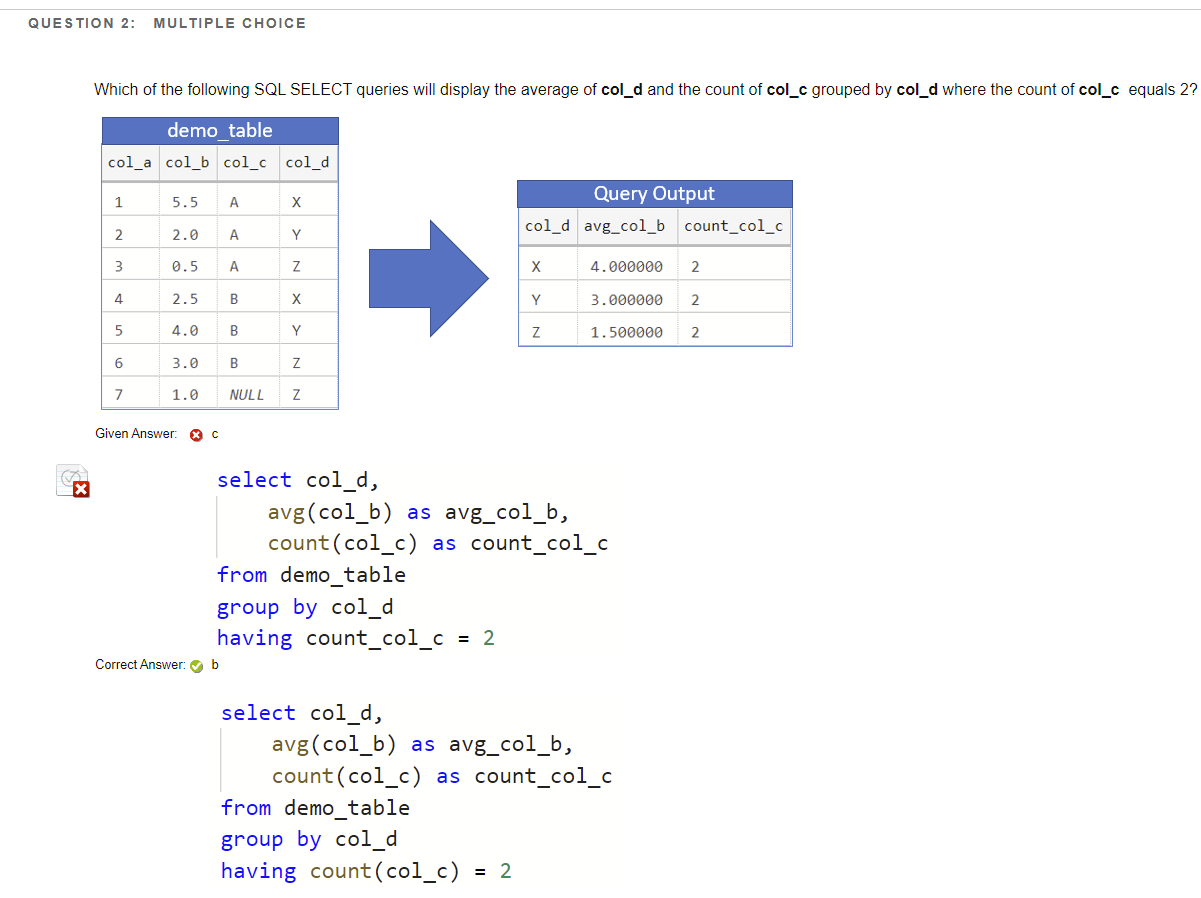
IST659 Quiz 3

Rayanna Harduarsingh

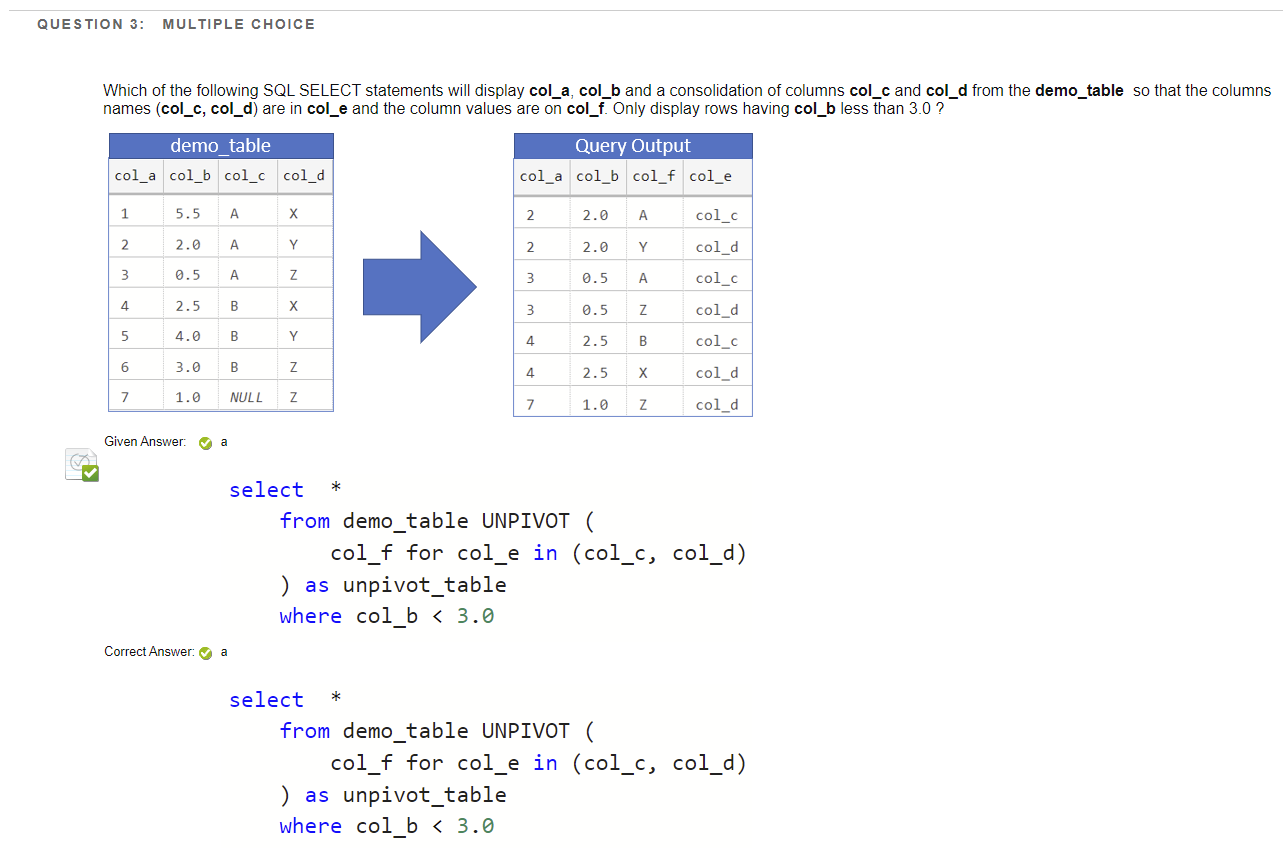
Text

Description automatically generated

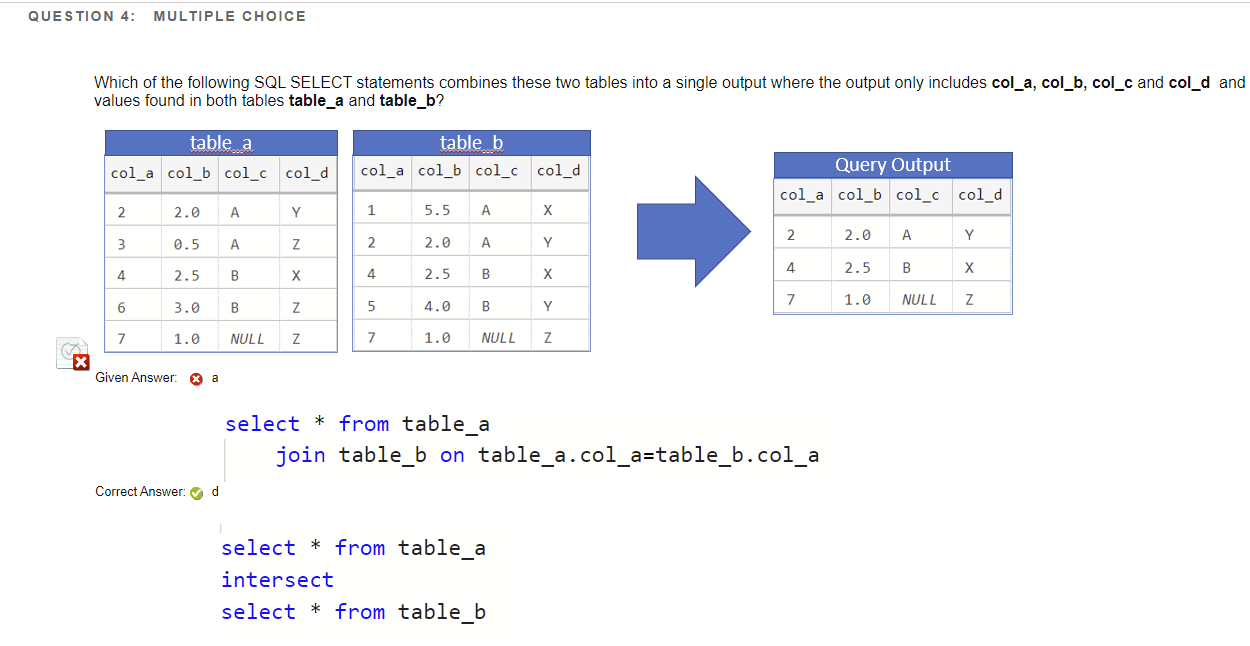
Answer D is not correct because according the question, it asked which statement uses a reducing operation to aggregate the query output. The over() clause is a window function and although it is a hybrid of an aggregate function (reducer functions), it is not necessary to include it. It does to require that we group a set of rows. Answer C does a perfectly good job of simply using a reducing operation to aggregrate the query output such as using the count function to count all values from the table and return values where it is less than 5 from column\_a.



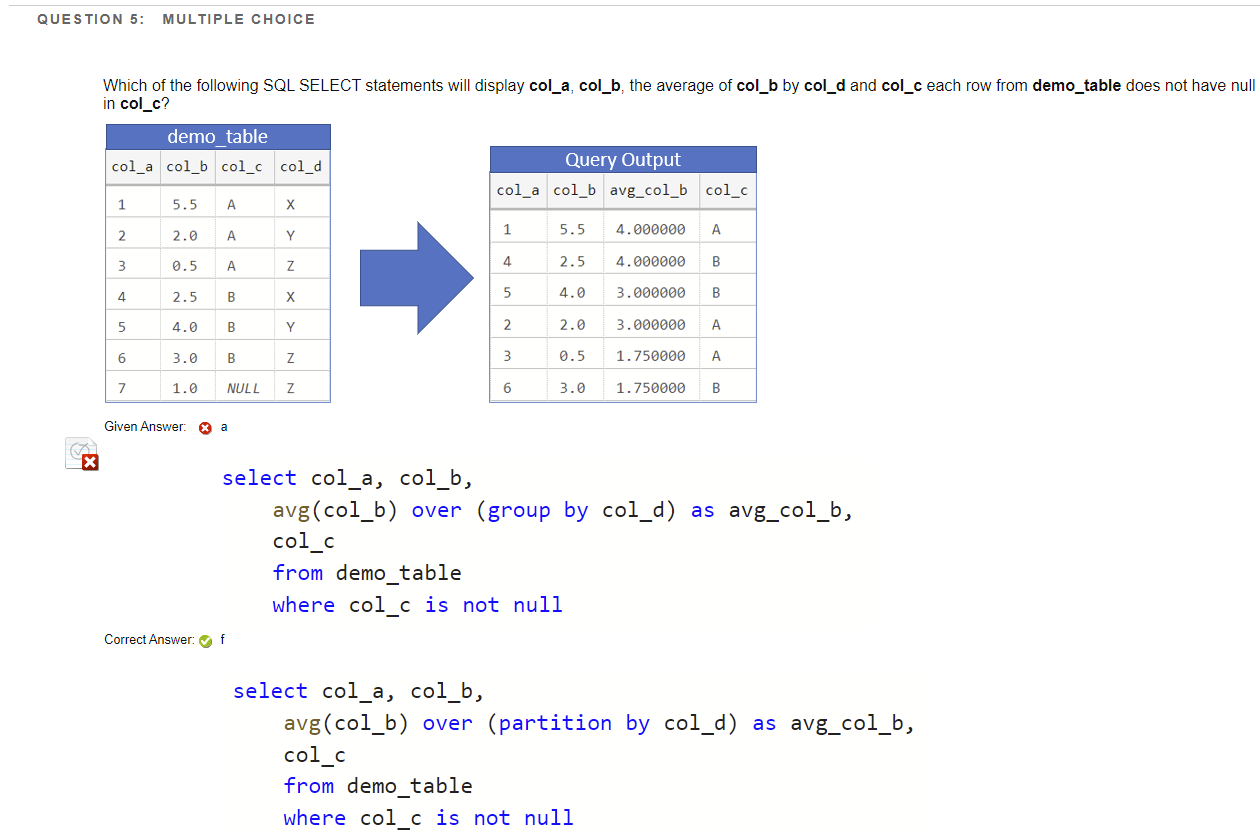
C is not the correct answer because although the format may be correct, the having clause is written wrong. The having clause as we know is similar to the where clause but it filters rows after the data aggregation has ran. Count\_col\_c is not a valid column name in the original table and so the query will not recognize that column, even though we renamed count(col\_c) as count\_col\_c, but that is only temporary as we just want to the column to have a name in the output. It does not change the column name and so B is the correct answer as we are using an aggregate function in the having clause in which it will read correctly.



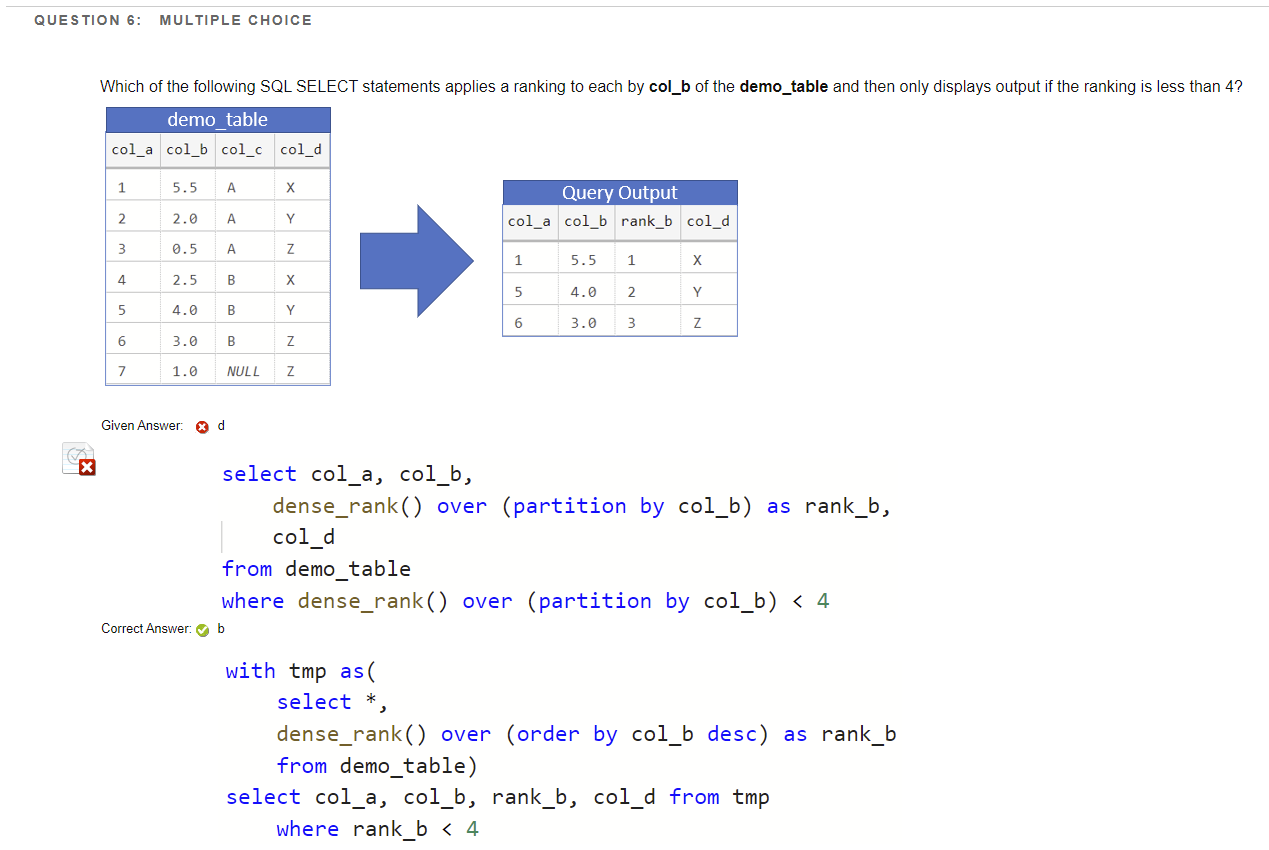
Unpivot transforms like-columns into rows. You start with the table-valued expression you would like to transform, the demo-table, and the columns we want to unpivot and consolidate, column c and column d into one column, column e. These two columns are now rows in column e. Because we are unpivoting those two columns into column e, we are also calling the values in each of those two columns (now column f) for column e. Finally, your unpivot operator is included in the FROM clause. Also, because we only want the rows with a value of less than 3.0, we specify in the where clause for column b to be less than 3.0 using ‘<’. This answer follows the correct syntax: select, from, an unpivot included in the from clause that includes the columns we’re calling for, and specifying our where clause after the from.



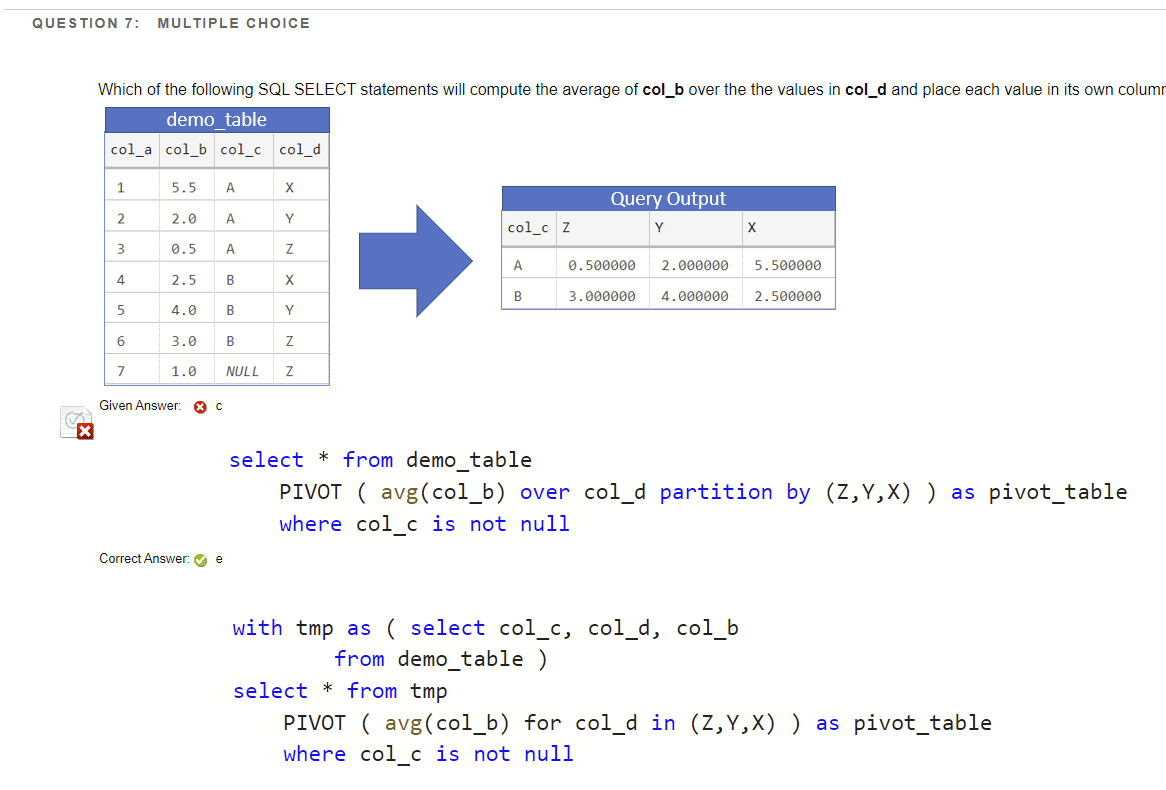
Because we want our output to include the same values found in both table a and b, we use the intersect clause. The join clause does indeed combine tables, but it does not find the values two tables have in common. The intersect function is a set operator that returns the same values found in both tables. For example, if table 1 has the colors blue, red, green, and table 2 has the colors yellow, pink, red, the intersect function will return the color red as both tables have those same values. So D is the correct answer because we use the select \* to return all the columns, and the intersect clause will return the rows that have the same values in each row from each table. In the diagram, we can see that query outputs all the rows the tables have in common. Taking the first row of the query output for example, we can see that for each column, both tables had a value of 2, 2.0, A, and Y for columns a, b, c, d, respectively.



Answer A is incorrect because off the bat, I noticed the syntax is wrong. In the window function over() the group by clause is placed and written incorrectly. Whenever using the group by clause, it is written after the where clause, and if there is no where, it’s written after the from clause. Also, when using over() it can be followed by either a partition by or an order by clause. Answer F is correct because it includes a partition by in the over() and is written correctly. Partition by organizes data into groups of rows and the window function will apply to each group. Here we wanted the avg of column b by col d and so using a partition clause will help us do that by applying the average function to each group. The group by clause is similar where the values of column specified are grouped together so any aggregate function, in this case average, can be applied to the group.



Answer D is incorrect because the function dense\_rank() must have an over clause with an order by, not partition by. Also, the question asks to first apply a ranking to each by column b, so we first have to use a with clause which allows us to name a query and then reuse it within a next execution. Answer B is correct because here it uses the with clause to name a query, in this case a query to rank each row in column b and using over() to determine the rows from the query and apply it to the rank function, and order it by column b to be in descending order. After temporarily naming that query, we can now use that query to execute a regular selection statement that outputs those rows with a ranking less than 4.



Answer C is incorrect in terms of syntax. Pivot allows us to transforms rows of data into columns. It includes an aggregate value in which it does do in answer C. However, when using pivot, rather than using the over() clause, you have to use for. For example, avg(col\_b) *for* col\_d *in* (values). Using an over with a partition by in a pivot is invalid. Answer E is correct because it follows the correct syntax of a Pivot operator. It’s using an aggregate function of a single column, the average of column\_b, and outputting a transformed table-value expression with values Z,Y,X for the column names and the output of the average aggregator for the values in each of those columns.

Graphical user interface

Description automatically generated

Temporal tables is a type of table that enables data to be collected from any point in time. Temporary tables allow us to query information as it has been stored at any point in time and to create a background of changes over time to the data. By using the for system\_time ALL clause, we can query data across the current and history tables. It returns a record for each row across those tables, but since we’re using ALL, it returns a union all or basically all records from the history and current table as it concatenates all the values, even returning duplicates. If we want to return rows from a specific time, then we can include a specified time. This answer is correct because we want to display all changes and using system\_time ALL it retrieves all those records for our where specifications with column\_a that has a value of 4 and column\_b with a value between 2.0 and 4.0.