Chapter 2
How to use Management Studio

Before you start the exercises…

Before you start these exercises, you need to install SQL Server and SQL Server Management Studio. The procedures for doing both of these tasks are provided in appendix A of *Murach's SQL Server 2022 for Developers*.

In addition, your instructor will provide you with scripts that contain starting code for many of these exercises. By default, these scripts will be in a directory named MC Exercise Starts. If your instructor has changed the name of the directory or makes the scripts available individually, adjust the following instructions accordingly.

Exercises

In these exercises, you'll use SQL Server Management Studio to create the MurachCollege database, review the tables in the MurachCollege database, and enter SQL statements and run them against this database.

Create the database

1. Start SQL Server Management Studio and open a connection using either Windows or SQL Server authentication.
2. Open the script file named CreateMurachCollege.sql that's in the MC Exercise Starts directory by clicking the Open File button in the toolbar and then using the resulting dialog box to locate and open the file.
3. Execute the entire script by clicking the Execute button in the SQL Editor toolbar or by pressing F5. When you do, the Messages tab indicates whether the script executed successfully.

Review the database

4. In the Object Explorer window, expand the node for the database named MurachCollege so you can see all of the database objects it contains. If it isn't displayed in the Object Explorer window, you may need to right-click on the name of the connection and select Refresh to display it.
5. View the data for the Courses and Instructors tables.
6. Navigate through the database objects and view the column definitions for the Courses and Instructors tables.

Enter and run some test SQL statements

7. Open a new Query Editor window by clicking the New Query button in the toolbar. Then, select the MurachCollege database from the Available Databases dropdown menu (Ctrl + U) to choose it as the default database.
8. Enter and run this SQL statement:

```
SELECT CourseDescription FROM Courses
```

9. Delete the *s* at the end of Courses and run the statement again. This shows you what an error caused by a typo looks like.
10. Open another Query Editor window and then enter and run this statement:

```
SELECT COUNT(*) AS NumberOfInstructors
FROM Instructors
```

Open and run scripts

11. Open the script named InstructorDetails.sql that's in the MC Exercise Starts directory. Note that this script contains one SQL statement. Then, run the statement.
12. Open the script named InstructorSummary.sql that's in the MC Exercise Starts directory. Note that this opens another Query Editor window. Then, run the script.
13. Open the script named InstructorStatements.sql that's in the MC Exercise Starts directory. Note that this script contains two SQL statements.
14. Press the F5 key or click the Execute button to run both of the statements in this script. Note that this displays the results in two Results tabs. Make sure to view the results of both SELECT statements.
15. Exit SQL Server Management Studio.

Chapter 3
How to retrieve data from a single table

Exercises

Enter and run your own SELECT statements

In these exercises, you'll enter and run your own SELECT statements.

1. Write a SELECT statement that returns all of the columns from the Courses table. Then, run this statement to make sure it works correctly.
2. Write a SELECT statement that returns three columns from the Courses table: CourseNumber, CourseDescription, and CourseUnits. Then, run this statement to make sure it works correctly.

   Add an ORDER BY clause to this statement that sorts the result set by CourseNumber in ascending sequence. Then, run this statement again to make sure it works correctly. This is a good way to build and test a statement, one clause at a time.
3. Write a SELECT statement that returns one column from the Students table named FullName that joins the LastName and FirstName columns.

   Format this column with the last name, a comma, a space, and the first name like this:

   ```
   Doe, John
   ```

   Sort the result set by last name in ascending sequence.

   Return only the students whose last name begins with a letter from A to M.
4. Write a SELECT statement that returns these column names and data from the Instructors table:

LastName	The LastName column
FirstName	The FirstName column
AnnualSalary	The AnnualSalary column

   Return only the rows with an annual salary that's greater than or equal to 60,000.

   Sort the result set in descending sequence by the AnnualSalary column.
5. Write a SELECT statement that returns these column names and data from the Instructors table:

LastName	The LastName column
FirstName	The FirstName column
HireDate	The HireDate column

   Return only the rows with a hire date that's in 2022.

   Sort the result set in ascending sequence by the HireDate column.

6. Write a SELECT statement that returns these column names and data from the Students table:

FirstName	The FirstName column
LastName	The LastName column
EnrollmentDate	The EnrollmentDate column
CurrentDate	The current date
MonthsAttended	A column that's calculated by getting the difference between the enrollment date and the current date

   To get the value of the months attended, use the DATEDIFF function with the month argument.

   Sort the result set in ascending sequence by the MonthsAttended column.

7. Write a SELECT statement that returns these column names and data from the Instructors table:

FirstName	The FirstName column
LastName	The LastName column
AnnualSalary	The AnnualSalary column

   Return only the top 20 percent of instructors based on annual salary.

8. Write a SELECT statement that returns these column names and data from the Students table:

LastName	The LastName column
FirstName	The FirstName column

   Return only the rows where the LastName column starts with the letter 'G'. To do that, use the LIKE phrase.

   Sort the result set by last name in ascending sequence.

9. Write a SELECT statement that returns these column names and data from the Students table:

LastName	The LastName column
FirstName	The FirstName column
EnrollmentDate	The EnrollmentDate column
GraduationDate	The GraduationDate column

   Return only the rows where the EnrollmentDate column is greater than 12-01-2022 and the GraduationDate column contains a null value.

10. Write a SELECT statement that returns these columns and data from the Tuition table, along with a constant value and two calculated values:

FullTimeCost	The FullTimeCost column
PerUnitCost	The PerUnitCost column
Units	12
TotalPerUnitCost	A column that's calculated by multiplying the per-unit cost by the units
TotalTuition	A column that's calculated by adding the full-time cost to the total per unit cost

Chapter 4

How to retrieve data from two or more tables

Exercises

1. Write a SELECT statement that joins the Courses table to the Departments table and returns these columns: DepartmentName, CourseNumber, CourseDescription.

   Sort the result set by DepartmentName and then by CourseNumber in ascending order.

2. Write a SELECT statement that joins the Instructors table to the Courses table and returns these columns: LastName, FirstName, CourseNumber, CourseDescription.

   Return all courses for each instructor with a status of "P" (part time).

   Sort the result set by LastName and then by FirstName in ascending order.

3. Write a SELECT statement that joins the Departments, Courses, and Instructors tables. This statement should return these columns: DepartmentName, CourseDescription, FirstName, and LastName.

   Use aliases for the tables, and return only those courses with three units.

   Sort the result set by DepartmentName and then by CourseDescription in ascending sequence.

4. Write a SELECT statement that joins the Departments, Courses, StudentCourses, and Students tables. This statement should return these columns: DepartmentName, CourseDescription, LastName, and FirstName.

   Return all courses in the English department.

   Sort the result set by CourseDescription in ascending sequence.

5. Write a SELECT statement that joins the Instructors and Courses tables and returns these columns: LastName, FirstName, and CourseDescription.

   Return at least one row for each instructor, even if that instructor isn't teaching any courses.

   Sort the result set by LastName and then by FirstName.

6. Use the UNION operator to generate a result set consisting of five columns from the Students table:

Status	A calculated column that contains a value of UNDERGRAD or GRADUATED
FirstName	The FirstName column
LastName	The LastName column
EnrollmentDate	The EnrollmentDate column
GraduationDate	The GraduationDate column

   If the student doesn't have a value in the GraduationDate column, the Status column should contain a value of UNDERGRAD. Otherwise, it should contain a value of GRADUATED.

   Sort the final result set by EnrollmentDate.

7. Write a SELECT statement that returns these two columns:

DepartmentName	The DepartmentName column from the Departments table
CourseID	The CourseID column from the Courses table

   Return all departments with no courses. (Hint: Use an outer join and only return rows where the CourseID column contains a null value.)

8. Write a SELECT statement that returns these columns:

InstructorDept	The DepartmentName column from the Departments table for an instructor
LastName	The LastName column from the Instructors table
FirstName	The FirstName column from the Instructors table
CourseDescription	The CourseDescription column from the Courses table
CourseDept	The DepartmentName column from the Departments table for a course

   Return one row for each course that's in a different department than the department of the instructor assigned to teach that course. (Hint: You will need to join the Departments table to both the Instructors table and the Courses table, which will require you to use table aliases to distinguish the two tables.)

Chapter 5
How to code summary queries

Exercises

1. Write a SELECT statement that returns these columns:

   The count of the number of instructors in the Instructors table

   The average of the AnnualSalary column in the Instructors table

   Include only those rows where the Status column is equal to "F" (Fulltime).

2. Write a SELECT statement that returns one row for each department that has instructors. The statement should return these columns:

   The DepartmentName column from the Departments table

   The count of the instructors in that department

   The annual salary of the highest paid instructor in that department

   Sort the result set so the department with the most instructors appears first.

3. Write a SELECT statement that returns one row for each instructor that has courses. The statement should return these columns:

   The instructor's first and last names from the Instructors table in this format: John Doe (Note: If the instructor first name has a null value, the concatenation of the first and last name will result in a null value.)

   A count of the number of courses for each instructor

   The sum of the course units for each instructor

   Sort the result set in descending sequence by the total course units for each instructor.

   (Hint: You will need to concatenate the instructor first and last names again in the GROUP BY clause.)

4. Write a SELECT statement that returns one row for each course that has students enrolled. The statement should return these columns:

   The DepartmentName column from the Departments table

   The CourseDescription from the Courses table

   A count of the number of students in the course

   Sort the result set by DepartmentName, then by the enrollment for each course.

5. Write a SELECT statement that returns one row for each student that has courses. The statement should return these columns:

   The StudentID column from the Students table

   The sum of the course units for each student

   Sort the result set in descending sequence by the total course units for each student.

6. Modify the solution to exercise 5 so it only includes students who haven't graduated and who are taking more than nine units.
7. Write a SELECT statement that answers this question: What is the total number of courses taught by **part-time** instructors only? Return these columns:

   The instructor's last name and first name from the Instructors table in this format: Doe, John (Note: If the instructor first name has a null value, the concatenation of the first and last name will result in a null value.)

   The total number of courses taught by each instructor

   Use the ROLLUP operator to include a row that gives the grand total.

Chapter 6
How to code subqueries

Exercises

1. Write a SELECT statement that returns the same result set as this SELECT statement, but don't use a join. Instead, use a subquery in a WHERE clause that uses the IN keyword.

```
SELECT DISTINCT LastName, FirstName
FROM Instructors i
  JOIN Courses c
    ON i.InstructorID = c.InstructorID
ORDER BY LastName, FirstName
```

2. Write a SELECT statement that answers this question: Which instructors have an annual salary that's greater than the average annual salary for all instructors?

   Return the LastName, FirstName, and AnnualSalary columns for each Instructor.

   Sort the result set by the AnnualSalary column in descending sequence.

3. Write a SELECT statement that returns the LastName and FirstName columns from the Instructors table.

   Return one row for each instructor who doesn't have any courses in the Courses table. To do that, use a subquery with the NOT EXISTS operator.

   Sort the result set by LastName and then by FirstName.

4. Write a SELECT statement that returns the LastName and FirstName columns from the Students table, along with a count of the number of courses each student is taking from the StudentCourses table.

   Return one row for each student who is taking more than one class. To do that, use a subquery with the IN operator that groups the student course by StudentID.

   Group and sort the result set by the LastName and then by the FirstName.

5. Write a SELECT statement that returns the LastName, FirstName, and AnnualSalary columns of each instructor who has a unique annual salary. In other words, don't include instructors who have the same annual salary as another instructor.

   Sort the results by LastName and then by FirstName.

6. Write a SELECT statement that returns one row for each course. It should return these columns:

   The CourseID column from the Courses table

   The most recent enrollment date for that course

   Change the SELECT statement to a CTE. Then, write a SELECT statement that returns one row per course that shows the CourseDescription for the course and the LastName, FirstName, and EnrollmentDate for the student with the most recent enrollment data.

7. Write a SELECT statement that returns one row for each student who has courses. It should return with these columns:

   The StudentID column from the Students table

   The sum of the course units for that student

   Include only those students who are taking more than 9 units (fulltime).

   Change the SELECT statement to a CTE. Then, write a SELECT statement that uses this CTE to return the student ID, sum of their course units, and their tuition. (The tuition is equal to the FullTimeCost column, plus the PerUnitCost column multiplied by the number of units.)

   (Hint: You can use a cross join to add the columns from the Tuition table to the query. This works because there's only one row in the Tuition table.)

Chapter 7
How to insert, update, and delete data

Exercises

1. Write an INSERT statement that adds a row to the Departments table for a department named "History".

   (Hint: Remember that you do not need to specify a value for identity columns.)

2. Write a single INSERT statement that adds two rows to the Instructors table with the following values:

InstructorID:	The next automatically generated ID
LastName:	Benedict
FirstName:	Susan
Status:	P
DepartmentChairman:	0
HireDate:	The current date
AnnualSalary:	34000.00
DepartmentID:	9

InstructorID:	The next automatically generated ID
LastName:	Adams
FirstName:	null
Status:	F
DepartmentChairman:	1
HireDate:	The current date
AnnualSalary:	66000.00
DepartmentID:	9

   The statement should not use a column list.

3. Write an UPDATE statement that changes the AnnualSalary column for Susan Benedict (the first instructor you added in exercise 2) from 34,000 to 35,0000. To identify the row, use the InstructorID column.

4. Write a DELETE statement that deletes Adams (the second instructor you added in exercise 2). To identify the row, use the InstructorID column.

5. Write a DELETE statement that deletes the row in the Departments table that has an ID of 9. Then, run the statement. It will produce an error since the department has related rows in the Instructors table. To fix that, precede the DELETE statement with another DELETE statement that deletes all instructors in this department from the Instructors table.

6. Write an UPDATE statement that increases the annual salary for all instructors in the Education department by 5%. (Hint: join the Departments and Instructors tables and then filter the rows by the department name.)

7. Write a DELETE statement that deletes instructors who aren't teaching any courses. To do that, use a subquery in the WHERE clause.

8. Open the script named CreateGradStudents.sql that's in the MC Exercise Starts directory. Run this file to create a table named GradStudents. This table has the same

columns as the Students table, but the StudentID column isn't defined as an identity column.

Make sure that when you run the script, the active database is set to MurachCollege.

9. Write an INSERT statement that inserts rows from the Students table into the GradStudents table. Include only the rows for students who have graduated, and don't use a column list.
10. Open the script named CreateMurachCollege.sql that's in the MC Exercise Starts directory. Then, run this script. That should restore the data that's in the database. If an error message is displayed indicating that the database is in use, you'll need to close and restart Management Studio and then run the script again.

Chapter 8
How to work with data types

Exercises

1. Write a SELECT statement that returns these columns from the Instructors table:

   The monthly salary (the AnnualSalary column divided by 12)

   A column that uses the CAST function to return the monthly salary with exactly 1 digit to the right of the decimal point

   A column that uses the CONVERT function to return the monthly salary as an integer

   A column that uses the CAST function to return the monthly salary as an integer

2. Write a SELECT statement that returns these columns from the Students table:

   The EnrollmentDate column

   A column that uses the CAST function to return the EnrollmentDate column with its date only (year, month, and day)

   A column that uses the CAST function to return the EnrollmentDate column with its time only (hour, minutes, seconds, and milliseconds)

   A column that uses the CONVERT function to return the EnrollmentDate column with just the month and day

3. Write a SELECT statement that returns these columns from the Students table:

   The EnrollmentDate column

   A column that uses the CONVERT function to return the EnrollmentDate column in this format: MM/DD/YY. In other words, use two digits for the month, day, and year, and separate each date component with slashes.

   A column that uses the CONVERT function to return the EnrollmentDate column in the Month DD, YYYY format with the hours and minutes on a 12-hour clock with an am/pm indicator.

   A column that uses the CONVERT function to return the EnrollmentDate column with just the time using a 24-hour format.

   A column that uses the CONVERT function to return the EnrollmentDate column with just the month and day.

Chapter 9
How to use functions

Exercises

1. Write a SELECT statement that returns these columns from the Instructors table:

   The AnnualSalary column

   A column named MonthlySalary that is the result of dividing the AnnualSalary column by 12

   A column named MonthlySalaryRounded that calculates the monthly salary and then uses the ROUND function to round the result to 2 decimal places

2. Write a SELECT statement that returns these columns from the Students table:

   The EnrollmentDate column

   A column that returns the four-digit year that's stored in the EnrollmentDate column

   A column that returns the day of the month that's stored in the EnrollmentDate column

   A column that returns the result from adding four years to the EnrollmentDate column. Use the CAST function so only the year is returned

3. Write a SELECT statement that returns these columns for each course:

   The DepartmentName column from the Departments table

   The CourseNumber column from the Courses table

   The FirstName column from the Instructors table

   The LastName column from the Instructors table

   A column that includes the first three characters from the DepartmentName column in uppercase, concatenated with the CourseNumber column, then either the first character of the FirstName column or an empty string if the column's value is null, and finally the LastName column. (Hint: you will need to cast the CourseNumber column to a character data type.)

4. Write a SELECT statement that returns these columns from the Students table:

   The FirstName column

   The LastName column

   The EnrollmentDate column

   The GraduationDate column

   A column that shows the number of months between the EnrollmentDate and GraduationDate columns

   Return one row for each student who has graduated.

5. Write a CTE with a SELECT statement that returns one row for each student who has at least one course. This query should return these columns:

   The StudentID column from the Students table

   The sum of the course units for that student

   Write a SELECT statement that uses this CTE to return these columns for each student:

   The StudentID column from the CTE

   The number of course units from the CTE

   Whether the student is fulltime or parttime (Hint: To determine whether a student is fulltime, use the IIF function to test if the sum of course units is greater than 9.)

   The student's total tuition (Hint: To calculate the tuition, use the IIF function to determine whether a student is fulltime or parttime. Then, multiply the sum of course units by the PerUnitCost column in the Tuition table and add that to either the FullTimeCost or PartTimeCost column in the Tuition table. To do that, use a cross join to join the CTE and the Tuition tables. This makes the columns from the Tuition table available to the SELECT statement.)

Chapter 10
How to design a database

Exercises

1. Create a database diagram that shows the relationships between the six tables in the MurachCollege database. (Hint: The Tuition table is not related to the other five tables.)
2. Design a database diagram for a database that stores information about the books downloaded by users from a book website.

   Each user must have an email address, first name, and last name.

   Each user can have one or more download.

   Each download must have a filename and download date/time.

   Each book can be related to one or more downloads.

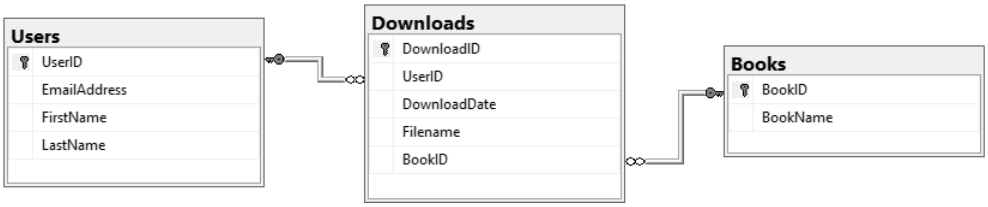
   Each book must have a name.

Chapter 11

How to create and maintain a database with SQL

Exercises

1. Write a script that creates a database named MyBookDB that implements this design:



   Include statements to drop the database if it already exists.

   In the Downloads table, the UserID and BookID columns are foreign keys.

   Define the BookName column so its value is unique.

   Include indexes for the two foreign keys.

   Choose appropriate data types for each column in the tables. (Hint: Use the int, varchar, and datetime2 data types.)

2. Write a script that adds rows to the database that you created in exercise 1.

   Add two rows to the Users table and two rows to the Books table. You can choose any names you like, or choose names that return the same results as in the example table below.

   Add three rows to the Downloads table: one row for user 1 and Book 1; one for user 2 and Book 1; and one for user 2 and book 2. Use the GETDATE function to insert the current date and time into the DownloadDate column. You can again choose any names you like for the files or choose the names from the example below.

   Write a SELECT statement that joins the three tables and retrieves the data from these tables like this:

	EmailAddress	FirstName	LastName	DownloadDate	Filename	BookName
1	johnsmith@gmail.com	John	Smith	2023-02-08 14:09:33.4866667	sq19_allfiles.exe	Murach's SQL Server 2019
2	janedoe@yahoo.com	Jane	Doe	2023-02-08 14:09:33.4866667	python_allfiles.zip	Murach's Python Programming
3	janedoe@yahoo.com	Jane	Doe	2023-02-08 14:09:33.4866667	sq19_ch03.pdf	Murach's SQL Server 2019

   Sort the results by email address in descending order and book name in ascending order.

3. Write an ALTER TABLE statement that adds the following new columns to the Books table created in exercise 1:

   Add one column for book price that provides for up to three digits to the left of the decimal point and two to the right. This column should have a default value of 59.50.

   Add one column for the date and time that the book was added to the database.

4. Write an ALTER TABLE statement that modifies the Users table created in exercise 1 so the EmailAddress column can store a maximum of 25 characters.

   In the same script, code an UPDATE statement that attempts to insert an email address that's longer than 25 characters. It should fail due to the length of the column.

5. Write an ALTER TABLE statement that modifies the Users table created in exercise 1 so the EmailAddress column must be unique.

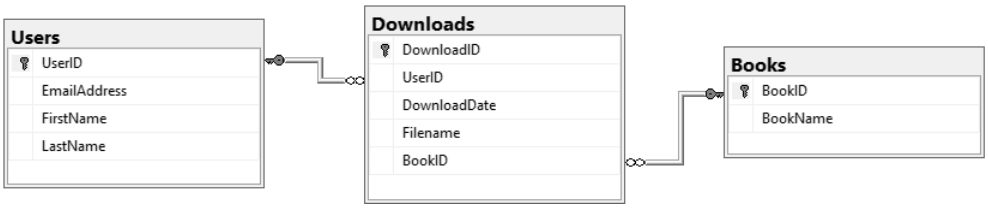
   In the same script, code an UPDATE statement that attempts to insert a non-unique value into this column. It should fail due to the unique constraint.

Chapter 12

How to create and maintain a database with Management Studio

Exercises

1. Use Management Studio to create a new database named MyBookDB using the default settings. (If the database already exists, use the Management Studio to delete it. Then, recreate it.)
2. Use the Management Studio to create the following tables and relationships.



3. Define the UserID column in the Users table, the BookID column in the Books table, and the DownloadID column in the Downloads table as primary keys and identity columns.
4. Choose appropriate data types for each column in the tables. (Hint: Use the int, varchar, and datetime2 data types.)
5. In the Downloads table, set the UserID and BookID columns as foreign keys.
6. Define the columns so none of them allow null values.
7. Use Management Studio to create indexes for the foreign keys in the Downloads table.
8. Use Management Studio to create a unique index on the EmailAddress and BookName columns and a regular index on the DownloadDate column.

Chapter 13
How to work with views

Exercises

1. Create a view named DepartmentInstructors that returns these columns: the DepartmentName column from the Departments table and the LastName, FirstName, Status, and AnnualSalary columns from the Instructors table.
2. Write a SELECT statement that returns all the columns from the DepartmentInstructors view that you created in exercise 1.

   Return one row for each fulltime instructor in the English department.
3. Write an UPDATE statement that updates the DepartmentInstructors view you created in exercise 1 so it increases the annual salary for each fulltime instructor in the English department by 10%. Then, run the SELECT statement you wrote in exercise 2 to be sure this worked correctly.
4. Create a view named StudentCoursesMin that returns these columns: FirstName and LastName from the Students table and CourseNumber, CourseDescription, and CourseUnits from the Courses table. (Hint: use the StudentCourses table to join the Students and Courses tables.)
5. Write a SELECT statement that returns these columns from the StudentCoursesMin view that you created in exercise 4: LastName, FirstName, CourseDescription.

   Return one row for each course with exactly three units.

   Sort the results by LastName and then by FirstName.
6. Create a view named StudentCoursesSummary that uses the view you created in exercise 4. This view should include these columns: LastName, FirstName, CourseCount (the number of courses the student is taking), and UnitsTotal (the total units for the student).

   Limit the view to only fulltime students (students with more than 9 course units).
7. Write a SELECT statement that uses the view that you created in exercise 6 to get the LastName, FirstName and total units for the five students with the most units.

   Sort the results by total units in descending order.

Chapter 14
How to code scripts

Exercises

1. Write a script that declares a variable and sets it to the count of all students in the Students table that haven't graduated. If the count is greater than or equal to 100, the script should display a message that says, "The number of undergrad students is greater than or equal to 100". Otherwise, it should say, "The number of undergrad students is less than 100".

2. Write a script that uses two variables to store (1) the count of all of the instructors in the Instructors table and (2) the average annual salary for those instructors. If the instructor count is greater than or equal to 10, the script should print a message that displays the values of both variables. Otherwise, the script should print a message that says, "The number of fulltime instructors is less than 10".

3. Write a script that creates a temporary table that includes all of the columns from the Instructors table. Include only parttime instructors who were hired in 2020.

   Loop through the temporary table as long as the average annual salary is less than $40,000, Within the loop, increase the salary of each instructor whose salary is less than $41,000 by $50.

   Write a SELECT statement that returns the FirstName, LastName, and AnnualSalary columns from the temporary table.

4. Write a script that attempts to delete the department with the name "Sociology" from the Departments table. If the deletion is successful, the script should display this message:

   ```
   SUCCESS: Record was deleted.
   ```

   If the deletion is unsuccessful, the script should use the ERROR_NUMBER and ERROR_MESSAGE functions to display a message similar to this one:

   ```
   FAILURE: Record was not deleted.

   Error 547: The DELETE statement conflicted with the REFERENCE
   constraint "FK__Instructo__Depar__267ABA7A". The conflict
   occurred in database "MurachCollege", table "dbo.Instructors",
   column 'DepartmentID'.
   ```

5. Write a script that determines if too few students (less than five) or too many students (more than 10) are enrolled in each course. To do that, use a cursor. This cursor should use a SELECT statement that gets the CourseID from the StudentCourses table and a calculated column with the count of students for each course.

   When you loop through the rows in the cursor, the script should display a message like this if there are too few students enrolled in a course:

   ```
   Too few students enrolled in course x
   ```

   where *x* is the course ID. The script should display a similar message if there are too many students enrolled in a course.

Chapter 15

How to code stored procedures, functions, and triggers

Exercises

1. Write a script that creates a stored procedure named spInsertDepartment. The procedure should accept one parameter for a department name, which it uses to add a new row to the Departments table.

   In the same script, code two EXEC statements to test the new procedure. One should succeed and one should fail. (Hint: Note that the Departments table doesn't allow duplicate department names.)

2. Write a script that creates a function named fnStudentUnits that calculates the total course units for a student in the StudentCourses table. The function should accept one parameter for the student ID, and it should return the sum of the course units for the student.

   In the same script, code a SELECT statement that returns the StudentID column from the StudentCourses table, the CourseNumber and CourseUnits columns from the Courses table, and the value returned by the fnStudentUnits function for that student.

3. Write a script that creates a function named fnTuition that calculates the total tuition for a student. To do that, this function should accept one parameter for the student ID. It should use the fnStudentUnits function that you created in exercise 2 and return the value of the tuition for that student depending on whether the student is fulltime (more than nine units) or parttime (nine or fewer units). (Hint: Use a cross join to work with data in the Students and Tuition tables.)

   Then, code a SELECT statement that calls the function. This statement should return the StudentID column from the Students table, the value returned by the fnStudentUnits function for that student, and the value returned by the fnTuition function for that student. Return only rows for students taking one or more units.

4. Write a script that creates a stored procedure named spInsertInstructor that inserts a row into the Instructors table. This stored procedure should accept a parameter for each of these columns: LastName, FirstName, Status, DepartmentChairman, AnnualSalary, and DepartmentID.

   This stored procedure should set the DateAdded column to the current date.

   If the value for the AnnualSalary column is a negative number, the stored procedure should raise an error that indicates that this column doesn't accept negative numbers.

   Code two EXEC statements that test this procedure, one that succeeds and one that fails.

5. Write a script that creates and calls a stored procedure named spUpdateInstructor that updates the AnnualSalary column in the Instructors table. This procedure should have two parameters, one for the instructor ID and another for the annual salary.

   If the value for the AnnualSalary column is a negative number, the stored procedure should raise an error that indicates that the value for this column must be a positive number.

   Code two EXEC statements that test this procedure, one that succeeds and one that fails

6. Create a trigger named Instructors_UPDATE that checks the new value for the AnnualSalary column of the Instructors table. This trigger should raise an error with an appropriate message if the annual salary is greater than 120,000 or less than 0.

   If the new annual salary is between 0 and 12,000, this trigger should modify the new annual salary by multiplying it by 12. That way, a monthly salary of 5,000 becomes an annual salary of 60,000.

   Test this trigger with an UPDATE statement that sets an instructor's salary to an amount less than 12,000.

7. Create a trigger named Instructors_INSERT that inserts the current date for the HireDate column of the Instructors table if the value for that column is null.

   Test this trigger with an INSERT statement that does not include a hire date.

8. Create a table named InstructorsAudit. This table should have all columns of the Instructors table. In addition, it should have an AuditID column for its primary key and a DATETIME2 column named DateUpdated.

   Create a trigger named Instructors_UPDATE. This trigger should insert the old data about the instructor into the InstructorsAudit table after the row is updated and set the DateUpdated column to the current date and time.

   Test this trigger with an UPDATE statement.

9. Re-run the script named CreateMurachCollege.sql to restore the database to its original state.

Chapter 16

How to manage transactions and locking

Exercises

1. Write a script that includes two SQL statements coded as a transaction to delete the row with a student ID of 10 from the Students table. To do this, you must first delete all courses for that student from the StudentCourses table.

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

2. Write a script that includes these statements coded as a transaction:

```
INSERT Students
VALUES ('Smith', 'John', GETDATE(), NULL);

SET @StudentID = @@IDENTITY;

INSERT StudentCourses
VALUES (@StudentID, @CourseID);
```

   Here, the @@IDENTITY variable is used to get the student ID value that's automatically generated when the first INSERT statement inserts a student.

   If these statements execute successfully, commit the changes. Otherwise, roll back the changes.

Chapter 17
How to manage database security

Exercises

1. Write a script that (1) creates a login ID named "JimRhodes" with the password "HelloJimR!"; (2) sets the default database for that login to the MurachCollege database, and (3) creates a user named "JimRhodes" for the new login.
2. Write a script that creates a user-defined database role named StudentEnrollment in the MurachCollege database. Give the new role INSERT, UPDATE, and DELETE permissions for the Students, StudentCourses, and Courses tables. Then, use a fixed database role to give the new StudentEnrollment role SELECT permission for all user tables. Finally, assign the user you created in exercise 1 to the new role.
3. Write a script that uses dynamic SQL and a cursor to loop through each row of the Instructors table. For each instructor who is a department chairman:

   Create a login ID that consists of the instructor's first and last name with no space between

   Set a temporary password of "TempPa$$1" for the new login ID

   Set the default database for the login ID to MurachCollege

   Create a user for the login ID with the same name as the login ID

   Assign the user to the StudentEnrollment role you created in exercise 2

   (Hint: Begin the script by declaring two variables, @DynamicSQL and @LoginName.)
4. Using Management Studio, create a login ID named "AnneBoehm" with the password "ABoe99999", and set the default database to the MurachCollege database. Then, grant the login ID access to the MurachCollege database, create a user for the login ID named "AnneBoehm", and assign the user to the StudentEnrollment role you created in exercise 2.

   Note: If you get an error that says "The MUST_CHANGE option is not supported", you can deselect the "Enforce password policy" option for the login ID.
5. Write a script that removes the user-defined database role named StudentEnrollment. (Hint: This script should begin by removing all users from this role.)
6. Write a script that does the following:

   Creates a schema named Admin

   Transfers the table named Tuition from the dbo schema to the Admin schema

   Assigns the Admin schema as the default schema for the user named JimRhodes that you created in exercise 1

   Grants all standard privileges except for REFERENCES and ALTER to JimRhodes for the Admin schema

Chapter 18
How to use Azure Data Studio

Exercises

In these exercises, you'll use Azure Data Studio to review the tables in the MurachCollege database and to enter SQL statements and run them against this database.

Review the database

1. Start Azure Data Studio.
2. In the Connections tab, expand the node for the database named MurachCollege so you can see all of the database objects it contains. If it isn't displayed in the Connections tab, you may need to click on the Refresh button to display it.
3. View the data for the Courses and Instructors tables.
4. Navigate through the database objects and view the column definitions for at least the Courses and Instructors tables.

Enter and run SQL statements

5. Open a new Query Editor window by right-clicking on the database and selecting New Query.
6. Enter and run this SQL statement:

```
SELECT CourseDescription FROM Courses
```

7. Delete the *s* at the end of Courses and run the statement again. Note the error number and the description of the error.
8. Open another Query Editor window and then enter and run this statement:

```
SELECT COUNT(*) AS NumberOfInstructors
FROM Instructors
```

Open and run scripts

9. Open the script named InstructorDetails.sql that's in the MC Exercise Starts directory. Note that this script contains just one SQL statement. Then, run the statement.
10. Open the script named InstructorSummary.sql that's in the MC Exercise Starts directory. Note that this opens another Query Editor window.
11. Open the script named InstructorStatements.sql that's in the MC Exercise Starts directory. Notice that this script contains two SQL statements that end with semicolons.
12. Press the F5 key or click the Run button to run both of the statements in this script. Note that this displays the results in two Results tabs. Make sure to view the results of both SELECT statements.
13. Exit from Azure Data Studio.