* Discuss how to build SQL statements with SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY clauses. Which are required and which are optional?
* Share any experience or conduct a research on SET operations such as UNION, INTERCEPT and EXCEPT using SQL statements. Elaborate them with examples.
* Share what set operations (e.g., UNION, INTERSECT, EXCEPT) supported by your RDBMS(s). If your RDBMSs don't support INTERSECT or EXCEPT; explain how you can resolve this issue with an alternate SQL with an example.
* Discuss any experience using subqueries and join, their pros and cons.
* If both ways can get the same results, explain which way you would prefer to use and why. If a query consists of complex logic; explain your approach to build the query.

SQL statements are built of formatted blocks of commands which pull data from the database. The SELECT clause lists the attributes/functions to be retrieved. The FROM clause specifies relations needed in the query. The WHERE clause specifies the conditions for the selection and joins with records specified in the FROM clause. The GROUP BY clause specifies groupings. The HAVING clause specifies a condition for the grouping selection, and the ORDER BY clause specifies an order for the query result. The only clauses that are required are SELECT and FROM, with the rest being optional.

SET operations such as UNION, INTERSECT and EXCEPT are similar to union operations discussed previoiusly in the course. UNION combines the result of two queries, INTERSECT is what two queries have in common, and EXCEPT includes everything in the first query that is not in the second query. An example would be if we had two zoos, where zoo1 had monkies, lions and penguins, and zoo2 had penguins, seals and dolphins, we can do the SET operation on the two zoos. UNION(zoo1, zoo2) would give monkies, lions, penguins, seals and dolphins, INTERSECT(zoo1, zoo2) would return penguins, and EXCEPT(zoo1, zoo2) would return monkies and lions. This concept is fairly simple and has been elaborated upon in earlier lectures.

The RDBMS that I am using is PostgreSQL, which seems to support each of UNION, INTERSECT and EXCEPT. This saves me the hassle of creating a workaround for all of the SET operations.

Subqueries in SQL are cool because they can introduce more complex logic to the SQL statements. The cons, however, include increased complexity for the queries. Subqueries allow the database manager to break out the logic of their SQL commands into discrete parts and create a logical order in which to process queries. Join is also a way to do this, where the database manager can do a query with embedded logic and join the results to create the same result as a subquery. The logic and thought process is similar between subqueries and join, however the syntax is different. For complex queries, I may lean toward trying subqueries first due to the logical nature of the query language using subqueries, but without any experience trying either, that opinion is open to change at any time.