Lesson 8: SQL & Databases:

**Questions for Mentor:**

**SQL & Databases Overview:**

* Database is an organized collection of structured information and a collection of virtual tables that store that information
* DBMS=Database management system
* Relational databases store data in a series of tables with links connecting each table
* Entity: a type of thing, not a thing itself. An entity would be the general class of things: Company, Tree, or State. An entity would not be Oracle, Redwood, or Virginia
* Record: a specific example of the entity (a row)
* Field: an attribute of a record
* Primary key: a unique value so we can differentiate records
* Create a table
  + CREATE TABLE products (
  + productID int NOT NULL,
  + productName varchar(70) NOT NULL,
  + productLine varchar(50) NOT NULL,
  + quantityInStock int NOT NULL,
  + price decimal(10,2) NOT NULL,
  + PRIMARY KEY (productID)
  + );
* Set theory
* Set = an **unordered** collection of **elements** with absolutely **no duplicates**
* Intersections = x y means anything in set x **AND** y



* Unions = x y means anything in set x **OR** y



* Set differences are how we can figure out difference between two sets (x – y)
* Relative complements are basically the opposite of set differences (y \ x) and will return what is in y that is not in X
* Symmetric difference or disjunctive union = all elements that exist in either x or y, but not both
  + Denoted with (aka x y will show whatever is in x or y, but not both)

**Introduction to SQL for Data Science:**

* Selecting columns syntax: SELECT column(s) FROM table;
* DISTINCT keyword (after SELECT) allows you to only select the unique values from a column
* COUNT() statement (after SELECT) allows you to count the number of records in one or more columns
  + Can also apply DISTINCT within the COUNT STATEMENT
* WHERE clause allows you to filter using conditional expressions (=, >, < etc)
  + Comes after FROM statement
* AND clause plays off of the WHERE clause and adds additional conditions
* OR clause plays off of the WHERE clause and adds additional conditions
* BETWEEN allows you to filter values within a specified range
  + It is inclusive, it includes the beginning and end of the range
* IN operator allows you to specify multiple values in a WHERE clause which is easier than many OR clauses
* IS NULL lets you check to see if a value is null
* LIKE / NOT LIKE operator can be used in a WHERE clause to search for a pattern in a column
  + Use wildcards like % (matches 1+ characters) or \_ (matches only one character)
* Can use some aggregate functions with SQL
  + Used after SELECT – AVG(), SUM() MIN() etc.
  + Can also be combined with WHERE clause to get further insight into data
* Can also use basic arithmetic with symbols like +, -, \*, /
  + Note: when dividing, if you want a float returned, your formula must contain floats
* Aliasing in SQL is assigning temporary aliases to something using AS keyword after the SELECT statement
* ORDER BY keyword will sort values
  + default is ascending order, can use DESC keyword to make descending
  + can also sort multiple columns by separating column names by a comma in ORDER BY statement – order is important!!
* GROUP BY keyword groups values based on the specified column
  + Always goes after the FROM clause (questionable)
  + Can combine GROUP BY and ORDER BY to group results, calculate something about them then order results
* HAVING keyword lets you use an aggregator function since you can’t use them with WHERE clause
  + Syntax: HAVING COUNT(title) > 10
* LIMIT clause lets you limit the results
  + Syntax: LIMIT 5 to limit to 5 results

**Joining Data in SQL:**

* INNER JOIN
  + It is good to use short aliases for longer table names using the first letter of their names
  + Syntax:
  + SELECT \*
  + FROM left\_table
  + INNER JOIN right\_table
  + ON left\_table.id = right\_table.id;
* INNER JOIN via USING
  + When the name of the key column is same in both tables, can use USING
  + Syntax: USING (ID)
* Self joins
  + In ON statement, can use AND to make sure multiple conditions are met
* CASE
  + CASE WHEN \*Boolean\* THEN ‘return value’
  + Creates new column
  + INTO command creates a new table with the new case command
* OUTER JOINs
  + LEFT, RIGHT and FULL joins
* LEFT JOIN
  + Left join keeps all values from left table, but only includes values with a match from right table
  + Syntax of left join is similar to inner join
* RIGHT JOIN
  + Much less common than left join
  + Mirror of left join
* FULL JOIN
  + Similar syntax to INNER and LEFT joins
  + Joins all rows of multiple tables
* CROSS joins
  + Create all possible combinations between tables
  + Similar syntax to INNER and LEFT joins
* UNION
  + A picture containing device

    Description automatically generated
  + Unions stack records on top of each other from one table to the next
  + Do not do the lookup step that joins do
* INTERSECT
  + Pull only records that are the same between tables
* EXCEPT
  + Only include records from the left table that are not in the right table
* Semi join
  + Similar to WHERE clause
  + Semi join chooses records in the first table where a condition is met in the second table
* Anti Join
  + Semi join chooses records in the first table where a condition is NOT met in the second table
* Subqueries
  + Can be inside SELECT and WHERE clauses
* Subqueries inside WHERE clause
  + Can be a part of another condition in the WHERE clause
  + Most common place for a subquery
* Subqueries inside SELECT clause
  + Always need to give the subquery an alias
  + Can be a part of the columns we want to use in SELECT clause
* Subqueries inside FROM clause
  + Create subquery in from clause to create temporary table with the columns you want to include
  + There must be a comma between tables or subqueries in FROM clause

**Intermediate SQL:**

* CASE statement
  + Contain WHEN, THEN, and ELSE statement finished with END syntax:
  + CASE WHEN x=1 THEN ‘a’
  + ELSE ‘b’ END AS new\_column
  + Can add AND inside WHEN clause
  + Add specific filters in WHERE clause so you’re only impacting rows you want to
  + Removing ELSE clause will fill the rest with NULL values
  + To filter by CASE statement, include whole CASE statement in the WHERE clause, excluding the alias
* CASE statements with aggregate functions
  + Can include inside an agg function
  + For COUNT - instead of a string in THEN clause, use the identifier that we can count
  + Can calculate percentage by using avg aggregator and 1s and 0s in the THEN clause
* Subqueries
  + Can create more than one subquery in the FROM statement
    - Make sure they have aliases
    - Make sure they are joined
  + Can join a subquery to any existing table in database
    - Must have a column to join on
  + Subqueries in select
    - Needs to return a single value
    - Properly filter both main query and subquery (WHERE clause)
  + Query best practices
    - Format queries clearly
    - Line up SELECT, FROM, WHERE, and GROUP BY
    - Annotate query with comments (/\* text \*/ for multiple lines or – text for inline comments)
    - Indent contents of a subquery
    - Addition subqueries require extra computing power – always check whether it’s necessary to get results needed
    - Properly filter subqueries
* Correlated Queries
  + Queries that use values from the outer query to generate a result
  + Can slow down performance because query is run in loops for every row of data
* Nested Subqueries
  + Can be correlated or uncorrelated or a combination of the two
* Common Table expressions (CTEs)
  + Can create common table expression to put subquery in using WITH statement. Can then call that WITH statement wherever you want to put your subquery
  + Syntax: WITH s AS (\*\*Subquery\*\*)
  + If multiple subqueries, you can list then in the same WITH statement, separating with commas
  + CTEs can reference other CTEs for itself in recursive CTE
* Deciding on techniques to use
  + There can be multiple ways to do the same thing, depends on processing time, reproducibility etc
  + Depends on question, field, database etc
* Window Functions
  + Class of functions that perform calc on already generated result set
  + Can perform agg functions without grouping data
  + OVER() clause tells query to apply aggregate function ‘over’ all rows
  + RANK() creates a rank of highest to lowest or vice versa of data set – default lowest to highest
* Window Partitions
  + PARTITION BY calculate separate values for diff categories
  + Calc diff calcs in same column
  + Syntax: AVG(home\_goal) OVER(PARTITION BY season)
  + Can partition by multiple columns
* Sliding Windows
  + Perform calcs relative to vurrent row
  + Can also be partitioned by one or more columns
  + Syntax: ROWS BETWEEN <start> AND <finish>
    - Keywords: PRECEDING, FOLLOWING, UNBOUNDED, PRECEDING, UNBOUNDED FOLLOWING, CURRENT ROW
* Bringing it all together

**Working with Relational Databases in Python:**