CHAPTER 7 Equijoins

Oracle SQL By Example, Fourth Edition by Alice Rischert. Published by Prentice Hall. Copyright © 2008 by Pearson Education, Inc.

CHAPTER OBJECTIVES

In this chapter, you will learn about:

- The Two-Table Join
- Joining Three or More Tables

So far, you have written SQL statements against a single table. In this chapter, you will learn about joining tables, one of the most important aspects of the SQL language. The *equijoin, which* is by far the most common form of join, allows you to connect two or more tables. Equijoins are based on equality of values in one or more columns. You will learn about other types of joins in Chapter 10, “Complex Joins.”

Lab 7.1 The Two-Table Join

LAB OBJECTIVES

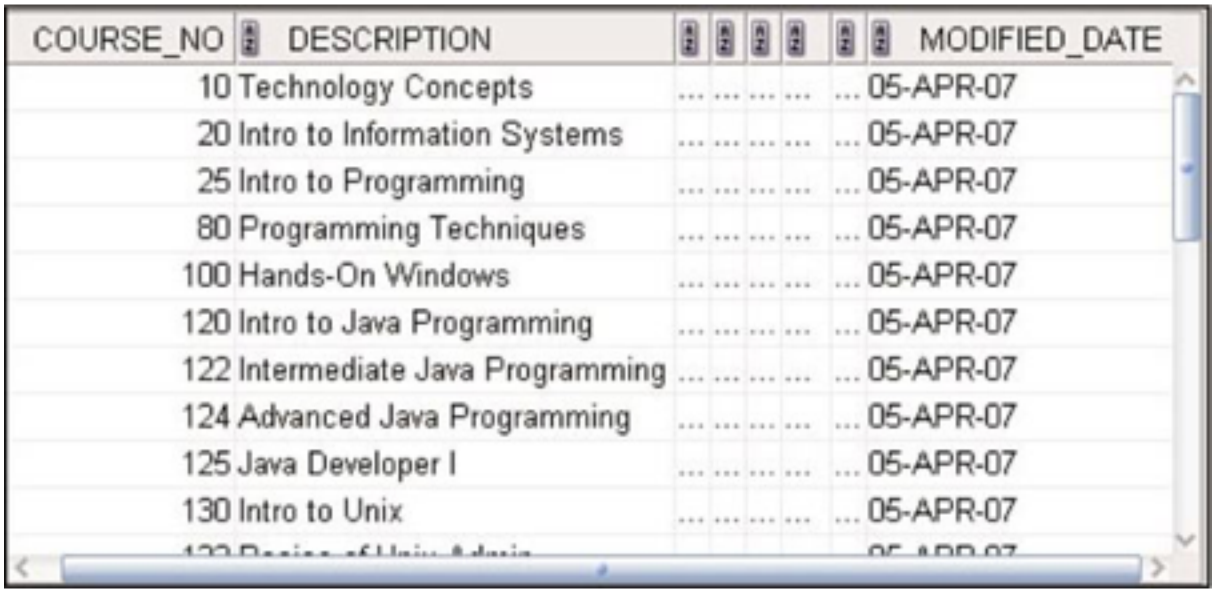
After this lab, you will be able to:

- ▶ Write Simple Join Constructs
- ▶ Narrow Down Your Result Set
- ▶ Understand the Cartesian Product

In this lab, you will join information from two tables into one meaningful result. Suppose you want to list the course number, course description, section number, location, and instructor ID for each section. This data is found in two separate tables: The course number and description are in the COURSE table; the SECTION table contains the course number, section number, location, and instructor ID. One approach is to query the individual tables and record the results on paper, then match every course number in the COURSE table with the corresponding course number in the SECTION table. The other approach is to formulate a SQL statement that accomplishes the join for you.

Figure 7.1 shows part of the COURSE table. Missing columns and rows are indicated with an ellipsis (…). The primary key of the COURSE table is COURSE_NO.

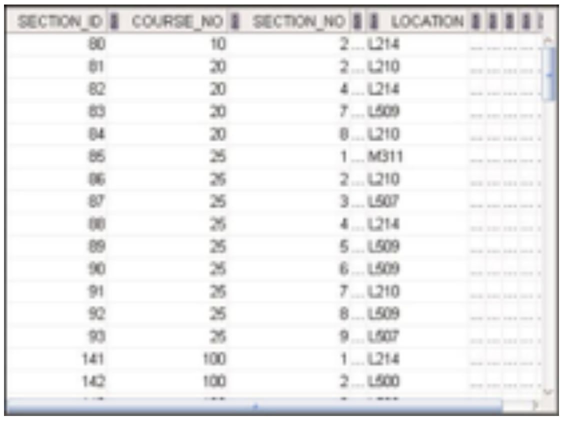
FIGURE 7.1 Excerpt of the COURSE table



COURSE_NO	DESCRIPTION						MODIFIED_DATE
10	Technology Concepts	...	...	...	...	...	05-APR-07
20	Intro to Information Systems	...	...	...	...	...	05-APR-07
25	Intro to Programming	...	...	...	...	...	05-APR-07
80	Programming Techniques	...	...	...	...	...	05-APR-07
100	Hands-On Windows	...	...	...	...	...	05-APR-07
120	Intro to Java Programming	...	...	...	...	...	05-APR-07
122	Intermediate Java Programming	...	...	...	...	...	05-APR-07
124	Advanced Java Programming	...	...	...	...	...	05-APR-07
125	Java Developer I	...	...	...	...	...	05-APR-07
130	Intro to Unix	...	...	...	...	...	05-APR-07

Figure 7.2 shows part of the SECTION table. The COURSE_NO column is the foreign key to the COURSE table.

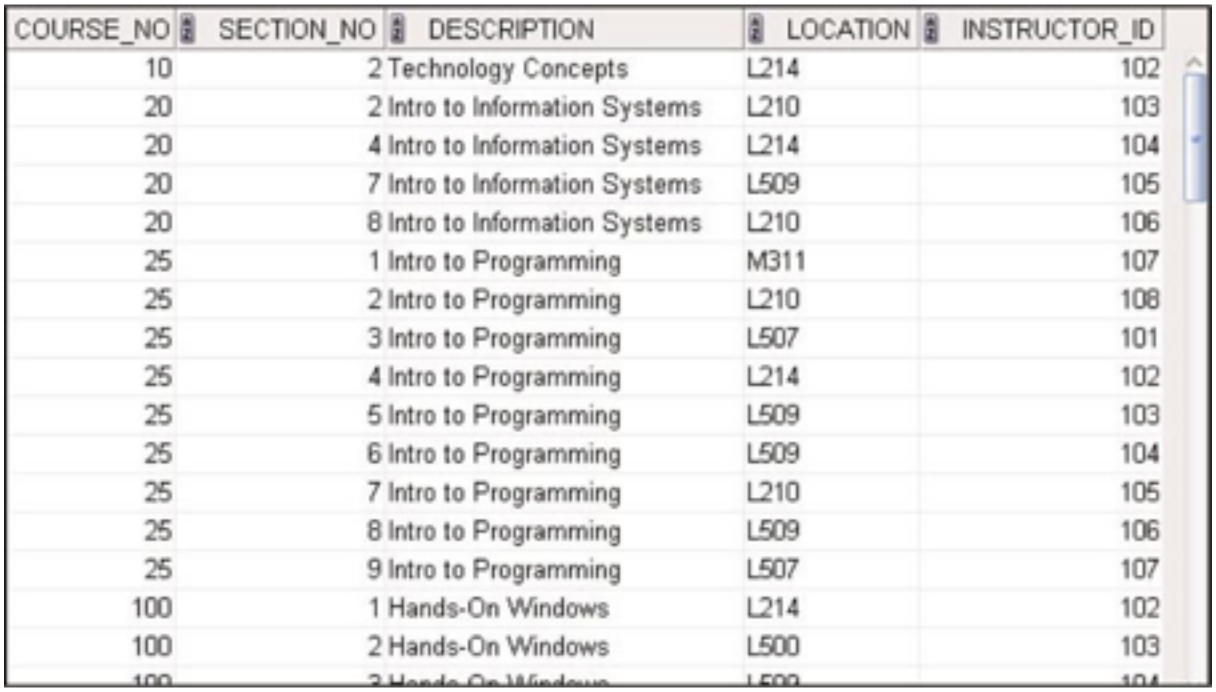
FIGURE 7.2 Excerpt of the SECTION table



SECTION_ID	COURSE_NO	SECTION_NO		LOCATION				
80	10	2	...	L214	...	...	...	...
81	20	2	...	L210	...	...	...	...
82	20	4	...	L214	...	...	...	...
83	20	7	...	L509	...	...	...	...
84	20	8	...	L210	...	...	...	...
85	25	1	...	M311	...	...	...	...
86	25	2	...	L210	...	...	...	...
87	25	3	...	L507	...	...	...	...
88	25	4	...	L214	...	...	...	...
89	25	5	...	L509	...	...	...	...
90	25	6	...	L509	...	...	...	...
91	25	7	...	L210	...	...	...	...
92	25	8	...	L509	...	...	...	...
93	25	9	...	L507	...	...	...	...
141	100	1	...	L214	...	...	...	...
142	100	2	...	L500	...	...	...	...

Examine the result set listed in Figure 7.3. For example, for course number 10, one section exists in the SECTION table. The result of the match is one row. Course number 20, Intro to Information Systems, has multiple rows in the section table because there are multiple classes/sections for the same course. Course number 80, Programming Techniques, is missing from the result. This course number has no matching entry in the SECTION table, and therefore this row is not in the result.

FIGURE 7.3 Result of the join between the COURSE and SECTION tables



COURSE_NO	SECTION_NO	DESCRIPTION	LOCATION	INSTRUCTOR_ID
10	2	Technology Concepts	L214	102
20	2	Intro to Information Systems	L210	103
20	4	Intro to Information Systems	L214	104
20	7	Intro to Information Systems	L509	105
20	8	Intro to Information Systems	L210	106
25	1	Intro to Programming	M311	107
25	2	Intro to Programming	L210	108
25	3	Intro to Programming	L507	101
25	4	Intro to Programming	L214	102
25	5	Intro to Programming	L509	103
25	6	Intro to Programming	L509	104
25	7	Intro to Programming	L210	105
25	8	Intro to Programming	L509	106
25	9	Intro to Programming	L507	107
100	1	Hands-On Windows	L214	102
100	2	Hands-On Windows	L500	103
100	3	Hands-On Windows	L500	104

Steps to Formulate the SQL Statement

Before you write the SQL join statement, first choose the columns you want to include in the result. Next, determine the tables to which the columns belong. Then identify the common columns between the tables. Finally, determine whether there is a one-to-one or a one-to-many relationship among the column values. Joins are typically used to join between the primary key and the foreign key. In the previous example, the COURSE_NO column in the COURSE table is the primary key, and the column COURSE_NO in the SECTION table is the foreign key. This represents a one-to-many relationship between the tables. (Joining tables related through a many-to-many relationship yields a *Cartesian product*. You'll learn more about the Cartesian product later in this chapter.)

Following is the SQL statement that achieves the result shown in Figure 7.3. It looks much like the SELECT statements you have written previously, but two tables, separated by commas, are listed in the FROM clause.

```
SELECTcourse. course_no,
section_no, description,
   location, instructor_id
 FROM course, section
```

```
WHEREcourse. course_no =
section.course_no
```

The WHERE clause formulates the *join criteria*, also called the *join condition*, between the two tables using the common COURSE_NO column. Because this is an equijoin, the values in the common columns must equal each other for a row to be displayed in the result set. Each COURSE_NO value from the COURSE table must match a COURSE_NO value from the SECTION table. To differentiate between columns of the same name, *qualify* the columns by prefixing the column with the table name and a period. Otherwise, Oracle returns the error ORA-00918: column ambiguously defined.

Instead of displaying the COURSE_NO column from the COURSE table in the SELECT list, you can use the COURSE_NO column from the SECTION table. Because it is an equijoin, it returns the same result.

The order in which the tables are listed in the FROM has no effect on the query result. But the join order can become important when optimizing SQL statement, and you will learn about this in Chapter 18, "SQL Optimization."

Table Aliases

Instead of using the table name as a prefix to differentiate between the columns, you can use a *table alias*, which qualifies the table using a short abbreviation.

```
SELECT c.course_no, s.section_no,
c.description,
   s.location, s.instructor_id
 FROM course c, section s
WHERE c.course_no = s.course_no
```

288

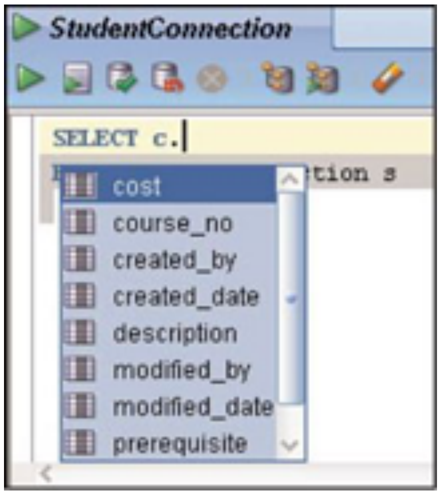
289

The table alias names are arbitrary. However, you cannot use any Oracle *reserved words*. (Reserved words have a special meaning in SQL or in the Oracle database and are typically associated with a SQL command. For example, SELECT and WHERE are reserved words.) It is best to keep the name short and simple, as in this example, where the COURSE table has the alias c, and the SECTION table has the alias s.

To easily identify the source table of a column and to improve the readability of a join statement, it is best to qualify all column names with the table alias. Furthermore, this avoids any future ambiguities that may arise if a new column with the same name is added later. Without a qualified table alias, a subsequently issued SQL statement referencing both tables results in the Oracle error ORA-00918: column ambiguously defined.

When you define a table alias in SQL Developer, any subsequent reference of the alias brings up a list of columns for the table, helping you remember the column names and avoid column misspellings (see Figure 7.4). Refer to Chapter 2, “SQL: The Basics,” for more information regarding the code completion feature.

FIGURE 7.4 Column list of the COURSE table



```
SELECT c.
```

Narrowing Down Your Result Set

The previous SQL statement lists all the rows in the SECTION and COURSE tables with matching COURSE_NO values. If you want to narrow down the criteria to specific rows, you can expand the WHERE clause to include additional conditions. The next statement chooses only those courses and their respective sections where the DESCRIPTION column starts with the text Intro to.

```
SELECT c.course_no, s.section_no,
c.description,
   s.location, s.instructor_id
 FROM course c, section s
WHERE c.course_no = s.course_no
  AND c.description LIKE ‘Intro to
%’
```

Nulls and Joins

In an equijoin, a null value in the common column has the effect of not including the row in the result. Look at the foreign key column ZIP on the INSTRUCTOR table, which allows nulls.

First, query for records with a null value.

```
SELECT instructor_id, zip, last_name, first_name
  FROM instructor
 WHERE zip IS NULL
INSTRUCTOR_ID ZIP   LAST_NAME  FIRST_NAME
------------- ----- ---------- ----------
          110       Willig     Irene

1 row selected.
```

Next, formulate the join to the ZIPCODE table via the ZIP column. Observe that instructor Irene Willig does not appear in the result.

```
SELECT i.instructor_id, i.zip, i.last_name, i.first_name
  FROM instructor i, zipcode z
 WHERE i.zip = z.zip
INSTRUCTOR_ID ZIP   LAST_NAME  FIRST_NAME
------------- ----- ---------- ----------
          101 10015 Hanks      Fernand
          105 10015 Morris     Anita
          109 10015 Chow       Rick
          102 10025 Wojick     Tom
          103 10025 Schorin    Nina
          106 10025 Smythe     Todd
          108 10025 Lowry      Charles
          107 10005 Frantzen   Marilyn
          104 10035 Pertez     Gary

9 rows selected.
```

A null value is not equal to any other value, including another null value. In this case, the zip code of Irene Willig's record is null; therefore, this row is not included in the result. In Chapter 10, you will learn how to include null values by formulating an *outer join* condition.

ANSI Join Syntax

Starting with Oracle 9*i*, Oracle implemented a number of additions to SQL to conform to many aspects of the ANSI SQL/92 and SQL:1999 standards. The advantage of the ANSI join syntax over the traditional comma-separated tables FROM clause is that SQL queries can run unmodified against other non-Oracle, ANSI-compliant databases.

INNER JOINS

The term *inner join* is used to express a join that satisfies the join condition. Typically, the join condition is based on equality, thus creating an equijoin. (The inner join is in contrast to the outer join. Besides the matched rows, the outer join also includes the unmatched rows from two tables.)

The ANSI syntax has a number of differences from the previously discussed join syntax. For example, the JOIN keyword replaces the comma between the tables and identifies the tables to be joined.

The keyword INNER is optional and typically omitted. To express a join condition, you can specify either the USING condition or the ON condition.

THE USING CONDITION

The USING condition, also referred as the USING clause, identifies the common column between the tables. Here the common column is the COURSE_NO column, which has the same name and compatible data type in both tables. An equijoin is always assumed with the USING clause.

```
SELECT course_no, s.section_no,
c.description,
   s.location, s.instructor_id
 FROM course c JOIN section s
USING (course_no)
```

Alternatively, you can include the optional INNER keyword, but as mentioned, this keyword is usually omitted.

```
SELECT course_no, s.section_no,
c.description,
   s.location, s.instructor_id
 FROM course c INNER JOIN section
s
```

```
USING (course_no)
```

The following query does not execute because you cannot use a table alias name with this syntax. The USING syntax implies that the column names are identical. The Oracle error identifies the C.COURSE_NO column as the column in the USING clause that has the problem.

```
 SQL> SELECT course_no,
s.section_no, c.description,
  2 s.location, s.instructor_id
  3 FROM course c JOIN section s
  4 USING (c.course_no)
  5 /
 USING (c.course_no)
  *
 ERROR at line 4:
  ORA-01748: only simple column
names allowed here
```

The next query shows the COURSE_NO column in the SELECT list prefixed with the alias name, thus resulting in an error as well. This alias must also be eliminated to successfully run the query.

```
 SQL> SELECT c.course_no,
s.section_no, c.description,
  2 s.location, s.instructor_id
  3 FROM course c JOIN section s
  4* USING (course_no)
 SQL> /
```

```
  SELECT c.course_no,
s.section_no, c.description,
  *
  ERROR at line 1:
  ORA-25154: column part of USING
clause cannot have qualifier
```

The next example illustrates the error you receive when you omit the parentheses around the column COURSE_NO in the USING clause.

```
 SQL> SELECT c.course_no,
s.section_no, c.description,
  2 s.location, s.instructor_id
  3 FROM course c JOIN section s
  4 USING course_no
  5 /
```

```
 USING course_no
  *
  ERROR at line 4:
 ORA-00906: missing left parenthesis
```

THE ON CONDITION

In case the column names on the tables are different, you use the ON condition, also referred to as the ON clause. The next query is identical in functionality to the previous query, but it is now expressed with the ON condition, and the column name is qualified with the alias both in the SELECT list and in the ON condition. This syntax allows for conditions other than equality and different column names; you will see many such examples in Chapter 10.

```
 SELECT c.course_no, s.section_no,
c.description,
  s.location, s.instructor_id
  FROM course c JOINsection s
  ON (c.course_no = s.course_no)
```

The pair of parentheses around the ON condition is optional. When comparing this syntax to the traditional join syntax, there are not many differences other than the ON clause and the JOIN keyword.

ADDITIONAL WHERE CLAUSE CONDITIONS

The ON and USING conditions let you specify the join condition separately from any other WHERE condition. One of the advantages of the ANSI join syntax is that it separates the join condition from the filtering WHERE clause condition.

```
 SELECT c.course_no, s.section_no,
c.description,
  s.location, s.instructor_id
  FROM course c JOIN section s
  ON (c.course_no = s.course_no)
  WHERE description LIKE 'B%'
```

292

293

NATURAL JOINS

A natural join joins tables based on the columns with the same name and data type. Here there is no need to prefix the column name with the table alias, and the join is indicated with the keywords NATURAL JOIN. There is not even a mention of which column(s) to join. This syntax figures out the common columns between the tables. Any use of the ON clause or the USING clause is not allowed with the NATURAL JOIN keywords, and the common columns cannot list a table alias.

```
 SELECT course_no, s.section_no,
c.description,
  s.location, s.instructor_id
  FROM course c NATURAL JOINsection
s
  no rows selected
```

You might be surprised that the query does not return any result. However, the COURSE_NO column is not the only column with a common name. The columns CREATED_BY, CREATED_DATE, MODIFIED_BY, and MODIFIED_DATE are also common to both tables. These columns record the name of the last user updating a row and the original user creating the row, including the respective date and time. The SQL statement does not return any results because there are no rows that have identical values for all five common columns.

Using the natural join within a program is somewhat risky. There is always a chance for adding to the table columns that happen to have the same column name, and then you may not get the desired result. Therefore, the natural join works best for ad hoc queries but not for repeated use within programs.

The Cartesian Product

The Cartesian product is rarely useful in the real world. It usually indicates either that the WHERE clause has no joining columns or that multiple rows from one table match multiple rows in another table; in other words, it indicates a many-to-many relationship.

To illustrate the multiplication effect of a Cartesian product, the following query joins the INSTRUCTOR table with the SECTION table. The INSTRUCTOR table contains 10 rows; the SECTION table has 78 rows. The multiplication of all the possible combinations results in 780 rows.

```
SELECT COUNT(*)
  FROM section, instructor
 COUNT(*)
---------
      780

1 row selected.
```

Following is a partial listing of the rows, showing all the different combinations of values between the two tables.

```
SELECT s.instructor_id s_instructor_id,
       i.instructor_id i_instructor_id
  FROM section s, instructor i
S_INSTRUCTOR_ID I_INSTRUCTOR_ID
--------------- ---------------
            101             101
            101             101
            101             101
            101             101
            101             101
            101             101
            101             101
            101             101
            101             101
            101             102
            101             102
            101             102
...
            108             110
            101             110

780 rows selected.
```

In SQL*Plus, if you want to stop and examine the rows one screen at a time, you can use the SQL*Plus SET PAUSE ON command. Appendix C, “SQL*Plus Command Reference,” provides more details about the various SQL*Plus commands.

THE ANSI STANDARD CROSS-JOIN

To formulate a Cartesian product using the ANSI JOIN syntax, you use the keyword CROSS JOIN in place of the comma between the two tables. Because of the nature of a cross-join as a combination of all possible values, the SQL statement does not have join criteria. The result is obviously identical to that of the Cartesian product.

```
SELECT COUNT(*)
FROM section CROSS JOINinstructor
```

LAB 7.1 EXERCISES

a) For all students, display last name, city, state, and zip code. Show the result ordered by zip code.

b) Select the first and last names of all enrolled students and order by last name in ascending order.

c) Execute the following SQL statement. Explain your observations about the WHERE clause and the resulting output.

```
 SELECT c.course_no,
c.description, s.section_no
```

```
  FROM course c, section s
  WHERE c.course_no =
s.course_no
  AND c.prerequisite IS NULL
  ORDER BY c.course_no,
s.section_no
```

d) Select the student ID, course number, enrollment date, and section ID for students who enrolled in course number 20 on January 30, 2007.

e) Select the students and instructors who live in the same zip code by joining on the common ZIP column. Order the result by the STUDENT_ID and INSTRUCTOR_ID columns. What do you observe?

LAB 7.1 EXERCISE ANSWERS

a) For all students, display last name, city, state, and zip code. Show the result ordered by zip code.

ANSWER: The common column between the ZIPCODE table and the STUDENT table is the ZIP column. The ZIP column in both tables is defined as NOT NULL. For each row in the

ZIPCODE table there may be zero, one, or multiple students living in one particular zip code. For each student's zip code there must be one matching row in the ZIPCODE table. Only records that satisfy the equality condition of the join are returned.

```
SELECT s.last_name, s.zip, z.state, z.city
  FROM student s, zipcode z
 WHERE s.zip = z.zip
 ORDER BY s.zip
LAST_NAME                 ZIP   ST CITY
------------------------- ----- -- ----------
Norman                    01247 MA North Adams
Kocka                     02124 MA Dorchester
...
Gilloon                   43224 OH Columbus
Snow                      48104 MI Ann Arbor

268 rows selected.
```

Because the ZIP column has the same name in both tables, you must qualify the column when you use the traditional join syntax. For simplicity, it is best to use an alias instead of the full table name because it saves you a lot of typing and improves readability. The ORDER BY clause lists the S.ZIP column, as does the SELECT clause. Choosing the Z.ZIP column instead of S.ZIP in the SELECT list or ORDER

BY clause displays the same rows because the values in the two columns have to be equal to be included in the result.

You can also write the query with the ANSI join syntax instead and use the ON condition or the USING condition. If you use the ON condition, you must prefix the common columns with either their full table name or their table alias, if one is specified.

```
 SELECT s.last_name, s.zip,
z.state, z.city
  FROM student s JOIN zipcode z
  ON (s.zip = z.zip)
 ORDER BY s.zip
```

If you choose the USING condition instead, do not alias the common column because doing so will cause an error.

```
 SELECT s.last_name, zip,
z.state, z.city
  FROM student s JOIN zipcode z
 USING (zip)
 ORDER BY zip
```

b) Select the first and last names of all enrolled students and order by last name in ascending order.

ANSWER: You need to join the ENROLLMENT and STUDENT tables. Only students who are enrolled have one or multiple rows in the ENROLLMENT table.

```
SELECT s.first_name, s.last_name, s.student_id
  FROM student s, enrollment e
 WHERE s.student_id = e.student_id
 ORDER BY s.last_name
FIRST_NAME LAST_NAME  STUDENT_ID
---------- ---------- ----------
Mardig     Abdou             119
Suzanne M. Abid              257
...
Salewa     Zuckerberg        184
Salewa     Zuckerberg        184
Salewa     Zuckerberg        184
Freedon    annunziato        206

226 rows selected.
```

Student Salewa Zuckerberg with STUDENT_ID 184 is returned three times. This is because Salewa Zuckerberg is enrolled in three sections. When the SECTION_ID column is included in the SELECT list, this fact becomes evident in the result set.

However, if you are not interested in the SECTION_ID and you want to list the names without the duplication, use DISTINCT in the SELECT statement.

```
 SELECT DISTINCT s.first_name,
s.last_name, s.student_id
  FROM student s, enrollment e
  WHERE s.student_id =
e.student_id
  ORDER BY s.last_name
```

The STUDENT_ID column is required in the SELECT clause because there may be students with the same first and last names but who are, in fact, different individuals. The STUDENT_ID column differentiates between these students; after all, it’s the primary key that is unique to each individual row in the STUDENT table.

The student with the last name annunziato is the last row. Because the last name is in lowercase, it has a higher sort order. (See Lab 4.3 regarding the sort order values and the ASCII function.)

If you use the ANSI syntax, your SQL statement may look similar to the following statement.

```
 SELECT s.first_name,
s.last_name, s.student_id
  FROM student s JOIN enrollment
e
  ON (s.student_id =
e.student_id)
  ORDER BY s.last_name
```

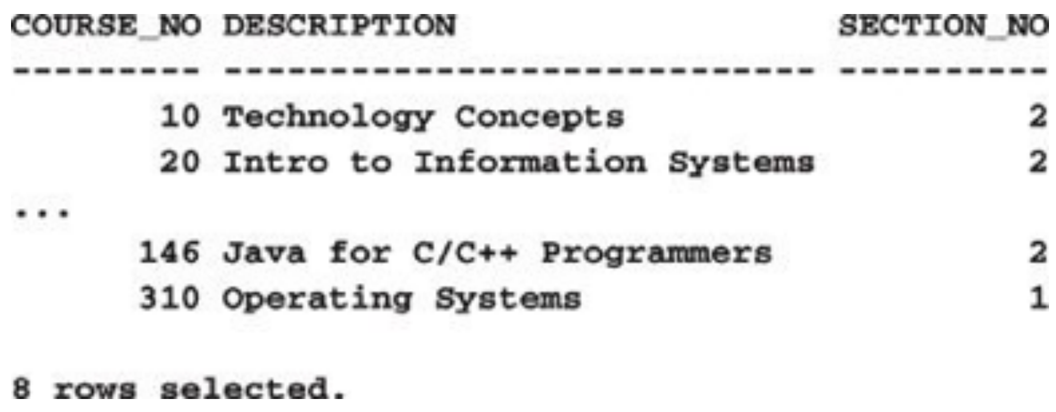
Or you may write the statement with the USING clause. In this particular query, all aliases are omitted, which has the disadvantage that you cannot easily recognize the source table for each column.

```
 SELECT first_name, last_name,
student_id
  FROM student JOIN enrollment
  USING (student_id)
  ORDER BY last_name
```

c) Execute the following SQL statement. Explain your observations about the WHERE clause and the resulting output.

```
 SELECT c.course_no,
c.description, s.section_no
  FROM course c, section s
  WHERE c.course_no =
s.course_no
  AND c.prerequisite IS NULL
  ORDER BY c.course_no,
section_no
```

ANSWER: This query includes both a join condition and a condition that restricts the rows to courses that have no prerequisite. The result is ordered by the course number and the section number.



```
COURSE_NO DESCRIPTION                  SECTION_NO
--------- ---------------------------- ----------
       10 Technology Concepts                   2
       20 Intro to Information Systems          2
...
      146 Java for C/C++ Programmers            2
      310 Operating Systems                     1

8 rows selected.
```

The COURSE and SECTION tables are joined to obtain the SECTION_NO column. The join requires the equality of values for the COURSE_NO columns in both tables. The courses without a prerequisite are determined with the IS NULL operator.

If the query is written with the ANSI join syntax and the ON clause, you see one advantage of the

ANSI join syntax over the traditional join syntax: The ANSI join distinguishes the join condition from the filtering criteria.

```
 SELECT c.course_no,
c.description, s.section_no
  FROM course c JOIN section s
  ON (c.course_no = s.course_no)
  WHERE c.prerequisite IS NULL
  ORDER BY c.course_no,
section_no
```

d) Select the student ID, course number, enrollment date, and section ID for students who enrolled in course number 20 on January 30, 2007.

ANSWER: The SECTION and ENROLLMENT tables are joined through their common column: SECTION_ID. This column is the primary key in the SECTION table and the foreign key column in the ENROLLMENT table. The rows are restricted to those records that have a course number of 20 and an enrollment date of January 30, 2007, by including this condition in the WHERE clause.

```
SELECT e.student_id, s.course_no,
       TO_CHAR(e.enroll_date,'MM/DD/YYYY HH:MI PM'),
       e.section_id
  FROM enrollment e JOIN section s
    ON (e.section_id = s.section_id)
 WHERE s.course_no = 20
   AND e.enroll_date >= TO_DATE('01/30/2007','MM/DD/YYYY')
   AND e.enroll_date < TO_DATE('01/31/2007','MM/DD/YYYY')
STUDENT_ID COURSE_NO TO_CHAR(ENROLL_DATE SECTION_ID
---------- --------- ------------------- ----------
       103        20 01/30/2007 10:18 AM         81
       104        20 01/30/2007 10:18 AM         81

2 rows selected.
```

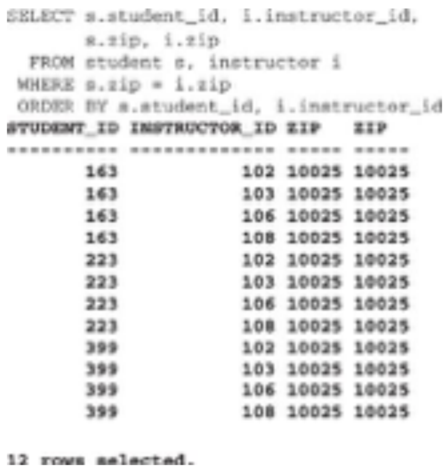
Alternatively, you can use the USING clause or the more traditional join syntax, listed here.

```
 SELECT e.student_id,
s.course_no,
 TO_CHAR(e.enroll_date,'MM/DD/
YYYY HH:MI PM’),
 e.section_id
 FROM enrollment e, section s
 WHERE e.section_id =
s.section_id
 AND s.course_no = 20
 AND e.enroll_date >=
TO_DATE(‘01/30/2007’, ‘MM/DD/
YYYY’)
 AND e.enroll_date <
TO_DATE(‘01/31/2007’,'MM/DD/
YYYY’)
```

The WHERE clause considers the date and time values of the ENROLL_DATE column. There are alternative WHERE clause solutions, such as applying the TRUNC function on the ENROLL_DATE column. Refer to Chapter 5,“Date and Conversion Functions,” for many examples of querying and displaying DATE data type columns.

e) Select the students and instructors who live in the same zip code by joining on the common ZIP column. Order the result by the STUDENT_ID and INSTRUCTOR_ID columns. What do you observe?

ANSWER: When you join the STUDENT and INSTRUCTOR tables, there is a many-to-many relationship, which causes a Cartesian product as a result.



```
SELECT s.student_id, i.instructor_id,
       s.zip, i.zip
  FROM student s, instructor i
 WHERE s.zip = i.zip
 ORDER BY s.student_id, i.instructor_id
STUDENT_ID INSTRUCTOR_ID ZIP   ZIP
---------- ------------- ----- -----
       163           102 10025 10025
       163           103 10025 10025
       163           106 10025 10025
       163           108 10025 10025
       223           102 10025 10025
       223           103 10025 10025
       223           106 10025 10025
       223           108 10025 10025
       399           102 10025 10025
       399           103 10025 10025
       399           106 10025 10025
       399           108 10025 10025

12 rows selected.
```

Initially, this query and its corresponding result may not strike you as a Cartesian product because the WHERE clause contains a join criteria. However, the relationship between the STUDENT table and the INSTRUCTOR table does not follow the primary key/foreign key path, and therefore a Cartesian product is possible. A look at the schema diagram reveals that no primary key/foreign key relationship exists between the two tables. To further illustrate the many-to-many relationship between the ZIP columns, select the students and instructors living in zip code 10025 in separate SQL statements.

```
SELECT student_id, zip
  FROM student
 WHERE zip = '10025'
STUDENT_ID ZIP
---------- -----
       223 10025
       163 10025
       399 10025

3 rows selected.

SELECT instructor_id, zip
  FROM instructor
 WHERE zip = '10025'
INSTRUCTOR_ID ZIP
------------- -----
          102 10025
          103 10025
          106 10025
          108 10025

4 rows selected.
```

These results validate the solution's output: the Cartesian product shows the three student rows multiplied by the four instructors, which results in 12 possible combinations. You can rewrite the query to include the DISTINCT keyword to select only the distinct student IDs. You can also write the query with a subquery construct, which avoids the Cartesian product. You will learn about this in Chapter 8, "Subqueries."

JOINING ALONG THE PRIMARY/ FOREIGN KEY PATH

You can join along the primary/foreign key path by joining the STUDENT table to the ENROLLMENT table, then to the SECTION table, and finally to the INSTRUCTOR table. This involves a multitable join, discussed in Lab 7.2. However, the result is different from the Cartesian product result because it shows only instructors who teach a section in which the student is enrolled. In other words, an instructor living in zip code 10025 is included in the result only if the instructor teaches that student also living in the same zip code. This is in contrast to the Cartesian product example, which shows all the instructors and students living in the same zip code, whether the instructor teaches this student or not. You will explore the differences between these two examples once more in the Workshop section at the end of the chapter.

Lab 7.1 Quiz

In order to test your progress, you should be able to answer the following questions.

1) Find the error(s) in the following SQL statement.

```
1 SELECT stud.last_name, stud.first_name,
2 stud.zip, zip.zip, zip.state, zip.city,
3 TO_CHAR(stud.student_id)
4 FROM student stud, zipcode zip
5 WHERE stud.student_id = 102
6 AND zip.zip = ‘11419'
7 AND zip.zip = s.zip
```

a) ______No error.

b) ______This is not an equijoin.

c) ______Lines 1, 2, and 3.

d) ______Line 4.

e) ______Lines 5 and 6.

f) ______Line 7.

2) Find the error(s) in the following SQL statement.

```
1 SELECT s.*, zipcode.zip,
2 DECODE(s.last_name, ‘Smith’, szip,
3 UPPER(s.last_name))
4 FROM student s, zipcode
5 WHERE stud.zip = zipcode.zip
6 AND s.last_name LIKE ‘Smi%'
```

a) ______Lines 1 and 2

b) ______Lines 1 and 4

c) ______Line 3

d) ______Lines 2 and 5

e) ______Line 4

3) A table alias is the name of a duplicate table stored in memory.

a) ______True

b) ______False

4) To equijoin a table with another table involves matching the common column values.

a) ______True

b) ______False

5) Find the error(s) in the following SQL statement.

1 SELECT TO_CHAR(w.modified_date, ‘dd-mon-yyyy’),

2 t.grade_type_code, description,

3 TO_NUMBER(TO_CHAR(number_per_section))

4 FROM grade_type t, grade_type_weight w

5 WHERE t.grade_type_code = w.grade_type_code_cd

6 AND ((t.grade_type_code = ‘MT'

7 OR t.grade_type_code = ‘HM’))

8 AND w.modified_date >=

9 TO_DATE(‘01-JAN-2007’, ‘DD-MON-YYYY’)

a) ______Lines 1 and 8

b) ______Line 4

c) ______Line 5

d) ______Lines 6 and 7

e) ______Lines 5, 6, and 7

6) Given two tables, T1 and T2, and their rows, as shown, which result will be returned by the following query?

```
SELECT t1.val, t2.val, t1.name, t2.location
  FROM t1, t2
 WHERE t1.val = t2.val
```

```
Table T1                Table T2
VAL NAME                VAL LOCATION
--- --------------- --- ---------
A   Jones               A   San Diego
B   Smith               B   New York
C   Zeta                B   New York
    Miller                  Phoenix
```

7) The USING clause of the ANSI join syntax always assumes an equijoin and identical column names.

a) ______True

b) ______False

8) The NATURAL JOIN keywords and the USING clause of the ANSI join syntax are mutually exclusive.

a) ______True

b) ______False

9) The common column used in the join condition must be listed in the SELECT list.

a) ______True

b) ______False

ANSWERS APPEAR IN APPENDIX A.

LAB 7. 2 Joining Three or More Tables

LAB OBJECTIVES

After this lab, you will be able to:

- Join Three or More Tables
- Join with Multicolumn Join Criteria

You often have to join more than two tables to determine the answer to a query. In this lab, you will practice these types of joins. In addition, you will join tables with multicolumn keys.

Joining Three or More Tables

The join example at the beginning of the chapter involves two tables: the COURSE and SECTION tables. The following SQL statement repeats that query and the result of the join (see Figure 7.5). To include the instructor's first and last names, you can expand this statement to join to a third table, the INSTRUCTOR table.

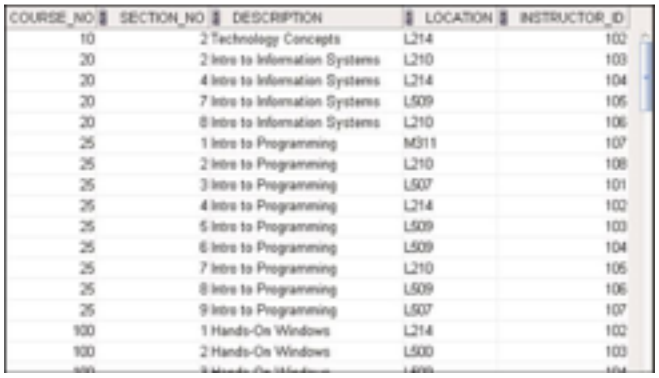
```
SELECT c.course_no, s.section_no,
c.description,
s.location, s.instructor_id
FROM course c, section s
WHERE c.course_no = s.course_no
```

Figure 7.6 shows a partial listing of the INSTRUCTOR table. The INSTRUCTOR_ID column is the primary key of the table and is the common column with the SECTION table. Every row in the SECTION table that has a value for the INSTRUCTOR_ID column must have one corresponding row in the INSTRUCTOR table. A particular INSTRUCTOR_ID in the INSTRUCTOR table may have zero, one, or multiple rows in the SECTION table.

To formulate the SQL statement, follow the same steps performed in Lab 7.1. First, determine the columns and tables needed for output. Then confirm whether a one-to-one or a one-to-many relationship exists between the tables to accomplish the join. The changes to the previous SQL statement are indicated here in bold.

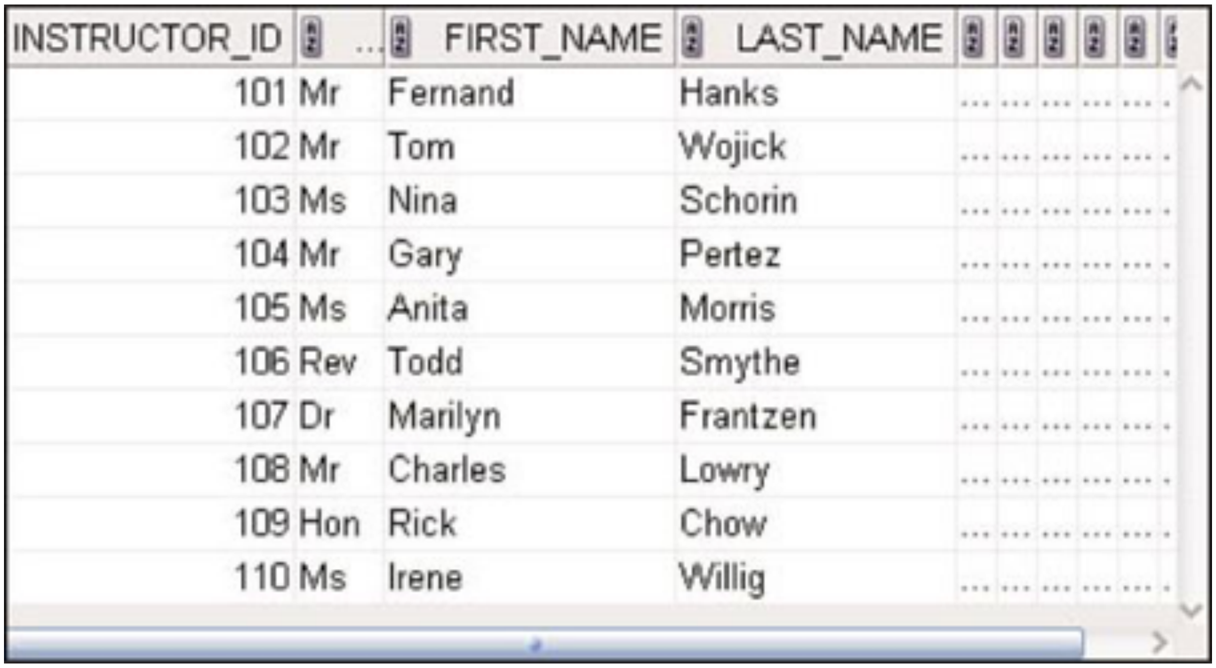
```
 SELECT c.course_no, s.section_no,
c.description, s.location,
 s.instructor_id, i.last_name,
i.first_name
 FROM course c, section s,
instructor i
 WHERE c.course_no = s.course_no
 AND s.instructor_id =
i.instructor_id
```

FIGURE 7.5 Result of join between COURSE and SECTION tables



COURSE_NO	SECTION_NO	DESCRIPTION	LOCATION	INSTRUCTOR_ID
10	2	Technology Concepts	L214	102
20	2	Intro to Information Systems	L210	103
20	4	Intro to Information Systems	L214	104
20	7	Intro to Information Systems	L509	105
20	8	Intro to Information Systems	L210	106
25	1	Intro to Programming	M311	107
25	2	Intro to Programming	L210	108
25	3	Intro to Programming	L507	101
25	4	Intro to Programming	L214	102
25	5	Intro to Programming	L509	103
25	6	Intro to Programming	L509	104
25	7	Intro to Programming	L210	105
25	8	Intro to Programming	L509	106
25	9	Intro to Programming	L507	107
100	1	Hands-On Windows	L214	102
100	2	Hands-On Windows	L500	103

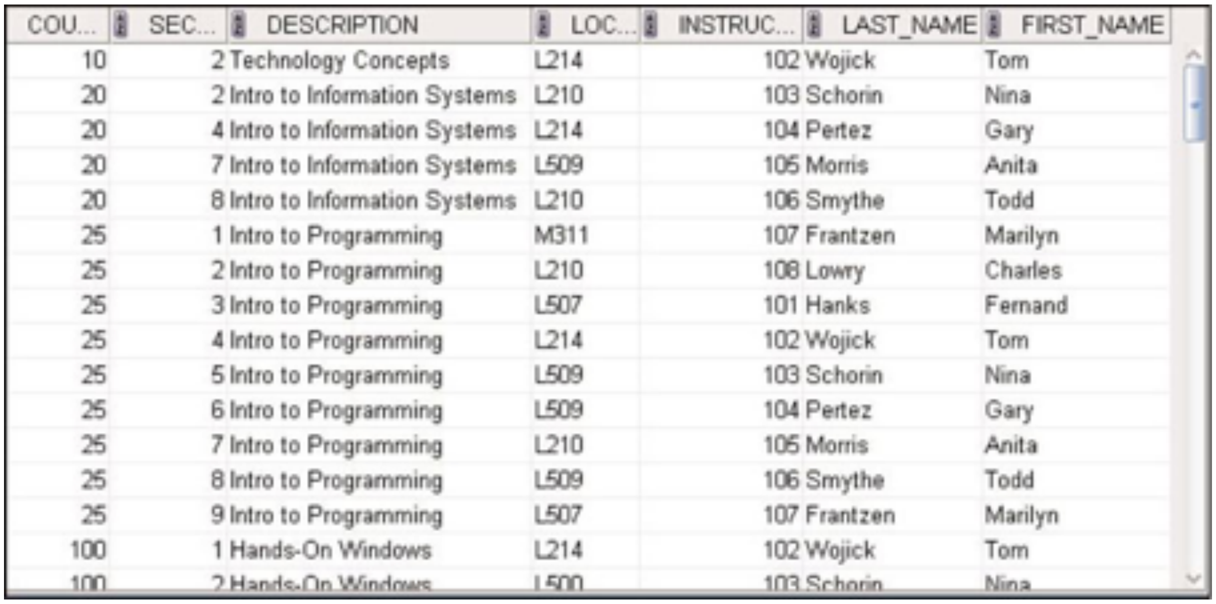
FIGURE 7.6 The INSTRUCTOR table



INSTRUCTOR_ID	...	FIRST_NAME	LAST_NAME					
101	Mr	Fernand	Hanks	...	...	...	...	...
102	Mr	Tom	Wojick	...	...	...	...	...
103	Ms	Nina	Schorin	...	...	...	...	...
104	Mr	Gary	Pertez	...	...	...	...	...
105	Ms	Anita	Morris	...	...	...	...	...
106	Rev	Todd	Smythe	...	...	...	...	...
107	Dr	Marilyn	Frantzen	...	...	...	...	...
108	Mr	Charles	Lowry	...	...	...	...	...
109	Hon	Rick	Chow	...	...	...	...	...
110	Ms	Irene	Willig	...	...	...	...	...

The join yields the result shown in Figure 7.7. The three-table join result now includes the instructor's first and last names. For example, INSTRUCTOR_ID 102 is listed multiple times in the SECTION table. This instructor teaches several sections; therefore, the INSTRUCTOR_ID's corresponding first and last names are repeated in the result.

FIGURE 7.7 Result of join between the COURSE, SECTION, and INSTRUCTOR tables



COU...	SEC...	DESCRIPTION	LOC...	INSTRUC...	LAST_NAME	FIRST_NAME
10	2	Technology Concepts	L214	102	Wojick	Tom
20	2	Intro to Information Systems	L210	103	Schorin	Nina
20	4	Intro to Information Systems	L214	104	Pertez	Gary
20	7	Intro to Information Systems	L509	105	Morris	Anita
20	8	Intro to Information Systems	L210	106	Smythe	Todd
25	1	Intro to Programming	M311	107	Frantzen	Marilyn
25	2	Intro to Programming	L210	108	Lowry	Charles
25	3	Intro to Programming	L507	101	Hanks	Fernand
25	4	Intro to Programming	L214	102	Wojick	Tom
25	5	Intro to Programming	L509	103	Schorin	Nina
25	6	Intro to Programming	L509	104	Pertez	Gary
25	7	Intro to Programming	L210	105	Morris	Anita
25	8	Intro to Programming	L509	106	Smythe	Todd
25	9	Intro to Programming	L507	107	Frantzen	Marilyn
100	1	Hands-On Windows	L214	102	Wojick	Tom
100	2	Hands-On Windows	L509	103	Schorin	Nina

ANSI Join Syntax for Joining Three or More Tables

A join across three tables can be expressed with the ANSI join syntax. Create the first join between the COURSE and SECTION tables via the JOIN keyword and the ON clause. To this result, you add the next table and join condition. The set of parentheses around the ON clause is optional.

```
 SELECT c.course_no, s.section_no,
c.description, s.location,
  s.instructor_id, i.last_name,
i.first_name
  FROM course c JOIN section s
  ON (c.course_no = s.course_no)
  JOIN instructor i
  ON (s.instructor_id =
i.instructor_id)
```

Alternatively, the query can be expressed with the USING clause. The table and column aliases in the SELECT and FROM clauses are optional, but the parentheses in the USING clause are required.

```
   SELECT course_no, s.section_no,
  c.description, s.location,
    instructor_id, i.last_name,
  i.first_name
    FROM course c JOIN section s
    USING (course_no)
    JOIN instructor i
    USING (instructor_id)
```

Multicolumn Joins

The basic steps of a multicolumn join do not differ from the previous examples. The only variation is to make multicolumn keys part of the join criteria.

One of the multikey column examples in the schema is the GRADE table. The primary key of the table consists of the four columns STUDENT_ID, SECTION_ID, GRADE_CODE_ OCCURRENCE, and GRADE_TYPE_CODE. The GRADE table also has two foreign keys: the GRADE_TYPE_CODE column, referencing the GRADE_TYPE_WEIGHT table, and the multi-column foreign keys STUDENT_ID and SECTION_ID, referencing the ENROLLMENT table.

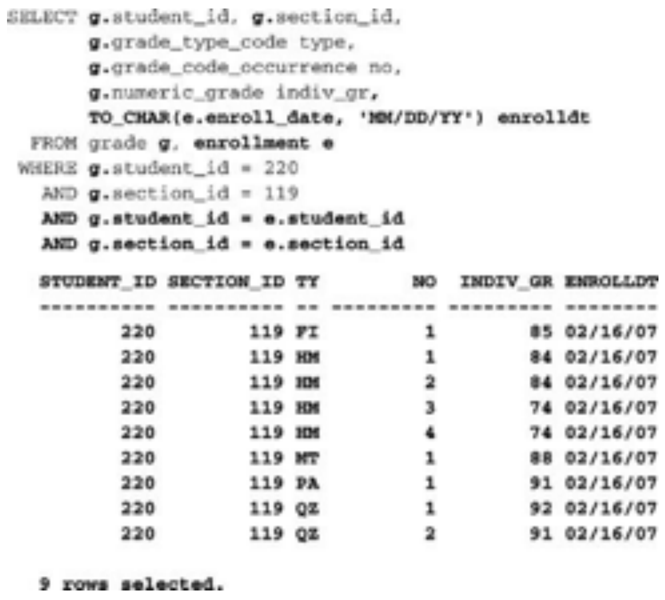
To help you understand the data in the table, examine a set of sample records for a particular student. The student with ID 220 is enrolled in SECTION_ID 119 and has nine records in the GRADE table: four homework assignments (HM), two quizzes (QZ), one midterm (MT), one final examination (FI), and one participation (PA) grade.

```
SELECT student_id, section_id, grade_type_code type,
       grade_code_occurrence no,
       numeric_grade indiv_gr
  FROM grade
 WHERE student_id = 220
   AND section_id = 119
STUDENT_ID SECTION_ID TY         NO  INDIV_GR
---------- ---------- -- ---------- --------
       220        119 FI          1       85
       220        119 HM          1       84
       220        119 HM          2       84
       220        119 HM          3       74
       220        119 HM          4       74
       220        119 MT          1       88
       220        119 PA          1       91
       220        119 QZ          1       92
       220        119 QZ          2       91

9 rows selected.
```

The next SQL query joins the GRADE table to the ENROLLMENT table, to include the values of the ENROLL_DATE column in the result set. All the changes from the previous SQL query are indicated in bold.

```
    SELECTg.student_id, g.section_id,
     g.grade_type_code type,
     g.grade_code_occurrence no,
     g.numeric_grade indiv_gr
     ,TO_CHAR(e.enroll_date, 'MM/DD/
   YY') enrolldt
     FROM grade g, enrollment e
     WHERE g.student_id = 220
     AND g.section_id = 119
     AND g.student_id = e.student_id
     AND g.section_id = e.section_id
```



```
SELECT g.student_id, g.section_id,
       g.grade_type_code type,
       g.grade_code_occurrence no,
       g.numeric_grade indiv_gr,
       TO_CHAR(e.enroll_date, 'MM/DD/YY') enrolldt
  FROM grade g, enrollment e
 WHERE g.student_id = 220
   AND g.section_id = 119
   AND g.student_id = e.student_id
   AND g.section_id = e.section_id

STUDENT_ID SECTION_ID TY         NO  INDIV_GR ENROLLDT
---------- ---------- -- ---------- --------- --------
       220        119 FI          1        85 02/16/07
       220        119 HM          1        84 02/16/07
       220        119 HM          2        84 02/16/07
       220        119 HM          3        74 02/16/07
       220        119 HM          4        74 02/16/07
       220        119 MT          1        88 02/16/07
       220        119 PA          1        91 02/16/07
       220        119 QZ          1        92 02/16/07
       220        119 QZ          2        91 02/16/07

9 rows selected.
```

To join between the tables ENROLLMENT and GRADE, use both the SECTION_ID and STUDENT_ID columns. These two columns represent the primary key of the ENROLLMENT table and the foreign key of the GRADE table; thus a one-to-many relationship between the tables exists.

The values for the ENROLL_DATE column are repeated, because for each individual grade you have one row showing ENROLL_DATE in the ENROLLMENT table.

EXPRESSING MULTICOLUMN JOINS USING THE ANSI JOIN SYNTAX

A join involving multiple columns on a table requires the columns to be listed in the ON or the USING clause as join criteria. The next SQL statement shows the ON clause.

```
 SELECT g.student_id, g.section_id,
 g.grade_type_code type,
 g.grade_code_occurrence no,
 g.numeric_grade indiv_gr,
 TO_CHAR(e.enroll_date, ‘MM/DD/YY’)
enrolldt
 FROM grade g JOINenrollment e
```

```
ON (g.student_id = e.student_id
AND g.section_id = e.section_id)
WHERE g.student_id = 220
AND g.section_id = 119
```

When you write the query with the USING clause, you list the join columns separated by commas.

```
 SELECT student_id, section_id,
 grade_type_code type,
 grade_code_occurrence no,
 numeric_grade indiv_gr,
 TO_CHAR(enroll_date, ‘MM/DD/YY’)
enrolldt
 FROM grade JOINenrollment
 USING (student_id, section_id)
 WHERE student_id = 220
 AND section_id = 119
```

Joining Across Many Tables

Joining across multiple tables involves repeating the same steps of a two-join or three-join table: You join the first two tables and then join the result to each subsequent table, using the common column(s). You repeat this until all the tables are joined.

To join *n* tables together, you need at least *n*–1 join conditions. For example, to join five tables, you need at least four join conditions. If your join deals with tables containing multicolumn keys, you will obviously need to include these multiple columns as part of the join condition.

The Oracle optimizer determines the order in which the tables are joined based on the join condition, the indexes on the table, and the various statistics about the tables (such as the number of rows or the number of distinct values in each column). The join order can have a significant impact on the performance of multitable joins. You can learn more about this topic and how to influence the optimizer in Chapter 18.

The ANSI Join Versus the Traditional Join Syntax

You might wonder which one of the join syntax options is better. The ANSI join syntax has a number of advantages.

- It helps you easily identify the join criteria and the filtering condition.

- You avoid an accidental Cartesian product because you must explicitly specify the join criteria, and any missing join conditions become evident because an error is generated.
- The syntax is easy to read and understand.
- The USING clause requires less typing, but the data types of the columns must match.
- SQL is understood by other ANSI-compliant non-Oracle databases.

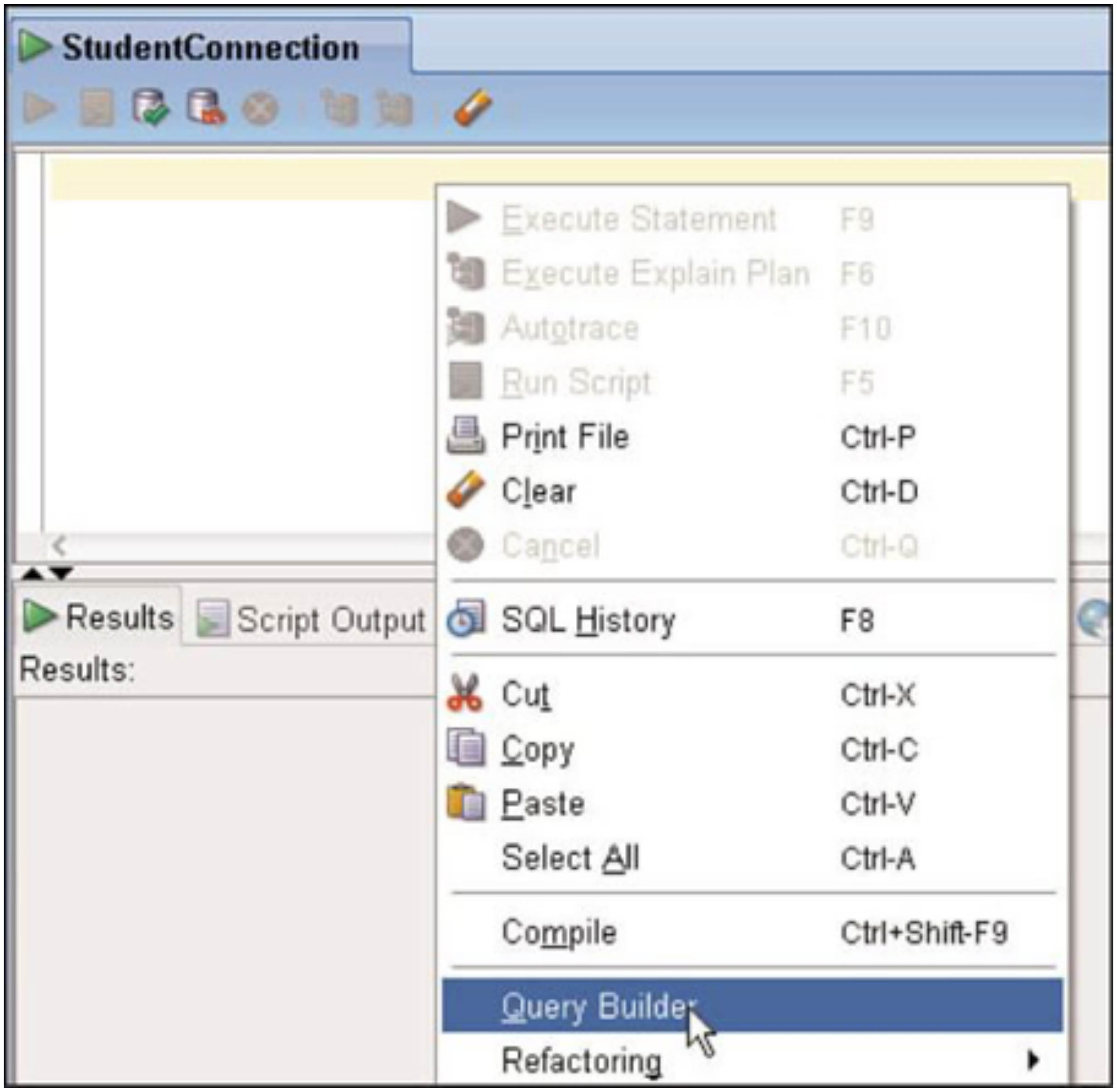
Although the traditional join syntax, with the columns separated by commas in the FROM clause and the join condition listed in the WHERE clause, may become the old way of writing SQL, you must nevertheless familiarize yourself with this syntax because millions of SQL statements already use it, and it clearly performs its intended purpose.

The ANSI join syntax has some distinct functional advantages over the traditional join syntax when it comes to outer joins, which you can learn about in Chapter 10.

Using SQL Developer's Query Builder

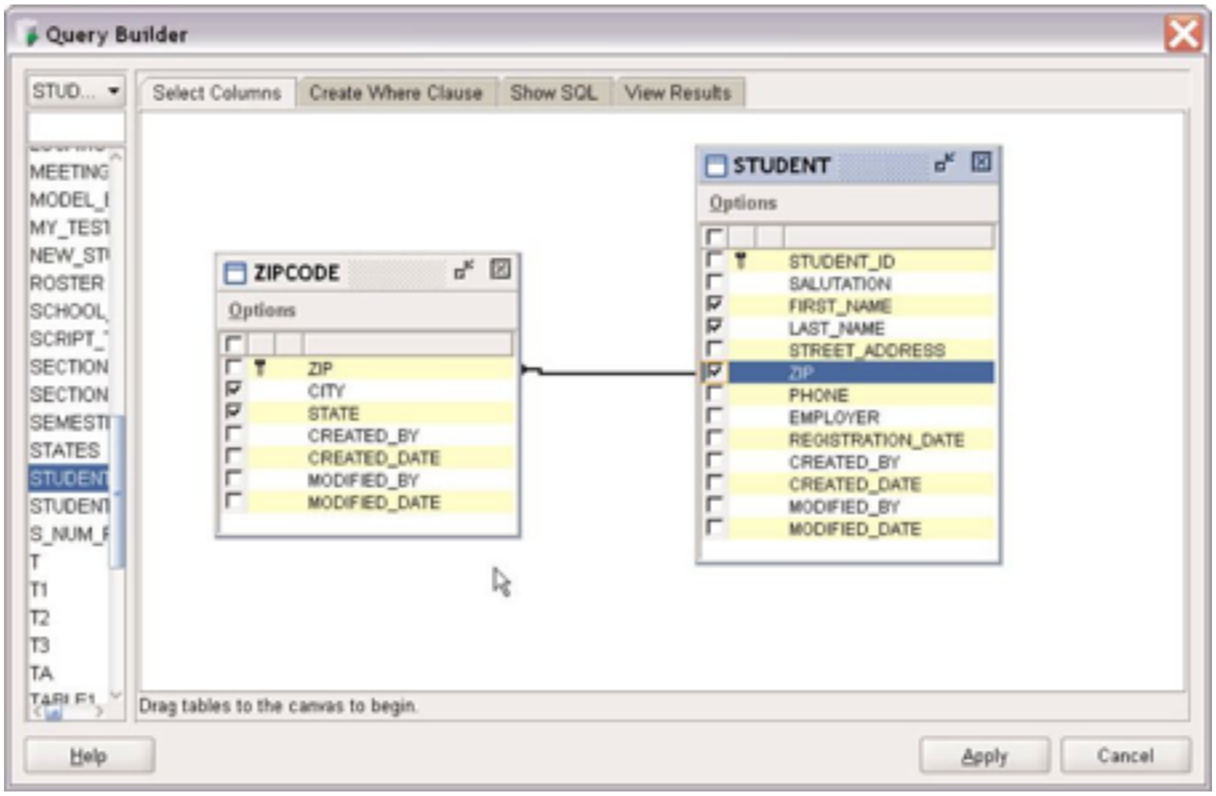
Instead of typing your SQL statement and figuring out the joining columns, SQL Developer performs these tasks for you with the help of the Query Builder. As Figure 7.8 shows, you invoke the Query Builder from the SQL Worksheet by right-clicking and selecting Query Builder.

FIGURE 7.8 Query Builder



Query Builder allows you to drag the desired tables to the canvas. It builds the join criteria based on the existing primary/foreign key relationship and visually displays the relationship line between the tables (see Figure 7.9). You can check the columns you want to include in the query. To narrow down the query output, use the Create Where Clause tab at the top of the screen. SQL Developer generates a SQL statement you can view and execute.

FIGURE 7.9 Query Builder canvas



Different Types of Joins

Most of the joins you will come across are based on equality, with the equijoin being the most dominant. In this chapter, you have learned about equijoins; there are other types of joins you must become familiar with, most notably the self-join, the nonequijoin, and the outer join. Table 7.1 lists the various types of joins.

TABLE 7.1 Types of Joins

JOIN TYPE	BASE OF JOIN CONDITION	LEARN ABOUT IT	SYNTAX
Equijoin or inner join	Equality	This chapter	Traditional comma-separated join or ANSI join syntax (including the optional INNER keyword).
Natural join	Equality	This chapter	NATURAL JOIN keyword.
Cross-join or Cartesian product	No join condition	This chapter	Traditional comma-separated with the missing join condition in the WHERE clause or CROSS JOIN keyword.
Self-join	Equality	Chapter 10	See equijoin or inner join.
Outer join (left, right, full)	Equality and extending the result set	Chapter 10	OUTER JOIN keywords or outer join operator (+).

Nonequijoin	Nonequality of values	Chapter 10	Traditional comma-separated join or ANSI join syntax with the ON clause. Join criteria is not based on equality.

LAB 7.2 EXERCISES

a) Display the student ID, course number, and section number of enrolled students where the instructor of the section lives in zip code 10025. In addition, the course should not have any prerequisites.

b) Produce the mailing addresses for instructors who taught sections that started in June 2007.

c) List the student IDs of enrolled students living in Connecticut.

d) Show all the grades student Fred Crocitto received for SECTION_ID 86.

e) List the final examination grades for all enrolled Connecticut students of course number 420. (Note that final examination grade does not mean final grade.)

f) Display the LAST_NAME, STUDENT_ID, PERCENT_OF_FINAL_GRADE, GRADE_TYPE_CODE, and NUMERIC_GRADE columns for students who received 80 or less for their class project (GRADE_TYPE_CODE = 'PJ'). Order the result by student last name.

LAB 7.2 EXERCISE ANSWERS

a) Display the student ID, course number, and section number of enrolled students where the instructor of the section lives in zip code 10025. In addition, the course should not have any prerequisites.

ANSWER: This query involves joining four tables. The course number is found in the SECTION and COURSE tables and the PREREQUISITE column in the COURSE table. To determine the zip code of an instructor, use the INSTRUCTOR table. To choose only enrolled students, join to the ENROLLMENT table.

```
SELECT c.course_no, s.section_no, e.student_id
  FROM course c, section s, instructor i, enrollment e
 WHERE c.prerequisite IS NULL
   AND c.course_no = s.course_no
   AND s.instructor_id = i.instructor_id
   AND i.zip = '10025'
   AND s.section_id = e.section_id
COURSE_NO SECTION_NO STUDENT_ID
--------- ---------- ----------
       10          2        128
      146          2        117
      146          2        140
...
       20          8        158
       20          8        199

12 rows selected.
```

To obtain this result, build the four-table join as you would any other join, step by step. First, start with one of the tables, such as the COURSE table.

```
SELECT course_no
FROM course
WHERE prerequisite IS NULL
```

For each of these courses, find the corresponding sections when you join the COURSE table with the SECTION table. Notice the bolded additions to the SQL statement.

```
SELECTc.course_no, s.section_no
FROM course c, section s
```

```
 WHEREc.prerequisite IS NULL
 AND c.course_no = s.course_no
```

Then include instructors who live in zip code 10025. The common column between SECTION and INSTRUCTOR is INSTRUCTOR_ID.

```
 SELECT c.course_no, s.section_no
 FROM course c, section s,
instructor i
 WHERE c.prerequisite IS NULL
 AND c.course_no = s.course_no
 AND s.instructor_id =
i.instructor_id
 AND i.zip = '10025'
```

Finally, join the results of the ENROLLMENT table via the SECTION_ID column, which leads you to the solution shown previously.

Instead of using the traditional join syntax to obtain the result, you can use the ANSI join syntax. The query may look similar to the following statement.

```
 SELECT course_no, section_no,
student_id
 FROM course JOIN section
 USING (course_no)
 JOIN instructor
 USING (instructor_id)
 JOIN enrollment
 USING (section_id)
 WHERE prerequisite IS NULL
 AND zip = '10025'
```

The following is another possible alternative using the ANSI join syntax, using the ON condition instead of the USING clause.

```
 SELECT c.course_no,
s.section_no, e.student_id
 FROM course c JOIN section s
 ON (c.course_no = s.course_no)
 JOIN instructor i
 ON (s.instructor_id =
i.instructor_id)
 JOIN enrollment e
 ON (s.section_id = e.section_id)
 WHERE c.prerequisite IS NULL
 AND i.zip = '10025'
```

b) Produce the mailing addresses for instructors who taught sections that started in June 2007.

ANSWER: This solution requires the join of three tables: You join the INSTRUCTOR, SECTION, and ZIPCODE tables to produce the mailing list.

```
SELECT i.first_name || ' ' ||i.last_name name,
       i.street_address, z.city || ', ' || z.state
       || ' ' || i.zip "City State Zip",
       TO_CHAR(s.start_date_time, 'MM/DD/YY') start_dt,
       section_id sect
  FROM instructor i, section s, zipcode z
 WHERE i.instructor_id = s.instructor_id
   AND i.zip = z.zip
   AND s.start_date_time >=
       TO_DATE('01-JUN-2007','DD-MON-YYYY')
   AND s.start_date_time <
       TO_DATE('01-JUL-2007','DD-MON-YYYY')
NAME          STREET_ADDRESS City State Zip     START_DT SECT
------------- -------------- ------------------ -------- ----
Fernand Hanks 100 East 87th  New York, NY 10015 06/02/07  117
Anita Morris  34 Maiden Lane New York, NY 10015 06/11/07   83
Anita Morris  34 Maiden Lane New York, NY 10015 06/12/07   91
Anita Morris  34 Maiden Lane New York, NY 10015 06/02/07  113
...
Gary Pertez   34 Sixth Ave   New York, NY 10035 06/12/07   90
Gary Pertez   34 Sixth Ave   New York, NY 10035 06/10/07  120
Gary Pertez   34 Sixth Ave   New York, NY 10035 06/03/07  143
Gary Pertez   34 Sixth Ave   New York, NY 10035 06/12/07  151
```

One of the first steps in solving this problem is to determine what columns and tables are involved. Look at the schema diagram in Appendix D, “STUDENT Database Schema,” or refer to the table and column comments listed in Appendix E, “Table and Column Descriptions.”

In this example, the instructor’s last name, first name, street address, and zip code are found in the INSTRUCTOR table. CITY, STATE, and ZIP are columns in the ZIPCODE table. The join also needs to include the SECTION table because the column START_DATE_TIME lists the date and time when the individual sections started. The next step is to determine the common columns. The ZIP column is the common column between the INSTRUCTOR and ZIPCODE tables. For every value in the ZIP column of the INSTRUCTOR table you have one corresponding ZIP value in the ZIPCODE table. For every value in the ZIPCODE table there may be zero, one, or multiple records in the INSTRUCTOR table. The join returns only the matching records.

The other common column is INSTRUCTOR_ID in the SECTION and INSTRUCTOR tables. Only instructors who teach have one or more rows in the SECTION table. Any section that does not have an instructor assigned is not taught.

As always, the query can be expressed with one of the ANSI join syntax variations.

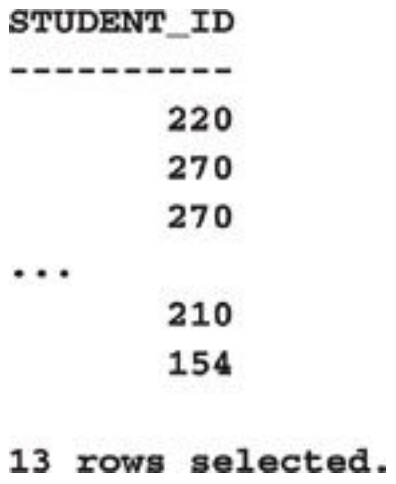
```
 SELECT first_name || ' ' ||
last_name name,
 street_address, city || ’, ‘ ||
state
 || ‘ ‘ || zip “City State Zip”,
 TO_CHAR(start_date_time, ‘MM/DD/
YY’) start_dt,
 section_id sect
 FROM instructor JOIN section s
 USING (instructor_id)
 JOIN zipcode
 USING (zip)
 WHERE start_date_time
>=TO_DATE(‘01-JUN-2007’,'DD-MON-
YYYY’)
 AND start_date_time <
TO_DATE(‘01-JUL-2007’,'DD-MON-
YYYY’)
```

You see that some instructors are teaching multiple sections. To see only the distinct addresses, use the DISTINCT keyword and drop the START_DATE_TIME and SECTION_ID columns from the SELECT list.

c) List the student IDs of enrolled students living in Connecticut.

ANSWER: Only students enrolled in classes are in the result; any student who does not have a row in the ENROLLMENT table is not considered enrolled. STUDENT_ID is the common column between the STUDENT and ENROLLMENT tables. The STATE column is in the ZIPCODE table. The common column between the STUDENT and ZIPCODE tables is the ZIP column.

```
SELECT student_id
FROM student JOIN enrollment
USING (student_id)
JOIN zipcode
USING (zip)
WHERE state = ‘CT’
```



```
STUDENT_ID
----------
       220
       270
       270
...
       210
       154

13 rows selected.
```

Because students can be enrolled in more than one class, add the DISTINCT keyword if you want to display each STUDENT_ID once.

Following is the SQL statement, expressed using the traditional join syntax. SELECT s.student_id

```
 FROM student s, enrollment e,
zipcode z
 WHERE s.student_id =
e.student_id
 AND s.zip = z.zip
 AND z.state = ‘CT’
```

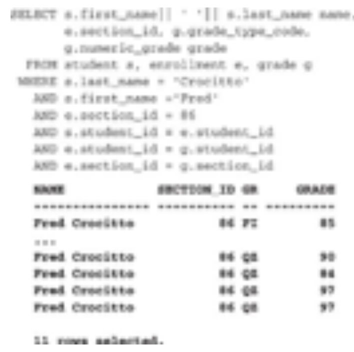
d) Show all the grades student Fred Crocitto received for SECTION_ID 86.

ANSWER: The grades for each section and student are stored in the GRADE table. The primary key of the GRADE table consists of the STUDENT_ID, SECTION_ID, GRADE_TYPE_CODE, and GRADE_CODE_OCCURRENCE columns. This means a student, such as Fred Crocitto, has multiple grades for each grade type.

```
 SELECT s.first_name|| ‘ ‘||
s.last_name name,
 e.section_id, g.grade_type_code,
 g.numeric_grade grade
 FROM student s JOIN enrollment e
 ON (s.student_id = e.student_id)
 JOIN grade g
 ON (e.student_id = g.student_id
 AND e.section_id = g.section_id)
 WHERE s.last_name = 'Crocitto'
 AND s.first_name = 'Fred'
 AND e.section_id = 86
```

The SQL statement using the traditional join syntax may look similar to the following query.

```
 SELECT s.first_name|| ' '||
s.last_name name,
 e.section_id, g.grade_type_code,
 g.numeric_grade grade
 FROM student s, enrollment e,
grade g
 WHERE s.last_name = 'Crocitto'
 AND s.first_name = 'Fred'
 AND e.section_id = 86
 AND s.student_id = e.student_id
 AND e.student_id = g.student_id
 AND e.section_id = g.section_id
```



```
SELECT s.first_name|| ' '|| s.last_name name,
       e.section_id, g.grade_type_code,
       g.numeric_grade grade
  FROM student s, enrollment e, grade g
 WHERE s.last_name = 'Crocitto'
   AND s.first_name ='Fred'
   AND e.section_id = 86
   AND s.student_id = e.student_id
   AND e.student_id = g.student_id
   AND e.section_id = g.section_id

NAME             SECTION_ID GR     GRADE
---------------- ---------- -- ---------
Fred Crocitto            86 FI        85
...
Fred Crocitto            86 QZ        90
Fred Crocitto            86 QZ        84
Fred Crocitto            86 QZ        97
Fred Crocitto            86 QZ        97

11 rows selected.
```

To build up the SQL statement step by step, you might want to start with the STUDENT table and select the record for Fred Crocitto.

```
SELECT last_name, first_name
FROM student
WHERE last_name = 'Crocitto'
AND first_name = 'Fred'
```

Next, choose the section with ID 86, in which Fred is enrolled. The common column between the two tables is STUDENT_ID.

```
SELECT s.first_name||' '||
s.last_name name
,e.section_id
FROM student s, enrollment e
WHEREs.last_name = 'Crocitto'
AND s.first_name = 'Fred'
AND e.section_id = 86
AND s.student_id = e.student_id
```

Next, retrieve the individual grades from the GRADE table. The common columns between the GRADE table and the ENROLLMENT table are SECTION_ID and STUDENT_ID. They represent the primary key in the ENROLLMENT table and are foreign keys in the GRADE table. Both columns need to be in the WHERE clause.

If you want to expand the query, add the DESCRIPTION column of the GRADE_TYPE table for each GRADE_TYPE_CODE. The common column between the tables GRADE and GRADE_TYPE is GRADE_TYPE_CODE.

```
 SELECT s.first_name||' `||
s.last_name name,
 e.section_id, g.grade_type_code
grade,
 g.numeric_grade, gt.description
 FROM student s, enrollment e,
grade g, grade_type gt
 WHERE s.last_name = ‘Crocitto’
 AND s.first_name = ‘Fred’
 AND e.section_id = 86
 AND s.student_id = e.student_id
 AND e.student_id = g.student_id
 AND e.section_id = g.section_id
 AND g.grade_type_code =
gt.grade_type_code
```

If you also show the COURSE_NO column, join to the SECTION table via the ENROLLMENT table column SECTION_ID.

```
SELECT s.first_name||' '|| s.last_name name,
       e.section_id, g.grade_type_code,
       g.numeric_grade grade, gt.description,
       sec.course_no
  FROM student s, enrollment e, grade g, grade_type gt,
       section sec
 WHERE s.last_name = 'Crocitto'
   AND s.first_name = 'Fred'
   AND e.section_id = 86
   AND s.student_id = e.student_id
   AND e.student_id = g.student_id
   AND e.section_id = g.section_id
   AND g.grade_type_code = gt.grade_type_code
   AND e.section_id = sec.section_id
NAME          SECTION_ID GR GRADE DESCRIPTION COURSE_NO
------------- ---------- -- ----- ----------- ---------
Fred Crocitto         86 FI    85 Final              25
...
Fred Crocitto         86 QZ    90 Quiz               25
Fred Crocitto         86 QZ    84 Quiz               25
Fred Crocitto         86 QZ    97 Quiz               25
Fred Crocitto         86 QZ    97 Quiz               25

11 rows selected.
```

e) List the final examination grades for all enrolled Connecticut students of course number 420. (Note that final examination grade does not mean final grade.)

ANSWER: To find the answer, you need to join five tables. The needed joins are the ZIPCODE table with the STUDENT table to determine the Connecticut students and the STUDENT and ENROLLMENT tables to determine the SECTION_IDs in which the students are enrolled. From these SECTION_IDs, you include only sections where the course number equals 420. This requires a join of the ENROLLMENT table to the SECTION table. Finally, join the ENROLLMENT table to the GRADE table to display the grades.

```
SELECT e.student_id, sec.course_no, g.numeric_grade
  FROM student stud, zipcode z,
       enrollment e, section sec, grade g
 WHERE stud.zip = z.zip
   AND z.state = 'CT'
   AND stud.student_id = e.student_id
   AND e.section_id = sec.section_id
   AND e.section_id = g.section_id
   AND e.student_id = g.student_id
   AND sec.course_no = 420
   AND g.grade_type_code = 'FI'
STUDENT_ID COURSE_NO NUMERIC_GRADE
---------- --------- -------------
       196       420            84
       198       420            85

2 rows selected.
```

You can list any of the columns you find relevant to solving the query. For this solution, the columns STUDENT_ID, COURSE_NO, and NUMERIC_GRADE were chosen.

The query can also be expressed with the ANSI join syntax, as in the following example, which shows the USING clause.

```
 SELECT student_id, course_no,
numeric_grade
 FROM student JOIN zipcode
 USING (zip)
 JOIN enrollment
 USING (student_id)
 JOIN section
 USING (section_id)
 JOIN grade g
 USING (section_id, student_id)
 WHERE course_no = 420
 AND grade_type_code = 'FI'
 AND state = 'CT'
```

f) Display the LAST_NAME, STUDENT_ID, PERCENT_OF_FINAL_GRADE, GRADE_TYPE_CODE, and NUMERIC_GRADE columns for students who received 80 or less for their class project (GRADE_TYPE_CODE = ‘PJ’). Order the result by student last name.

ANSWER: Join the tables GRADE_TYPE_WEIGHT, GRADE, ENROLLMENT, and STUDENT.

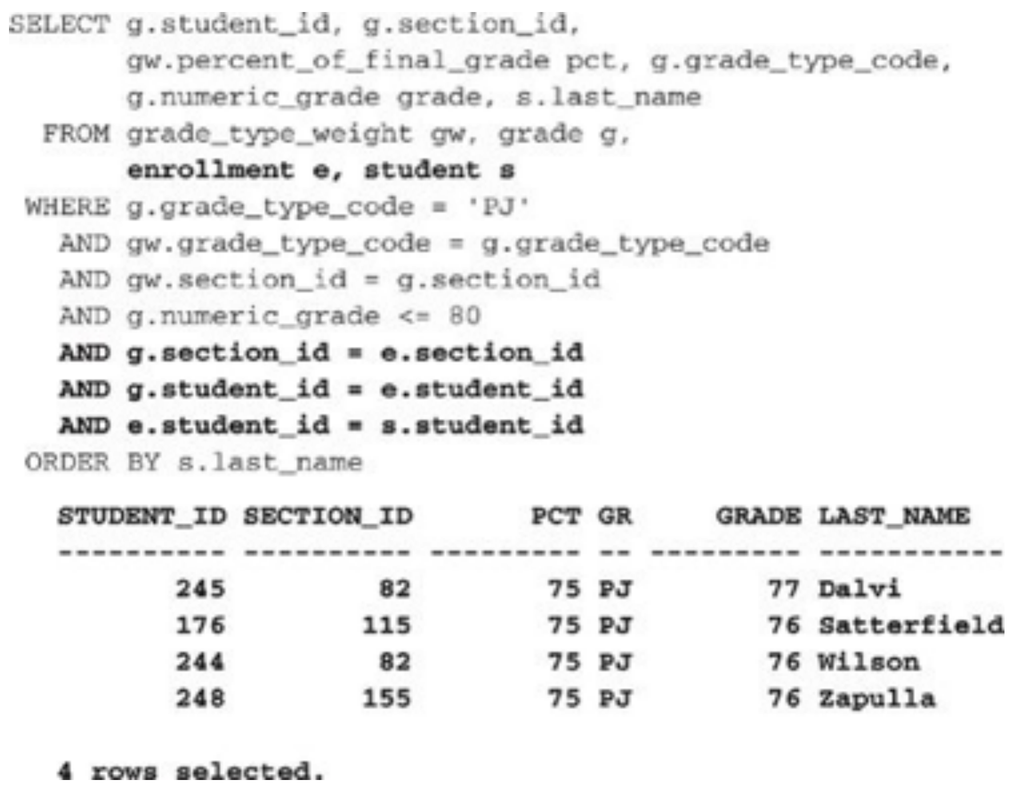
The column PERCENT_OF_FINAL_GRADE of the GRADE_TYPE_WEIGHT table stores the weighted percentage a particular grade has on the final grade. One of the foreign keys of the GRADE table is the combination of GRADE_TYPE_CODE and SECTION_ID; these columns represent the primary key of the GRADE_TYPE_WEIGHT table.

To include the student’s last name, you have two choices. Either follow the primary and foreign key relationships by joining the tables GRADE and ENROLLMENT via the STUDENT_ID and SECTION_ID columns and then join the ENROLLMENT table to the STUDENT table via the STUDENT_ID column, or skip the ENROLLMENT table and join GRADE directly to the STUDENT table via STUDENT_ID. Examine the first option of joining to the ENROLLMENT table and then joining it to the STUDENT table.

```
 SELECT g.student_id,
g.section_id,
 gw.percent_of_final_grade pct,
g.grade_type_code,
 g.numeric_grade grade,
s.last_name
 FROM grade_type_weight gw, grade
g,
 enrollment e, student s
 WHERE g.grade_type_code = 'PJ'
 AND gw.grade_type_code =
g.grade_type_code
 AND gw.section_id = g.section_id
 AND g.numeric_grade <= 80
 AND g.section_id = e.section_id
 AND g.student_id = e.student_id
 AND e.student_id = s.student_id
 ORDER BY s.last_name
```

317

318



```
SELECT g.student_id, g.section_id,
       gw.percent_of_final_grade pct, g.grade_type_code,
       g.numeric_grade grade, s.last_name
  FROM grade_type_weight gw, grade g,
       enrollment e, student s
 WHERE g.grade_type_code = 'PJ'
   AND gw.grade_type_code = g.grade_type_code
   AND gw.section_id = g.section_id
   AND g.numeric_grade <= 80
   AND g.section_id = e.section_id
   AND g.student_id = e.student_id
   AND e.student_id = s.student_id
 ORDER BY s.last_name

STUDENT_ID SECTION_ID       PCT GR     GRADE LAST_NAME
---------- ---------- --------- -- --------- -----------
       245         82        75 PJ        77 Dalvi
       176        115        75 PJ        76 Satterfield
       244         82        75 PJ        76 Wilson
       248        155        75 PJ        76 Zapulla

4 rows selected.
```

SKIPPING THE PRIMARY/FOREIGN KEY PATH

The second choice is to join the STUDENT_ID from the GRADE table directly to the STUDENT_ID of the STUDENT table, thus skipping the ENROLLMENT table entirely. The following query returns the same result as the query in the preceding section.

```
SELECT g.student_id, g.section_id,
       gw.percent_of_final_grade pct, g.grade_type_code,
       g.numeric_grade grade, s.last_name
  FROM grade_type_weight gw, grade g,
       student s
 WHERE g.grade_type_code = 'PJ'
   AND gw.grade_type_code = g.grade_type_code
   AND gw.section_id = g.section_id
   AND g.numeric_grade <= 80
   AND g.student_id = s.student_id
 ORDER BY s.last_name
STUDENT_ID SECTION_ID         PCT GR      GRADE LAST_NAME
---------- ---------- ------------ -- ---------------------
       245         82           75 PJ         77 Dalvi
       176        115           75 PJ         76 Satterfield
       244         82           75 PJ         76 Wilson
       248        155           75 PJ         76 Zapulla

4 rows selected.
```

This shortcut is perfectly acceptable, even if it does not follow the primary key/foreign key relationship path. In this case, you can be sure not to build a Cartesian product because you can guarantee only one STUDENT_ID in the STUDENT table for every STUDENT_ID in the GRADE table. In addition, it also eliminates a join; thus, the query executes a little faster and requires fewer resources. The effect is probably fairly negligible with this small result set.

Lab 7.2 Quiz

In order to test your progress, you should be able to answer the following questions.

1) Which SQL statement shows the sections that have instructors assigned to them?

______a)
```
 SELECT c.course_no,
s.section_id,
i.instructor_id

 FROM course c, section s,
instructor i
 WHERE c.course_no =
s.course_no
 AND i.instructor_id =
s.section_id
```

______b)
```
 SELECT c.course_no,
s.section_id,
i.instructor_id
```

```
 FROM course c, section s,
instructor i
 WHERE c.course_no =
s.course_no
 AND i.instructor_id =
s.instructor_id
```

______**c)**

```
 SELECT course_no,
section_id,
instructor.instructor_id

 FROM section, instructor
 WHERE
instructor.instructor_id =
section.section_id
```

______**d)**

```
 SELECT c.section_id,
i.instructor_id

 FROM course c, instructor
i
 WHERE i.instructor_id =
c.section_id
```

______**e)**

```
 SELECT c.course_no,
i.instructor_id

 FROM course c JOIN
instructor
 USING (instructor_id)
```

2) How do you resolve the Oracle error ORA-00918: column ambiguously defined?

a) ______Correct the join criteria and WHERE clause condition.

b) ______Choose another column.

c) ______Add the correct table alias.

d) ______Correct the spelling of the column name.

3) Joins involving multiple columns must always follow the primary key/foreign key relationship path.

a) ______True

b) ______False

4) Find the error(s) in the following SQL statement.

```
1 SELECT g.student_id, s.section_id,
2 g.numeric_grade, s.last_name
3 FROM grade g,
4 enrollment e, student s
5 WHERE g.section_id = e.section_id
```

```
6 AND g.student_id = e.student_id
7 AND s.student_id = e.student_id
8 AND s.student_id = 248
9 AND e.section_id = 155
```

a) ______Line 1

b) ______Line 5

c) ______Line 6

d) ______Lines 5 and 6

e) ______Lines 1, 5, and 6

f) ______No error

5) Equijoins are the most common type of joins and are always based on equality of values.

a) ______True

b) ______False

6) To join four tables, you must have at least three join conditions.

a) ______True

b) ______False

WORKSHOP

The projects in this section are meant to prompt you to utilize all the skills you have acquired throughout this chapter. The answers to these projects can be found at the companion Web site to this book, located at www.oraclesqlbyexample.com.

1) Select the course description, section number, and location for sections meeting in location L211.

2) Show the course description, section number, and starting date and time of the courses Joseph German is taking.

3) List the instructor ID, last name of the instructor, and section ID of sections where class participation contributes to 25 percent of the total grade. Order the result by the instructor's last name.

4) Display the first and last names of students who received 99 or more points on the class project.

5) Select the grades for quizzes of students living in zip code 10956.

6) List the course number, section number, and instructor first and last names for classes with course number 350 as a prerequisite.

7) What problem do the following two SELECT statements solve? Explain the difference between the two results.

```
 SELECT stud.student_id,
i.instructor_id,
 stud.zip, i.zip
 FROM student stud, instructor i
 WHERE stud.zip = i.zip
 SELECT stud.student_id,
i.instructor_id,
 stud.zip, i.zip
 FROM student stud, enrollment e,
section sec,
 instructor i
 WHERE stud.student_id =
e.student_id
 AND e.section_id =
sec.section_id
 AND sec.instructor_id =
i.instructor_id
 AND stud.zip = i.zip
```