can be pieced together to create pages*
    Visualforce - develop entire pages at once

- Lightning Platform APIs:
  - SOAP API - Integrate your org's data with other applications using standard SOAP protocols.
  - REST API - Access objects in your org using standard REST protocols.
  - Metadata API - Manage customizations in your org and build tools that manage your metadata model.
  - Tooling API - Build custom development tools for platform applications.
  - Marketing Cloud API - Expose Marketing Cloud capabilities with the REST API and get comprehensive access to most email functionality with the SOAP API.
  - Bulk API - Load, delete, and perform asynchronous queries on large data sets.
  - Streaming API - Send and receive notifications securely and efficiently. Notifications can reflect data changes in your org, or custom events.
  - Chatter REST API - Build UI for Chatter, Communities, Recommendations, Files, Topics, and more.
  - Mobile SDK - Integrate Native or Hybrid mobile apps directly with Salesforce.

- Heroku:
  - Web dev platform that lets you quickly build, deploy, and scale web apps.
  - Built on AWS
  - Heroku Connect - unifies Salesforce data with Heroku Postgres data

Apex & .NET Basics

- Object Oriented design
- Data Types:
  - Primitive types: Integer, Double, Long, Date, Datetime, String, Boolean
  - ID - 18 digit record ID
  - sObjects
- Collections:
  - List: Ordered, non-unique
  - Set: Unordered, unique
  - Map: key-value

- In Apex, all variables are initialized to null by default
- Differences:
  - .NET strings are actually references even though they behave like value types, because they're immutable. But in Apex, strings are always treated as a primitive value type.
  - Apex is not case sensitive
  - Apex and Database are Tightly Coupled
    - Each standard or custom object in the database is represented as an Apex class
    - The class and its underlying object are mirror images kept in sync.
    - You cannot add a reference to a non-existent class member (field).
    - You cannot delete an object or field that is referenced in Apex
  - Different Design Patterns
  - Unit Tests Are Required - must have 75% test coverage to deploy Apex code to a production org
  - No Solution, Project, or Config Files
    - There is also no such thing as a config file in the Lightning Platform world, because the database is baked in. You don't need connection strings
  - A much smaller Class library in Apex
  - Security Model
    - Identity is handled by the platform.
    - You can control access to data at many different levels, including object level, record level, and field level.
  - Integration
    - SOAP / REST APIs
    - Expose / Invoke web services using Apex
    - Third-party - AppExchange

- Methods of Invoking Apex:

Database Trigger	Invoked for a specific event on a custom or standard object.
Anonymous Apex	Code snippets executed on the fly in Dev Console & other tools.
Asynchronous Apex	Occurs when executing a future or queueable Apex, running a batch job, or scheduling Apex to run at a specified interval.
Web Services	Code that is exposed via SOAP or REST web services.
Email Services	Code that is set up to process inbound email.
Visualforce / Lightning Pages	Visualforce controllers and Lightning components can execute Apex code automatically or when a user initiates an action, such as clicking a button. Lightning components can also be executed by Lightning processes and flows.

  ➢ *A public method declared as static can be called w/out instantiating the class*
  ➢ *By default, Apex executes in system context (access to all objects and fields). Object permissions, field-level security, and sharing rules aren't applied for the current user, unless you use 'with sharing'*

- Triggers:
  - Only use trigger if you can't use OOB automation
  - before: insert, update, delete
  - after: insert, update, delete, undelete
  - trigger TriggerName on ObjectName (trigger_events) {
    ```
        // code_block
    }
    ```
- Execution Context: Everything between the EXECUTION_STARTED and EXECUTION_FINISHED in a log file
- Synchronous Limits:
  - 100 SOQL queries per transaction (context)
  - 150 DML statements per transaction (context)
  ➢ *Always Bulkify Code*

➢ *There is no such thing as an application or session variable in the Lightning Platform. If you need data to persist between classes, you have static variables but they can only persist information within a single execution context.*

- Asynchronous Apex:
  - When to use:
    ✓ Processing a very large number of records
    ✓ Making callouts to external web services
    ✓ Creating a better and faster user experience
  - Future methods
    - Use when: web service callout / offload processing
    - @future annotation
    - static and returns void
    - You can't track execution because no Apex job ID is returned.

        - Parameters must be primitive data types, arrays of primitive data types, or collections of primitive data types.
        - Can't take objects as arguments.
        - You can't chain future methods and have one call another.
    - Batches
        - Use when: large no. of records
        - Class implements Database.Batchable interface.
        - Define start(), execute(), and finish() methods.
        - Invoke using the Database.executeBatch method.
    - Queueable Apex
        - **Non-primitive types** - Classes can accept parameter variables of non-primitive data types, such as sObjects or custom Apex types.
        - **Monitoring** - jobId is returned
        - **Chaining jobs** - You can chain one job to another job by starting a second job from a running job.
        - *Id jobId = System.**enqueueJob**(new MyQueueableClass(contacts));*
    - Debug Log
        - System.debug('My Debug Message');
            - Logging Levels (cumulative, i.e. FINEST includes all levels)
                - NONE / ERROR / WARN / INFO / DEBUG / FINE / FINER / FINEST
            - Each debug log must be 20 MB or smaller
            - Each org can retain up to 1,000 MB of debug logs
            - The oldest logs are overwritten
    - Checkpoints - similar to breakpoints but they don't stop execution on that line.

Aura Components Core Concepts

Aura vs. Visualforce

- Page vs. Bundle
    - Visualforce pages - single entity [1 page + 1 metadata]
    - Aura component - bundle [8 code artifacts + 1 metadata]
- Server side vs. Client Side
    - Visualforce: runs "on the server side." All the "framework processing" happens **on the server.** User requests a page -> The server executes the page's underlying code and sends the resulting HTML to the browser -> The browser displays the HTML
    - Aura: User requests a component -> the component's resources are packaged up and sent to the requesting user's browser -> The browser loads the bundle -> The JavaScript application generates the UI
- Page-by-Page vs. Single Page Application
    - Visualforce: create set of pages, navigate from one to another

    - Aura: SPA loads, the JavaScript runs and updates the user interface of the component from then on. (State-to-state navigation)
- Page vs. Component
    - Visualforce page: large building block, stand alone page
    - Aua component: piece of a larger whole which is a collection of smaller components
- Controller Architecture
    - Visualforce: server side controller (Apex)
    - Aura: Client side controller (Javascript), can also have server side (Apex)

Data Modeling and Management

Data Modeling

- Standard / Custom Objects
- Object Relationships
    - Lookup:
        - Loosely coupled
        - Rollup summary not supported
        - Parent field optional on child
        - Can be one-to-one or one-to-many
    - Master-Detail:
        - Tightly coupled
        - Detail(child) record is deleted when master(parent) record is deleted
        - Master reference is always required on detail
        - Detail inherits security from master
        - Can choose if detail can be reparented
        - Detail object must be custom object
        - Rollup summary is supported
        - Lookup field on detail page layout is required
- Junction Object
- Schema Builder - tool to visualize and edit your data model

Data Management

- Import data
    - Data Import Wizard
        - Through Setup
        - Standard / custom objects
        - Upto 50,000 records
        - Use when:
            - You need to load less than 50,000 records
            - The objects you need to import are supported by the wizard

        - You don’t need the import process to be automated
  - Data Loader
    - Client application - UI/Command line
    - Up to 5 million records
    - Of any data type, from files or database connection
    - Possible to automate the import process using API calls.
    - Use when:
      - You need to load more than 50,000 records (up to 5 million)
      - The objects you need to import are not supported by the wizard
      - You need the import process to be automated (e.g. nightly imports)
    - ➢ *If you need to load more than 5,000,000 records, use an ETL tool / AppExchange*
  - Considerations:
    - If you import a picklist value that doesn’t match an existing picklist value:
      - Unrestricted picklist - Data Import Wizard uses the value
      - Restricted picklist - Data Import Wizard uses picklist’s default value
    - Multi-Select Picklists - Import multiple values into a multi-select picklist using semicolon between values
    - Checkboxes - checked = 1, unchecked = 0 in input file
    - Default Values - For picklist, multi-select picklist, and checkbox fields, if you do not map the field in the import wizard, the default value for the field, if any, is automatically inserted into the new or updated record
    - Date/Time Fields - Ensure that the format of any date/time fields you are importing matches how they display in Salesforce per your locale setting
    - Formula Fields - cannot accept imported data because they are read-only
    - Field Validation Rules - Salesforce runs validation rules on records before they are imported. Records that fail validation aren’t imported.
- Export Data
  - Data Export Service
    - In-browser wizard, accessible through the Setup menu
    - Export manually (Export Now) - Once every 7 days or 29 days
    - Export automatically (Schedule Export) - Weekly or monthly intervals
    - Can include images, documents and attachments
    - Salesforce creates a zip archive of CSV files, emails you when it's ready
    - Large exports are broken up into multiple files
    - Zip files are deleted 48 hours after the email is sent

    - Data Loader

Formulas & Validations

- Formulas:
    - Whitespace and line breaks do not matter
    - Formulas text comparisons are case sensitive
    - When working with numbers, the standard order of operations applies
    - Cross-Object Formulas
- Roll-up Summary fields:
    - Created on Master side of a master-detail relationship
    - Summarizes Date or Numerical data from detail record
    - COUNT / SUM / MIN / MAX
- Validation Rules:
    - Expression to evaluate data in one or more fields
    - Returns True / False
    - Can also return Error message to be displayed

Logic and Process Automation

Lightning Flow

- Introduction
    - Product - Lightning Flow
    - Tools - Process Builder, Flow Builder
- When to use what
    - Guided visual experience - Flow Builder
    - Behind-the-scenes automation - Process Builder / Flow Builder / Apex
    - Approval automation - Approvals
- Process Builder
    - Trigger - when the process should run
        - Only when a record is created / Anytime a record is created or edited
        - Process type
            - Record Change
            - Invocable by another process
            - Platform Event message is received
    - Criteria - whether to execute actions or not
        - Set filter conditions, or
        - Enter a custom formula, or
        - Opt out of criteria and always execute the associated actions
    - Action - what the process should do
        - It can:
            - Create records

                - Update the record that started the process or any related record
                - Submit that record for approval
                - Update one or more related records
                - Send emails using a specified email template
                - Post to a Chatter feed
            - Each immediate action is executed as soon as the criteria evaluates to true.
            - Each scheduled action is executed at the specified time. At the specified time, Salesforce makes sure that the associated criteria node still evaluates to true. If so, the scheduled action is executed. You can schedule actions based on either:
                - A specific date/time field on the record that started the process. For example, a month before an account's service contract expires.
                - The time that the process ran. For example, 3 days from now.

- Flow Builder
    - Flow – an application that automates a business process by collecting data and doing something in your Salesforce org or an external system.
    - Building Blocks:
        - Elements - appear on the canvas
        - Connectors - tell the flow which element to execute next
        - Resources - containers that represent a given value, such as field values or formulas
    - Types of Flow Variables
        - Variable - Single value
        - Collection Variable - Multiple values of the same data type
        - Record Variable - A set of field values for a single record
        - Record Collection Variable - A set of records of the same object type
    - ➢ *Add the Run Flows user permission to a permission set or profile*
    - ➢ *Only flow admins with Manage Flow user permission can run inactive flows*
    - ➢ *A flow interview is a running instance of a flow*
- Process Builder + Flow Builder
    - Process Builder can't:
        - Post to a community feed.
        - Submit a related record for approval.
        - Delete records.
        - Create a bunch of records and associate them with each other.
        - Perform complex logic.
        - You can't grab the ID of the created record and use it elsewhere. You can do that in a flow.

    - You can configure more complex functionality in a flow, and then add a flow action to your process. If necessary, you can write Apex and then add an Apex action.
- Approvals
    - Record submitted for approval
    - Initial submission action
        - Default = lock the record
        - Other options:
            - Update a field
            - Create a task
            - Send an email
            - Send outbound message
    - Approval steps
        - Assign approval requests to various users
        - Define the chain of approval for a particular approval process
    - Final approval actions
    - Final rejection actions

Apex Basics & Database

- Apex
    - Hosted – saved, compiled, and executed on the server
    - Object oriented – classes, interfaces, inheritance
    - Strongly typed – validates references to objects at compile time
    - Multitenant aware – enforces limits, which prevent code from monopolizing shared resources
    - Integrated with the database – provides direct access to records and their fields, provides statements and query languages to manipulate those records, provides triggers/transactions/rollbacks
    - Data focused – provides transactional access to the database, allowing you to roll back operations.
    - Easy to use – based on familiar Java idioms
    - Easy to test – provides built-in support for unit test creation, execution, and code coverage
    - Versioned – can be saved against different versions of the API
    - The 'global' access modifier, more permissive than the 'public' modifier and allows access across namespaces and applications
    - Case Insensitive
- sObjects
    - Every record in Salesforce is natively represented as an sObject in Apex
    - API names:
        - custom objects and custom fields - end with __c
        - custom relationship fields - end with __r

    - Generic sObject type:
        - can be cast to specific sObject
        - can be created only through the newSObject() method
        - fields can be accessed only through the put() and get() methods
- DML
    - Data Manipulation Language - Create and modify records
    - Statements: insert / update / upsert / delete / undelete / merge
        - upsert - creates new records and updates sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified
            - If the key is not matched, a record is inserted
            - If the key is matched, the record is updated
            - If the key is matched multiple times, an error is generated (no insert or update)
        - merge - merges up to three records of the same sObject type into one of the records, deleting the others, and re-parenting any related records
    - ID field is auto-assigned to new records
    - Bulk DML - DML on a list of sObjects counts as one DML statement
    - Two Ways to Execute DML in Apex:
        - Using the statement itself e.g. insert accts;
        - Database Methods:
            - Database.insert() / Database.update() / Database.upsert() / Database.delete() / Database.undelete() / Database.merge()
            - allOrNone - optional parameter to specify whether the operation can partially succeed.
                - FALSE - if an error occurs on a partial set of records, successful records are committed. Errors for failed records are returned. No exceptions are thrown.
                - TRUE - 1 error will cause the entire transaction to rollback, this is default
            - Database.SaveResult[] results = Database.insert(recordList, false);
    - Insert related records - by specifying foreign key ID for a defined relationship
    - Update related records - Fields on related records cannot be updated within the same DML call
    - Delete related records - supports cascading delete.
    - ➢ *You can enable cascading delete for lookup relationships (Cascade-delete bypasses security and sharing settings, which means users can delete records when the target lookup record is deleted even if they don't have access to the child records.)*
- SOQL
    - Basic syntax: SELECT fields FROM ObjectName [WHERE Condition]

- Cannot use * to query all fields since it is multi tenant
- WHERE / ORDER BY / LIMIT
- Bind variable: local variable within a SOQL statement, access by preceding with a colon (:).
- Child-to-Parent Query
  - dot notation
  - SELECT FirstName, LastName, Account.Name FROM Contact
  - Can go upto 5 levels up
- Parent-to-Children Query
  - nested select query
  - SELECT Name, (Select FirstName, LastName FROM Contacts) FROM Account
  - When working with relationship queries, the parent-to-child relationship name must be a plural name.
  - The results from the subquery consist of a JSON array, as one parent can have many children
  - Custom child: My_Object__c => Child relationship: My_Objects__r (Check the child relationship name on the related field, can be modified)
- Aggregates:
  - Results are returned as an AggregateResult type
  - Functions:
    - AVG() - average of a numeric field
    - COUNT(), COUNT(fieldName), COUNT_DISTINCT()
      - Returns the number of rows matching the query criteria
      - COUNT() returns an integer value, similar to COUNT(*) in SQL
      - Count(fieldName) returns an AggregateResult that counts the number of rows where the fieldName has a non-null value.
    - MIN() - minimum value of a field
    - MAX() - maximum value of a field
    - SUM() - sum of a numeric field
  - GROUP BY clause:
    - GROUP BY field
    - GROUP BY ROLLUP
    - ● GROUP BY CUBE
      - Add subtotals for all combinations of grouped fields in the query results
    - GROUPING
  - HAVING clause - filter the results that an aggregate function returns
- SOQL For Loops:
  - for (variable_list / variable : [soql_query]) { code_block }

        - It is preferable to use the sObject list format of the SOQL for loop as the loop executes once for each batch of 200 sObjects.
        - Avoids hitting governor limits
- SOSL
    - FIND 'SearchQuery' [IN SearchGroup] [RETURNING ObjectsAndFields]
    - SearchQuery
        - Can be grouped using AND / OR
        - Case insensitive
        - Wildcards:
            - * matches zero or more characters at the middle or end
            - ? matches only one character at the middle or end
            - Can negatively impact performance, especially if in the middle
    - SearchGroup
        - optional, default = ALL
        - (ALL / NAME / EMAIL / PHONE / SIDEBAR) FIELDS
    - ObjectsAndFields
        - optional, default - all objects
        - If you specify an object name without a field name in parenthesis, the search returns only the ID
        - can use WHERE / ORDER BY / LIMIT
    - Use when:
        - you don't know the exact fields and objects that your data resides in
        - you need to search across multiple objects, specially when they are not related (as SOQL only works with related objects)
    - It's possible for a SOSL search to NOT find all the matching results because of limits. e.g. searches that exceed 20,000 characters can tax the system

Apex Triggers

- Syntax:
```
trigger TriggerName on ObjectName (trigger_events) {
        Code_block
}
```
- Trigger events:
    - before (insert / update / delete)
    - after (insert / update / delete / undelete)
- Types:
    - Before - update or validate record values before they're saved to the database
    - After - access field values that are set by the system / affect changes in other records. The records that fire the after trigger are read-only.
- Context Variables:
    - isExecuting - true if the current context for the Apex code is a trigger
    - isInsert - true if fired due to an insert operation

- isUpdate - true if fired due to an update operation
- isDelete - true if fired due to a delete operation
- isBefore - true if fired before any record was saved.
- isAfter - true if fired after all records were saved.
- isUndelete - true if fired after a record is recovered from Recycle Bin
- New - list of the new versions of the sObject records.
  - only available in insert, update, and undelete triggers, and the records can only be modified in before triggers.
- newMap - map of IDs to the new versions of the sObject records.
  - only available in before update, after insert, after update, and after undelete triggers.
- old - list of the old versions of the sObject records.
  - only available in update and delete triggers.
- oldMap - map of IDs to the old versions of the sObject records.
  - only available in update and delete triggers.
- operationType - enum of type System.TriggerOperation corresponding to the current operation.
  - BEFORE_INSERT, BEFORE_UPDATE, BEFORE_DELETE, AFTER_INSERT, AFTER_UPDATE, AFTER_DELETE, AFTER_UNDELETE
- size - total number of records in a trigger invocation, both old and new.

- Trigger Exceptions:
  - addError() method throws a fatal error inside a trigger.
  - Error message is displayed in the user interface and is logged.
  - Affects all records if non-bulk DML or bulk DML without partial success
  - Affects only specific records if bulk DML with partial success
  - If a DML statement in Apex spawned the trigger, any error rolls back the entire operation. However, the runtime engine still processes every record in the operation to compile a comprehensive list of errors.
  - If a bulk DML call in the Lightning Platform API spawned the trigger, the runtime engine sets the bad records aside. The runtime engine then attempts a partial save of the records that did not generate errors.
- Callouts:
  - Calls to external Web services in Apex
  - Must be done asynchronously
  - Use @future(callout=true) annotation
- Bulk Trigger Design Patterns:
  - Triggers execute on batches of 200 records at a time. So if 400 records cause a trigger to fire, the trigger fires twice, once for each 200 records.
  - Limits:
    - SOQL queries: Synchronous = 100 , Asynchronous = 200
    - DML: 150 (sync or async)
  - Bulkify triggers:
    - Iterate over all available sObjects

            - Perform bulk SOQL and DML on collections of sObjects instead of single sObjects at a time

Database & .NET Basics

- SQL vs. SOQL:
    - SOQL has no such thing as SELECT *
    - SOQL has relationship queries instead of JOIN clause (Child-to-Parent / Parent-to-Children)
- Best Practices:
    - Build selective queries:
        - WHERE Clause
        - Use index fields to prevent full table scans
            - Certain fields are automatically indexed: Id / Name / OwnerId / CreatedDate / SystemModStamp / RecordType / Master-Detail / Lookup / Unique / External ID Fields
        - Exceptions: The following will make the query non-selective
            - Querying for null rows – Queries that look for records in which the field is empty or null
            - Negative filter operators – Using operators such as !=, NOT LIKE, or EXCLUDES in your queries
            - Leading wildcards – Queries that use a leading wildcard
            - Text fields with comparison operators – Using comparison operators, such as >, <, >=, or <=, with text-based fields
    - Always strive to avoid creating queries with non-deterministic formula fields (formula value can vary over time
- Query Plan Tool
    - Available in Developer Console
    - Help > Preferences > Enable Query Plan (select)
    - Provides a view into the Query Optimizer

Search Solution Basics

- Salesforce Search
    - Most used feature on Salesforce
    - How it works:
        - All records are stored as data fields in the org's database. When you update or create a record, the search engine comes along, makes a copy of the data, and breaks up the content into smaller pieces called tokens. We store these tokens in the search index, along with a link back to the original record.
        - When users enter a term in the search field
            - Search engine breaks up the search term into tokens

        - It matches those tokens to the record information stored in the search index
        - Ranks the associated records by relevance (based on search term frequency, order, and uniqueness, record activity, access permissions)
        - Returns the results that users have access to
- Connect with APIs:
    - Suggested Records API - Search Suggested Records and Search Suggested Articles REST methods
    - Salesforce Federated Search - Search for items stored outside of Salesforce from the Salesforce UI, making the global search box an external search engine
- Common Use cases:
    - Search Within a Single Object
        - FIND {term} RETURNING ObjectTypeName
    - Search Within Multiple Objects
        - FIND {term} RETURNING sObject1, sOjbect2, ..., sObjectN
- Optimize Searches:
    - Strategies:
        - Limit which data you're searching through
        - Limit which data you're returning
    - IN SearchGroup
    - RETURNING ObjectsAndFields
    - LIMIT n - max number of records returned
    - OFFSET n - sets the starting row offset into the result set
    - WITH - filter records by certain predefined fields
- Display Suggested Results
    - Search Suggested Records - Returns a list of suggested records (with shortcuts to same) whose names match the user's search string prior to performing a full search
    - Search Suggested Article Title Matches - Returns a list of Salesforce Knowledge articles (with shortcuts to same) whose titles match the user's search string prior to performing full search
- Create Synonym Groups
    - Setup > [Quick Find] > Synonyms > Custom Synonym Groups: New
    - 2 to 6 synonyms per group
    - Can be any word or phrase, no special characters
    - Upload Synonyms in Bulk via the Metadata API

Triggers and Order of Execution

On the server, Salesforce:

1. **Loads the original record** from the database or initializes the record for an upsert statement.
2. **Overwrites the old field values** with new values from request.
3. **If request from a standard UI edit page, system validations:**
    - Compliance with layout-specific rules
    - Required values at the layout level and field-definition level
    - Valid field formats
    - Maximum field length
4. **If other sources, (Apex/SOAP), validates only foreign keys** (any custom foreign keys do not refer to the object itself)
5. **If multiline items created, runs custom validation rules** (eg. Quote Line Item)
6. **Record-triggered flows** that are configured to run before the record is saved.
7. **Before triggers**
8. **System validation steps again** (except enforcement of layout-specific rules when request from standard UI edit page)
9. **User-defined validation rules**
10. **Duplicate rules** (If it identifies record as duplicate and uses block action, record is not saved and no further steps)
11. **Saves to database, but doesn't commit yet.**
12. **After triggers**
13. **Assignment rules**
14. **Auto-response rules**
15. **Workflow rules**
16. **If workflow field updates, updates record again**
17. If record updated with workflow field updates, **before update and after update triggers *one more time*** + **standard validations.** (Custom validation rules, flows, duplicate rules, processes, escalation rules not run again.)
18. **Processes / Flows launched by processes / Flows launched by workflow rules** (When a process/flow executes DML, affected record goes through save process)
19. **Escalation rules**
20. **Entitlement rules**

21. **Record-triggered flows** that are configured to run after the record is saved.
22. **Updates roll-up summary fields in parent record**. Parent record goes through save process. (similarly if grandparent updated, it goes through save process)
23. **Criteria Based Sharing** evaluation
24. **Commits all DML to database**
25. **Post-commit logic**, such as sending email

Platform Events Basics

- Event-Driven Software Architecture:
  - Publisher-subscriber model – a sender broadcasts a message that one or more receivers capture.
  - Events get sent whether or not receivers are listening
  - Near-real time
  - Decouples event producers from event consumers
- Components of Event-Driven Systems:
  - Event
    - A change in state that is meaningful in a business process.
  - Event message/notification
    - A message that contains data about the event.
  - Event producer
    - The publisher of an event message over a channel.
  - Event channel
    - A stream of events on which an event producer sends event messages and event consumers read those messages.
  - Event consumer
    - A subscriber to a channel that receives messages from the channel.
- Platform Events:
  - Special kind of Salesforce entity, similar in many ways to an sObject.
  - An event message is an instance of a platform event
  - Use platform events in the following cases:
    - To send and receive custom event data with a predefined schema
    - To publish or subscribe to events in Apex
    - For the flexibility of publishing and processing events on and off the Salesforce platform
  - Salesforce stores high-volume platform events for 72 hours
  - You can't query records through SOQL or SOSL
  - You can retrieve past events only when subscribing in CometD and using a ReplayId option
  - ReplayId - refers to the position of the event in the event stream

    - API name - system appends the __e suffix
    - Two kinds of publish behaviour:
        - Publish After Commit
            - transactional and can be rolled back
            - published only after a transaction commits successfully
        - Publish Immediately
            - not transactional and can not be rolled back
            - published when the publish call executes
    - Publish Events:
        - Apex - create an instance of the event and pass it to the EventBus.publish() method
        - No Code - record creation functionality in a process or flow to create platform event records
        - Salesforce APIs - create platform event records using SOAP/REST/Bulk API
    - Subscribe to Events:
        - Triggers - PEs support only after insert triggers
        - Aura/LWC - Lightning apps can use the empApi Lightning Web or Aura component to subscribe to events in the app
        - No Code - create a process that starts when a platform event message is received
    - ➢ *A trigger processes platform event notifications sequentially in the order they're received*
    - ➢ *A platform event trigger runs in its own process asynchronously*
    - ➢ *Platform event triggers run under the Automated Process system user*
    - ➢ *Platform event triggers are subject to Apex governor limits.*
    - ➢ *The maximum batch size in a platform event trigger is 2,000 event messages*

User Interface

Visualforce Basics

- Where to use:
    - Open a Visualforce Page from the App Launcher
    - Add a Visualforce Page to the Navigation Bar
    - Display within a Standard Page Layout
    - Component in the Lightning App Builder
    - Quick Action
    - Overriding Standard Buttons or Links
    - Using Custom Buttons or Links
- Introduction to Visualforce Pages:
    - Basic building blocks
    - Powerful features to access, display, update SF data

    - Can be referenced and invoked via a unique URL
    - Uses a tag-based markup language that's similar to HTML
    - Each VF tag corresponds to a coarse or fine-grained user interface component, such as a section of a page, a list view, or an individual field
    - Can be freely mixed with HTML, CSS and JavaScript
    - Includes 150 built-in components with the ability to create custom components
- Expressions:
    - Syntax: {! expression }
    - Anything inside the {! } delimiters is evaluated and dynamically replaced when the page is rendered or when the value is used.
    - Whitespace is ignored.
- Global Variables: (reference)
    - $Action - standard Salesforce actions such as displaying the Accounts tab home page or creating/editing/deleting accounts
    - $Api - API URLs
    - $Asset - images and other assets in the SLDS
    - $Cache.Org - Access an org cache
    - $Cache.Session - Access an org's session cache
    - $Component - Visualforce component
    - $ComponentLabel - label of an inputField component on a VF page associated with a message
    - $CurrentPage - current Visualforce page or page request
    - $FieldSet - Provides access to a field set defined in your organization
    - $Label - custom label
    - $Label.Site - standard Sites label in a Visualforce page
    - $Network - community details in a Visualforce email template
    - $ObjectType - standard or custom objects and the values of their fields
    - $Organization - information about your company profile
    - $Page - Visualforce page
    - $Permission - current user's custom permission access
    - $Profile - current user's profile
    - $Resource - existing static resource by name.
    - $SControl - existing custom s-control by name
    - $Setup - custom setting of type "hierarchy"
    - $Site - current Salesforce site
    - $System.OriginDateTime - literal value of 1900-01-01 00:00:00.
    - $User - current user info
    - $User.UITheme and $User.UIThemeDisplayed - Identify the Salesforce look and feel a user sees on a given Web page
    - $UserRole - current user's role
- MVC:
    - VF uses the traditional model–view–controller (MVC) paradigm

        - Model: Salesforce Database
        - View: Visualforce Page
        - Controller: Standard or Custom Controller
- Standard Controller:
    - Provide data access to the related sObject
    - Provide a set of standard actions (create, edit, save, and delete)
    - Provides simple and tight integration with the Lightning Platform database
    - Must use standard controller:
        - to embed in the sObject's page layout
        - to use as a custom action for the sObject
        - to use as the sObject mobile card in SF mobile
    - When you give ID parameter in url:
        - When page is loaded, and the <apex:page> component is activated, it activates a standard controller for the sObject.
        - The std. cont. sees that there's an ID parameter in the URL, and searches for and retrieves the matching record.
        - The std. cont. puts the record in a variable with the same name as the sObject, and dot notation can be used to access individual fields
- Output Components:
    - Coarse-grained components - provide a significant amount of functionality in a single component
    - Fine-grained components - provide more focused functionality
    - <apex:detail /> - will show the complete details for a record in a VF page using the standard controller for the related sObject
    - <apex:relatedList> - displays lists of records related to the current record
    - <apex:outputField> - displays individual fields from a record
    - <apex:pageBlockTable> - adds a table of data to a page
- Forms:
    - <apex:form> - Packages everything inside it into something that can be sent back to the server as part of a page action, used most of the time to send data back to Salesforce
    - <apex:inputField> - Creates an onscreen form field for the record data field that's associated with it via an expression. Renders the appropriate input widget based on a standard or custom field's type
    - <apex:commandButton> - Adds a button to the UI that fires a specific action when clicked
    - <apex:pageMessages> - displays any form handling errors or messages
- Standard List Controller:
    - Allows you to create Visualforce pages that can display or act on a set of records
    - Provides Query / Filter / Pagination of records
    - Loads a list (collection) of records into a variable

- Use with an iteration component (<apex:pageBlockTable> / <apex:dataList> / <apex:repeat>)
- Filtering:
  - Properties provided by the standard list controller:
    - {! listViewOptions } - get a list of list view filters available for an object
    - {! filterId } - set the list view filter to use for a standard list controller's result
  - Filter:

```
<apex:selectList value="{! filterId }" size="1">
    <apex:selectOptions value="{! listViewOptions }"/>
    <apex:actionSupport event="onchange" reRender="contacts_list"/>
</apex:selectList>
```

- Pagination:
  - By default, standard controller only shows the first 20 records
  - Previous/Next - action methods provided by standard list controller
  - Use 'rendered' with 'HasPrevious' and 'HasNext' to conditionally display Previous/Next

- Static Resources:
  - Upload content that you can reference in a Visualforce page (Archives/Images/Stylesheets/JavaScript/Other files)
  - Referenced using the $Resource global variable
  - Include different kinds of static resources:
    - <apex:includeScript> - JavaScript files
    - <apex:stylesheet> - CSS stylesheets
    - <apex:image> - graphics files
  - The URLFOR() function can combine a reference to a zipped static resource and a relative path to an item within it to create a URL that can be used with Visualforce components that reference static assets
- Custom Controllers:
  - Contain custom logic and data manipulation that can be used by a VF page
  - When your page uses a custom controller, you can't use a standard controller
  - Getter method: {! someExpression } in your Visualforce markup automatically connects to a method named getSomeExpression() in your controller.
  - Setter method: use the same name for the action method as defined in the controller class
- Apex Properties:
  - public MyObject__c myVariable { get; set; }
  - Alternative to getters and setters in Apex
  - Like a variable with getter and setter methods
  - Can be public or private

        - Can be read-only (omit set) or write-only (omit get)
        - Can perform additional logic before return/save
    - ➢ *There is no specific order in which getters, setters or properties are called*

Lightning Experience Development

- Classic VF
    - **UI Generation -** Server-side
    - **Data and Business Logic -** Apex standard or custom controller
    - **Workflow**
        - User requests a page
        - The server executes the page's underlying code and sends the resulting HTML to the browser
        - The browser displays the HTML
        - When the user interacts with the page, return to step one
    - **Pros**
        - Tried and true model
        - Easy to implement for greater productivity
        - Naturally splits large applications into small, manageable pages
        - Has built-in metadata integration
    - **Caveats**
        - Limited interactivity (aside from added JavaScript functionality)
        - Higher latency, which degrades mobile performance
- Visualforce as a JavaScript Application Container
    - **UI Generation -** Client-side (mostly JavaScript)
    - **Data and Business Logic -** Remote Objects or JavaScript Remoting, Apex controller
    - **Workflow**
        - The user requests the "empty" Visualforce page containing a page skeleton and JavaScript includes
        - The page is returned to the browser
        - The browser loads the JavaScript application
        - The JavaScript application generates the UI
        - When a user interacts with the application, the JavaScript modifies the user interface as needed (return to the previous step)
    - **Pros**
        - Enables highly interactive and immersive user experiences
    - **Caveats**
        - Complex
        - No built-in metadata integration

        - Lack of an integrated developer experience. The Lightning Platform Developer Console doesn't explicitly support these JavaScript applications so you have to load them as static resources.
- Lightning Components
    - **UI Generation -** Client-side (JavaScript)
    - **Data and Business Logic -** Lightning Data Service, Apex
    - **Workflow**
        - The user requests an application or a component
        - The application or component bundle is returned to the client
        - The browser loads the bundle
        - The JavaScript application generates the UI
        - When the user interacts with the page, the JavaScript application modifies the user interface as needed (return to previous step)
    - **Pros**
        - Enables highly interactive and immersive user experiences
        - Aligns with Salesforce user interface strategy
        - Built on metadata from the foundation, accelerating development
- VF in Lightning Experience:
    - What works:
        - Fundamental mechanisms are the same
        - Whether your pages use the standard controller, custom controllers, JavaScript remoting, or Remote Objects, your connection to Salesforce works the same.
        - Visualforce markup remains the same
        - Most customizations remain the same
    - What Works, but Needs Updating and Testing:
        - In LEX, Visualforce pages are embedded in an HTML iframe that's displayed inside the Lightning Experience app.
        - Can't directly access the 'window' global object in Javascript
        - Re-test if page makes use of iframe, either with <apex:iframe> or static HTML
        - Re-test if page embeds a Canvas app, using Canvas APIs
    - What Doesn't Work:
        - In LEX, Object tab and Object list are combined into Object Home
        - showHeader and sidebar attributes of <apex:page> have no effect on Visualforce pages in LEX
        - A number of related lists available in Classic aren't supported in LEX
- Lightning Components in LEX:
    - Advantages:
        - Out-of-the-Box Components
        - Performance (more responsive and efficient) - stateful client (JavaScript) and a stateless server (Apex) allows client to call server only when absolutely necessary

        - Event-Driven Architecture
        - Rapid Development
        - Device-Aware and Cross-Browser Compatibility
    - Where to use:
        - Lightning Experience
        - Salesforce Mobile App
        - Standalone Apps
        - Visualforce Pages
        - Lightning Out (Beta) - anywhere you can serve a web page
        - Screen Flows
    - ➢ *Apps listed on AppExchange are marked with a "Lightning Ready" sash if they're fully compatible with LEX.*
- User experience
    - VF - $User.UITheme and $User.UIThemeDisplayed global variables or <apex:stylesheet>
    - Javascript - UITheme.getUITheme()
    - Apex - UserInfo.getUiTheme() and UserInfo.getUiThemeDisplayed() system methods
- Static urls:
    - Visualforce - {!URLFOR($Action.Contact.Edit, recordId)}
    - JavaScript - navigateToSObject(recordId)

Lightning Web Components Basics

- LWC uses core Web Components standards and provides only what's necessary to perform well in browsers supported by Salesforce
- 3 steps:
    - HTML - provides the structure for your component.
    - JavaScript - defines the core business logic and event handling.
    - CSS - provides the look, feel, and animation for your component.
- Aura components can contain Lightning web components (though not vice-versa)
- Creating LWC
    - A component = a folder and its files with the same name
    - Namespace = separated from the folder name by a hyphen
        - eg. the markup for the Lightning web component with the folder name 'app' in the default namespace c is <c-app>
    - Camel case component folder names map to kebab-case in markup.
        - eg. In markup, to reference a component with the folder name myComponent, use <c-my-component>
    - HTML
        - <template> - defines the structure of your component

        - if:false / if:true - conditional directives within your template to determine which visual elements are rendered
    - Javascript
        - Must include at least this code, where MyComponent is the name you assign your component class.

            -
              ```
              import { LightningElement } from 'lwc';
              export default class MyComponent extends
              LightningElement { }
              ```
        - As a best practice, the name of the class usually matches the file name of the JavaScript class, but it's not a requirement.
    - Modules
        - LWC uses modules (built-in modules were introduced in ECMAScript 6) to bundle core functionality and make it accessible to the JavaScript in your component file.
        - Core module for Lightning web components is 'lwc'
        - import functionalities from a module
            - `import { LightningElement} from 'lwc';`
    - Lifecycle Hooks
        - Methods to 'hook' your code to events in the component lifecycle, including when it is
            - Created
            - Added to the DOM
            - Rendered in the browser
            - Encountering errors
            - Removed from the DOM
        - Callback methods
            - connectedCallback() is invoked when a component is inserted into the DOM.
            - disconnectedCallback() is invoked when a component is removed from the DOM.
    - Decorators
        - modify the behavior of a property or function
        - To use a decorator, import it from the lwc module and place it before the property or function.
        - Types:
            - @api:
                - Marks a field as public
                - Public properties define the API for a component
                - Reactive (When the property's value changes, the framework reacts and rerenders the component)

            - only fields that a component author decorates with @api are publicly available to consumers as object properties
        - @track:
            - Tells the framework to observe changes to the properties of an object or to the elements of an array
            - If a change occurs, the framework rerenders the component.
            - Reactive
            - Use @track only if a field contains an object or an array and if you want the framework to observe changes to the properties of the object or to the elements of the array.
        - @wire:
            - Gives you an easy way to get and bind data from a Salesforce org.
- Config file
    - Extension - .js-meta.xml
    - Required:
        - apiVersion - binds the component to a Salesforce API version.
        - isExposed - (true or false) makes the component available from other namespaces. Only set this to true to make a component usable in a managed package or by Lightning App Builder in another org.
    - Optional:
        - targets - specify which types of Lightning pages the component can be added to in the Lightning App Builder.
        - targetConfigs - let you specify behavior specific to each type of Lightning page, including things like which objects support the component.
- Displaying LWC
    - Set the component to support various flexipage types (home, record home) then add it to a flexipage using Lightning App Builder.
    - Create a tab which points to an Aura component containing your Lightning web component.
- Handling Events
    - Passing Information Up
        - Events and event listeners
        - The child component dispatches the event and the parent component listens for it.
        - Child: create and dispatch
            - this.dispatchEvent(new CustomEvent('**next**'));
        - Parent: listen and handle

            - <c-todo-item **onnext**={nextHandler}></c-child>
            - nextHandler() { ... }
        - Passing Information Down
            - Public properties and public methods
            - Child:
                - @api **itemName**;
                - @api **play**() { .... }
            - Parent:
                - <c-todo-item **item-name**="Milk"></c-todo-item>
                - this.template.querySelector('c-child').**play**();
    - Getting Salesforce Data
        - @wire - delivers a stream of data
        - How to use:
            - Import a wire adapter in the JavaScript file.
            - Decorate a property or function with the @wire decorator.
        - Syntax:
            - `import { adapterId } from 'adapter-module';`
              `@wire(adapterId, adapterConfig) propertyOrFunc;`
            - adapterId (Identifier)–The identifier of the wire adapter.
            - adapter-module (String)–The identifier of the module that contains the wire adapter function.
            - adapterConfig (Object)–A configuration object specific to the wire adapter.
            - propertyOrFunction–A private property or function that receives the stream of data from the wire service.
        - If a property is decorated with @wire, the results are returned to the **property's data** property or **error** property.
        - If a function is decorated with @wire, the results are returned in an **object** with a **data** property and an **error** property.
        - Example:

```
import {LightningElement, wire} from 'lwc';
import {getRecord} from 'lightning/uiRecordApi';
@wire(getRecord, {recordId: '$userId', [field]})
user;
```

Testing, Debugging, and Deployment

Apex Testing

- Apex Hammer - Before each major service upgrade, Salesforce runs all Apex tests in select customer orgs (which are copied and secured) through a process called Apex

Hammer, which runs tests in the current and next version of the service and then compares the results

- Code Coverage Requirement for Deployment
    - 75% of Apex Code
    - Each Trigger must have some coverage
- Test class - @isTest annotation
    - Private - unit testing only
    - Public - test data factory classes
- Test Method
    - @**isTest** static void testName() , OR
    - static **testMethod** void testName()
    - @isTest allows you to include annotations that you cannot with testMethod
    - The testing framework is always able to access test methods so declaring test methods as public or private does not matter
    - Declare them as **static** so that you do not need to create an instance of the test class
- Test Suite - Collection of Apex test classes that you run together (Dev Console > Test > New Suite)
- Test Data Creation
    - Apex tests don't have access to pre-existing data in the org, except setup and metadata objects (User/Profile)
    - Salesforce records that are created in test methods aren't committed to the database
    - @isTest(SeeAllData=true) - access existing records in the org
    - TestDataFactory - public class that is annotated with isTest and can be accessed only from a running test
    - Test utility classes contain methods that can be called by test methods to perform useful tasks, such as setting up test data.
    - Test utility classes are excluded from the org's code size limit.
- Test.startTest() / Test.stopTest()
    - Delimits a block of code that gets a fresh set of governor limits
    - Prevents test data setup from consuming limits the actual code consumes
    - Asynchronous Apex called after startTest() to immediately executes after stopTest()

➢ *Test classes annotated with @isTest don't count toward the 6mb limit for the total amount of Apex that can be stored in an org*
➢ *Test methods don't send emails*
➢ *Test methods can't make callouts to external services (use mock callouts)*
➢ *SOSL searches performed in a test return empty results*
➢ *To test Apex Triggers - Create records and perform DML*

Execute Anonymous Blocks

- An anonymous block is Apex code that does not get stored in the metadata, but that can be compiled and executed.
- Where to execute:
  - Developer Console (Debug > Execute Anonymous)
  - Salesforce extensions for Visual Studio Code
  - SOAP API call: ExecuteAnonymousResult executeAnonymous(String code)
- Can include user-defined methods and exceptions
- User-defined methods **cannot be static**
- If errors (i.e. in trigger, etc), any changes made to the database are **rolled back**
- Unlike classes and triggers, anonymous blocks **execute as the current user** and can fail to compile if the code violates the user's object- and field-level permissions
- Do not have a scope other than **local**
  - Though it is legal to use the global access modifier, it has no meaning
  - The scope of the method is limited to the anonymous block
- When you define a class or interface (a custom type) in an anonymous block, the class or interface is considered **virtual by default** when the anonymous block executes
- Even though a user-defined method can refer to itself or later methods without the need for forward declarations, variables cannot be referenced before their actual declaration.

Developer Console Basics

- Integrated Development Environment (IDE)
- Browser-based (nothing to install)
- Attached to a single org
- Changes effective immediately
- VS Code with Salesforce Extensions
  - Need to install and configure
  - Connect to multiple orgs
  - Compare or synchronize files
  - Use version control
- Set up different workspaces to organize each dev task (Developer Console > Workspace > New Workspace)
- Log Inspector
  - Debug > View Log Panels
  - Stack Tree - Displays log entries within the hierarchy of their objects and their execution using a top-down tree view
  - Execution Stack - Displays a bottom-up view of the selected item. It displays the log entry, followed by the operation that called it.
  - Execution Log - Displays every action that occurred during the execution of your code

    - Source - Displays the contents of the source file, indicating the line of code being run when the selected log entry was generated
    - Source List - Displays the context of the code being executed when the event was logged
    - Variables - Displays the variables and their assigned values that were in scope when the code that generated the selected log entry was run
    - Execution Overview - Displays statistics for the code being executed (e.g. time, heap)

Asynchronous Apex

- Used to run processes in a separate thread, at a later time
- Advantages
    - User efficiency (they don't have to wait)
    - Scalability (handle more jobs in parallel)
    - Higher Limits (start a new thread with higher governor and execution limits)
- Types
    - Future Methods
        - Run in their own thread
        - Do not start until resources are available
        - Good for, e.g. web service callout
    - Batch Apex
        - Run large jobs that would exceed normal processing limits
        - Good for, e.g. eata cleansing or archiving of records
    - Queueable Apex
        - Similar to future methods, but provide additional job chaining and allow more complex data types to be used
        - Good for, e.g. sequential processing with external Web services
    - Scheduled Apex
        - Schedule Apex to run at a specified time
        - Good for, e.g. daily or weekly tasks
- You can use a combination of types (e.g. scheduled apex invoking batch apex), but there are constraints (e.g. cannot call an @future method from an @future method)
- Increased Governor and Execution Limits
    - SOQL query limit increases from 100 to 200
    - Heap size increases from 6mb to 12mb
    - Max CPU time increases from 10,000 milliseconds to 60,000 milliseconds
- Constraints
    - 50 @future method invocations in synchronous mode
        - Reduced to 0 if asynchronous via @future
        - Reduced to 1 if asynchronous via Queueable
    - 50 Maximum number of Apex jobs added to the queue with System.enqueueJob

        - Reduced to 1 if asynchronous

- Future Methods:
    - Used for:
        - Callouts to external Web services (@future(callout=true)
        - Callouts from a trigger or after performing DML in Apex requires @future, to avoid holding the database connection for the lifetime of the callout
        - Resource intensive operations that can run in their own thread
        - Isolating DML operations on different sObject types to prevent mixed DML errors
    - Not guaranteed to execute in the same order they are called
        - If you need this type of functionality, use Queueable Apex
        - Two future methods could run concurrently, which could result in record locking and a nasty runtime error if the two methods were updating the same record
    - Future methods can't be used in Visualforce controllers in getMethodName(), setMethodName(), nor in the constructor
    - The getContent() and getContentAsPDF() methods can't be used in @future methods
    - Syntax
        - Must be **static**
        - Can only **return void**
        - Input parameters can only be **primitives** or collections of primitives
    - Best Practices
        - Avoid designs that add large numbers of future requests over a short period of time
        - Ensure that future methods execute as fast as possible
        - If using Web service callouts, try to bundle all callouts together from the same future method, rather than using a separate future method for each callout
        - Conduct thorough testing at scale, e.g. test that a trigger enqueuing the @future calls is able to handle a trigger collection of 200 records.
        - Consider using Batch Apex instead of future methods to process large number of records asynchronously - more efficient than creating a future request for each record
- Batch Apex:
    - Class includes three methods
        - start
        - execute
        - Finish
    - Syntax
        - Class **implements Database.Batchable**

- Invoking a Batch Class
  - MyBatchClass myBatchObject = new MyBatchClass();
    Id batchId = **Database.executeBatch**(myBatchObject);
  - Include scope parameter to stipulate # of records to process in each batch

    Id batchId = Database.executeBatch(myBatchObject, **100**);
  - Track the job:

    AsyncApexJob job = [SELECT Id, Status, JobItemsProcessed,
    TotalJobItems, NumberOfErrors
    FROM AsyncApexJob WHERE ID = :batchId];
- Using State in Batch Apex
  - Database.Stateful
  - Only instance member variables retain their values between transactions
  - Make sure that the number of records inserted is less than the batch size of 200 because test methods can execute only one batch total
- Best Practices
  - Ensure fast execution of batch jobs
  - Minimize Web service callout times
  - Only use Batch Apex if you have more than one batch of records. If not, use Queueable Apex
  - Tune any SOQL query to gather the records to execute as quickly as possible
  - Minimize the number of asynchronous requests to minimize the chance of delays
  - You must guarantee that the trigger won't add more batch jobs than the limit

- Queueable Apex:
  - Queueable vs @future:
    - Accept **non-primitive types** as parameters (sObjects/Apex Types)
    - **Monitoring**
      - Invoke via System.enqueueJob to get AsyncApexJob Id
      - Use Id to identify your job and monitor its progress from UI (Setup > Apex Jobs) or query (AsyncApexJob)
    - **Chaining** Jobs
      - Submit another job from the execute() method of your queueable class
      - Can add only one job from an executing job
      - There is no limit on the depth of chaining
  - Syntax
    - public class SomeClass **implements Queueable** {
      public void **execute**(QueueableContext context) { ... }

    - Queueable Apex jobs count against the shared limit for Asynchronous Apex
        - Can add up to 50 jobs to the queue with System.enqueueJob in a single transaction
        - Can only chain 1 job from an executing job with System.enqueueJob
        - No limit is enforced on the depth of chained jobs
        - For Developer Edition and Trial orgs, the maximum stack depth for chained jobs is 5

- Scheduled Apex:
    - Schedule execution of Apex to a specific time
    - Syntax
        - **global** class SomeClass **implements Schedulable** {
          global void **execute**(SchedulableContext ctx) { ... }
    - After a class has been scheduled, a **CronTrigger** object is created for the scheduled job, which provides a getTriggerId method that returns the ID of a CronTrigger API object
    - **System.Schedule** Method
        - Use extreme care when planning to schedule a class from a trigger. Must be able to guarantee that the trigger won't add more scheduled job classes than the limit
        - System.schedule('name', cron_exp, schedulable_class')
    - Scheduling a Job from the UI
        - Setup > [Quick Find] Apex > Apex Classes > Schedule Apex
    - **Max 100** scheduled Apex jobs at one time
    - There is also a maximum number of scheduled Apex executions per 24-hours
    - Synchronous Web service callouts are not supported from scheduled Apex
        - Batch Apex called from Scheduled Apex can make Synchronous Callouts

Debug Logs

- Debug logs have the following limits.
    - Each debug log must be 20 MB or smaller. Debug logs that are larger than 20 MB are reduced in size by removing older log lines
    - System debug logs are retained for 24 hours. Monitoring debug logs are retained for seven days.
    - If you generate more than 1,000 MB of debug logs in a 15-minute window, your trace flags are disabled.
    - When your org accumulates more than 1,000 MB of debug logs, we prevent users in the org from adding or editing trace flags.

Miscellaneous Notes

- **Ask these questions for each feature for a better understanding:**

  1. When should this feature be used?
  2. How does this feature work?
  3. What is the customisation potential of this feature?
  4. When should you NOT use this feature?
  5. What are similar features, and when should they be used instead?
  6. How do you report / monitor the feature?
  7. How does this feature work within Salesforce's security model?
  8. Are there any limits around this feature?

- If your organization uses multiple currencies, the currency of the master record determines the currency of the roll-up summary field. For example, if the master and detail records are in different currencies, the detail record value is converted into the currency of the master record.

- **Sharing**:
  - By default, Apex executes in system context. Apex code has access to all objects and fields. Object permissions, field-level security, and sharing rules aren't applied for the current user. - Using the with sharing or without sharing keywords.
  - executeAnonymous always executes using the full permissions of the current user
  - The sharing setting of the class where the method is defined is applied, not of the class where the method is called.
  - Inner classes do **not** inherit the sharing setting from their container class.
  - Classes inherit this setting from a parent class when one class extends or implements another.

- Create a dependent relationship that causes the values in a picklist or multi-select picklist to be dynamically filtered based on the value selected by the user in another field.
  - The field that drives filtering is called the "controlling field." Standard and custom checkboxes and picklists with at least one and **less than 300 values** can be controlling fields.
  - The field that has its values filtered is called the "dependent field." **Custom picklists and multi-select picklists can be dependent fields.**

- Salesforce evaluates the setup items in this order:
    1. Validation rules
    2. Assignment rules
    3. Auto-response rules
    4. Workflow rules and processes (and their immediate actions)
    5. Escalation rules

- **Processes Don't Evaluate Record Changes When:**
    - Campaign statistic fields, such as individual campaign statistics or campaign hierarchy statistics, are updated.
    - Picklist values are mass replaced.
    - Address fields are mass updated.
    - Divisions are mass updated.
    - Territory assignments of accounts and opportunities are modified.
    - Self-Service Portal, Customer Portal, or partner portal users are deactivated.
    - State and country data are converted with the Convert tool.
    - Values for state and country picklists are modified using AddressSettings in the Metadata API.

- Each process runs in the context of a transaction. A transaction represents a set of operations that are executed as a single unit. When a process is triggered more than once in a single transaction, Salesforce executes similar actions in one batch.