VIEWs and Subqueries: Tips for Workint with Data in SQL

* Defining Subqueries
  + Essentially, a subquery is a query within a query. This allows you to use the results of one query within another
  + For this lesson we’ll be using three types of subqueries:
    - The first allows us to use the results of one query as a table for the outer query.
      * The general structure of our first subquery looks like this:

SELECT something

FROM

(SELECT something else

FROM somewhere) AS temp;

* + - * + Where we normally have a table, we can supply an entire query between parentheses
        + The outer query then treats the results of the inner query as if it were another table
        + These are very simple queries, but each SELECT query can be as complex as you like
        + Remember, you must give an alias name to the inner query with the AS keyword, even if you never refer to the inner query by this alias
    - The other allows us to filter the results of one query by the results of another questions using the IN operator.
    - Lastly, we will use WITH AS to combine sub queries into one query
* VIEWs
  + Defining VIEWs
    - A VIEW, in general, is just a shortcut for a SELECT statement. It does not imply that the results are ever run and processed. If you use a VIEW< the results will need to be regenerated each item it is used. A temporary table actually stores these results
  + Benefits of VIEWs
    - VIEWs can hide complexity
      * If you have a query that requires joining several tables or has complex logic or calculations, you can code all of that logic into a VIEW< then SELECT from the VIEW just like you would a table.
    - VIEWs can be used as a security mechanism
      * A VIEW can SELECT certain columns and / or rows from a table, and permissions can be set on the VIEW instead of the underlying tables. This allows the VIEW to only surface the data that a user needs to see.
    - VIEWS can simplify supporting legacy code
      * If you need to refactor a table that would break a lot of code, you can replace the table with a VIEW of the same name.
      * The VIEW provides the exact same schema as the original table, while the actual schema has changed
      * This keeps the legacy code that references the table from breaking, allowing you to update the code at your leisure.
  + Creating a VIEW
    - The basic CREATE VIEW syntax is as follows:

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

* + - * You can include multiple tables in your SELECT statement similar to how you would in a normal SQL SELECT query
  + Updating VIEWs
    - You can update a VIEW using the following syntax:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* + Dropping VIEWs
    - Obviously, where you have a VIEW, you need a way to drop it if it is no longer needed. The syntax for this is very simple:

DROP VIEW view\_name;