

CHAPTER ONE



Getting Oriented

SQL is an elegant tool used to query Oracle databases. This chapter provides examples of the reports and SQL programs you'll be able to construct after completing this book. The chapter also introduces basic database terminology and assesses the strengths and weaknesses of SQL as a reporting solution.



Where We're Headed

Let's start by taking a look ahead. By the time you complete the topics and exercises in this book, what types of SQL queries and reports should you be able to do on your own?

The reports you'll see in this chapter are based on a population of students registered for a higher algebra course in Fall 1999, part of a registration database at a fictitious college called Komenda College. We'll explore that database in Chapter 2 and then use it throughout the book. For now, just examine the reports and get a feeling for some of the things that can be done with SQL.

Figure 1-1 illustrates the list report. As the name suggests, the report simply lists students in the population. In this case, the report shows their identification numbers, name, sex, and academic degree program. List reports are standard fare for day-to-day operations and get used frequently in an organization. Figure 1-1, for example, might serve as a class roster for the course instructor.

Students Registered in Higher Algebra: Fall 1999			
Date: 29-MAY-2000		Time: 06:47	
Report name: t5.lis		Page: 1	
		SQL name: t5.sql	
id	name	sex	degree
@493665	Bertaut, Steven	M	BA
@416193	De Loatch, Janet	F	BA
@173082	Elsbernd, Srinivasa		BA
@136088	Haffey, Joseph	M	BS
@009673	Heise, Norman	M	BA
@139982	Hetherington, Jimmy X	M	BS
@232233	McClellan, William G	M	BS
@213830	Oliver, Terry Q		BS
@042078	Rao, Joseph O	M	BS
@207613	Slaughter, Patricia	F	BA
@790367	Smith, Larry	M	BA
@350039	Washington, Larry	M	BS
@134152	Weinstein, Kirk J	M	BA
@753128	Zartner, Mary Z	F	BA

FIGURE 1-1 An example of a list report.

Policy decisions frequently get based on reports that summarize the detailed data appearing in lists. Figure 1-2 illustrates an example of a one-way frequency distribution. It provides counts of students in the population grouped by their sex. It also provides a percentage breakdown, so we know what percentage are women and what percentage are men.

Summary reports come in a variety of flavors. Figure 1-3 shows a two-way crosstabulation of the students by their sex and by their degree program. There are, for example, 4 men in the BA program. The report also shows column totals, row totals, and a grand total. Managers faced with policy or program decisions most frequently request this type of summary report.

```

Count of Students Registered for Higher Algebra, by Gender: Fall 1999
Date: 29-MAY-2000      Time: 11:26      Page: 1
Report name: t7.lis      SQL name: t7.sql
    
```

sex	count	pct
F	3	21
M	9	64
Unk	2	14
sum	14	100

FIGURE 1-2 An example of a one-way frequency distribution report.

```

Students Registered for Higher Algebra, by Gender and Degree: Fall 1999
Date: 29-MAY-2000      Time: 11:33      Page: 1
Report name: t9.lis      SQL name: t9.sql
    
```

sex	BA	BS	count
F	3	0	3
M	4	5	9
Unk	1	1	2
sum	8	6	14

FIGURE 1-3 An example of a crosstabulation report.

Figure 1-4 shows an example of a report that most of us would not label a report. Sometimes you simply want to extract data in a format so that they can be read and used by another program. A common example is the extraction of name and address data for use with the mail merge feature of a word processor. Or the extraction of data that will be used for research purposes and read by a statistical program. In either case the report (i.e., the output) would be written to a data file and then used as input to another application program.

At the conclusion of this book, you will be able to produce all these types of reports and others as well. To do this, however, means learning to write SQL programs. Figure 1-5 shows the SQL that produced the standard

4 | ONE Getting Oriented

list report shown earlier in Figure 1-1. Not shown in the figure are several other commands called SQL*Plus commands used to format the results (e.g., adding a title to the report). The SQL probably looks like a mess at this point. Luckily, it's not really a mess. Each program can be systematically assembled in small steps that are clear and understandable. This is part of the craft in SQL—constructing programs that provide accurate results, return results quickly, and are easy to write. It's what this book is about. After completing this book, you'll be able to scan the SQL in Figure 1-5 and read it as easily as if it were written in English sentences.

```
@493665,Bertaut,Steven,,M,BA,x
@416193,De Loatch,Janet,,F,BA,x
@173082,Elsbernd,Srinivasa,,BA,x
@136088,Haffey,Joseph,,M,BS,x
@009673,Heise,Norman,,M,BA,x
@139982,Hetherington,Jimmy,X,M,BS,x
@232233,McClellan,William,G,M,BS,x
@213830,Oliver,Terry,Q,,BS,x
@042078,Rao,Joseph,O,M,BS,x
@207613,Slaughter,Patricia,,F,BA,x
@790367,Smith,Larry,,M,BA,x
@350039,Washington,Larry,,M,BS,x
@134152,Weinstein,Kirk,J,M,BA,x
@753128,Zartner,Mary,Z,F,BA,x
```

FIGURE 1-4 An example of a data extract.

```
SELECT      reg_id "id",
            name_Last||', '||name_First||' '||name_Middle "name",
            demog_sex "sex",
            student_degree_code "degree"
FROM        student,
            demog,
            name,
            reg
WHERE       reg_term = '199909'
AND        reg_catalog_code = '19725'
AND        reg_status_code = 'RG'
AND        name_id = reg_id
AND        name_seqno =
            (SELECT  MAX(n2.name_seqno)
             FROM    name n2
             WHERE   n2.name_id = reg_id)
AND        demog_id(+) = reg_id
AND        student_id = reg_id
AND        student_effective_term =
            (SELECT  MAX(s2.student_effective_term)
             FROM    student s2
             WHERE   s2.student_id = reg_id
             AND     s2.student_effective_term <= reg_term)
ORDER BY   name_Last,
            name_First,
            name_Middle;
```

FIGURE 1-5 SQL that produced the list report in Figure 1-1.

Basic Terms

Before we can do anything with SQL, we need to discuss four basic concepts. These concepts are *tables*, *rows*, *columns*, and *data values*. They're the lingua franca of all relational databases. Luckily, the definition of each concept is intuitively obvious.

Data in Oracle databases get stored in objects called *tables*. In common usage, information arranged in tables consists of rows and columns with values appearing in the cells where rows and columns intersect. This is exactly analogous to an Oracle table. In fact, think of an Oracle table as a spreadsheet where the *rows* identify people or widgets or something else entirely and the *columns* identify specific items of information about those entities, such as the first name of a person or the cost for a widget. The table cells contain the *data values* themselves.

Suppose we construct a table of students called **STUDENT** to store data about the student's identification number, name, and address of residence. Each row in the table corresponds to a single person; each column corresponds to an item of information about the students, such as their identification numbers, last names, street addresses, and so on. Specific data values occur at the intersection of each row and column. So, if data about Sharon Davenport occur in row 3 and if column 2 refers to first name, the intersection of this row and column contains the data value "Sharon" (see Figure 1-6).

id	first	last	street	city	state	zip
@928141	Roger	Thompson	6076 Cleary Dr	Fresno	CA	93727
@215419	Rowena	Lamagna	548 Mullica Hill Rd	Conway	MA	01341
@545687	Sharon	Davenport	1687 Franklin St	Greensboro	NC	27412
@159619	Joseph	Medaris	714 Central Ave	Albany	NY	12208

FIGURE 1-6 Data in a **STUDENT** table.

- ▶ Column 2 in the **STUDENT** table refers to the first name of each student.
- ▶ Row 3 refers to the student whose identification number is @545687. The intersection of row 3 and column 2 contains the data value "Sharon," which is the first name of this student.

6 | ONE Getting Oriented

Sometimes data values are unknown. For example, someone might not report his or her middle name. Other times a data value may not exist. For example, suppose our database stores two lines of a street address. A specific person, however, may have an address that requires only a single street line. The address table will contain no entry in the second street line for this person. In cases where data are missing or do not exist, the data value in a spreadsheet would appear blank. These blank values are called *null* values in Oracle databases. When database designers set up Oracle tables, some data columns are restricted from ever containing null values. This can simplify queries. In many situations, however, null values can occur, and we'll need to pay particular attention to them.

Queries against a single table are simple. Everything exists in one place, the population for the query is the whole table or a subset of the table, and all the items for our report also exist in the table as columns. If all SQL queries occurred against single tables, there would be no need for books like this. So you know it must get more complicated.

Consider the simple **STUDENT** table again. Things start to get interesting when it's possible for any one student to have multiple values for a single data column. Let's use last name as an example. Suppose that Sharon Davenport gets married and changes her last name to Lee. Do we replace the last name of Davenport with Lee and thus lose the fact that her previous last name was Davenport? Or do we add another row to the table for Lee and thus repeat her identification number, her first name, her city of address, and other data in this second row?

Either solution might work depending on our needs, but neither solution is ideal. In one case we keep things simple but lose the history; in the second case we keep the history but complicate things with a second row and all the duplication of data values this requires.

Through a process called *normalization*, Oracle database designers isolate instances where you can get repeating rows through database transactions. We've already seen one such example with name changes. If it's important to capture the history of name changes, as distinct from merely updating the old name to a new name, then name data will be isolated during the database design into a separate table.

Historical transactions represent one situation where a single entity (a student in this example) can have multiple rows. Here's another common

situation. Suppose that in addition to the address of residence, we want to maintain an address where we can send bills for tuition, fees, and other charges. In other words, the database must now accommodate two different types of addresses. A single student might have any number of address rows—none if we have no address information, one if we have only a residence or a billing address, two if we have both a residence and a billing address, two if we have data about a current residence and a former residence, and so on. The presence of multiple address data means that during the database design and normalization, addresses also will be isolated into a separate table.

Do you see what's happening? From a single **STUDENT** table we now have a **NAME** table and an **ADDRESS** table. This is all very clever and quite useful because we can now track historical transactions like name or address changes as well as maintain data about different types of addresses. So we're definitely better off. The downside is that it's now harder to query the database. Instead of a single table that contains all the data, we must query two tables to produce a name and address report. To do this, you must *join* the two tables together so that the names and addresses match correctly (much more on this later).

■ Exercises

Figure 1-7a through Figure 1-7h show data on college students and some of the courses they've taken. The following questions all refer to these figures.

1. How many tables exist? What are their names?
2. What data columns occur in the **STUDENT** table?
3. The second row in the **REG** table refers to a course taken by a student whose identification number is @042078. What was the course?
4. Why does the student with identification number @930300 appear twice in the **NAME** table?
5. Why does @034576 appear twice in the **STUDENT** table?
6. Are there any **NULL** values in the **REG** data shown in Figure 1-7c?

8 | ONE Getting Oriented

name_ id	name_ seqno	name_ first	name_ last	name_ update
@034576	1	Nancy	Heinbokel	11-Jun-1998
@042078	1	Joseph	Rao	11-Jun-1998
@150179	1	Eugene	Kimble	11-Jun-1998
@930300	1	Jean	Meadows	11-Jun-1998
@930300	2	Jean	Warren	12-Jun-1998

FIGURE 1-7a Selected data in the NAME table.

student_ id	student_ effective_ term	student_ major_ code	student_ update
@034576	199809	BIO	11-Jun-1998
@034576	199906	MAT	15-Jun-1998
@042078	199909	CHE	11-Jun-1998
@150179	199909	ANT	11-Jun-1998
@930300	199909	HIS	11-Jun-1998

FIGURE 1-7b Selected data in the STUDENT table.

reg_ id	reg_ term	reg_ catalog_ code	reg_ section_ number	reg_ status_ code	reg_ credit_ hours
@042078	199909	16635	01	RG	3.00
@042078	199909	19725	01	RG	3.00
@042078	199909	91177	01	DR	.00
@042078	199909	35566	01	DR	.00
@042078	199909	93835	01	DR	.00
@150179	199909	19672	01	RG	3.00
@150179	199909	32884	01	RG	3.00
@150179	199909	96151	01	DR	.00
@930300	199909	26461	01	RG	3.00
@930300	199909	49006	01	NC	.00
@930300	199909	82961	01	DR	.00
@930300	199909	91177	01	RG	3.00

FIGURE 1-7c Selected data in the registration table REG.

term_ code	term_ description	term_ codes_ update
199809	Fall 1998	26-May-1998
199906	Summer 1999	26-May-1998
199909	Fall 1999	26-May-1998

FIGURE 1-7d Selected data in the TERM_CODES table.

catalog_ code	catalog_ effective_ term	catalog_ subject_ code	catalog_ course_ number	catalog_ description
16635	199509	GEO	106	The Chemical Evolution of the Earth
19672	199509	GOV	106	Contemporary British Politics
19725	199509	MAT	215	Higher Algebra
49006	199509	SOC	240	New Institutionalism in Asia: Seminar
82961	199509	SPA	104	Intensive Elementary Spanish: Special Course
91177	199509	HIS	156	Private Life in Byzantium (7th-15th centuries): Conference Course
93835	199509	PSY	122	Social Cognitive Neuroscience: Seminar
96151	199509	ART	100	Introduction to Theatre Arts

FIGURE 1-7e Selected data in the CATALOG table.

major_ code	major_ description	major_ codes_ update
ANT	Anthropology	11-Jun-1998
BIO	Biology	11-Jun-1998
CHE	Chemistry	11-Jun-1998
HIS	History	11-Jun-1998
MAT	Mathematics	11-Jun-1998

FIGURE 1-7f Selected data in the MAJOR_CODES table.

reg_ status_ code	reg_ status_ description	reg_ status_ codes_ update
DR	Dropped	29-MAY-1998
NC	No credit	29-MAY-1998
RG	Registered	29-MAY-1998
WL	Wait listed	29-MAY-1998

FIGURE 1-7g Selected data in the REG_STATUS_CODES table.

subject_ code	subject_ description	subject_ codes_ update
ART	Fine Arts	05-JUN-1998
GEO	Geology	05-JUN-1998
GOV	Government	05-JUN-1998
HIS	History	05-JUN-1998
MAT	Mathematics	05-JUN-1998
PSY	Psychology	05-JUN-1998
SOC	Sociology	05-JUN-1998
SPA	Spanish	05-JUN-1998

FIGURE 1-7h Selected data in the SUBJECT_CODES table.

Other Reporting Objects

Reporting in Oracle databases occurs against three types of objects. You've already been introduced to tables, which form the foundation for reports. Normalization during the database design process frequently produces hundreds or even thousands of tables. This complicates reporting because the tables must be joined together properly when constructing SQL to query the tables.

Database designers frequently construct two other types of objects that improve your ability to query Oracle databases. The first object is called a *data view*. The second is called by various names, sometimes a *warehouse table* or a *snapshot table* or a *static table*. Both types of objects share one feature. They're constructed so that you can access data from many different tables in just one location. Rather than needing to join a **NAME** table to an **ADDRESS** table to get a report with both names and addresses, a data view and a static table can each be constructed so that the table join is not necessary. In other words, you access name and address data from one object.

One important difference between a data view and a static table is that the static table actually contains data, while a data view does not. Frequently you'll hear a view called a *virtual table*. The word *virtual* means that real data are not stored there. The view looks and acts like it contains data, but in fact, the data exist in the original tables (e.g., **NAME** and **ADDRESS**), and you access it through SQL that the database designers have constructed. This SQL contains all the logic needed to join the tables, saving you the problem of doing the joins yourself.

A static table has one important distinguishing characteristic. As the language implies, the data in a static table are a snapshot at one particular point in time. The table gets populated by taking extracts of data from many different tables and inserting them into the static table. But if data in the database tables change, you do not see those changes until the next time the extract is run and the static table is repopulated.

Figure 1-8 summarizes several important differences between tables, views, and static tables. It uses four points of comparison:

- Is the SQL easy to write when you want to query the data?
- Are the data current so that you get up-to-the-second results?

- Is the object comprehensive so that data from many different tables can be accessed from a single source?
- Is the response time from the query fast so that you don't need to wait long for the results? From a database administrator's standpoint, this question can be asked another way. Is the impact on system resources minimal?

Feature	Tables	Views	Static Tables
SQL Easy?	No	Yes	Yes
Current Data?	Yes	Yes	No
Comprehensive?	No	Yes	Yes
Fast Response?	Yes	No	Yes

FIGURE 1-8 Advantages and disadvantages of database objects.

As shown in Figure 1-8, each database object has at least one flaw. There is no single object where the SQL is easy, the data are current, the object is comprehensive, and the response time is fast. Now, of course, I'm talking generalities here. You may encounter a specific database (e.g., a very small one) where you can achieve an optimal reporting environment with a single type of object. But most frequently, reporting solutions involve the use of several different types of objects.

Here are the important limitations and advantages of each of the three reporting objects.

Database tables contain very focused data. Names will be in a **NAME** table; addresses will be in an **ADDRESS** table; and so on. You won't find comprehensive tables that contain everything you might want to include in your query. Consequently, the query will involve joining the tables in ways that sometimes can be quite complex. However, the results are both current and relatively fast. The response time, of course, is relative to the size of the database and the system traffic that occurs when you run the query. But compared with data views, for example, most SQL queries against tables run significantly faster.

Data views improve reporting in three significant ways. The SQL to

query the database is often trivial; the results are current; and the view, if constructed properly and used for some time in a production environment, contains most data that you'll want to include in a query. There is a limit of 255 data columns that database designers can include in a view. Typically, views contain data specific to a given reporting need (i.e., to one functional area in an organization). So, if you suddenly need to query across functional areas, the view may not contain the required data.

But the main disadvantage of views is their response time. A view might provide access to data in 15 or 20 different tables. If you only need to see data from two of those tables, that's too bad. The view still does the join to all 15 or 20 tables and then just returns the results you requested. This slows things down for you and also can adversely impact system performance.

A static table also contains one potentially important problem. It is static; the data are not current. However, the SQL needed to query a static table is very simple; the table frequently contains data from many different sources; and accessing the data is very fast because you don't need to do any table joins. Static tables frequently get used in policy and strategic research, when summary data are the main ingredient of the analysis. In this situation, data need not be absolutely current just so long as they have been refreshed fairly recently.

SQL as a Reporting Tool

SQL is only one among many reporting tools that you can use to query Oracle databases. Let's examine how SQL fits into the spectrum of reporting solutions and discuss the strengths and weaknesses of each class of tools. SQL may not be appropriate for you. Now is the time to determine that, not after you've read 300 pages of this book.

First, some history. Relational databases, of which Oracle is one example, got their start in the late 1960s and 1970s in an IBM research laboratory in California. Researchers there created a language called SEQUEL to access data in relational databases. As vendors began offering commercial relational database systems, the SQL language evolved from SEQUEL (SQL is sometimes still pronounced as "sequel").

Today SQL is an ANSI and ISO standard language used to query, modify, and manage relational databases. While the language is standardized, each vendor adds its own unique flavor. Oracle added SQL*Plus. This product allows you to enter and execute SQL commands, format query results, access databases, and even modify the SQL*Plus environment so that it's customized to your liking.

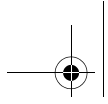
When I talk about Oracle SQL as a reporting tool, I'm really talking about SQL in combination with SQL*Plus.

The major strength of SQL is its antiquity. It goes back to the inception of relational databases and is the conduit into the data in those databases. Other products simply build interfaces between users and the database, hiding but not eliminating the need to access data via SQL. Think of other reporting tools as the branches of a tree. SQL is the trunk. Everything feeds through it to the data. Knowing SQL provides you with this foundation.

But SQL is also limited. It produces a few standard types of reports very well. But it cannot be used for large, complicated production reports, especially if those reports require use of color, the addition of charts and graphs, variations in fonts, and other formatting requirements. It also has very limited statistical and research capabilities. And it seldom finds wide adoption as an end-user query solution because most end-users perceive it as a programmer's tool.

So what are some of the other options? Well, if you're a developer and need to build complicated production reports, you'll use a full-fledged procedural language (e.g., SQR or Pro*C). These give you row-by-row access to the data and allow you to manipulate and format to your heart's content. They access the database using SQL statements embedded in the program. This solution provides ultimate control when that control is absolutely necessary. They are not, of course, end-user query products.

For research and statistical purposes, you'll want to use products like SAS or SPSS. Data can be accessed directly with embedded SQL statements. Or you can write an SQL program that extracts data in a format readable by the statistical package. Regardless of how you retrieve the data, you can perform countless analyses and summarize the results with a wide variety of visual displays like graphs, charts, and maps. Again, this solution tends toward the specialized. It is not a general end-user reporting solution but finds considerable use in market research, strategic planning, and insti-



tutional research departments.

Then there are a wide variety of products designed for end-user reporting. Examples include Crystal Reports, Impromptu, and even Microsoft Access. These products have a graphical interface, so they have a look and feel that's comfortable to users familiar with personal computers. Constructing a query consists of clicking, dragging, dropping, and other procedures familiar to anyone who's ever used a mouse. The strength of these products lies in their ease of use. And when applied to database objects like comprehensive data views and static tables, these are superior reporting solutions.

Problems, however, can occur when you need to access data from tables themselves. As we'll see later, preparing an accurate report often requires complex SQL to define the query population. Or requires the use of multiple queries and subqueries. If users don't understand the complexities of SQL, they can misapply these tools and get results that are misleading or wrong. So some end-user reporting tools can pose a danger precisely because of their ease of use.

