9

OUTER JOINs

"The only difference between a problem and a solution is people understand the solution."
—Charles Franklin Kettering
Inventor, 1876–1958

Topics Covered in This Chapter

What Is an OUTER JOIN?

The LEFT/RIGHT OUTER JOIN

The FULL OUTER JOIN

Uses for OUTER JOINs

Sample Statements

Summary

Problems for You to Solve

In the previous chapter, we covered all the "ins" of JOINs—linking two or more tables or result sets using INNER JOIN to find all the rows that match. Now it's time to talk about the "outs"—linking tables and finding out not only the rows that match but also the rows that don't match.

What Is an OUTER JOIN?

As we explained in the previous chapter, the SQL Standard defines several types of JOIN operations to link two or more tables or result sets. An OUTER JOIN asks your database system to return not only the rows that match on the criteria you specify but also the unmatched rows from either one or both of the two sets you want to link.

Let's suppose, for example, that you want to fetch information from the School Scheduling database about students and the classes for which they're registered. As you learned in the previous chapter, an INNER JOIN returns only students who have registered for a class and classes for which a student has registered. It won't return any students who have been accepted at the school but haven't signed up for any classes yet, nor will it return any classes that are on the schedule but for which no student has yet shown an interest.

What if you want to list *all* students and the classes for which they are registered, if any? Conversely, suppose you want a list of *all* the classes and the students who have registered for those classes, if any. To solve this sort of problem, you need to ask for an OUTER JOIN.

Figure 9–1 uses a set diagram to show one possible relationship between students and classes. As you can see, a few students haven't registered for a class yet, and a few classes do not yet have any students signed up to take the class.

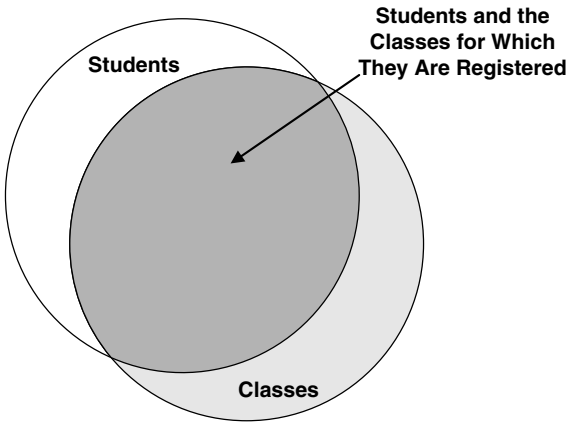

Figure 9–1 *A possible relationship between students and classes*

If you ask for *all* students and the classes for which they are registered, you'll get a result set resembling Figure 9–2.

You might ask, "What will I see for the students who haven't registered for any classes?" If you remember the concept of a Null or "nothing" value discussed in Chapter 5, Getting More Than Simple Columns, you know what you'll see: When you ask for all students joined with any classes, your database system will return a Null value in all columns from the Classes table

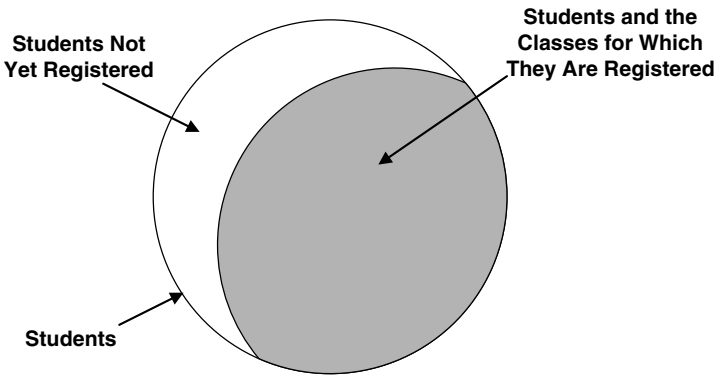

Figure 9–2 *All students and the classes for which they are registered*

when it finds a student who is not yet registered for any classes. If you think about the concept of a difference between two sets (discussed in Chapter 7, Thinking in Sets), the rows with a Null value in the columns from the Classes table represent the difference between the set of all students and the set of students who have registered for a class.

Likewise, if you ask for all classes and any students who registered for classes, the rows with Null values in the columns from the Students table represent the difference between the set of all classes and the set of classes for which students have registered. As we promised, using an OUTER JOIN with a test for Null values is an alternate way to discover the difference between two sets. Unlike a true EXCEPT operation that matches on entire rows from the two sets, you can specify the match in a JOIN operation on just a few specific columns (usually the primary key and the foreign key).

The LEFT/RIGHT OUTER JOIN

You'll generally use the OUTER JOIN form that asks for all the rows from one table or result set and any matching rows from a second table or result set. To do this, you specify either a LEFT OUTER JOIN or a RIGHT OUTER JOIN.

What's the difference between LEFT and RIGHT? Remember from the previous chapter that to specify an INNER JOIN on two tables, you name the first table, include the JOIN keyword, and then name the second table. When you begin building queries using OUTER JOIN, the SQL Standard considers the

first table you name as the one on the "left," and the second table as the one on the "right." So, if you want all the rows from the first table and any matching rows from the second table, you'll use a LEFT OUTER JOIN. Conversely, if you want all the rows from the second table and any matching rows from the first table, you'll specify a RIGHT OUTER JOIN.

Syntax

Let's examine the syntax needed to build either a LEFT or RIGHT OUTER JOIN.

Using Tables

We'll start simply with defining an OUTER JOIN using tables. Figure 9–3 shows the syntax diagram for creating a query with an OUTER JOIN on two tables.

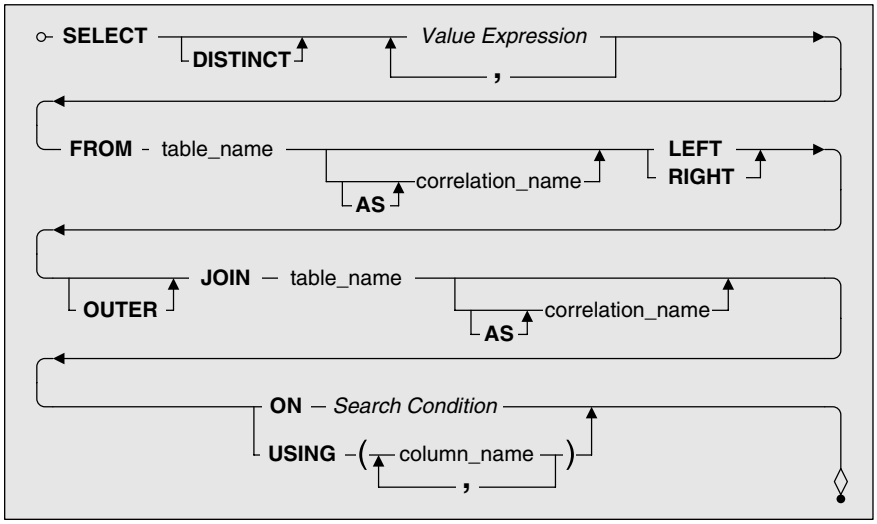

Figure 9–3 *Defining an OUTER JOIN on two tables*

Just like INNER JOIN (covered in Chapter 8), all the action happens in the FROM clause. (We left out the WHERE and ORDER BY clauses for now to simplify things.) Instead of specifying a single table name, you specify two table names and link them with the JOIN keyword. If you do not specify the type of JOIN you want, your database system assumes you want an INNER JOIN. In

this case, because you want an OUTER JOIN, you must explicitly state that you want either a LEFT JOIN or a RIGHT JOIN. The OUTER keyword is optional.

❖ **Note** For those of you following along with the complete syntax diagrams in Appendix A, SQL Standard Diagrams, note that we've pulled together the applicable parts (from *Select Statement, Table Reference,* and *Joined Table*) into simpler diagrams that explain the specific syntax we're discussing.

The critical part of any JOIN is the ON or USING clause that follows the second table and tells your database system how to perform the JOIN. To solve the JOIN, your database system logically combines every row in the first table with every row in the second table. (This combination of all rows from one table with all rows from a second table is called a *Cartesian product*.) It then applies the criteria in the ON or USING clause to find the matching rows to be returned. Because you asked for an OUTER JOIN, your database system also returns the unmatched rows from either the "left" or "right" table.

You learned about using a search condition to form a WHERE clause in Chapter 6, Filtering Your Data. You can use a search condition in the ON clause within a JOIN to specify a logical test that must be true in order to return any two linked rows. It only makes sense to write a search condition that compares at least one column from the first table with at least one column from the second table. Although you can write a very complex search condition, you can usually specify a simple equals comparison test on the primary key columns from one table with the foreign key columns from the other table.

To keep things simple, let's start with the same recipe classes and recipes example we used in the last chapter. Remember that in a well-designed database, you should break out complex classification names into a second table and then link the names back to the primary subject table via a simple key value. In the Recipes sample database, recipe classes appear in a table separate from recipes. Figure 9–4 shows the relationship between the Recipe_Classes and Recipes tables.

When you originally set up the kinds of recipes to save in your database, you might have started by entering all the recipe classes that came to mind. Now that you've entered a number of recipes, you might be interested in finding

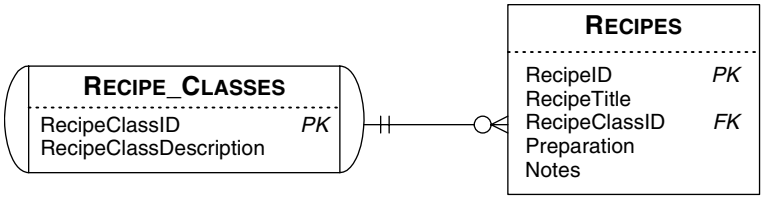

Figure 9–4 *Recipe classes are in a separate table from recipes.*

out which classes don't have any recipes entered yet. You might also be interested in listing *all* the recipe classes along with the names of recipes entered so far for each class. You can solve either problem with an OUTER JOIN.

❖ **Note** Throughout this chapter, we use the "Request/Translation/Clean Up/SQL" technique introduced in Chapter 4, Creating a Simple Query.

"Show me all the recipe types and any matching recipes in my database."

Translation: Select recipe class description and recipe title from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table

Clean Up:
Select recipe class description ~~and~~ recipe title
from ~~the~~ recipe classes ~~table~~
left outer join~~ed with the~~ recipes ~~table~~
on recipe_ classes.recipe class ID ~~in the recipe classes table matching~~ = recipes.recipe class ID ~~in the recipes table~~

SQL:

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

When using multiple tables in your FROM clause, remember to qualify fully each column name with the table name wherever you use it so that it's absolutely clear which column from which table you want. Note that we *had to* qualify the name of RecipeClassID in the ON clause because there are two columns named RecipeClassID—one in the Recipes table and one in the Recipe_Classes table.

❖ **Note** Although most commercial implementations of SQL support OUTER JOIN, some do not. If your database does not support OUTER JOIN, you can still solve the problem by listing all the tables you need in the FROM clause, then moving your search condition from the ON clause to the WHERE clause. You must consult your database documentation to learn the specific nonstandard syntax that your database requires to define the OUTER JOIN. For example, earlier versions of Microsoft SQL Server support this syntax. (Notice the asterisk in the WHERE clause.)

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID *=
   Recipes.RecipeClassID
```

If you're using Oracle, the optional syntax is as follows. (Notice the plus sign in the WHERE clause.)

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID(+)
```

Quite frankly, these strange syntaxes were invented by database vendors that wanted to provide this feature long before a clearer syntax was defined in the SQL Standard. Thankfully, the SQL Standard syntax allows you to fully define the source for the final result set entirely within the FROM clause. Think of the FROM clause as fully defining a linked result set from which the database system obtains your answer. In the SQL Standard, you use the WHERE clause only to filter rows out of the result set defined by the FROM clause. Also, because the specific syntax for defining an OUTER JOIN via the WHERE clause varies by product, you might have to learn several different syntaxes if you work with multiple nonstandard products.

If you execute our example query in the Recipes sample database, you should see 16 rows returned. Because we didn't enter any soup recipes in the database, you'll get a Null value for RecipeTitle in the row where RecipeClassDescription is 'Soup'. To find only this one row, use this approach.

"List the recipe classes that do not yet have any recipes."

Translation: Select recipe class description from the recipe classes table left outer joined with the recipes table on recipe class ID where recipe ID is empty

Clean Up:
Select recipe class description
from ~~the~~ recipe classes ~~table~~
left outer join~~ed with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipes table matches~~
= recipes.recipe class ID ~~in the recipes table~~
where recipe ID is ~~empty~~ NULL

SQL:
```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
WHERE Recipes.RecipeID IS NULL
```

If you think about it, we've just done a difference or EXCEPT operation (see Chapter 7) using a JOIN. It's somewhat like saying, *"Show me all the recipe classes except the ones that already appear in the recipes table."* The set diagram in Figure 9–5 should help you visualize what's going on.

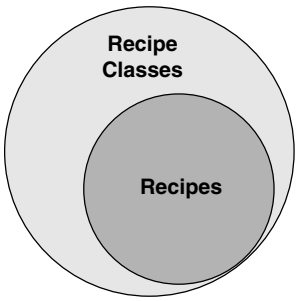

Figure 9–5 *A possible relationship between recipe classes and recipes*

In Figure 9–5, all recipes have a recipe class, but some recipe classes exist for which no recipe has yet been defined. When we add the IS NULL test, we're asking for all the rows in the lighter outer circle that don't have any matches in the set of recipes represented by the darker inner circle.

Notice that the diagram for an OUTER JOIN on tables in Figure 9–3 also has the optional USING clause. If the matching columns in the two tables have the same name and you want to join only on equal values, you can use the USING clause and list the column names. Let's do the previous problem again with USING.

"Display the recipe classes that do not yet have any recipes."

Translation	Select recipe class description from the recipe classes table left outer joined with the recipes table using recipe class ID where recipe ID is empty
Clean Up	Select recipe class description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~ using recipe class ID where recipe ID is ~~empty~~ NULL

SQL	`SELECT Recipe_Classes.RecipeClassDescription`
	`FROM Recipe_Classes`
	`LEFT OUTER JOIN Recipes`
	`USING (RecipeClassID)`
	`WHERE Recipes.RecipeID IS NULL`

The USING syntax is a lot simpler, isn't it? There's one small catch: Any column in the USING clause loses its table identity because the SQL Standard dictates that the database system must "coalesce" the two columns into a single column. In this example, there's only one RecipeClassID column as a result, so you can't reference Recipes.RecipeClassID or Recipe_Classes.RecipeClassID in the SELECT clause or any other clause.

Be aware that some database systems do not yet support USING. If you find that you can't use USING with your database, you can always get the same result with an ON clause and an equals comparison.

❖ **Note** The SQL Standard also defines a type of JOIN operation called a NATURAL JOIN. A NATURAL JOIN links the two specified tables by matching all the columns with the same name. If the only common columns are the linking columns and your database supports NATURAL JOIN, you can solve the example problem like this:

```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
NATURAL LEFT OUTER JOIN Recipes
WHERE Recipes.RecipeID IS NULL
```

Do not specify an ON or USING clause if you use the NATURAL keyword.

Embedding a SELECT Statement

As you recall from Chapter 8, most SQL implementations let you substitute an entire SELECT statement for any table name in your FROM clause. Of course, you must then assign a correlation name (see the section on assigning alias

names in Chapter 8) so that the result of evaluating your embedded query has a name. Figure 9-6 shows how to assemble an OUTER JOIN clause using embedded SELECT statements.

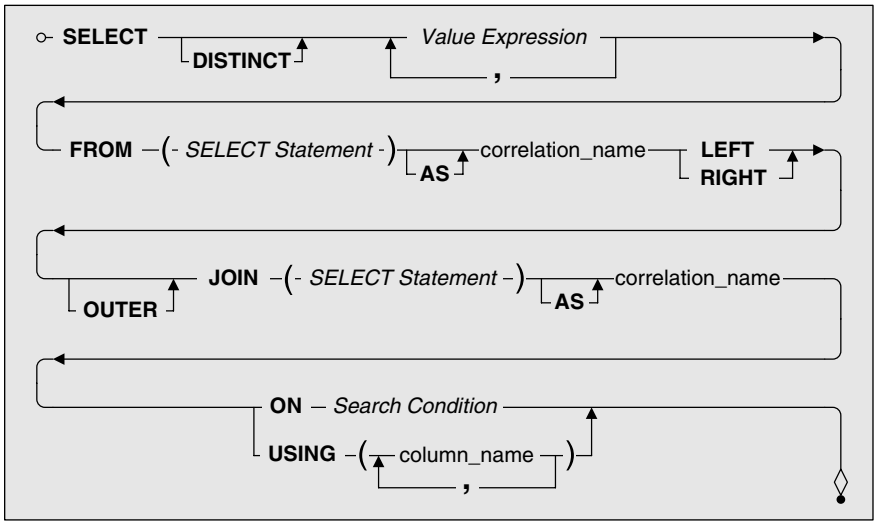

Figure 9–6 *An OUTER JOIN using SELECT statements*

Note that a SELECT statement can include all query clauses *except* an ORDER BY clause. Also, you can mix and match SELECT statements with table names on either side of the OUTER JOIN keywords.

Let's look at the Recipes and Recipe_Classes tables again. For this example, let's also assume that you are interested only in classes Salads, Soups, and Main courses. Here's the query with the Recipe_Classes table filtered in a SELECT statement that participates in a LEFT OUTER JOIN with the Recipes table.

```
SQL         SELECT RCFiltered.ClassName, R.RecipeTitle
            FROM
               (SELECT RecipeClassID,
                   RecipeClassDescription AS ClassName
                FROM Recipe_Classes AS RC
                WHERE RC.ClassName = 'Salads'
                   OR RC.ClassName = 'Soup'
                   OR RC.ClassName = 'Main Course')
              AS RCFiltered
            LEFT OUTER JOIN Recipes AS R
            ON RCFiltered.RecipeClassID = R.RecipeClassID
```

You must be careful when using a SELECT statement in a FROM clause. First, when you decide to substitute a SELECT statement for a table name, you must be sure to include not only the columns you want to appear in the final result but also any linking columns you need to perform the JOIN. That's why you see both RecipeClassID and RecipeClassDescription in the embedded statement. Just for fun, we gave RecipeClassDescription an alias name of ClassName in the embedded statement. As a result, the SELECT clause asks for ClassName rather than RecipeClassDescription. Note that the ON clause now references the correlation name (RCFiltered) of the embedded SELECT statement rather than the original name of the table or the correlation name we assigned the table inside the embedded SELECT statement.

As the query is stated for the actual Recipes sample database, you see one row with RecipeClassDescription of Soup with a Null value returned for RecipeTitle because there are no soup recipes in the sample database. We could just as easily have built a SELECT statement on the Recipes table on the right side of the OUTER JOIN. For example, we could have asked for recipes that contain the word "beef" in their titles, as in the following statement.

```
SQL         SELECT RCFiltered.ClassName, R.RecipeTitle
            FROM
               (SELECT RecipeClassID,
                   RecipeClassDescription AS ClassName
                FROM Recipe_Classes AS RC
                WHERE RC.ClassName = 'Salads'
                   OR RC.ClassName = 'Soup'
                   OR RC.ClassName = 'Main Course')
              AS RCFiltered
            LEFT OUTER JOIN
               (SELECT Recipes.RecipeClassID, Recipes.Recipe
                   Title
                FROM Recipes
                WHERE Recipes.RecipeTitle LIKE '%beef%')
              AS R
            ON RCFiltered.RecipeClassID = R.RecipeClassID
```

Keep in mind that the LEFT OUTER JOIN asks for *all* rows from the result set or table on the left side of the JOIN, regardless of whether any matching rows exist on the right side. The previous query not only returns a Soup row with a Null RecipeTitle (because there are no soups in the database at all) but also a Salad row with a Null. You might conclude that there are no salad recipes in the database. Actually, there *are* salads in the database but no salads with "beef" in the title of the recipe!

❖ **Note** You might have noticed that you can enter a full search condition as part of the ON clause in a JOIN. This is absolutely true, so it is perfectly legal in the SQL Standard to solve the example problem as follows.

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
AND
   (Recipe_Classes.RecipeClassDescription = 'Salads'
OR Recipe_Classes.RecipeClassDescription = 'Soup'
OR Recipe_Classes.RecipeClassDescription =
   'Main Course')
AND Recipes.RecipeTitle LIKE '%beef%'
```

Unfortunately, we have discovered that some major implementations of SQL solve this problem incorrectly or do not accept this syntax at all! Therefore, we recommend that you always enter in the search condition in the ON clause only criteria that compare columns from the two tables or result sets. If you want to filter the rows from the underlying tables, do so with a separate search condition in a WHERE clause in an embedded SELECT statement.

Embedding JOINs within JOINs

Although you can solve many problems by linking just two tables, many times you'll need to link three, four, or more tables to get all the data to solve your request. For example, you might want to fetch all the relevant information about recipes—the type of recipe, the recipe name, and all the ingredients for the recipe—in one query. Now that you understand what you can do with an OUTER JOIN, you might also want to list *all* recipe classes—even those that have no recipes defined yet—and all the details about recipes and their ingredients. Figure 9-7 shows all the tables needed to answer this request.

Looks like you need data from *five* different tables! Just as in Chapter 8, you can do this by constructing a more complex FROM clause, embedding JOIN clauses within JOIN clauses. Here's the trick: Everywhere you can specify a table name, you can also specify an entire JOIN clause surrounded with parentheses. Figure 9-8 shows a simplified version of joining two tables. (We've left off the correlation name clauses and chosen the ON clause to form a simple INNER or OUTER JOIN of two tables.)

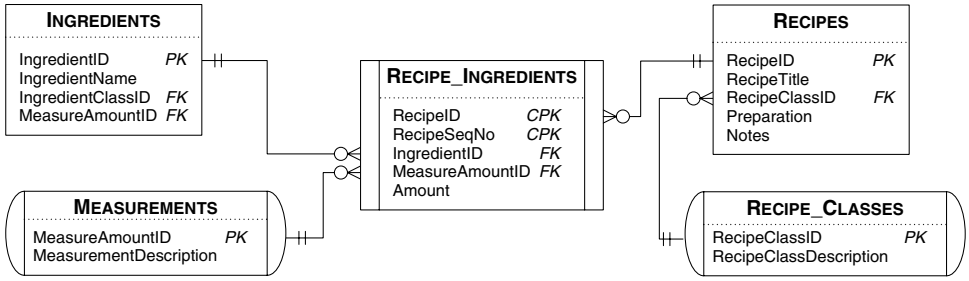

Figure 9–7 *The tables you need from the Recipes sample database to fetch all the information about recipes*

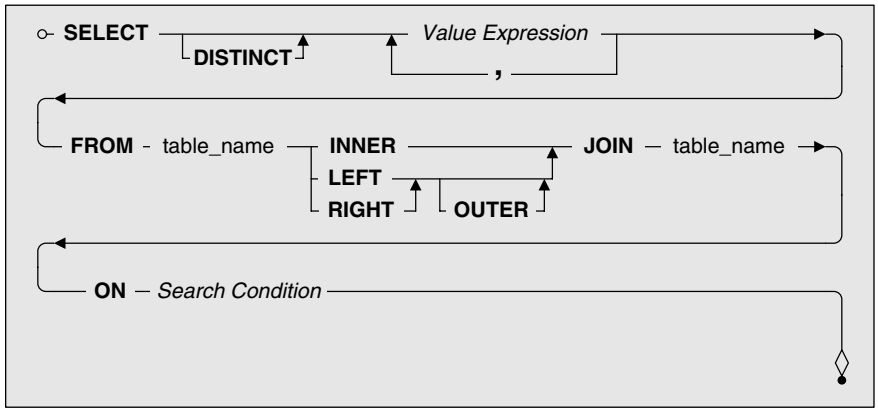

Figure 9–8 *A simple JOIN of two tables*

To add a third table to the mix, just place an open parenthesis before the first table name, add a close parenthesis after the search condition, and then insert another JOIN, a table name, the ON keyword, and another search condition. Figure 9–9 (on page 306) shows how to do this.

If you think about it, the JOIN of two tables inside the parentheses forms a logical table, or inner result set. This result set now takes the place of the first simple table name in Figure 9–8. You can continue this process of enclosing an entire JOIN clause in parentheses and then adding another JOIN keyword, table name, ON keyword, and search condition until you have all the result sets you need. Let's make a request that needs data from all the tables shown in Figure 9–7 and see how it turns out. (You might use this type of request for a report that lists all recipe types with details about the recipes in each type.)

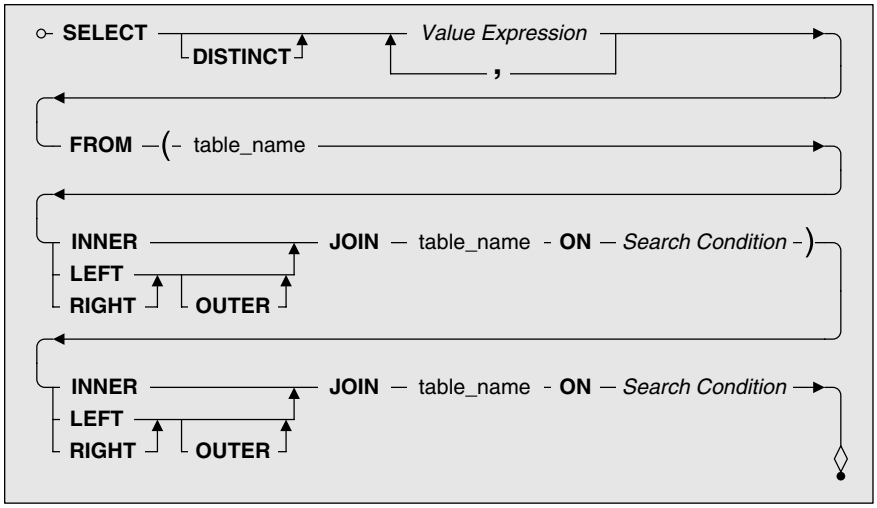

Figure 9–9 *A simple JOIN of three tables*

"I need all the recipe types, and then the matching recipe names, preparation instructions, ingredient names, ingredient step numbers, ingredient quantities, and ingredient measurements from my recipes database, sorted in recipe title and step number sequence."

Translation Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe ID in the recipes table, then joined with the recipe ingredients table on recipe ID in the recipes table matching recipe ID in the recipe ingredients table, then joined with the ingredients table on ingredient ID in the ingredients table matching ingredient ID in the recipe ingredients table, and then finally joined with the measurements table on measurement amount ID in the measurements table matching measurement amount ID in the recipe ingredients table, order by recipe title and recipe sequence number

Clean Up Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~

on recipe_classes .recipe class ID ~~in the recipe classes table matching~~ = recipes .recipe class ID ~~in the recipes table, then~~ inner join~~ed with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table, then~~
inner join~~ed with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matching~~
= recipe_ingredients.ingredient ID
~~in the recipe ingredients table, and then~~
~~finally~~ inner join~~ed with the~~ measurements ~~table~~
on measurements.measurement amount ID ~~in the measurements table matching~~
= recipe_ingredients.measurement amount ID
~~in the recipe ingredients table,~~
order by recipe title, ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

In truth, you can substitute an entire JOIN of two tables anywhere you might otherwise place only a table name. In Figure 9–9, we implied that you must first join the first table with the second table and then join that result with the third table. You could also join the second and third tables first (as long as the third table is, in fact, related to the second table and not the first one) and then perform the final JOIN with the first table. Figure 9–10 (on page 308) shows you this alternate method.

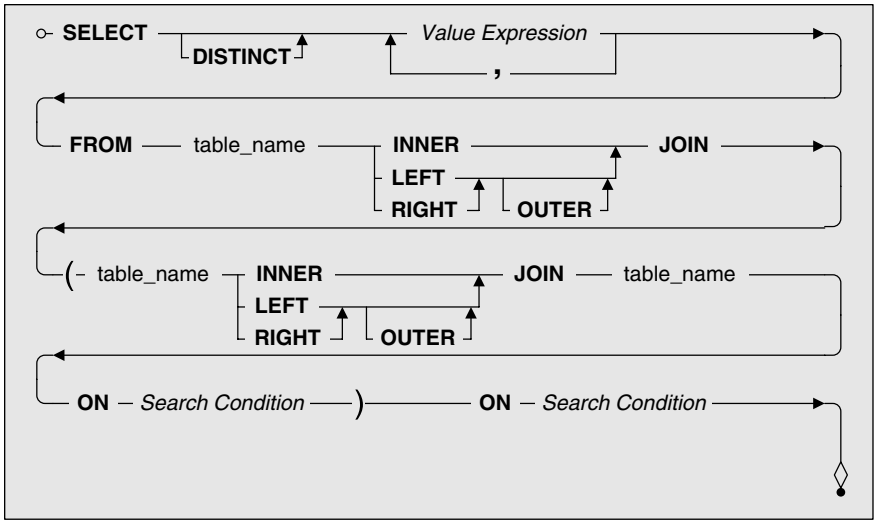

Figure 9–10 *Joining more than two tables in an alternate sequence*

To solve the request we just showed you using five tables, we could have also stated the SQL as follows.

```
SQL         SELECT Recipe_Classes.RecipeClassDescription,
               Recipes.RecipeTitle, Recipes.Preparation,
               Ingredients.IngredientName,
               Recipe_Ingredients.RecipeSeqNo,
               Recipe_Ingredients.Amount,
               Measurements.MeasurementDescription
            FROM Recipe_Classes
            LEFT OUTER JOIN
               (((Recipes
               INNER JOIN Recipe_Ingredients
               ON Recipes.RecipeID = Recipe_Ingredients.RecipeID)
               INNER JOIN Ingredients
               ON Ingredients.IngredientID =
                  Recipe_Ingredients.IngredientID)
               INNER JOIN Measurements
               ON Measurements.MeasureAmountID =
                  Recipe_Ingredients.MeasureAmountID)
            ON Recipe_Classes.RecipeClassID =
               Recipes.RecipeClassID
            ORDER BY RecipeTitle, RecipeSeqNo
```

Remember that the optimizers in some database systems are sensitive to the sequence of the JOIN definitions. If your query with many JOINs is taking a

long time to execute on a large database, it might run faster if you change the sequence of JOINs in your SQL statement.

You might have noticed that we used only one OUTER JOIN in the previous multiple-JOIN examples. You're probably wondering whether it's possible or even makes sense to use more than one OUTER JOIN in a complex JOIN. Let's assume that there are not only some recipe classes that don't have matching recipe rows but also some recipes that don't have any ingredients defined yet. In the previous example, you won't see any rows from the Recipes table that do not have any matching rows in the Recipe_Ingredients table because the INNER JOIN eliminates them. Let's ask for all recipes as well.

"I need all the recipe types, and then all the recipe names and preparation instructions, and then any matching ingredient names, ingredient step numbers, ingredient quantities, and ingredient measurements from my recipes database, sorted in recipe title and step number sequence."

Translation	Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table, then left outer joined with the recipe ingredients table on recipe ID in the recipes table matching recipe ID in the recipe ingredients table, then joined with the ingredients table on ingredient ID in the ingredients table matching ingredient ID in the recipe ingredients table, and then finally joined with the measurements table on measurement amount ID in the measurements table matching measurement amount ID in the recipe ingredients table, order by recipe title and recipe sequence number
Clean Up	Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~ on recipe_classes.recipe class ID ~~in the recipe classes table matching~~ = recipes.recipe class ID ~~in the recipes table, then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~

on recipes.recipe ID ~~in the recipes table matching~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table, then~~
inner join~~ed with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matching~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table, and then finally~~ inner join~~ed with the~~ measurements ~~table~~
on measurement.measurement amount ID
~~in the measurements table matching~~
= recipe_ingredients.measurement amount ID
~~in the recipe ingredients table,~~
order by recipe title ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
LEFT OUTER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

Be careful! This sort of multiple OUTER JOIN works as expected only if you're following a path of one-to-many relationships. Let's look at the relationships between Recipe_Classes, Recipes, and Recipe_Ingredients again, as shown in Figure 9–11.

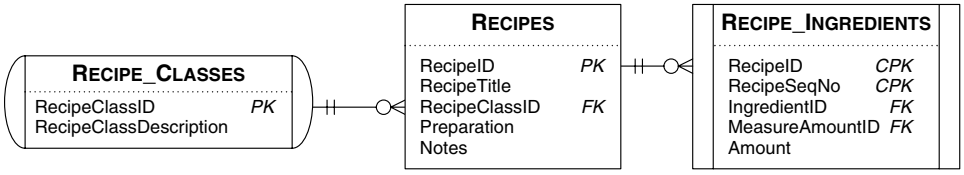

Figure 9–11 *The relationships between the Recipe_Classes, Recipes, and Recipe_Ingredients tables*

You might see a one-to-many relationship sometimes called a *parent-child relationship*. Each parent row (on the "one" side of the relationship) might have zero or more children rows (on the "many" side of the relationship). Unless you have orphaned rows on the "many" side (for example, a row in Recipes that has a Null in its RecipeClassID column), *every* row in the child table should have a matching row in the parent table. So it makes sense to say `Recipe_Classes LEFT JOIN Recipes` to pick up any parent rows in Recipe_Classes that don't have any children yet in Recipes. `Recipe_Classes RIGHT JOIN Recipes` should (barring any orphaned rows) give you the same result as an INNER JOIN.

Likewise, it makes sense to ask for `Recipes LEFT JOIN Recipe_Ingredients` because you might have some recipes for which no ingredients have yet been entered. `Recipes RIGHT JOIN Recipe_Ingredients` doesn't work because the linking column (RecipeID) in Recipe_Ingredients is also part of that table's compound primary key. Therefore, you are guaranteed to have no orphaned rows in Recipe_Ingredients because no column in a primary key can contain a Null value.

Now, let's take it one step further and ask for all ingredients, including those not yet included in any recipes. First, take a close look at the relationships between the tables, including the Ingredients table, as shown in Figure 9–12.

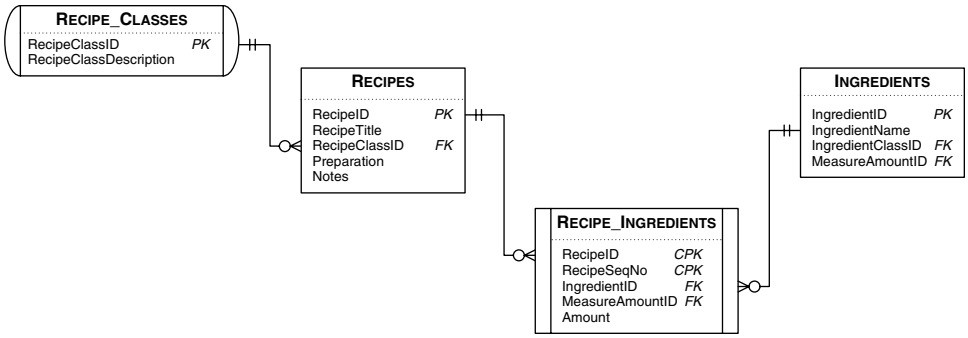

Figure 9–12 *The relationships between the Recipe_Classes, Recipes, Recipe_Ingredients, and Ingredients tables*

Let's try this request. (Caution: There's a trap here!)

> *"I need all the recipe types, and then all the recipe names and preparation instructions, and then any matching ingredient step numbers, ingredient quantities, and ingredient measurements, and finally all ingredient names from my recipes database, sorted in recipe title and step number sequence."*

Translation

Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID in the recipe classes table matches class ID in the recipes table, then left outer joined with the recipe ingredients table on recipe ID in the recipes table matches recipe ID in the recipe ingredients table, then joined with the measurements table on measurement amount ID in the measurements table matches measurement amount ID in the measurements table, and then finally right outer joined with the ingredients table on ingredient ID in the ingredients table matches ingredient ID in the recipe ingredients table, order by recipe title and recipe sequence number

Clean Up

Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~
on recipe_classes .recipe class ID ~~in the recipe classes table matches~~
= recipes .class ID ~~in the recipes table,~~
~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner join~~ed with the~~ measurements ~~table~~
on measurements.measurement amount ID
~~in the measurements table matches~~
= measurements.measurement amount ID
~~in the measurements table,~~
~~and then finally~~ right outer join~~ed with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table,~~
order by recipe title, ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID)
LEFT OUTER JOIN Recipe_Ingredients
ON Recipes.RecipeID =
   Recipe_Ingredients.RecipeID)
INNER JOIN Measurements
ON Measurements.MeasureAmountID =
   Recipe_Ingredients.MeasureAmountID)
```

```
RIGHT OUTER JOIN Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
ORDER BY RecipeTitle, RecipeSeqNo
```

Do you think this will work? Actually, the answer is a resounding NO! Most database systems analyze the entire FROM clause and then try to determine the most efficient way to assemble the table links. Let's assume, however, that the database decides to fully honor how we've grouped the JOINs within parentheses. This means that the database system will work from the innermost JOIN first (Recipe_Classes joined with Recipes) and then work outward.

Because some rows in Recipe_Classes might not have any matching rows in Recipes, this first JOIN returns rows that have a Null value in RecipeClassID. Looking back at Figure 9–12, you can see that there's a one-to-many relationship between Recipe_Classes and Recipes. Unless some recipes exist that haven't been assigned a recipe class, we should get *all* the rows from the Recipes table anyway! The next JOIN with the Recipe_Ingredients table also asks for a LEFT OUTER JOIN. We want all the rows, regardless of any Null values, from the previous JOIN (of Recipe_Classes with Recipes) and any matching rows in Recipe_Ingredients. Again, because some rows in Recipe_Classes might not have matching rows in Recipes or some rows in Recipes might not have matching rows in Recipe_Ingredients, several of the rows might have a Null in the IngredientID column from the Recipe_Ingredients table. What we're doing with both JOINs is "walking down" the one-to-many relationships from Recipe_Classes to Recipes and then from Recipes to Recipe_Ingredients. So far, so good. (By the way, the final INNER JOIN with Measurements is inconsequential—we know that all Ingredients have a valid MeasureAmountID.)

Now the trouble starts. The final RIGHT OUTER JOIN asks for all the rows from Ingredients and *any matching* rows from the result of the previous JOINs. Remember from Chapter 5 that a Null is a very special value—it cannot be equal to any other value, not even another Null. When we ask for *all* the rows in Ingredients, the IngredientID in all these rows has a non-Null value. None of the rows from the previous JOIN that have a Null in IngredientID will match at all, so the final JOIN throws them away! You will see any ingredient that isn't used yet in any recipe, but you won't see recipe classes that have no recipes or recipes that have no ingredients.

If your database system decides to solve the query by performing the JOINs in a different order, you might see recipe classes that have no recipes and recipes that have no ingredients, but you won't see ingredients not yet used in any

recipe because of the Null matching problem. Some database systems might recognize this logic problem and refuse to solve your query at all—you'll see something like an "ambiguous OUTER JOINs" error message. The problem we're now experiencing results from trying to "walk back up" a many-to-one relationship with an OUTER JOIN going in the other direction. Walking down the hill is easy, but walking back up the other side requires special tools. What's the solution to this problem? Read on to the next section to find out!

The FULL OUTER JOIN

A FULL OUTER JOIN is neither "left" nor "right"—it's both! It includes *all* the rows from both of the tables or result sets participating in the JOIN. When no matching rows exist for rows on the "left" side of the JOIN, you see Null values from the result set on the "right." Conversely, when no matching rows exist for rows on the "right" side of the JOIN, you see Null values from the result set on the "left."

Syntax

Now that you've been working with JOINs for a while, the syntax for a FULL OUTER JOIN should be pretty obvious. You can study the syntax diagram for a FULL OUTER JOIN in Figure 9–13.

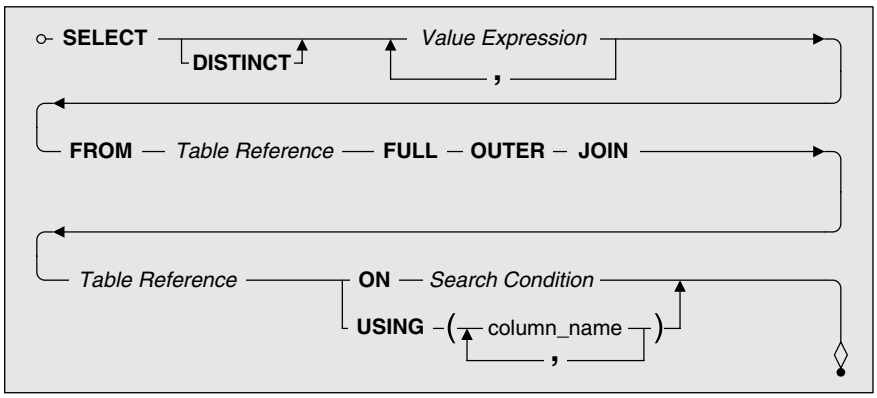

Figure 9–13 *The syntax diagram for a FULL OUTER JOIN*

To simplify things, we're now using the term *table reference* in place of a table name, a SELECT statement, or the result of another JOIN. Let's take another look at the problem we introduced at the end of the previous section. We can now solve it properly using a FULL OUTER JOIN.

"I need all the recipe types, and then all the recipe names and preparation instructions, and then any matching ingredient step numbers, ingredient quantities, and ingredient measurements, and finally all ingredient names from my recipes database, sorted in recipe title and step number sequence."

Translation Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table full outer joined with the recipes table on recipe class ID in the recipe classes table matches recipe class ID in the recipes table, then left outer joined with the recipe ingredients table on recipe ID in the recipes table matches recipe ID in the recipe ingredients table, then joined with the measurements table on measurement amount ID in the measurements table matches measurement amount ID in the recipe ingredients table, and then finally full outer joined with the ingredients table on ingredient ID in the ingredients table matches ingredient ID in the recipe ingredients table, order by recipe title and recipe sequence number

Clean Up Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
full outer join~~ed with the~~ recipes ~~table~~
on recipe_classes .recipe class ID ~~in the recipe classes table matches~~
= recipes .recipe class ID ~~in the recipes table,~~
~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner join~~ed with the~~ measurements ~~table~~
on measurements.measurement amount ID
~~in the measurements table matches~~
= recipe_ingredients.measurement amount ID
~~in the recipe ingredients table,~~
~~and then finally~~ full outer join~~ed with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients .ingredient ID ~~in the recipe ingredients table,~~
order by recipe title ~~and~~ recipe sequence number

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle, Recipes.Preparation,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
```

```
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
FULL OUTER JOIN Recipes
  ON Recipe_Classes.RecipeClassID =
     Recipes.RecipeClassID)
  LEFT OUTER JOIN Recipe_Ingredients
  ON Recipes.RecipeID =
     Recipe_Ingredients.RecipeID)
  INNER JOIN Measurements
  ON Measurements.MeasureAmountID =
     Recipe_Ingredients.MeasureAmountID)
  FULL OUTER JOIN Ingredients
  ON Ingredients.IngredientID =
     Recipe_Ingredients.IngredientID
ORDER BY RecipeTitle, RecipeSeqNo
```

The first and last JOINs now ask for *all* rows from both sides of the JOIN, so the problem with Nulls not matching is solved. You should now see not only recipe classes for which there are no recipes and recipes for which there are no ingredients but also ingredients that haven't been used in a recipe yet. You might get away with using a LEFT OUTER JOIN for the first JOIN, but because you can't predict in advance how your database system decides to nest the JOINs, you should ask for a FULL OUTER JOIN on both ends to ensure the right answer.

❖ **Note** As you might expect, database systems that do not support the SQL Standard syntax for LEFT OUTER JOIN or RIGHT OUTER JOIN also have a special syntax for FULL OUTER JOIN. You must consult your database documentation to learn the specific nonstandard syntax that your database requires to define the OUTER JOIN. For example, earlier versions of Microsoft SQL Server support the following syntax. (Notice the asterisks in the WHERE clause.)

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle
FROM Recipe_Classes, Recipes
WHERE Recipe_Classes.RecipeClassID *=*
   Recipes.RecipeClassID
```

Products that do not support any FULL OUTER JOIN syntax but do support LEFT or RIGHT OUTER JOIN yield an equivalent result by performing a UNION on a LEFT and RIGHT OUTER JOIN. We'll discuss UNION in more detail in the next chapter. Because the specific syntax for defining a FULL OUTER JOIN using the WHERE clause varies by product, you might have to learn several different syntaxes if you work with multiple nonstandard products.

FULL OUTER JOIN on Non-Key Values

Thus far, we have been discussing using OUTER JOINs to link tables or result sets on related key values. You can, however, solve some interesting problems by using an OUTER JOIN on non-key values. For example, the previous chapter showed how to find students and staff who have the same first name in the School Scheduling database. Suppose you're interested in listing *all* staff members and *all* students and showing the ones who have the same first name as well. You can do that with a FULL OUTER JOIN.

"Show me all the students and all the teachers and list together those who have the same first name."

Translation
Select student full name and staff full name from the students table full outer joined with the staff table
on first name in the students table matches
first name in the staff table

Clean Up
Select student full name ~~and~~ staff full name
from ~~the~~ students ~~table~~
full outer join~~ed with the~~ staff ~~table~~
on students.first name ~~in the students table matches~~
= staff.first name ~~in the staff table~~

SQL

```
SELECT (Students.StudFirstName || ' ' ||
   Students.StudLastName) AS StudFullName,
   (Staff.StfFirstName || ' ' ||
   Staff.StfLastName) AS StfFullName
FROM Students
FULL OUTER JOIN Staff
ON Students.StudFirstName =
   Staff.StfFirstName
```

UNION JOIN

No discussion of OUTER JOINs would be complete without at least an honorable mention to UNION JOIN. In the SQL Standard, a UNION JOIN is a FULL OUTER JOIN with the matching rows removed. Figure 9-14 (on page 318) shows the syntax.

As you might expect, not many commercial implementations support a UNION JOIN. Quite frankly, we're hard pressed to think of a good reason why you would want to do a UNION JOIN.

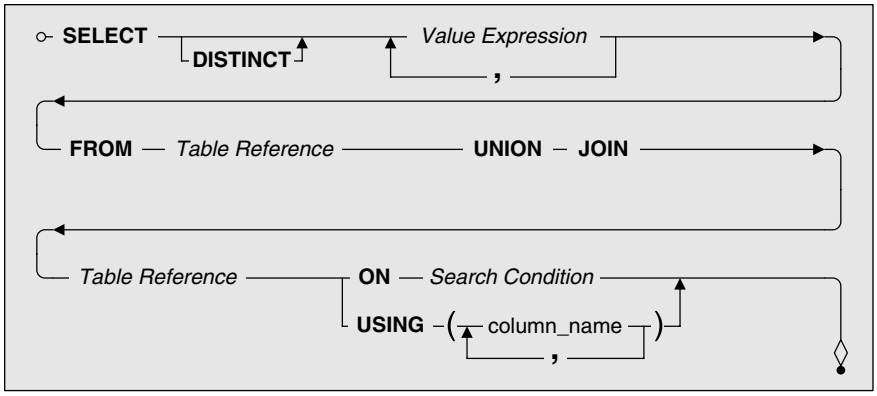

Figure 9–14 *The SQL syntax for a UNION JOIN*

Uses for OUTER JOINs

Because an OUTER JOIN lets you see not only the matched rows but also the unmatched ones, it's great for finding out which, if any, rows in one table do not have a matching related row in another table. It also helps you find rows that have matches on a few rows but not on all. In addition, it's useful for creating input to a report where you want to show all categories (regardless of whether matching rows exist in other tables) or all customers (regardless of whether a customer has placed an order). Following is a small sample of the kinds of requests you can solve with an OUTER JOIN.

Find Missing Values

Sometimes you just want to find what's missing. You do so by using an OUTER JOIN with a test for Null. Here are some "missing value" problems you can solve.

"What products have never been ordered?"
"Show me customers who have never ordered a helmet."
"List entertainers who have never been booked."
"Display agents who haven't booked an entertainer."
"Show me tournaments that haven't been played yet."
"List the faculty members not teaching a class."
"Display students who have never withdrawn from a class."
"Show me classes that have no students enrolled."

"List ingredients not used in any recipe yet."

"Display missing types of recipes."

Find Partially Matched Information

Particularly for reports, it's useful to be able to list all the rows from one or more tables along with any matching rows from related tables. Here's a sample of "partially matched" problems you can solve with an OUTER JOIN.

"List all products and the dates for any orders."

"Display all customers and any orders for bicycles."

"Show me all entertainment styles and the customers who prefer those styles."

"List all entertainers and any engagements they have booked."

"List all bowlers and any games they bowled over 160."

"Display all tournaments and any matches that have been played."

"Show me all subject categories and any classes for all subjects."

"List all students and the classes for which they are currently enrolled."

"Display all faculty and the classes they are scheduled to teach."

"List all recipe types, all recipes, and any ingredients involved."

"Show me all ingredients and any recipes they're used in."

Sample Statements

You now know the mechanics of constructing queries using OUTER JOIN and have seen some of the types of requests you can answer with an OUTER JOIN. Let's look at a fairly robust set of samples, all of which use OUTER JOIN. These examples come from each of the sample databases, and they illustrate the use of the OUTER JOIN to find either missing values or partially matched values.

We've also included sample result sets that would be returned by these operations and placed them immediately after the SQL syntax line. The name that appears immediately above a result set is the name we gave each query in the sample data on the companion CD you'll find bound into the back of the book. We stored each query in the appropriate sample database (as indicated within the example) and prefixed the names of the queries relevant to this chapter with "CH09." You can follow the instructions in the Introduction of this book to load the samples onto your computer and try them.

❖ **Note** Because many of these examples use complex JOINs, the optimizer for your database system might choose a different way to solve these queries. For this reason, the first few rows might not exactly match the result you obtain, but the total number of rows should be the same. To simplify the process, we have combined the Translation and Clean Up steps for all the following examples.

Sales Orders Database

"What products have never been ordered?"

Translation/ Clean Up: Select product number ~~and~~ product name from ~~the~~ products ~~table~~ left outer join~~ed with the~~ order details ~~table~~ on products.product number ~~in the products table matches~~ = order_details .product number ~~in the order details table~~ where ~~the~~ order detail order number is null

SQL:

```
SELECT Products.ProductNumber,
   Products.ProductName
FROM Products LEFT OUTER JOIN Order_Details
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Order_Details.OrderNumber IS NULL
```

CH09_Products_Never_Ordered (2 rows)

ProductNumber	ProductName
4	Victoria Pro All Weather Tires
23	Ultra-Pro Rain Jacket

"Display all customers and any orders for bicycles."

Translation 1: Select customer full name, order date, product name, quantity ordered, and quoted price from the customers table left outer joined with the orders table on customer ID, then joined with the order details table on order number, then joined with the products table on product number, then finally joined with the categories table on category ID where category description is 'Bikes'

Translation 2/ Clean Up: Select customer full name, order date, product name, quantity ordered, ~~and~~ quoted price from ~~the~~ customers ~~table~~ left outer join~~ed with~~

(Select customer ID, order date, product name,
quantity ordered, ~~and~~ quoted price
from ~~the~~ orders ~~table~~
inner join~~ed with the~~ order details ~~table~~
on orders .order number ~~in the orders table matches~~
= order_details .order number ~~in the order details table,~~
~~then~~ join~~ed with the~~ products ~~table~~
on order_details.product number ~~in the order details table matches~~ = products.product number ~~in the products table,~~
~~then finally~~ join~~ed with the~~ categories ~~table~~
on categories.category ID ~~in the categories table matches~~
= products.category ID ~~in the products table~~
where category description ~~is~~ = 'Bikes') as rd
on customers.customer ID in the customers table matches
= rd.customerID ~~in the embedded SELECT statement~~

❖ **Note** Because we're looking for specific orders (bicycles), we split the translation process into two steps to show that the orders need to be filtered before applying an OUTER JOIN.

SQL

```
SELECT Customers.CustFirstName || ' ' ||
   Customers.CustLastName AS CustFullName,
   RD.OrderDate, RD.ProductName,
   RD.QuantityOrdered, RD.QuotedPrice
FROM Customers
LEFT OUTER JOIN
  (SELECT Orders.CustomerID, Orders.OrderDate,
      Products.ProductName,
      Order_Details.QuantityOrdered,
      Order_Details.QuotedPrice
    FROM ((Orders
    INNER JOIN Order_Details
    ON Orders.OrderNumber =
       Order_Details.OrderNumber)
    INNER JOIN Products
    ON Order_Details.ProductNumber =
       Products.ProductNumber)
    INNER JOIN Categories
    ON Categories.CategoryID =
       Products.CategoryID
    WHERE Categories.CategoryDescription =
       'Bikes')
  AS RD
ON Customers.CustomerID = RD.CustomerID
```

❖ **Note** This request is really tricky because you want to list *all* customers OUTER JOINed with only the orders for bikes. If you turn Translation 1 directly into SQL, you won't find any of the customers who have not ordered a bike! An OUTER JOIN from Customers to Orders *will* return all customers and any orders. When you add the filter to select only bike orders, that's all you will get—customers who ordered bikes.

Translation 2 shows you how to do it correctly—create an inner result set that returns only orders for bikes, and then OUTER JOIN that with Customers to get the final answer.

CH09_All_Customers_And_Any_Bike_Orders (913 rows)

CustFullName	OrderDate	ProductName	QuantityOrdered	QuotedPrice
Suzanne Viescas				
William Thompson	2007-12-23	Trek 9000 Mountain Bike	5	$1,164.00
William Thompson	2008-01-15	Trek 9000 Mountain Bike	6	$1,164.00
William Thompson	2007-10-11	Viscount Mountain Bike	2	$635.00
William Thompson	2007-10-05	Viscount Mountain Bike	5	$615.95
William Thompson	2008-01-15	Trek 9000 Mountain Bike	4	$1,200.00
William Thompson	2007-10-11	Trek 9000 Mountain Bike	3	$1,200.00
William Thompson	2008-01-07	Trek 9000 Mountain Bike	2	$1,200.00
<< more rows here >>				

(Looks like William Thompson is a really good customer!)

Entertainment Agency Database

"List entertainers who have never been booked."

Translation/ Clean Up

Select entertainer ID ~~and~~ entertainer stage name
from ~~the~~ entertainers ~~table~~
left outer join~~ed with the~~ engagements ~~table~~
on entertainers.entertainer ID ~~in the entertainers table matches~~
= engagements.entertainer ID ~~in the engagements table~~
where engagement number is null

SQL

```
SELECT Entertainers.EntertainerID,
   Entertainers.EntStageName
FROM Entertainers
LEFT OUTER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID
WHERE Engagements.EngagementNumber IS NULL
```

CH09_Entertainers_Never_Booked
(1 row)

EntertainerID	EntStageName
1009	Katherine Ehrlich

"Show me all musical styles and the customers who prefer those styles."

Translation/ Clean Up

Select style ID, style name, customer ID, customer first name, ~~and~~ customer last name
from ~~the~~ musical styles ~~table~~
left outer join~~ed with~~
(~~the~~ musical preferences ~~table~~ inner join~~ed with the~~ customers ~~table~~
on musical_preferences.customer ID
~~in the musical preferences table matches~~
= customers.customer ID ~~in the customers table~~)
on musical_styles.style ID ~~in the musical styles table matches~~
= musical_preferences.style ID ~~in the musical preferences table~~

SQL

```
SELECT Musical_Styles.StyleID,
   Musical_Styles.StyleName,
   Customers.CustomerID,
   Customers.CustFirstName,
   Customers.CustLastName
FROM Musical_Styles
LEFT OUTER JOIN (Musical_Preferences
   INNER JOIN Customers
   ON Musical_Preferences.CustomerID =
      Customers.CustomerID)
ON Musical_Styles.StyleID =
   Musical_Preferences.StyleID
```

CH09_All_Styles_And_Any_Customers (41 rows)

StyleID	StyleName	CustomerID	CustFirstName	CustLastName
1	40s Ballroom Music	10015	Carol	Viescas
1	40s Ballroom Music	10011	Joyce	Bonnicksen
2	50s Music			
3	60s Music	10002	Deb	Waldal
4	70s Music	10007	Liz	Keyser
5	80s Music	10014	Mark	Rosales
6	Country	10009	Sarah	Thompson
7	Classical	10005	Elizabeth	Hallmark
<< more rows here >>				

(Looks like nobody likes 50s music!)

❖ **Note** We very carefully phrased the FROM clause to influence the database system to first perform the INNER JOIN between Musical_Preferences and Customers, and then OUTER JOINed that with Musical_Styles. If your database tends to process JOINs from left to right, you might have to state the FROM clause with the INNER JOIN first followed by a RIGHT OUTER JOIN to Musical_Styles. In Microsoft Office Access, we had to state the INNER JOIN as an embedded SELECT statement to get it to return the correct answer.

School Scheduling Database

"List the faculty members not teaching a class."

Translation/ Clean Up

Select staff first name ~~and~~ staff last name
from ~~the~~ staff ~~table~~ left outer join~~ed with the~~ faculty classes ~~table~~
on staff.staff ID ~~in the staff table matches~~
= faculty_classes.staff ID ~~in the faculty classes table~~
where class ID is null

SQL

```
SELECT Staff.StfFirstName, Staff.StfLastName,
FROM Staff
LEFT OUTER JOIN Faculty_Classes
ON Staff.StaffID = Faculty_Classes.StaffID
WHERE Faculty_Classes.ClassID IS NULL
```

CH09_Staff_Not_Teaching (4 rows)

StfFirstName	StfLastName
Jeffrey	Smith
Tim	Smith
Kathryn	Patterson
Joe	Rosales III

"Display students who have never withdrawn from a class."

Translation/ Clean Up

Select student full name
from ~~the~~ students ~~table~~ left outer join~~ed with~~
(Select student ID from ~~the~~ student schedules ~~table~~
inner join~~ed with the~~ student class status ~~table~~
on student_class_status.class status
~~in the student class status table matches~~
= student_schedules.class status ~~in the student schedules table~~
where class status description ~~is~~ = 'withdrew') as withdrew
on students.student ID ~~in the students table matches~~
= withdrew.student ID ~~in the embedded SELECT statement~~
where ~~the~~ student_schedules.student ID ~~in the student schedules table~~ is null

SQL

```
SELECT Students.StudLastName || ', ' ||
   Students.StudFirstName AS StudFullName
FROM Students
LEFT OUTER JOIN
   (SELECT Student_Schedules.StudentID
    FROM Student_Class_Status
    INNER JOIN Student_Schedules
    ON Student_Class_Status.ClassStatus =
       Student_Schedules.ClassStatus
    WHERE Student_Class_Status.ClassStatus
       Description = 'withdrew')
  AS Withdrew
ON Students.StudentID = Withdrew.StudentID
WHERE Withdrew.StudentID IS NULL
```

CH09_Students_Never_Withdrawn (15 rows)

StudFullName
Hamilton, David
Stadick, Betsy
Galvin, Janice
Hartwig, Doris
Bishop, Scott
Hallmark, Elizabeth
Sheskey, Sara
Wier, Marianne
<< *more rows here* >>

"Show me all subject categories and any classes for all subjects."

Translation/ Clean Up

Select category description, subject name, classroom ID,
start time, ~~and~~ duration
from ~~the~~ categories ~~table~~
left outer join~~ed with the~~ subjects ~~table~~
on categories.category ID ~~in the categories table matches~~
= subjects.category ID ~~in the subjects table,~~
~~then~~ left outer join~~ed with the~~ classes ~~table~~
on subjects .subject ID ~~in the subjects table matches~~
= classes.subject ID ~~in the classes table~~

SQL

```
SELECT Categories.CategoryDescription,
   Subjects.SubjectName, Classes.ClassroomID,
   Classes.StartTime, Classes.Duration
FROM (Categories
LEFT OUTER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
LEFT OUTER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID
```

❖ **Note** We were very careful again to construct the sequence and nesting of JOINs to be sure we got the answer we expected.

CH09_All_Categories_All_Subjects_Any_Classes (82 rows)

CategoryDescription	SubjectName	ClassroomID	StartTime	Duration
Accounting	Financial Accounting Fundamentals I	3313	9:00	50
Accounting	Financial Accounting Fundamentals I	3313	13:00	50
Accounting	Financial Accounting Fundamentals II	3415	8:00	50
Accounting	Fundamentals of Managerial Accounting	3415	10:00	50
Accounting	Intermediate Accounting	3315	11:00	50
Accounting	Business Tax Accounting	3313	14:00	50
Art	Introduction to Art	1231	10:00	50
Art	Design	1619	15:30	110
<< more rows here >>				

Further down in the result set, you'll find no classes scheduled for Developing a Feasibility Plan, Computer Programming, and American Government. You'll also find no subjects scheduled for categories Psychology, French, or German.

Bowling League Database

"Show me tournaments that haven't been played yet."

Translation/ Clean Up
Select tourney ID, tourney date, ~~and~~ tourney location
from ~~the~~ tournaments ~~table~~
left outer join~~ed with the~~ tourney matches ~~table~~
on tournaments.tourney ID ~~in the tournaments table matches~~
= tourney_matches.tourney ID ~~in the tourney matches table~~
where match ID is null

SQL

```
SELECT Tournaments.TourneyID,
   Tournaments.TourneyDate,
   Tournaments.TourneyLocation
FROM Tournaments
LEFT OUTER JOIN Tourney_Matches
ON Tournaments.TourneyID =
   Tourney_Matches.TourneyID
WHERE Tourney_Matches.MatchID IS NULL
```

CH09_Tourney_Not_Yet_Played (6 rows)

TourneyID	TourneyDate	TourneyLocation
15	2008-07-11	Red Rooster Lanes
16	2008-07-18	Thunderbird Lanes
17	2008-07-25	Bolero Lanes
18	2008-08-01	Sports World Lanes
19	2008-08-08	Imperial Lanes
20	2008-08-15	Totem Lanes

"List all bowlers and any games they bowled over 180."

Translation 1
Select bowler name, tourney date, tourney location, match ID, and raw score from the bowlers table left outer joined with the bowler scores table on bowler ID, then inner joined with the tourney matches table on match ID, then finally inner joined with the tournaments table on tournament ID where raw score in the bowler scores table is greater than 180

Can you see why the above translation won't work? You need a filter on one of the tables that is on the right side of the left join, so you need to put the filter in an embedded SELECT statement. Let's restate the Translation step, clean it up, and solve the problem.

Translation 2/ Clean Up

Select bowler name, tourney date, tourney location,
match ID, ~~and~~ raw score
from ~~the~~ bowlers ~~table~~ left outer join~~ed with~~
(Select tourney date, tourney location, match ID,
bowler ID, ~~and~~ raw score
from ~~the~~ bowler scores ~~table~~
inner join~~ed with the~~ tourney matches ~~table~~
on bowler_scores .match ID ~~in the bowler scores table matches~~
= tourney_ matches.match ID ~~in the tourney matches table,~~
~~then~~ inner join~~ed with the~~ tournaments ~~table~~
on tournaments.tournament ID ~~in the tournaments table matches~~
= tourney_ matches.tournament ID ~~in the tourney matches table~~
where raw score ~~is greater than~~ > 180) as ti
on bowlers.bowler ID ~~in the bowlers table matches~~
= ti.bowler ID ~~in the embedded SELECT statement~~

SQL

```
SELECT Bowlers.BowlerLastName || ', ' ||
   Bowlers.BowlerFirstName AS BowlerName,
   TI.TourneyDate, TI.TourneyLocation,
   TI.MatchID, TI.RawScore
FROM Bowlers
LEFT OUTER JOIN
   (SELECT Tournaments.TourneyDate,
       Tournaments.TourneyLocation,
       Bowler_Scores.MatchID,
       Bowler_Scores.BowlerID,
       Bowler_Scores.RawScore
    FROM (Bowler_Scores
    INNER JOIN Tourney_Matches
    ON Bowler_Scores.MatchID =
       Tourney_Matches.MatchID)
    INNER JOIN Tournaments
    ON Tournaments.TourneyID =
       Tourney_Matches.TourneyID
    WHERE Bowler_Scores.RawScore > 180)
  AS TI
ON Bowlers.BowlerID = TI.BowlerID
```

CH09_All_Bowlers_And_Scores_Over_180 (106 rows)

BowlerName	TourneyDate	TourneyLocation	MatchID	RawScore
Black, Alastair				
Cunningham, David				
Ehrlich, Zachary				
Fournier, Barbara				
Fournier, David				
Hallmark, Alaina				
Hallmark, Bailey				
Hallmark, Elizabeth				
Hallmark, Gary				
Hernandez, Kendra				
Hernandez, Michael				
Kennedy, Angel	2007-11-20	Sports World Lanes	46	185
Kennedy, Angel	2007-10-09	Totem Lanes	22	182
<< more rows here >>				

❖ **Note** You guessed it! This is another example where you must build the filtered INNER JOIN result set first and then OUTER JOIN that with the table from which you want "all" rows.

Recipes Database

"List ingredients not used in any recipe yet."

Translation/ Clean Up

Select ingredient name from ~~the~~ ingredients ~~table~~
left outer join~~ed with the~~ recipe ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ingredients.ingredient ID ~~in the recipe ingredients table~~
where recipe ID is null

SQL

```
SELECT Ingredients.IngredientName
FROM Ingredients
LEFT OUTER JOIN Recipe_Ingredients
ON Ingredients.IngredientID =
   Recipe_Ingredients.IngredientID
WHERE Recipe_Ingredients.RecipeID IS NULL
```

CH09_Ingredients_ Not_Used (20 rows)

IngredientName
Halibut
Chicken, Fryer
Bacon
Iceberg Lettuce
Butterhead Lettuce
Scallop
Vinegar
Red Wine
<< *more rows here* >>

"I need all the recipe types, and then all the recipe names, and then any matching ingredient step numbers, ingredient quantities, and ingredient measurements, and finally all ingredient names from my recipes database."

Translation/ Clean Up

Select ~~the~~ recipe class description, recipe title,
ingredient name, recipe sequence number,
amount, ~~and~~ measurement description
from ~~the~~ recipe classes ~~table~~
full outer join~~ed with the~~ recipes ~~table~~
on recipe_classes.recipe class ID ~~in the recipe classes table matches~~
= recipes.recipe class ID ~~in the recipes table,~~
~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~
on recipes.recipe ID ~~in the recipes table matches~~
= recipe_ingredients.recipe ID ~~in the recipe ingredients table,~~
~~then~~ inner join~~ed with the~~ measurements ~~table~~
on measurements.measurement amount ID
~~in the measurements table matches~~
= recipe_ ingredients.measurement amount ID
~~in the recipe ingredients table,~~
~~and then finally~~ full outer join~~ed with the~~ ingredients ~~table~~
on ingredients.ingredient ID ~~in the ingredients table matches~~
= recipe_ ingredients.ingredient ID ~~in the recipe ingredients table,~~

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
   Recipes.RecipeTitle,
   Ingredients.IngredientName,
   Recipe_Ingredients.RecipeSeqNo,
   Recipe_Ingredients.Amount,
   Measurements.MeasurementDescription
FROM (((Recipe_Classes
FULL OUTER JOIN Recipes
  ON Recipe_Classes.RecipeClassID =
     Recipes.RecipeClassID)
  LEFT OUTER JOIN Recipe_Ingredients
  ON Recipes.RecipeID =
     Recipe_Ingredients.RecipeID)
  INNER JOIN Measurements
  ON Measurements.MeasureAmountID =
     Recipe_Ingredients.MeasureAmountID)
  FULL OUTER JOIN Ingredients
  ON Ingredients.IngredientID =
     Recipe_Ingredients.IngredientID
ON Recipe_Classes.RecipeClassID =
   Recipes.RecipeClassID
```

❖ **Note** This sample is a request you saw us solve in the section on FULL OUTER JOIN. We decided to include it here so that you can see the actual result. You won't find this query saved using this syntax in the Microsoft Access or MySQL version of the sample database because neither product supports a FULL OUTER JOIN. Instead, you can find this problem solved with a UNION of two OUTER JOIN queries that achieves the same result. You'll learn about using UNION in the next chapter. The result shown here is what you'll see when you run the query in Microsoft SQL Server.

CH09_All_Recipe_Classes_All_Recipes (109 rows)

RecipeClass Description	RecipeTitle	Ingredient Name	RecipeSeq No	Amount	Measurement Description
Starch	Yorkshire Pudding	Flour	1	1.5	Cup
Starch	Yorkshire Pudding	Water	2	1	Cup
Starch	Yorkshire Pudding	Eggs	3	2	Piece
Starch	Yorkshire Pudding	Salt	4	0.5	Teaspoon
Starch	Yorkshire Pudding	Milk	5	0.5	Cup
Starch	Yorkshire Pudding	Beef drippings	6	4	Teaspoon
Dessert	Trifle	Sponge Cake	1	1	Package
Dessert	Trifle	Raspberry Jello	2	1	Package
Dessert	Trifle	Bird's Custard Powder	3	1	Package
Dessert	Trifle	Raspberry Jam	4	1	Jar
<< more rows here >>					

SUMMARY

In this chapter, we led you through the world of OUTER JOINs. We began by defining an OUTER JOIN and comparing it to the INNER JOIN you learned about in Chapter 8.

We next explained how to construct a LEFT or RIGHT OUTER JOIN, beginning with simple examples using two tables, and then progressing to embedding SELECT statements and constructing statements using multiple JOINs. We showed how an OUTER JOIN combined with a Null test is equivalent to the difference (EXCEPT) operation we covered in Chapter 7. We also discussed some of the difficulties you might encounter when constructing statements using multiple OUTER JOINs. We closed the discussion of the LEFT and RIGHT OUTER JOIN with a problem requiring multiple OUTER JOINs that can't be solved with only LEFT or RIGHT.

In our discussion of FULL OUTER JOIN, we showed how you might need to use this type of JOIN in combination with other INNER and OUTER JOINs to get the correct answer. We also briefly explained a variant of the FULL OUTER JOIN—the UNION JOIN.

We explained how OUTER JOINs are useful and listed a variety of requests that you can solve using OUTER JOINs. The rest of the chapter showed nearly a dozen examples of how to use OUTER JOIN. We provided several examples for each of the sample databases and showed you the logic behind constructing the solution statement for each request.

The following section presents a number of requests that you can work out on your own.

Problems for You to Solve

Below, we show you the request statement and the name of the solution query in the sample databases. If you want some practice, you can work out the SQL you need for each request and then check your answer with the query we saved in the samples. Don't worry if your syntax doesn't exactly match the syntax of the queries we saved—as long as your result set is the same.

Sales Orders Database

1. *"Show me customers who have never ordered a helmet."*
   (Hint: This is another request where you must first build an INNER JOIN to find all orders containing helmets and then do an OUTER JOIN with Customers.)
   You can find the solution in CH09_Customers_No_Helmets (2 rows).
2. *"Display customers who have no sales rep (employees) in the same ZIP Code."*
   You can find the solution in CH09_Customers_No_Rep_Same_Zip (18 rows).
3. *"List all products and the dates for any orders."*
   You can find the solution in CH09_All_Products_Any_Order_Dates (2,682 rows).

Entertainment Agency Database

1. *"Display agents who haven't booked an entertainer."*
   You can find the solution in Agents_No_Contracts (1 row).
2. *"List customers with no bookings."*
   You can find the solution in CH09_Customers_No_Bookings (2 rows).
3. *"List all entertainers and any engagements they have booked."*
   You can find the solution in CH09_All_Entertainers_And_Any_Engagements (112 rows).

School Scheduling Database

1. *"Show me classes that have no students enrolled."*
   (Hint: You need only "enrolled" rows from Student_Classes, not "completed" or "withdrew.")
   You can find the solution in CH09_Classes_No_Students_Enrolled (63 rows).
2. *"Display subjects with no faculty assigned."*
   You can find the solution in CH09_Subjects_No_Faculty (1 row).
3. *"List students not currently enrolled in any classes."*
   (Hint: You need to find which students have an "enrolled" class status in student schedules and then find the students who are not in this set.)
   You can find the solution in CH09_Students_Not_Currently_Enrolled (2 rows).
4. *"Display all faculty and the classes they are scheduled to teach."*
   You can find the solution in CH09_All_Faculty_And_Any_Classes (79 rows).

Bowling League Database

1. *"Display matches with no game data."*
   You can find the solution in CH09_Matches_Not_Played_Yet (1 row).
2. *"Display all tournaments and any matches that have been played."*
   You can find the solution in CH09_All_Tourneys_Match_Results (174 rows).

Recipes Database

1. *"Display missing types of recipes."*
   You can find the solution in CH09_Recipe_Classes_No_Recipes (1 row).
2. *"Show me all ingredients and any recipes they're used in."*
   You can find the solution in CH09_All_Ingredients_Any_Recipes (108 rows).
3. *"List the salad, soup, and main course categories and any recipes."*
   You can find the solution in CH09_Salad_Soup_Main_Courses (9 rows).
4. *"Display all recipe classes and any recipes."*
   You can find the solution in CH09_All_RecipesClasses_And_Matching_Recipes (16 rows).