Now that you know how to write simple queries using one table and how to use functions and expressions in queries, it is time to learn how to write queries involving two or more tables. In a properly designed relational database, a table contains data about one thing or entity. For example, an order-entry application will have a table storing customer information, a table containing data about orders, and a table containing detail information about each item ordered. The order table has a column, called a foreign key, that points to a row in the customer table. The detail table has a foreign key column that points to the order table. By using joins, you can link these tables together so that you can display columns from each table in the same result set.

You can also use multiple tables with subqueries and union queries. You might use a subquery in place of an IN list in the WHERE clause, for example. A union query allows you to combine the result of two or more queries into one result set. For example, a database may contain archive tables with old sales data. By using a UNION query, you can combine the data from both the production tables and the archived tables so that it looks like the results are from the same table.

Finally, this chapter demonstrates two useful techniques: derived tables and common table expressions. These techniques allow you to isolate the logic used to query one table from the rest of the main query.

Learning how to join tables is a critical skill for T-SQL developers because it allows you to combine the relational data stored in multiple tables and present it as a single result set. Make sure you understand all the example code and complete the exercises in this chapter before moving on to the next chapter.

Writing INNER JOINS

Most of the time, to join tables together, you will use INNER JOIN. When connecting two tables with INNER JOIN, only the rows from the tables that match on the joining columns will show up in the results. If you join the customer and order tables, the query will return only the customers who have placed orders, along with the orders that have been placed. Only the rows where the customer ID is common in both tables will show up in the results.

Joining Two Tables

To join tables together, you might think that another clause will be added to the SELECT statement. This is not the case. Instead, the FROM clause contains information about how the tables join together. Here is the syntax for joining two tables (the keyword INNER is optional):
SELECT <select list>  
FROM <table1>  
[INNER] JOIN <table2> ON <table1>.<col1> = <table2>.<col2>  

Figure 4-1 shows how the Sales.SalesOrderHeader and Sales.SalesOrderDetail tables connect and shows some of the columns in the tables. You will see these tables joined in the first example query, so make sure you understand how they connect before typing Listing 4-1.

![Figure 4-1. The Sales.SalesOrderHeader and Sales.SalesOrderDetail tables](image)

The Sales.SalesOrderHeader table has a primary key called SalesOrderID. The Sales.SalesOrderDetail table has a composite primary key, one that is made up of more than one column, consisting of SalesOrderDetailID and SalesOrderID. The SalesOrderID column in the Sales.SalesOrderDetail table is also a foreign key pointing back to the Sales.SalesOrderHeader table. The arrow points from the foreign key in the Sales.SalesOrderDetail table to the primary key in the Sales.SalesOrderHeader table.

Take a look at the code in Listing 4-1. Type in and execute the code to learn how to join the two tables.

Listing 4-1. Joining Two Tables

USE AdventureWorks2012;  
GO  
SELECT s.SalesOrderID, s.OrderDate, s.TotalDue, d.SalesOrderDetailID,  
  d.ProductID, d.OrderQty  
FROM Sales.SalesOrderHeader AS s  
INNER JOIN Sales.SalesOrderDetail AS d ON s.SalesOrderID = d.SalesOrderID;  

Figure 4-2 displays the results. The SELECT list may contain columns from either of the tables. In the FROM clause, you list one of the tables followed by the words INNER JOIN and the second table name. To define how the two tables join together, use the keyword ON and an equality expression. Each Sales.OrderHeader row contains a unique SalesOrderID. Each Sales.SalesOrderDetail row contains a SalesOrderID column that determines to which order the detail belongs. When you join these two tables together, the query displays every row from the Sales.SalesOrderHeader table that matches a row in the Sales.SalesOrderDetail table.