CHAPTER 10 Complex Joins

CHAPTER OBJECTIVES

In this chapter, you will learn about:

- Outer Joins
- Self-Joins

Outer joins and self-joins are extensions of the equijoins you learned about in Chapter 7, "Equijoins." An outer join includes the result rows returned by an equijoin, plus extra rows where no matches are found. The self-join, as implied by the name, joins a table to itself. This type of join is useful for tables with a self-referencing relationship or when you want to determine data inconsistencies. Self-joins are useful for analyzing and exploring the relationships within your data.

LAB 10.1 Outer Joins

LAB OBJECTIVES

After this lab, you will be able to:

- Write Outer Joins with Two Tables
- Write Outer Joins with Three Tables

An outer join is similar to an equijoin because it returns all the records an equijoin returns. But it also returns records that are in one of the tables with no matching records in another table. The following example shows an equijoin and its result. The SQL statement returns all the rows where a match for the COURSE_NO column is found in both the COURSE and SECTION tables.

```
SELECT course_no, description,
       section_id
  FROM course JOIN section
 USING (course_no)
 ORDER BY course_no
COURSE_NO DESCRIPTION                    SECTION_ID COURSE_NO
--------- ------------------------------ ---------- ---------
       10 Technology Concepts                    80        10
       20 Intro to Information Systems           81        20
...
      420 Database System Principles            108       420
      450 DB Programming with Java              109       450

78 rows selected.
```

Some courses are not included in the result because there are no matching course numbers in the SECTION table. To determine which courses are not assigned to any

sections, write a NOT EXISTS subquery, a NOT IN subquery, or use the MINUS operator.


```
SELECT course_no, description
  FROM course c
 WHERE NOT EXISTS
       (SELECT 'X'
          FROM section
         WHERE c.course_no = course_no)
COURSE_NO DESCRIPTION
--------- --------------------------------
       80 Programming Techniques
      430 Java Developer III

2 rows selected.
```

The previous equijoin does not return the two courses because there are no matches for course numbers 80 and 430 in the SECTION table. To include these courses in the result, you need to perform an outer join.

The ANSI Outer Join

An outer join is typically formed with one of three different syntax options: Either you can use the ANSI join syntax, Oracle's outer join operator denoted with the (+), or you can express the query as a UNION. In this lab you will learn how to write the queries in various ways.

Typically, it is best to use the ANSI outer join syntax, as it greatly increases the SQL functionality and flexibility.

It can also be easily understood by any non-Oracle databases and is not subject to the many limitations the Oracle-specific outer join operator imposes. If your SQL needs to run against Oracle versions prior to 9*i*, you have no choice but to use Oracle's outer join operator or a UNION ALL set operator.

The following query shows the use of the ANSI outer join syntax. The keywords LEFT OUTER are added to the JOIN keyword, indicating that the rows in the table to the left side of the JOIN keyword are to be listed. This left table is the COURSE table, and all rows are shown, including any rows where there is no matching COURSE_NO value in the SECTION table.

```
SELECT c.course_no, c.description,
       s.section_id, s.course_no
  FROM course c LEFT OUTER JOIN section s
    ON c.course_no = s.course_no
 ORDER BY c.course_no
COURSE_NO DESCRIPTION                    SECTION_ID COURSE_NO
--------- ------------------------------ ---------- ---------
       10 Technology Concepts                    80        10
       20 Intro to Information Systems           81        20
...
       80 Programming Techniques
...
      430 Java Developer III
      450 DB Programming with Java              109       450

80 rows selected.
```

Look closely at the result for course numbers 80 and 430. These courses have no sections assigned. For example, COURSE_NO 430 of the COURSE table (c.course_no) shows the COURSE_NO value, but COURSE_NO from the SECTION table (s.course_no) displays a null.

The outer join displays null values for the columns s.course_no and s.section_id where no match exists.

If you want to include all the rows in the SECTION table, you can use the RIGHT OUTER JOIN syntax. This does not really make any sense when reviewing the schema diagram: Every row in the SECTION table must have a corresponding row in the COURSE table. Orphan rows, which are rows that exist in the SECTION table but not in the COURSE table, are not allowed.

However, if you switch the order of the tables to list the SECTION table first, you can write a RIGHT OUTER JOIN. This is essentially the same as the previous LEFT OUTER JOIN. The only differences are the order of the tables in the FROM clause and the RIGHT keyword. Based on the order of the tables in the FROM clause, you must choose either the RIGHT or the LEFT keyword to include all the rows of the outer joined table.

```
SELECT c.course_no, c.description,
       s.section_id, s.course_no
  FROM section s RIGHT OUTER
       JOIN course c
    ON c.course_no = s.course_no
ORDER BY c.course_no
```

You can also write the outer join with another ANSI join syntax, such as the USING clause.

```
SELECT course_no, description,
       section_id
  FROM section RIGHT OUTER JOIN course
 USING (course_no)
 ORDER BY course_no
COURSE_NO DESCRIPTION                    SECTION_ID
--------- ------------------------------ ----------
       10 Technology Concepts                    80
       20 Intro to Information Systems           81
...
       80 Programming Techniques
...
      430 Java Developer III
      450 DB Programming with Java              109

80 rows selected.
```

The query and the returned result do not include both COURSE_NO columns because you are not allowed to alias the joined column when writing the query with the USING clause. The column now also contains a non-null

value, unlike the previous result, where the corresponding COURSE_NO column from the SECTION table showed a null. For the SECTION_ID column, you continue to see a null value because there is obviously no matching value in the SECTION table.

The Oracle Outer Join Operator (+)

The second way to express an outer join is with Oracle's outer join operator (+). The next query looks very much like the equijoin you are already familiar with, except for the (+) next to the SECTION table's COURSE_NO column in the WHERE clause. Oracle uses the outer join operator (+) to indicate that nulls are shown for non-matching rows. As in the previous result set, for those rows of the COURSE table where a match does not exist in the SECTION table (course numbers 80 and 430), there are null values displayed in the SECTION table columns.

You place the outer join operator on the table for which you want to return null column values when a match is not found. In this case, all the rows from the COURSE table are desired, and for every row in the COURSE table for which a match cannot be found in the SECTION table, a null is shown. Therefore, the (+)

operator is placed on the COURSE_NO column of the SECTION table.

```
SELECT c.course_no, c.description,
       s.section_id, s.course_no
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
 ORDER BY c.course_no
```

If the order of the tables in the FROM clause changes, the (+) operator still needs to remain on the S.COURSE_NO column.

The Outer Join and the UNION ALL Operator

Instead of using the (+) operator or the ANSI join syntax, you can achieve the same result with two SQL statements: an equijoin and a correlated subquery, with the results combined using the UNION ALL operator.

```
SELECT c1.course_no, c1.description,
       s.section_id, s.course_no
  FROM course c1, section s
 WHERE c1.course_no = s.course_no
 UNION ALL
SELECT c2.course_no, c2.description,
       TO_NUMBER(NULL),
       TO_NUMBER(NULL)
  FROM course c2
 WHERE NOT EXISTS
       (SELECT 'X'
        FROM section
       WHERE c2.course_no =
       course_no)
```

In this example, the UNION ALL operator is used to combine the result of the equijoin (all courses with sections) with the result of the correlated subquery (courses with no match in the SECTION table). Duplicate rows are not returned between the two SELECT statements; each SELECT statement returns a different set. Therefore, it is more efficient to use UNION ALL rather than the UNION operator because UNION ALL avoids the sort required by the UNION operator to eliminate the duplicates.

The TO_NUMBER data type conversion is performed to match the data types of the columns in each of the SELECT statements in the set operation. Alternatively, you can substitute CAST (NULL AS NUMBER) for the TO_NUMBER(NULL) function.

The outer join (+) syntax and the UNION ALL syntax options are the only way to express outer joins in Oracle versions prior to Oracle9*i*. If you have a choice, use the ANSI join syntax instead; it is easy and, overall, more flexible and functional.

The Full Outer Join

A full outer join includes rows from both tables. Oracle does not support a full outer join with the (+) outer join operator. To accomplish a full outer join, you need to use either the ANSI full outer join syntax or the UNION operator.

ANSI FULL OUTER JOIN

To fully illustrate the effects of an outer join, the following are tables named T1 and T2 and the data in

them. Table T1 has one numeric column named COL1, and table T2 also consists of a numeric column called COL2. (These tables are not found in the STUDENT schema unless you installed the additional tables from the companion Web site.)

```
SELECT col1
  FROM t1
     COL1
---------
         1
         2
         3

3 rows selected.


SELECT col2
  FROM t2
     COL2
---------
         2
         3
         4

3 rows selected.
```

To understand the result of a full outer join on tables T1 and T2, first write an outer join on table T1 with the following SELECT statement, which is a left outer join. The result includes all the rows from table T1.

```
SELECT col1, col2
  FROM t1 LEFT OUTER JOIN t2
    ON t1.col1 = t2.col2
     COL1 COL2
--------- ----
        1
        2    2
        3    3

3 rows selected.
```

The next SELECT statement returns all the rows from T2, whether a match is found or not. This outer join is a right outer join. All the rows on the right table are returned, including non-matching rows.

```
SELECT col1, col2
  FROM t1 RIGHT OUTER JOIN t2
    ON t1.col1 = t2.col2
     COL1 COL2
--------- ----
        2    2
        3    3
             4

3 rows selected.
```

The full outer join includes all the rows from both tables, whether a match is found or not.

FULL OUTER JOIN USING THE UNION OPERATOR

You can express the full outer join from the preceding section by using the Oracle outer join operator and combining the two SELECT statements with the UNION operator. The UNION operator eliminates the duplicate rows from the two statements. If you want to include duplicates, use UNION ALL.

```
SELECT col1, col2
    From t1, t2
WHERE t1.col1 = t2.col2(+)
 UNION
SELECT col1, col2
    From t1, t2
WHERE t1.col1(+) = t2.col2
```


The first SELECT statement performs an outer join on table T1; the second SELECT statement performs an outer join on table T2. The results of the queries are combined, and duplicates are eliminated with the UNION operator.

LAB 10.1 EXERCISES

a) Explain why Oracle returns an error message when you execute the following SELECT statement.

```
SELECT c.course_no,
       s.course_no, s.section_id,
       c.description,
       s.start_date_time
  FROM course c, section s
 WHERE c.course_no(+) =
       s.course_no(+)
```

b) Show the description of all courses with the prerequisite course number 350. Include in the result the location where the sections meet. Return course rows even if no corresponding row in the SECTION table is found.

c) Rewrite the following SQL statement using an outer join.

```
SELECT course_no, description
  FROM course c
 WHERE NOT EXISTS
       (SELECT 'X'
          FROM section
         WHERE c.course_no = course_no)
COURSE_NO DESCRIPTION
--------- --------------------------------
       80 Programming Techniques
      430 Java Developer III

2 rows selected.
```

d) Show all the city, state, and zip code values for Connecticut. Display a count of how many students live in each zip code. Order the result alphabetically by city. The result should look similar to the following output. Note that the column STUDENT_COUNT displays a zero when no student lives in a particular zip code.

```
CITY                      ST ZIP   STUDENT_COUNT
------------------------- -- ----- -------------
Ansonia                   CT 06401             0
Bridgeport                CT 06605             1
...
Wilton                    CT 06897             0
Woodbury                  CT 06798             1

19 rows selected.
```


e) Display the course number, description, cost, class location, and instructor's last name for all the courses. Also include courses where no sections or instructors have been assigned.

f) For students with the student ID 102 and 301, determine the sections they are enrolled in. Also show the numeric grades and grade types they received, regardless of whether they are enrolled or received any grades.

LAB 10.1 EXERCISE ANSWERS

a) Explain why Oracle returns an error message when you execute the following SELECT statement.

```
SELECT c.course_no,
       s.course_no, s.section_id,
       c.description,
       s.start_date_time
  FROM course c, section s
 WHERE c.course_no(+) =
       s.course_no(+)
```

ANSWER: The outer join symbol can be used only on one side of the equation, not both.

ERROR at line 4:

ORA-01468: a predicate may reference only one outer-joined table

This SQL statement attempts to include rows from the COURSE table for which no match exists in the SECTION table and include rows from the SECTION table where no match is found in the COURSE table. This is referred to as a full outer join; you want to include the rows

from both tables, including rows for which a match cannot be found in either table.

If you want to write such an outer join, use the ANSI join syntax instead.

```
SELECT c.course_no,
       s.course_no, s.section_id,
       c.description,
       s.start_date_time
  FROM course c FULL OUTER
       JOIN section s
    ON c.course_no =
       s.course_no
```

When you look at the relationship between the SECTION and COURSE tables, you notice a section cannot exist unless a corresponding course exists. Therefore, finding any sections for which no course exists is impossible unless the foreign key constraint is disabled or dropped. (To learn how to determine whether the foreign key is disabled or enabled, see Chapter 12,"Create, Alter, and Drop Tables.")

b) Show the description of all courses with the prerequisite course number 350. Include in the result the location where the sections meet.

Return course rows even if no corresponding row in the SECTION table is found.

ANSWER: To show all the courses with this prerequisite, you need to write an outer join. For any records where no match in the SECTION table is found, null values are displayed for the respective SECTION table columns.

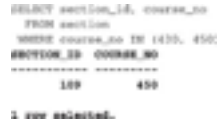
There are only two courses with a PREREQUISITE value of 350: courses 430 and 450.


```
SELECT course_no, description
FROM course
WHERE prerequisite = 350
```

```
 COURSE_NO DESCRIPTION
---------- ------------------------------
       430 Java Developer III
       450 DB Programming with Java

2 rows selected.
```

Only course number 450 has a matching course number in the SECTION table. Course number 430, Java Developer III, does not exist in the SECTION table.



The solution can be written as follows.

```
SELECT c.course_no cno, s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c LEFT OUTER JOIN section s
    ON c.course_no = s.course_no
 WHERE c.prerequisite = 350
CNO SNO DESCRIPTION               PREREQ LOC  SECTION_ID
--- --- ----------------------- ------ ---- ----------
430     Java Developer III         350
450 450 DB Programming with Java   350 L507        109

2 rows selected.
```

Alternatively, it can be written as follows.

```
SELECT c.course_no cno,
s.course_no sno,
   c.description,
   c.prerequisite prereq,
   s.location loc, s.section_id
 FROM course c, section s
WHERE c.course_no =
s.course_no(+)
 AND c.prerequisite = 350
```

As you see from the result, the columns of the SECTION table, such as S.LOCATION (with LOC as the column alias), S.COURSE_NO (with SNO as the column alias), and

S.SECTION_ID show null values for the unmatched row.

The solution can also be expressed with the USING clause as shown in the next statement. The COURSE_NO is listed here only once because the USING clause does not you to reference the COURSE_NO column on both tables.


```
SELECT course_no cno,
   description,
   prerequisite prereq,
   location loc, section_id
 FROM course LEFT OUTER JOIN
section
USING (course_no)
WHERE prerequisite = 350
```

ORACLE OUTER JOIN OPERATOR RESTRICTIONS

Oracle imposes a number of restrictions and caveats on using the (+) outer join operator. For example, the outer join operator restricts the use of an outer joined column involving a subquery. You also cannot use the OR logical operator and IN comparison. If you incorrectly

place the necessary (+) symbols on the correct WHERE clause conditions and join criteria, you form an equijoin rather than an outer join. This occurs without warning, as you will see in the following examples.

WHERE CONDITIONS AND THE ORACLE OUTER JOIN OPERATOR

There are some things you need to watch out for when you use Oracle's proprietary outer join operator, particularly when it comes to conditions in WHERE clauses. The previously listed outer join is repeated here for your reference. The condition in the WHERE clause is applied to the prerequisite column. This column is in the COURSE table, the outer joined table that includes all rows, including non-matching rows.

```
SELECT c.course_no cno,
       s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
   AND c.prerequisite = 350
```

The next SQL statement modifies the WHERE condition and adds a condition specific to the SECTION table. The query retrieves classes that meet only in S.LOCATION L507. Observe the output of the query and compare it to the previous result.

```
SELECT c.course_no cno, s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
   AND c.prerequisite = 350
   AND s.location = 'L507'
CNO SNO DESCRIPTION              PREREQ LOC  SECTION_ID
--- --- ------------------------ ------ ---- ----------
450 450 DB Programming with Java    350 L507        109

1 row selected.
```


You might wonder what happened to course number 430. The course is no longer included in the result, even though the outer join operator is applied to return all the rows, whether a match is found in the SECTION table or not.

When a WHERE clause contains a condition that compares a column from the outer joined table to a literal, such as the text literal ‘L507’, you also need to include the outer join operator on the column. Otherwise, Oracle returns only the results of the

equijoin rather than displaying nulls for the columns. The following query adds the outer join symbol to the LOCATION column.

```
SELECT c.course_no cno, s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
   AND c.prerequisite = 350
   AND s.location(+) = 'L507'
CNO SNO DESCRIPTION                PREREQ LOC  SECTION_ID
--- --- ------------------------ ------ ---- ----------
430     Java Developer III            350
450 450 DB Programming with Java      350 L507        109

2 rows selected.
```

These two records satisfy the condition of the prerequisite. The outer join operator applied to the S.LOCATION column includes records where either (a) the location equals L507, (b) the location is null, or (c) the location is different from L507. You will see an example shortly of why the location can be different.

When you apply the outer join operator to a column on the outer joined table, you need to understand that the conditions are processed in a particular order. First, the records on the table where you want to include all the rows are processed. This is the condition

C.PREREQUISITE = 350. Next, the matching records in the SECTION table are identified. If a match is not found, the records with the prerequisite 350 are still returned. The next condition, LOCATION(+) = 'L507', shows rows in the SECTION table that satisfy this condition; otherwise, a null is displayed.

What happens when you choose a different location, such as L210? Neither course meets in this location.

```
SELECT c.course_no cno, s.course_no sno,
       SUBSTR(c.description, 1,20),
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
   AND c.prerequisite = 350
   AND s.location(+) = 'L210'
CNO SNO DESCRIPTION          PREREQ LOC  SECTION_ID
--- --- -------------------- ------ ---- ----------
430     Java Developer III      350
450     DB Programming with Java    350

2 rows selected.
```

410

411

Here, you see both courses with this prerequisite. This contrasts with the earlier output because now both the LOCATION and SECTION_ID columns display nulls. When the WHERE clause is evaluated, the PREREQUISITE condition is evaluated first, and then matches are found in the SECTION table with the

condition s.location(+) = ‘L210’. Because none of the sections matches this LOCATION condition for this course number, nulls are shown for SECTION_ID and LOCATION.

WHERE CONDITIONS AND ANSI OUTER JOINS

When you compare the previous, proprietary Oracle syntax to the ANSI outer join syntax, you see that there are some differences. Following is the query of the first outer join with the condition location = 'L507'. The ANSI join returns the result of the equijoin, which is one row. This is not the desired result.

```
SELECT c.course_no cno, s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c LEFT OUTER JOIN section s
    ON c.course_no = s.course_no
 WHERE c.prerequisite = 350
   AND location = 'L507'
CNO SNO DESCRIPTION                PREREQ LOC  SECTION_ID
--- --- ------------------------- ------ ---- ----------
450 450 DB Programming with Java      350 L507         109

1 row selected.
```

Instead, the query needs to be changed with the use of parentheses to obtain the correct result. The order of

execution matters, and the order is determined by the parentheses. The join on the COURSE_NO column together with the condition location = 'L507' is enclosed by parentheses; the join and LOCATION condition are then executed first. This intermediate result includes all matching rows between the COURSE and SECTION tables based on the COURSE_NO column and those rows from the SECTION table where the LOCATION column has the value L507.

```
SELECT c.course_no cno,
       s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM course c LEFT OUTER JOIN
       section s
    ON (c.course_no = s.course_no
   AND location = 'L507')
 WHERE c.prerequisite = 350
```

Here is an intermediate result listing of the join condition without the WHERE clause condition applied.

411

412

Based on this intermediate result set, the WHERE clause is applied; only rows with a prerequisite value of 350 are chosen for output, and just two rows qualify for the final result. This works because the LOCATION column condition is part of the outer join criteria on the table for which nulls are to be displayed.

USING INLINE VIEWS AND OUTER JOINS

You can use inline views to control the execution order. You can get the same result as in the preceding section by using the following query, which chooses all the rows from the COURSE table with a value of 350 in the PREREQUISITE column. This result set is then left outer joined with the inline view of the SECTION table, which retrieves only sections with the location column value 'L507'.

```
SELECT c.course_no cno,
       s.course_no sno,
       c.description,
       c.prerequisite prereq,
       s.location loc, s.section_id
  FROM (SELECT *
        FROM course
        WHERE prerequisite = 350) c
       LEFT OUTER JOIN
       (SELECT * FROM section
        WHERE location = 'L507') s
    ON (c.course_no = s.course_no)
```

WHERE clauses and outer joins may give you unexpected results unless you carefully craft your conditions. Inline views and the ANSI join syntax are best for complicated conditions because they give you control over the execution order of the conditions and subsequent joins. ANSI joins are preferable over Oracle's proprietary outer join operator because the ANSI join syntax is less restrictive and easier to read, and it allows your SQL statements to be portable to non-Oracle databases.

c) Rewrite the following SQL statement using an outer join.

```
SELECT course_no, description
  FROM course c
 WHERE NOT EXISTS
       (SELECT 'X'
          FROM section
         WHERE c.course_no = course_no)
COURSE_NO DESCRIPTION
--------- --------------------------------
       80 Programming Techniques
      430 Java Developer III

2 rows selected.
```


ANSWER: You can rewrite a NOT EXISTS condition as an outer join condition by querying the SECTION table for nulls.

You can write your query in many different ways. The following example shows the use of the (+) outer join operator.

```
SELECT c.course_no, c.description
  FROM course c, section s
 WHERE c.course_no =
       s.course_no(+)
   AND s.course_no IS NULL
```

Or you can write it with the ANSI outer join syntax and the USING clause.

```
SELECT course_no, description
  FROM course LEFT OUTER JOIN
       section
 USING (course_no)
 WHERE section_id IS NULL
```

d) Show all the city, state, and zip code values for Connecticut. Display a count of how many students live in each zip code. Order the result alphabetically by city. The result should look similar to the following output. The column STUDENT_COUNT displays a zero when no student lives in a particular zip code.

```
CITY                      ST ZIP   STUDENT_COUNT
------------------------- -- ----- -------------
Ansonia                   CT 06401             0
Bridgeport                CT 06605             1
...
Wilton                    CT 06897             0
Woodbury                  CT 06798             1

19 rows selected.
```

ANSWER: The query that achieves the correct solution requires the use of an outer join on the ZIPCODE table and the use of the aggregate function COUNT. When using an aggregate function together with outer joins, you must be

careful to apply the aggregate function to the correct column.

```
SELECT city, state, z.zip,
       COUNT(s.zip) AS
       student_count
  FROM zipcode z LEFT OUTER
       JOIN student s
    ON (z.zip = s.zip)
 WHERE state = 'CT'
 GROUP BY city, state, z.zip
```

In this query, the parameter in the COUNT function is the S.ZIP column instead of Z.ZIP. The COUNT function requires the STUDENT table's ZIP column as a parameter to ensure that if the zip code is not found in the STUDENT table, the COUNT function will return a zero.

To illustrate this important issue, the following query shows the result of both the Z.ZIP and S.ZIP columns in the SELECT list and as a parameter in the COUNT function. The SZIP column in the result is null, and the column WRONG_VALUE has a count of one even though this zip code does not exist in the STUDENT table. The COUNT function for this column is counting the

occurrence of the zip code in the ZIPCODE table, not the desired STUDENT table's zip code.


```
SELECT city, state, z.zip AS zzip, s.zip AS szip,
       COUNT(s.zip) AS student_count,
       COUNT(z.zip) AS wrong_value
  FROM zipcode z LEFT OUTER JOIN student s
    ON (z.zip = s.zip)
 WHERE state = 'CT'
 GROUP BY city, state, z.zip, s.zip
CITY                 ST ZZIP  SZIP  STUDENT_COUNT WRONG_VALUE
-------------------- -- ----- ----- ------------- -----------
Ansonia              CT 06401                   0           1
...
Woodbury             CT 06798 06798             1           1

19 rows selected.
```

ALTERNATIVE SOLUTION WITH A SCALAR SUBQUERY

A scalar subquery, discussed in Chapter 8,"Subqueries," can provide some simplicity and less chance for errors. The query is correlated as the scalar subquery is executed for each individual zip code. However, scalar subqueries can be notoriously inefficient because Oracle can often execute table joins more quickly, particularly when scans of the entire result set are involved.

e) Display the course number, description, cost, class location, and instructor's last name for all the courses. Also include courses where no sections or instructors have been assigned.

ANSWER: This outer join involves three tables: COURSE, SECTION, and INSTRUCTOR. You want to include all the courses from the COURSE table, whether a section exists for it or not. Also, if no instructor is assigned to a section or no match is found, the rows of the SECTION table should still be included.

```
SELECT course_no cou, description, cost,
       location, last_name
  FROM course LEFT OUTER JOIN section
 USING (course_no)
  LEFT OUTER JOIN instructor
 USING (instructor_id)
 ORDER BY course_no
```

```
COU DESCRIPTION                    COST LOCA LAST_NAME
---- ---------------------------- ---- ---- ---------
  10 Technology Concepts          1195 L214 Wojick
  20 Intro to Information Systems 1195 L210 Schorin
  20 Intro to Information Systems 1195 L214 Pertez
  20 Intro to Information Systems 1195 L509 Morris
  20 Intro to Information Systems 1195 L210 Smythe
...
 430 Java Developer III           1195
 450 DB Programming with Java          L507 Hanks

80 rows selected.
```

When you review the result, recall from the previous examples that course number 430 does not have a section assigned. Therefore, the column LOCATION displays a null value. Also, the instructor's last name shows a null value because there cannot be an instructor assigned if a row in the SECTION table does not exist.

Alternatively, you can use the Oracle outer join operator. In this case, the SQL statement looks similar to the following.

```
SELECT c.course_no cou,
       c.description, c.cost,
       s.location, i.last_name
  FROM course c, section s,
       instructor i
 WHERE c.course_no =
       s.course_no(+)
   AND s.instructor_id =
       i.instructor_id(+)
ORDER BY c.course_no
```

The SELECT statement requires the outer join operator to be placed on the COURSE_NO column of the SECTION table. This indicates that you want to see all the courses, whether there are

corresponding sections or not. The outer join operator is also applied to the INSTRUCTOR_ID column of the INSTRUCTOR table. This directs Oracle to include rows from the SECTION table even if it doesn't find a matching record in the INSTRUCTOR table.

f) For students with the student ID 102 and 301, determine the sections they are enrolled in. Also show the numeric grades and grade types they received, regardless of whether they are enrolled or received any grades.

ANSWER: You can write outer joins that include rows from all three tables: STUDENT, ENROLLMENT, and GRADE.

```
SELECT student_id, section_id,
       grade_type_code,
       numeric_grade
  FROM student LEFT OUTER JOIN
       enrollment
 USING (student_id)
  LEFT OUTER JOIN grade
 USING (student_id, section_id)
 WHERE student_id IN (102, 301)
```

Or you can write them as follows.

```
    ON (s.student_id = en.student_id)
  LEFT OUTER JOIN grade g
    ON (s.student_id = g.student_id
   AND en.section_id = g.section_id)
 WHERE s.student_id IN (102, 301)
STUDENT_ID SECTION_ID GR NUMERIC_GRADE
---------- ---------- -- -------------
       102         86 FI             85
       102         86 HM             90
       102         86 HM             99
       102         86 HM             82
       102         86 HM             82
       102         86 MT             90
       102         86 PA             85
       102         86 QZ             90
       102         86 QZ             84
       102         86 QZ             97
       102         86 QZ             97
       102         89 FI             92
       102         89 MT             91
       301

14 rows selected.
```

The student with ID 102 is enrolled and received grades. His rows are returned as part of an equi-join. However, student 301 is not enrolled in any section and does not have any grades.

You can also write the query using the traditional join syntax and the Oracle (+) outer join operator.

```
SELECT s.student_id, e.section_id,
       g.grade_type_code,
       g.numeric_grade
  FROM student s, enrollment e,
       grade g
 WHERE s.student_id IN (102, 301)
   AND s.student_id =
       e.student_id(+)
   AND e.student_id =
       g.student_id(+)
   AND e.section_id =
       g.section_id(+)
```

Because the outer join operator is applied to both the ENROLLMENT and the GRADE tables, STUDENT_ID 301 is included in the result. The condition s.student_id IN (102, 301) does not require an outer join operator because it is based on the STUDENT table, and this is the table from which you want all the rows that satisfy this condition.

Lab 10.1 Quiz

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

1) A WHERE clause containing an outer join (+) operator cannot contain another condition with the OR operator, as in the following example.

```
SELECT *
  FROM course c, section s
 WHERE c.course_no =
       s.course_no(+)
   OR c.course_no = 100
```

_____ **a)** True

_____ **b)** False

2) A column with the outer join (+) operator cannot use the IN operator, as in the following example.

```
SELECT *
  FROM course c, section s
 WHERE c.course_no =
       s.course_no(+)
   AND c.course_no(+) IN (100, 200)
```

_____ **a)** True

_____ **b)** False

3) An outer join between two tables returns all rows that satisfy the equijoin condition plus records from

the outer joined tables for which no matches are found.

_____**a)** True

_____**b)** False

4) Which of the WHERE clauses results in the following error message?

ORA-01468: a predicate may reference only one outer joined table

```
SELECT c.course_no, s.course_no,
       SUBSTR(c.description,
       1,20), s.start_date_time
  FROM course c, section s
```

_____**a)** WHERE course_no = course_no

_____**b)** WHERE c.course_no(+) = s.course_no

_____**c)** WHERE c.course_no = s.course_no(+)

_____**d)** WHERE c.course_no(+) = s.course_no(+)

ANSWERS APPEAR IN APPENDIX A.

LAB 10.2 Self-Joins

LAB OBJECTIVES

After this lab, you will be able to:

- Write Self-Joins
- Detect Data Inconsistencies

An equijoin always joins one or multiple tables. A self-join joins a table to itself by pretending there are different tables involved. This is accomplished by using table aliases. One table has one alias, and the same table also has another alias. For the purpose of executing the query, Oracle treats them as two different tables.

Constructing a Self-Join

Self-joins are quite useful for performing comparisons and checking for inconsistencies in data. Sometimes a self-join is needed to report on recursive relationships. Chapter 16, "Regular Expressions and Hierarchical Queries," provides detailed examples of reporting on recursive relationships using the CONNECT BY operator.

One example that lends itself very well to illustrating the functionality of self-joins is the COURSE table. The PREREQUISITE column is a foreign key to the primary key column COURSE_NO of the COURSE table, reflecting a recursive relationship between the two columns. PREREQUISITE is valid only if it is also a valid COURSE_NO; otherwise, the data manipulation operation on the table is rejected.

Many queries executed on the COURSE table so far in this book typically show only the prerequisite number, as in the following example.

```
SELECT course_no, description,
       prerequisite
FROM course
```

If you also want to show the description of the prerequisite, you need to write a self-join. This is accomplished by pretending to have two separate tables via table aliases, such as C1 and C2. You join the PREREQUISITE column of table C1 with the COURSE_NO column of table C2, If matching records are found, the description of the prerequisite is displayed.

418

419

Examine the first row of the result, COURSE_NO 230, with the prerequisite course number of 10. The course description for course number 10 is Technology Concepts. This join works just like the equijoins you learned about in Chapter 7. If a prerequisite is NULL or a match is not found, the self-join, just like an equijoin, does not return the record.

The self-join acts like other joins with primary key and foreign key columns. However, here the relationship is to the table itself. The PREREQUISITE column is a foreign key to the primary key COURSE_NO. The PREREQUISITE values come from the child table, and the COURSE_NO values come from the parent table. Every COURSE_NO may have zero or one PREREQUISITE.

To qualify as a prerequisite, the PREREQUISITE course number must be listed in the PREREQUISITE column for at least one course.

The USING clause cannot be applied to the self-join because this clause requires identical column names on both tables. This is obviously a problem because the join

needs to be executed on the columns PREREQUISITE and COURSE_NO.

The query can also be expressed in the traditional join format with the following SQL statement.

```
SELECT c1.course_no,
       c1.description course_descr,
       c1.prerequisite,
       c2.description pre_req_descr
  FROM course c1, course c2
 WHERE c1.prerequisite =
       c2.course_no
 ORDER BY 3, 1
```

The Nonequijoin

Occasionally, you need to construct joins that are not based on equality of values. The following query illustrates such an example, using the BETWEEN operator, where you have values that fall into a range. The result shows a list of grades for student ID 107, including the respective letter grades.


```
SELECT grade_type_code, numeric_grade, letter_grade
  FROM grade g JOIN grade_conversion c
    ON (g.numeric_grade BETWEEN c.min_grade AND c.max_grade)
 WHERE g.student_id = 107
 ORDER BY 1, 2 DESC
   GR NUMERIC_GRADE LE
   -- ------------- --
   FI            76 C
   HM            96 A
   HM            96 A
   ...
   HM            73 C
   MT            91 A-

12 rows selected.
```

The BETWEEN operator checks for each value in the NUMERIC_GRADE column to see if the individual grade is between the values found in the columns MIN_GRADE and MAX_GRADE of the GRADE_CONVERSION table. If a match is found, the corresponding letter grade is returned. For example, the first row of the result shows the value 76 in the NUMERIC_GRADE column for a final examination. The appropriate letter grade for the value 76 is C.

You can also express the query with the traditional join syntax instead.

```
SELECT grade_type_code,
       numeric_grade, letter_grade,
       min_grade, max_grade
  FROM grade g, grade_conversion c
 WHERE g.numeric_grade BETWEEN
       c.min_grade AND
       c.max_grade
   AND g.student_id = 107
ORDER BY 1, 2 DESC
```

LAB 10.2 EXERCISES

a) For SECTION_ID 86, determine which students received a lower grade on their final than on their

midterm. In your result, list the STUDENT_ID and the grade for the midterm and final.

b) What problem does the following query solve?

```
SELECT DISTINCT a.student_id,
       a.first_name, a.salutation
  FROM student a, student b
 WHERE a.salutation <>
       b.salutation
   AND b.first_name =
       a.first_name
   AND a.student_id <>
       b.student_id
ORDER BY a.first_name
```

c) Display the student ID, last name, and street address of students living at the same address and zip code.

d) Write a query that shows the course number, course description, prerequisite, and description of the prerequisite. Include courses without any prerequisites. (This requires a self-join and an outer join.)

Lab 10.2 Exercise Answers

a) For SECTION_ID 86, determine which students received a lower grade on their final than on their midterm. In your result, list the STUDENT_ID and the grade for the midterm and final.

ANSWER: Using a self-join, you can compare the grade for the midterm with the grade for the final and determine whether the final is lower than the midterm grade.

```
SELECT fi.student_id, mt.numeric_grade "Midterm Grade",
       fi.numeric_grade "Final Grade"
  FROM grade fi JOIN grade mt
    ON (fi.section_id = mt.section_id
   AND fi.student_id = mt.student_id)
 WHERE fi.grade_type_code = 'FI'
   AND fi.section_id = 86
   AND mt.grade_type_code = 'MT'
   AND fi.numeric_grade < mt.numeric_grade
STUDENT_ID Midterm Grade Final Grade
---------- ------------- -----------
       102            90          85
       108            91          76
       211            92          77

3 rows selected.
```

Three students have a lower grade in the final than the grade they achieved in the midterm. Using a self-join allows you to easily determine the correct result. Imagine that you are actually

joining to a different table, even though it is really the same table. Visualize one table as the midterm table and the other as the final table, and the formulation of your SQL statement falls into place.

Start with the table representing the final grade for SECTION_ID 86. Then compare the result with the table representing the midterm grade (grade_type_code = 'MT'). Also join STUDENT_ID and SECTION_ID to make sure you match the same individuals and sections. Finally, compare the numeric grades between the midterm and final.

Using the traditional join syntax, you can also write the query as follows.

```
SELECT fi.student_id,
       mt.numeric_grade
       ”Midterm Grade”,
       fi.numeric_grade ”Final
       Grade”
  FROM grade fi, grade mt
 WHERE fi.grade_type_code = 'FI'
   AND fi.section_id = 86
   AND mt.grade_type_code =
       'MT'
   AND fi.section_id =
       mt.section_id
   AND fi.student_id =
       mt.student_id
   AND fi.numeric_grade <
       mt.numeric_grade
```

Alternatively, you can obtain a somewhat similar solution by using the ANY operator and a correlated subquery (see Chapter 8).

```
SELECT student_id, section_id,
       numeric_grade
  FROM grade g
 WHERE grade_type_code = 'FI'
   AND section_id = 86
   AND numeric_grade < ANY
       (SELECT numeric_grade
  FROM grade
 WHERE grade_type_code = 'MT'
   AND g.section_id = section_id
   AND g.student_id = student_id)
```

421

422

b) What problem does the following query solve?

```
SELECT DISTINCT a.student_id,
       a.first_name, a.salutation
  FROM student a, student b
 WHERE a.salutation <>
       b.salutation
   AND a.first_name =
       b.first_name
   AND a.student_id <>
       b.student_id
 ORDER BY a.first_name
```

ANSWER: The query determines the students who might have inconsistent salutations for their respective first names.

This self-join is used to check for errors and inconsistency of data. A number of students have different salutations for the same first name. According to the query result, Kevin is both a female and male name. The same holds true for Daniel, Roger, and some other students as well.

```
STUDENT_ID FIRST_NAME                     SALUT
---------- ------------------------------ -----
       124 Daniel                         Mr.
       242 Daniel                         Mr.
       315 Daniel                         Ms.
...
       272 Kevin                          Ms.
       341 Kevin                          Mr.
       368 Kevin                          Mr.
       238 Roger                          Mr.
       383 Roger                          Ms.

17 rows selected.
```

The query self-joins by the first name and shows only those having a different salutation for the same name. Because there are multiple names for each table alias, this results in a Cartesian product. Eliminate any records where the

student_ids are identical with the condition a.student_id <> b.student_id. Further eliminate duplicate rows by using DISTINCT.

c) Display the student ID, last name, and street address of students living at the same address and zip code.

ANSWER: You can use a self-join to compare the street address and the zip code.

```
SELECT DISTINCT a.student_id, a.last_name,
       a.street_address
  FROM student a, student b
 WHERE a.street_address = b.street_address
   AND a.zip = b.zip
   AND a.student_id <> b.student_id
 ORDER BY a.street_address
```

```
STUDENT_ID LAST_NAME            STREET_ADDRESS
---------- -------------------- --------------------
       390 Greenberg            105-34 65th Ave.  #6B
       392 Saliternan           105-34 65th Ave.  #6B
       234 Brendler             111 Village Hill Dr.
       380 Krot                 111 Village Hill Dr.
...
       217 Citron               PO Box 1091
       182 Delbrun              PO Box 1091

22 rows selected.
```

422

423

The condition a.student_id <> b.student_id eliminates the student itself from the result.

Alternatively, your ANSI join solution may be similar to the following SELECT statement.

```
SELECT DISTINCT a.student_id, a.last_name,
       a.street_address
  FROM student a JOIN student b
    ON (a.street_address = b.street_address
   AND a.zip = b.zip
   AND a.student_id <> b.student_id)
 ORDER BY a.street_address
```

Or, your join and WHERE clause may look like the following, which provides the same result. The ON clause and the WHERE condition all need to be true and are connected by the logical AND.

```
    ON (a.street_address = b.street_address
  AND a.zip = b.zip)
WHERE a.student_id <> b.student_id
ORDER BY a.street_address
```

You can also expand the query to include the city and state information for the particular zip code by joining to a third table, the ZIPCODE table.

As always, you can use many alternatives to achieve the same result; for example, you can also write a subquery like the following.

```
SELECT DISTINCT student_id id, last_name,
       street_address ||' '|| city || ', '
       || state address
  FROM student s, zipcode z
 WHERE s.zip = z.zip
   AND (street_address, s.zip) IN
       (SELECT street_address, zip
       FROM student
       GROUP BY street_address, zip
       HAVING COUNT(*) > 1)
 ORDER BY address
```

d) Write a query that shows the course number, course description, prerequisite, and description of the prerequisite. Include courses without any prerequisites. (This requires a self-join and an outer join.)

ANSWER: The SELECT statement joins the courses and their corresponding prerequisites. It also includes courses that do not have any prerequisites, using an outer join, and displays a

NULL for the prerequisite description column labeled PRE_REQ_DESCR.

```
SELECT c1.course_no,
       SUBSTR(c1.description, 1,15) course_descr,
       C1.prerequisite,
       SUBSTR(c2.description,1,15) pre_req_descr
  FROM course c1 LEFT OUTER JOIN course c2
    ON c1.prerequisite = c2.course_no
 ORDER BY 1
COURSE_NO COURSE_DESCR    PREREQUISITE PRE_REQ_DESCR
--------- --------------- ------------ ---------------
       10 Technology Conc
       20 Intro to Inform
       25 Intro to Progra          140 Systems Analysi
...
      145 Internet Protoc          310 Operating Syste
      146 Java for C/C++
      147 GUI Design Lab            20 Intro to Inform
...
      430 Java Developer           350 Java Developer
      450 DB Programming           350 Java Developer

30 rows selected.
```

Using the traditional syntax, you can write the query as follows.

```
SELECT c1.course_no,
       SUBSTR(c1.description, 1,15) course_descr,
       C1.prerequisite,
       SUBSTR(c2.description,1,15) pre_req_descr
  FROM course c1, course c2
 WHERE c1.prerequisite = c2.course_no(+)
 ORDER BY 1
```


Or, you can write the query with a UNION ALL.

```
SELECT c1.course_no, c1.description course_descr,
       c1.prerequisite, c2.description
       pre_req_descr
  FROM course c1 JOIN course c2
    ON (c1.prerequisite = c2.course_no)
 UNION ALL
SELECT course_no, description, prerequisite, NULL
  FROM course
 WHERE prerequisite IS NULL
```


Lab 10.2 Quiz

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

1) A self-join requires you to always join the foreign key with the primary key in the same table.

_____**a)** True

_____**b)** False

2) Self-joins work only when you have a recursive relationship in your table.

_____**a)** True

_____**b)** False

3) You cannot use subqueries or ORDER BY clauses with self-joins.

_____**a)** True

_____**b)** False

4) You need to use a table alias to be able to write a self-join.

_____**a)** True

_____**b)** False

ANSWERS APPEAR IN APPENDIX A.

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) Write a query that shows all the instructors who live in the same zip code.

2) Are any of the rooms overbooked? Determine whether any sections meet at the same date, time, and location.

3) Determine whether there is any scheduling conflict for instructors: Are any instructors scheduled to teach one or more sections at the same date and time? Order the result by the INSTRUCTOR_ID and the starting date and time of the sections.

4) Show the course number, description, course cost, and section ID for courses that cost 1195 or more. Include courses that have no corresponding section.

5) Write a query that lists the section numbers and students IDs of students enrolled in classes held in location L210. Include sections for which no students are enrolled.