The Labs are from the Book

To Do The Labs: Follow the progression of explanatory material in the book. When you reach a code section (example or demonstration) copy the section from here into the query editor window.

(NOTE: Sometimes you may have Create Table and INSERT statements to run before the query.)

You could run your code now. However, a far more effective technique would be to now type the query again directly under the copied query (what you are typing is what is directly in front of you on the screen.) Then, run what you typed. This helps a lot with retaining the skill / knowledge!

Practice, Practice, and More Practice

The only way to really learn T-SQL is by doing it.

You will not learn T-SQL without practicing it yourself.

The more you practice and experiment, the faster you will learn.

Complete these labs:

Type and Execute each query in SSMS.

These Labs are taken directly from the text, Beginning T-SQL 2012.

The labs are presented here for your convenience, but they are the same as found throughout the current reading chapter in the book.

You will be best served if you follow along in the current chapter and complete the examples provided along with working to understand the explanation of how it works.

This technique (Learning By Example) has been found to be highly effective when used as described in this guide.

As much as possible, type each query.

Manually typing in each query will help you to develop a solid sense of the syntax and some muscle memory (best for retention) and is a much more effective learning strategy than cut and paste will be!
Assignment Grading Rubric

Course: IT350 Unit: 7    Points: 15

Note ** Using the split screen feature in Windows 8 can be a great time saving strategy.

Also, for all your online education work, an optimal setup will be to have two separate monitors so that you can open and see the book, the course (Lab) and SSMS all at the same time. This is a low cost solution that can pay big dividends in your productivity, time and learning effectiveness!

Provide a screenshot from SSMS showing the queries for each main section below.

--- Using Aggregate Functions

USE AdventureWorks2012;
GO

--1
SELECT COUNT(*) AS CountOfRows,
       MAX(TotalDue) AS MaxTotal,
       MIN(TotalDue) AS MinTotal,
       SUM(TotalDue) AS SumOfTotal,
       AVG(TotalDue) AS AvgTotal
FROM Sales.SalesOrderHeader;

--1. Write a query that provides the total of sales dollars for all orders placed in December 2004 along with the number of orders. Use Sales.SalesOrderHeader.

--2
SELECT MIN(Name) AS MinName,
       MAX(Name) AS MaxName,
       MIN(SellStartDate) AS MinSellStartDate
FROM Production.Product;

--2. Write a query that shows the lowest cost product in Production.Product table.

--- Using the GROUP BY Clause

USE AdventureWorks2012;
GO

--1
SELECT CustomerID, SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID;

--2
SELECT TerritoryID, AVG(TotalDue) AS AveragePerTerritory
FROM Sales.SalesOrderHeader
GROUP BY TerritoryID;

-- 3. Write a Query that provides a count of the products on each order along with the total order amount using only Sales.SalesOrderDetail table.

--- Listing 5-3. How to Group on an Expression

Use AdventureWorks2012;
GO

--1
SELECT COUNT(*) AS CountOfOrders, YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
GROUP BY OrderDate;

Copyright Kaplan University
--2
SELECT COUNT(*) AS CountOfOrders, YEAR(OrderDate) AS OrderYear
FROM Sales.SalesOrderHeader
GROUP BY YEAR(OrderDate);

--Using ORDER BY
USE AdventureWorks2012;
GO
--1
SELECT CustomerID,SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
ORDER BY CustomerID;

--2
SELECT TerritoryID,AVG(TotalDue) AS AveragePerTerritory
FROM Sales.SalesOrderHeader
GROUP BY TerritoryID
ORDER BY TerritoryID;

--3
SELECT CustomerID,SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
ORDER BY SUM(TotalDue) DESC;

--4. Write a query that shows the average order amount by month and territoryID.

--Using the WHERE Clause
USE AdventureWorks2012;
GO
SELECT CustomerID,SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
WHERE TerritoryID in (5,6)
GROUP BY CustomerID;

-- 5. Write a query that shows the average order amount by month and territory number for all territories except (5 and 6.)

--Using the HAVING Clause
USE AdventureWorks2012;
GO
--1
SELECT CustomerID,SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING SUM(TotalDue) > 5000;

--2
SELECT CustomerID,SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING COUNT(*) = 10 AND SUM(TotalDue) > 5000;

Copyright Kaplan University
SELECT CustomerID, SUM(TotalDue) AS TotalPerCustomer
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING CustomerID > 27858;

--6. Write a Query that provides a count of the products on each order along with the total order amount using only Sales.SalesOrderDetail table only for orders with more than six products.

--Using DISTINCT and GROUP BY

USE AdventureWorks2012;
GO
--1
SELECT DISTINCT SalesOrderID
FROM Sales.SalesOrderDetail;
--2
SELECT SalesOrderID
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID;

--Using DISTINCT in an Aggregate Expression

USE AdventureWorks2012;
GO
--1
SELECT COUNT(*) AS CountOfRows,
    COUNT(SalesPersonID) AS CountOfSalesPeople,
    COUNT(DISTINCT SalesPersonID) AS CountOfUniqueSalesPeople
FROM Sales.SalesOrderHeader;
--2
SELECT SUM(TotalDue) AS TotalOfAllOrders,
    SUM(Distinct TotalDue) AS TotalOfDistinctTotalDue
FROM Sales.SalesOrderHeader;

-- 7.Provide a brief accurate explanation as to how the results from the above query would differ if the DISTINCT keyword was not used in the above two queries.

--Writing Aggregate Queries with Two Tables

USE AdventureWorks2012;
GO
--1
SELECT c.CustomerID, c.AccountNumber, COUNT(*) AS CountOfOrders,
    SUM(TotalDue) AS SumOfTotalDue
FROM Sales.SalesOrderHeader
GROUP BY c.CustomerID, c.AccountNumber;
FROM Sales.Customer AS c
INNER JOIN Sales.SalesOrderHeader AS s ON c.CustomerID = s.CustomerID
GROUP BY c.CustomerID, c.AccountNumber
ORDER BY c.CustomerID;

--2
SELECT c.CustomerID, c.AccountNumber, COUNT(*) AS CountOfOrders,
   SUM(TotalDue) AS SumOfTotalDue
FROM Sales.Customer AS c
LEFT OUTER JOIN Sales.SalesOrderHeader AS s ON c.CustomerID = s.CustomerID
GROUP BY c.CustomerID, c.AccountNumber
ORDER BY c.CustomerID;

--3
SELECT c.CustomerID, c.AccountNumber, COUNT(s.SalesOrderID) AS CountOfOrders,
   SUM(COALESCE(TotalDue, 0)) AS SumOfTotalDue
FROM Sales.Customer AS c
LEFT OUTER JOIN Sales.SalesOrderHeader AS s ON c.CustomerID = s.CustomerID
GROUP BY c.CustomerID, c.AccountNumber
ORDER BY c.CustomerID;

-- 8. Explain what Coalesce does in the query above. How do the results differ if you do not use it?

-- 9. Why is a Left Outer Join used in the above query,
-- How do the results differ if you use an Inner Join instead? Which results would be considered most correct? Why?

-- 10. Write the question that would lead to the above query being the answer

-- To return a list of the customer names along with a count of the orders placed.

-- To display the total sum of products (line total) by ProductID and OrderDate.

Submit to the appropriate Dropbox.

You will earn 15 pts. for submitting your best attempt.

The labs are intended to help you practice and think about the concepts in this course.
You will receive full grades for submitting your attempted answers.

Remember to type them in and avoid cut and paste where possible / practical.

**Directions:**

- When you have completed the Reading:
  - Open a word document.
  - Copy / Paste the section Descriptor.
  - Paste a screenshot (snipping tool) of the code executed in SSMS.

Your professor will review your Lab work in order to determine how you are progressing in the course. He/she may or may not provide specific feedback. In addition, the professor will use the Labs to determine if there are areas of covered material that many students are having difficulty with, and may then decide to provide extra explanation in the next Seminar or Discussion. Review the grading rubric below before beginning this Activity.

**Assignment grading rubric = 15 points**

<table>
<thead>
<tr>
<th>Assignment Criteria</th>
<th>Points possible</th>
<th>Points earned by student</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Includes a student response to all practice questions code samples / demonstrations showing that the code was run in SSMS questions.</td>
<td>0 - 15</td>
<td></td>
</tr>
</tbody>
</table>

**Column Total**