* Apex object and field level security
  + On SOQL queries, we can use the “WITH SECURITY\_ENFORCED” to enforce object and field level security we have set for the org
  + This is overkill
    - If you don’t have permission to see a field or the object returned in a query, it will throw an exception rather than filter the fields or objects you don’t have permission for
    - Throws a system.QueryException
  + If want just want to remove the fields or objects instead of an error, use Security.stripInaccessible()
    - Takes 2 parameters
    - An access type
      * This is an enum
      * Either AccessType.Createable, AccessType.Readable, AccessType.Updateable or AccessType.Upsertable
    - A list of sObjects
      * This is the list to strip
    - Returns an instance of sObjectAccessDecision class which has a method getRecords that returns the list of objects only containing what you have access to see
      * Id fields are never stripped unless you do not have access to the object
  + Using the schema namespace
    - Contains several classes
      * describeSObjectResult class
        + Contains methods to check if an object is accessible, createable, deleteable and updateable for each object in your database
        + Return true if you have those permission and false if you do not
        + Also has member variables for each schema object in your database
        + This also works for fields where we have the describeFieldResults class that has a fields instance/member variable that we can use with dot notation to access the describe field objects for the sObject fields

Each field will also have isAccessable, isCreateable and isUpdateable methods that return a Boolean defining the access that the user has to that fields

* Manual apex sharing
  + Process of programmatically sharing records with users
  + Unlike manual sharing, this will persist across ownership changes
  + System objects are things salesforce uses behind the screen to keep track of certain information
    - Approval objects
    - Share objects
  + Share objects
    - Created for all custom and standard objects by the system
    - Naming convention for these objects is:
      * Custom objects
        + NameOfTheObject\_\_Share
      * Standard objects
        + NameOfTheObjectShare
    - These objects have 4 fields we can see and interact with
      * AccessLevel
        + Determines what access we are giving to the user
        + Can be read or edit
      * ParentID
        + Holds the ID of the record we wish to share
        + NameOfObject(D for standard objects
      * RowCause
        + Interesting
        + Do not have to set it
        + Is a picklist field that determines the reason why we are sharing this record
      * UserOrGroupID
        + ID of the user or group that we wish to share the record with
* Asynchronous Apex
  + Asynchronous code doesn’t need to return or finish executing while the rest of the program finishes
  + They will execute in parallel rather than in series
  + With asynchronous apex, we get additional governor limits
    - New set
    - Larger set
  + Different tools
    - Use @future methods
      * Simplest
      * Static methods that have @future annotation above the method
      * Must return void
      * Can have any number of primitives as parameters but cannot use sObjects as parameters
        + Instead, pass the ID as a parameter
      * Most common use case are DML operations
      * Some setup objects can only be operated on in their own transaction in which o other record of a different is operated on
      * Testing
        + Writing a generic test for the future methods, they will fail
        + To fix this, write it in Test.startTest() and Test.stopTest()

Put it in the test method, not the class itself

* + - * Future methods cannot call one another
    - Queueable apex
      * We can queue one queueable job and only one from another
      * Each job has an ID to keep track of execution
      * Must implement the queueable interface
      * Must be public or global
      * Must implement all methods in that implement execute()
      * Method must be public or global and have a void return type
      * It has a single parameter and instance of the QueueableContext interface
      * Inside the execute method we perform our logic we want to execute async
      * We can invoke the queueable class by instantiating it and calling System.enqueueJob(IDofJob)
    - Batchable apex
      * Allows us to process large amounts of records by breaking them into smaller groups
      * Think millions
      * Create a class that implements the database.batchable<sObject> interface
      * Need to implement
        + Start method

Must be public or global and must return instance of database.queryLocator class

Can be created by database.getQueryLocator(‘mySOQLQuery’)

Has a parameter of batchable context

Taken case of for us

* + - * + Execute method

This is the logic we actually want to perform on our records

It must be public or global

Must have void return type

Takes two parameters

Instance of the database.batchableContext

List of sObjects

This is called once per batch

By default, breaks into batches of 200

* + - * + Finish method

Must be public or global

Return void

Contain clean u finish logic

Has a single parameter of database.batchableContext

* + - Schedulable apex
      * If we want to have our code execute at some point in the future on a specific date/time we can use schedulable apex
      * You create one by implementing the schedulable interface
      * We only have to implement the execute method
        + Must be public or global
        + Must return void
        + Must take in a parameter of the schedulable context
      * To invoke, we pass the name for our job, a CRON expression and an instance of our class into the System.schedule() method
      * CRON expression is a series of 7 characters to determine when to run the job
      * Run based on your default time zone for your org
* Dynamic Apex
  + Making our code more flexible to changes over time
  + Allowing for anything the user might enter
  + Can get the picklist values of a field without having to hard code it every time
    - Use the schema class to get this information
* Dynamic SOQL
  + We want to be input agnostic
    - People are stupid, account for it
  + Variable binding using the query() method of the database class
  + Conditionally creating our queries
* SOQL injection
  + Inputting malicious code intended to break what you wrote
  + Your website is less secure when you allow for input to your queries
  + Use string.escapeSingleQuotes() to get rid of single quotes
    - Also works in SOSL
* Apex best practices
  + Write self documenting code
  + Always write documentation
  + Need to write good tests4focus on getting to 75% test coverage but we need t make sure we are also following best testing practices
  + Do not test against production data
  + Bulkify code
  + Write efficient code
  + Do not put DML, SOQL or SOSL operations in loops
  + Perform database operations on bulk records
  + Be selective in your queries
  + Use the limits class to monitor usage
  + Use asynchronous apex if you are getting close to the limits
  + Use one trigger per object
    - Bulkify triggers
    - Make trigger logicless
    - Avoid recursion
  + Never hard code an ID when creating a new record
* Integrating with external systems
  + APIs
    - Application programming interface
    - Part of the server that gets requests from a client, parses them, tells the rest of the system what to do and returns a response from the server based on the results
    - When we interact with it programmatically it will send us back XML or JSON in the response body of the request
    - There are different types of web services that communicate in different ways
      * We will talk with SOAP
    - The request knows where to go based on the endpoint or URL that is attached to the request
    - A URL consists of a domain and a URI (unique resource identifier)
      * URL is uniform resource locator
    - HTTP
      * Hypertext transfer protocol
      * Common communication protocol seen on the internet
      * SOAP is built on HTTP and REST uses HTTP for communication
      * Methods/verbs are what we use to explain what we are trying to do
      * Get
        + Idempotent and not safe
        + It is used to retrieve a resource at a target endpoint
      * Put
        + Will either create a resource at a URL or overwrite the resource that exists at that URL
        + Idempotent and safe
      * Post
        + Same as put but instead of updating at that resource we will be redirected to that resource we want to update
        + Not idempotent or safe
      * Patch
        + Updates parts of the existing resource at a URL
        + Doesn’t overwrite entirely
        + Not idempotent or safe
      * Delete
        + Deletes a record at a resource
        + Is idempotent but not safe
      * Idempotent
        + Regardless of how many times you perform an action the result of that action will not be affected by previous calls
      * Safe
        + Means you do not affect the database
  + HTTP status codes
    - Is contained in the response sent back from the serer, it indicated the outcome of the operation
    - 100s
      * Information
      * Indicate how the operation invoked by our request is proceeding while it is being processed
    - 200s
      * Means success
    - 300s
      * Redirect
      * The resource you are looking for has been moved
    - 400s
      * Client side errors
      * Things the client does wrong
    - 500s
      * Server error
      * Things that are wrong with the server of the application