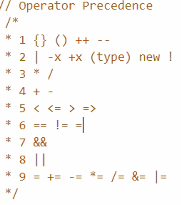
* Continuing Apex
  + Operator precedence
    - 
* SOQL
  + Similar to SQL
  + Salesforce Object Query Language
  + Stores query results in a list
  + Not case sensitive
  + Example
    - List<Contact> ExampleContacts = [SELECT id FROM Contact]
  + Optional clauses
    - Where
      * Criteria the FROM must meet to be added to the returned list
      * [SELECT id FROM Contact WHERE FirstName = ‘Steve’]
    - Like
      * Can be used in place of an equals
      * Accounts for wildcards
      * [SELECT id FROM Contact WHERE LastName LIKE ‘Rogers%’
        + % allows for 0 or more characters
      * \_ accounts for 0 or 1 characters
    - Limit
      * Limits number of records that are returned
      * [SELECT id FROM Contact LIMIT 1]
        + This would return the first record
    - Offset
      * Determines the starting row for the query
      * [SELECT LastName FROM Contact LIMIT 100 OFFSET 50]
        + Skips records 1-50 and starts at 51 and still returns 100 records
    - With
      * [SELECT LastName FROM Contact WITH SECURITY\_ENFORCED]
        + Ensures the person accessing the object or field has access to it
    - AGGREGATE
      * Get back an aggregate function from a query
      * Average
      * Count
      * Min
      * Max
      * Sum
      * AggregateResult Object is what is returned when working with aggregate functions
    - Group By
      * Group records by specific values
      * Great for picklists
      * List<AggregateResult> ExampleContact = [SELECT Count(LastName), account.name FROM Contact GROUP BY account.name];
    - Having
      * Allows for filtration of results of an aggregate function
      * List<AggregateResult> ExampleContact = [SELECT Count(LastName), account.name FROM Contact GROUP BY account.name HAVING Count(LastName) > 2];
    - Date Functions
      * Allow you to parse date fields to get specific information with aggregates
      * List<AggregateResult> ExampleContact = [SELECT Count\_Distinct (LastName), Calendar\_Year(Created\_Date) FROM Contact GROUP BY Calendar\_Year(Created\_Date)];
    - Format
      * [SELECT FORMAT(lastmodifieddate) FROM Contact]
    - For Reference
      * [SELECT LastName FROM Contact LIMIT 1 FOR REFERENCE]
    - For Update
      * [SELECT LastName FROM Contact FOR UPDATE]
      * Locks a record while being transacted
    - Relationship Queries
      * Return information from related records
      * 2 types
        + Child to parent

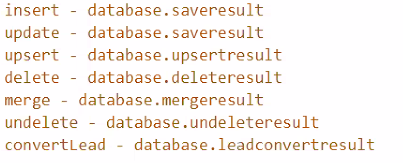
List<contact> ExampleContacts = [SELECT lastname, account.name FROM Contact]

Can go up to 5 levels

* + - * + Parent to child

List<Account> ExampleContacts = [SELECT name, (SELECT lastname FROM Contacts) FROM Account]

Note that contacts is plural here

* DML
  + Data manipulation language
  + Insert
    - Insert a new record into the org
    - Must have all required fields for the object
      * Don’t need every field, just the required one
    - Example
      * Account teamShield = new Account(Name=’Captain America’);
      * Account teamHydra = new Account(Name=’Baron Zemo’);
      * INSERT teamShield;
      * System.debug(treamShield.ID);
        + Will have an ID at this point
      * INSERT teamHydra;
  + Update
    - Edit an existing record
  + Upsert
    - Updates records in the org and inserts any ones that do not already exist
  + Delete
    - Deletes records from the database
    - Cannot drop tables, can only delete records
  + Merge
    - Combination of update and delete
    - Takes 2 arguments
      * The record that will stay and the list to update
    - Example
      * Merge teamShield teamHydra;
  + Undelete
    - Recover a deleted record
  + Database class methods
    - database.insert()
    - database.update()
    - database.upsert()
    - database.delete()
    - database.merge()
    - database.undelete()
    - database.convertLead()
      * Converts lead to account or contact
      * Takes 2 arguments, the second of which is allOrNone
    - Perform the same functionality as the database statements themselves
    - The biggest difference is the “all or none” parameter
      * database.insert(teamShield, allOrNone=false);
      * Keeps the operation atomic
      * Allows bypass of issues
    - Each method has a return type you can access that will tell you the success or failure of the operation
    - 
    - 3 methods
      * getID()
        + Returns ID of associated records
      * getErrors()
        + Returns errors associated with a record
      * isSuccess()
        + Returns Boolean if the operation was successful for the corresponding record
    - Transaction control
      * savepoint sp = database.setSavepoint();
      * database.rollback(sp);