

## CHAPTER SIX



# Query Construction

*As mentioned in Chapter 3, SQL queries get constructed in a series of steps structured around Who, Where, and What. Chapter 3 provided a simple example. This chapter describes in detail the steps to use when writing SQL queries. It will provide a framework for the four chapters to follow on Who, Where, What, and Formatting. Each of these chapters discusses the same example so that by the conclusion of them we'll have constructed an entire SQL query, tested it, debugged it, and run it with confidence.*

## Query Construction

The six clauses in the `SELECT` command provide structure to each query, but they also entice you as a query writer to construct the query in an inappropriate fashion. It's the old top-down fallacy. If the order is `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, and `ORDER BY`, many people assume that this is the order in which a query should be written. This is not the case.

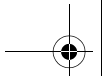
In fact, the order of the query is tied more to the 3W's than it is to the six clauses in the `SELECT` command. As mentioned previously, you actually begin in the middle of a query with the Who. You then bounce to the top of the query (i.e., the What `SELECT` clause) and identify additional tables needed for the report. Then you make the joins to these additional tables in the Where section. And finally, you write the SQL for the What data columns.

But within this general flow of Who, What, and Where, there are times when you pass through the same steps iteratively until you get things correct. Here's how a query actually gets assembled:

- **Step 1 (Who):** Using words, begin by describing the population of the query. In the next chapter we'll spend quite a bit of time on this step, because getting it right is essential, and there can be nuances that often get overlooked.
- **Step 2 (Who):** Identify the tables that must appear in the `FROM` clause to define the population you just described. From this list of tables, determine the one table that is most central to your population. This is known as the *driving table* because it essentially gets used to drive the remainder of the query. We'll see how this works in the next chapter. When you write the `FROM` clause, the driving table should appear last in the list of tables; i.e., it should be the table that appears right before the `WHERE` clause starts.
- **Step 3 (Who):** Based on the list of tables needed to define the population, join the tables, and add any modifying `WHERE` conditions. For example, if the report only includes women, then the `DEMOG` table will be joined, and `demog_sex = 'F'` will be added.
- **Step 4 (Who):** At this point you've defined your population in SQL. Add a dummy `SELECT` clause to count the items in the population.

This is important because at the conclusion of the query construction you want to ensure that you end up with this base population count.

- **Step 5 (Where):** Identify the tables needed to access the data columns, expressions, or group summaries that will appear in the report.
- **Step 6 (Where):** From the list of additional tables obtained in step 5, add one table to the **FROM** clause. Also add the join criteria for this table in the **WHERE** clause. Note that the only thing you should be adding to the **WHERE** clause is the join criteria; if you need to add other modifying conditions (e.g., `demog_sex = 'F'`), then you've improperly defined your population, and you'll need to return to step 2.
- **Step 7 (Where):** Test your join by running the SQL query to ensure that the population counts obtained in step 4 remain the same. Three things can happen here. One is that the counts are identical to the base counts. If so, congratulations! But the counts could be less than the population counts. If so, you've written a query where population items fall through the join you just added. Or the counts could be more than the population counts. If so, you've written a query where population items multiply through the join you added. Falling through a join and multiplying through a join are the two types of join errors. Both need to be fixed before proceeding with the query.
- **Step 8 (Where):** Repeat steps 6 and 7 taking one table at a time until all the additional tables have been added to the **FROM** clause, joined in the **WHERE** clause, and tested against the baseline population counts. This process is called *stepping through the joins*. It takes a little longer than merely writing the query in one massive step, but you test and debug the query as it's being constructed so that the overall process is actually quicker and easier.
- **Step 9 (What):** Delete the dummy counts in the **SELECT** clause used to test the query against the baseline population counts. Insert the data columns, expressions, and group summaries that you want to appear in the final report. If you require group summaries in the report, add the necessary **GROUP BY** clause to form the groups on which the summaries will be based. Also add the **ORDER BY** clause to sort the report. If you're using group summaries and want to limit the population in



## 76 | SIX Query Construction

the report to only those groups meeting certain criteria, then add a **HAVING** clause.

- **Step 10:** Use SQL\*Plus commands to add titles, page numbers, date and time stamps, customized column headings, footers, and other items that format the report to your liking.
- **Step 11:** Run the query.

Constructing queries in a stepwise approach like this is a large part of the craft in SQL. It will save you time, frustration, and possibly even the embarrassment of providing incorrect information to the person who requested the report. You'll just feel a whole lot more confident that the query you constructed is accurate.

