**1. GROUPING and AGGREGATING Data.**

**What do these ideas mean?**

Grouping is taking rows and arranging them into groups specified in the GROUP BY clause. Aggregating is performing a function on each group, usually to find the COUNT, SUM, AVG, MIN, or MAX of items in each group.

**How would you compare and contrast grouping with aggregating?**

Grouping is arranging rows according to similarities. Aggregating is performing a function on each group to gain further insight into the group.

**How do you do these kinds of things in everyday life, even if you've not recognized that this is what you're doing?**

When shopping for groceries, I like to arrange my shopping cart into groups of fruit, cans, and meat to prevent cross-contamination. I can perform an aggregate COUNT function on each of these groups to determine how many pieces of fruit I have, how many cans, and how much meat.

**Is it natural to think about data (or the things that the data represents) as being in groups, and then aggregating the members of that group to produce... well, what, exactly, do you produce when you "aggregate" things?**

Humans are especially prone to grouping data (sometimes maliciously). By aggregating the members of each group, we attempt to gain some additional insight into our data. For example, statisticians group participants into brackets according to race, social status, etc. They then aggregate members of that group to produce averages and attempt to understand a problem or predict a trend.

**How do you deal with groups that may have "distinct" elements as you think about aggregating?**

In T-SQL, you deal with groups that have distinct elements by including the DISTINCT keyword in parenthesis. For example, if given five groups with the following quantities 30, 10, NULL, 10, 10, then the expression COUNT(DISTINCT qty) returns 2 because there are only two distinct groups: 30 and 10 because NULL is ignored.

**2. SUBQUERIES.**

**Why do we use subqueries?**

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

**What advantages does this give you as the query builder?**

One advantage that subqueries give the builder is avoiding the need for separate steps to store intermediate query results in variables. Another advantage is that they can be easier to read than other types of operations because each part of the statement is isolated and performs the same operation that would ordinarily require complex joins and unions.

**How do creating and using subqueries compare with how you "get things done" in programming languages and building applications programs?**

Using subqueries is like using nested statements to get things done in other programming languages. You can use an inner operation (like an if statement) to find a value or set of values, and then use an outer operation (like a for statement) to perform an operation on the value or values returned from the inner operation.

**How are they different?**

SQL is a declarative language so it does not require the use of loops like other programming languages. Additionally, SQL subqueries avoid the use of variables, whereas other programming languages typically require the use of variables in their nested statements.

**3.  SETS and SET THEORY.**

**Much of what we do in computer science and information systems is all about sets.  Humans naturally want to put things into separate piles based on perceived similarities or differences; set theory is what gives us the rules of the road to do this with. "SQL and relational databases are built on set theory," we are told. Explain what this means, and then assess whether it is important (or not) to you to have a better understanding of set theory.**

A set is an unordered collection of elements with no duplicates. By using primary keys, SQL attempts to produce sets of data with no duplicates. Moreover, terms like union and intersection that are typically used in set theory are also used in SQL. However, duplication can occur in SQL when selecting data from multiple tables, as well as from a single table if the primary key is omitted from the selection. For these reasons it is important to have a better understanding of set theory.

**4.  CROSS APPLY and CROSS JOIN.**

**At first glance these seem similar, right? How do they work?**

CROSS JOIN returns a Cartesian product so if you have 10 rows in each table the query will return 100 rows, 1 row for each combination. The CROSS APPLY operator CROSS APPLY is based on correlated subquery since it is able to pass one or more arguments from left part of query to right part. The right part may be a query or a UDF. In some cases it works like an INNER JOIN operator. It retrieves those records from the table valued function and the table being joined, where it finds matching rows between the two.

**What purpose(s) are you trying to accomplish with them?**

Use CROSS JOIN to retrieve all possible matches between two sets. Use CROSS APPLY when using a correlated subquery.