CONTENTS

Exercise 1 Introduction to TSQL

Overview

The main purpose of this lab is to familiarise yourself with key features of SQL Server Management Studio (SSMS), which we will be using throughout the course.

In this lab, you will investigate the help system, connect to a database server using SSMS and investigate the Query Designer tool.

Objectives

At the end of this lab, you will be able to:

- locate and launch SQL Server Management Studio
- create and save a new query script file
- use the Query Designer window to create a simple query.

Setup: Launch SQL Server Management Studio

1. Launch the QASQL virtual machine.
2. Log on as Administrator with a password of Pa55w.rd.
3. On the Start menu, click SQL Server Management Studio 18.
4. The Microsoft SQL Server Management Studio window opens, and then the Connect to Server dialog box appears.
5. In the Connect to Server dialog box, click Connect to accept the default settings.

Exercise 1: Explore the components of and execute queries in SSMS

In this exercise, you will learn how to use the components of and execute queries in SQL Server Management Studio.

The main tasks for this exercise are as follows:

1. Resize and hide Object Explorer and the Properties window.
2. Use the Query Designer to return rows from the Products table.

Task 1: Resize and hide the Object Explorer window

1. Reduce the size of the Object Explorer window in SSMS by moving your mouse to the right-hand edge until the cursor changes into a resize cursor, then hold down the left mouse button and move it to the left.
2. In SSMS, on the View menu, click Properties Window.
3. In Properties, click Auto Hide (the little drawing pin icon in the top right corner) to hide the pane.

Task 2: Use the Query Designer to return rows from a table.

1. In Object Explorer, expand the Databases folder.
2. Right-click on the Northwind database, and then choose “New Query” from the context menu.
3. On the Query menu, click “Design Query in Editor”.
4. The Query Designer and Add Table dialog boxes appear.
5. In the Add Table dialog box, select the Products table, and then click Add, then Close.
6. The Products table appears in the top pane of the Query Designer.
7. The columns of the Products table are listed in the designer.
8. In the Products table, select the “* (All Columns)” check box.
9. In the Query Designer dialog box, click OK.
10. Notice the query in the query editor window.
11. On the toolbar, click Execute.
12. All the columns in the Products table are displayed in the Results pane.
13. Now select all of the text in the query editor and from the Query menu choose “Design Query in Editor” again.
14. Note that the query designer now doesn’t have the “All columns” check box checked and that it selects each individual column by name.
15. Click OK.
16. Note the new contents of the Query Editor window.
17. From the File menu, save the query file, giving it a filename of “ProductDetails.sql”
18. Click the Close button to close SQL Server Management Studio.
19. Save all changes.

Exercise 2 Retrieving data

Overview

In this lab, you will be working with the dbo.Products table to produce a list of products and their stock levels.

If you're feeling confident, there is a second exercise which uses some string concatenation on the dbo.Employees table.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- write a simple select statement
- write a select statement that picks out only certain columns from the source table
- write a select statement that uses calculated values

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Administrator.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Retrieve data by using the SELECT statement

Northwind Traders are attempting to learn more about their stock levels. You have been asked to produce a list of products.

In this exercise, you will create simple SELECT statement queries.

The main tasks for this exercise are as follows:

1. Create a report using the SELECT * statement against the Products table.
2. Create a report that selects the ProductID, ProductName, UnitPrice and UnitsInStock columns from the Products table.
3. Add columns to the second report so that it also calculates the current value of stock and the future value of stock.

Task 1: Create a report using SELECT *

1. Create a new query and save it with a filename of "SelectAllProductDetails.sql"

2. Create a query that uses the Northwind database and then displays all columns and all rows from the dbo.Products table.
3. Execute the query by hitting the <F5> key.
4. Browse the result set in the Results pane. You should see 77 rows returned.

Task 2: Create a report that selects individual columns from a table

1. Create another new query and call it “StockList.sql”.
2. It should access the dbo.Products table in the Northwind database.
3. It should only include the ProductID, ProductName, UnitPrice and UnitsInStock columns.
4. Execute the query by hitting the <F5> key.
5. Browse the result set in the Results pane and confirm that only the specified columns appear.

Task 3: Create a report that selects individual columns from a table

1. Edit the existing “StockList.sql” query.
2. Add the UnitsOnOrder column to the select list.
3. Add a calculated column that multiplies UnitPrice by UnitsInStock and alias it as CurrentStockValue.
4. Add a second calculated column that adds UnitsOnOrder to UnitsInStock and multiplies the result by UnitPrice. Alias the new column as FutureStockValue.
5. Execute the query by hitting the <F5> key.
6. Browse the result set in the Results pane and confirm that the current stock value of ProductID 2, Chang, is 323 and its future value is 1083.

(Optional) Exercise 2: Concatenate strings in a select list

Write a query that selects the FullName and telephone Extension from the dbo.Employees table in the Northwind database.

You’ll need to concatenate the FirstName and LastName columns and you’ll need to put a space between them. There are 9 rows in the Employees table.

Completed Scripts

Exercise 1 Task 1:

```
USE Northwind

SELECT *
FROM dbo.Products
```

Exercise 1 Task 2:

```
USE Northwind

SELECT ProductID, ProductName, UnitPrice, UnitsInStock
FROM dbo.Products
```

Exercise 1 Task 3:

```
SELECT
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
```

Exercise 2:

```
SELECT
  FirstName + ' ' + LastName AS FullName,
  Extension
FROM
  dbo.Employees
```

Exercise 3 Filtering rows

Overview

In this lab, you will work with the dbo.Products table to produce a list of products in a certain category, adding various different filters to the report.

Then you will work with the dbo.Orders table to find orders that were placed between certain dates for certain customers.

The third exercise involves working with string comparisons on the dbo.Customers table.

The last exercise involves finding rows containing NULL values in the dbo.Customers table.

The "optional extras" are some final suggestions for additional tasks that you may choose to attempt if you are feeling confident. They are intended to inspire you to play around with the various filter operations once you've completed the core tasks and are entirely optional.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- write a query that uses an equality filter
- write a query that uses a comparison filter
- write a query that uses the IN and BETWEEN operators
- write a query statement that looks for text within a column
- write a query that correctly identifies NULL values

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Administrator.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Basic WHERE clauses

Northwind Traders are attempting to learn more about their currently-stocked beverages. You have been asked to produce a list of current products that are in that category and that have a high value.

In this exercise, you will use basic WHERE clauses.

The main tasks for this exercise are as follows:

1. Create a report that selects only the rows in the dbo.Products table which have a value of 0 (zero) in the Discontinued column.
2. Modify the report so that it only returns non-discontinued Products with a CategoryID of 1 (beverages).
3. Modify the report to include an additional filter that removes Products with a unit price less than 40.

Task 1: Create a report that only returns current products

1. Create a new query and save it with a name of "CurrentBeverageProducts.sql".
2. Write a report that uses the Northwind database and displays the ProductID, ProductName, CategoryID, Discontinued and UnitPrice columns from the dbo.Products table.
3. Execute the query and verify that it returns 77 rows.
4. Add a WHERE clause to the query that asks for only those products with a Discontinued value of 0 (zero).
5. Execute the query and verify that it returns only 69 rows.

Task 2: Write a report that only returns current products in category 1

1. Modify your existing report.
2. Add an additional statement to the WHERE clause that further limits the results to products with a CategoryID of 1.
3. Execute the query and verify that it now returns only 11 rows.

Task 3: Write a report that only returns expensive, current products in category 1

1. Modify your existing report.
2. Add an additional statement to the WHERE that further limits the results to products with a unit price greater than or equal to 40.
3. Execute the query and verify that it now retrieves just 2 products – Cote de Blaye and Ipoh Coffee.

Exercise 2: Using IN and BETWEEN

Northwind Traders have had some complaints about orders that were placed in a certain month for certain customers. You have been asked to produce a report of all the orders that were placed by those customers in that month.

In this exercise, you will use the IN and BETWEEN operators.

The main tasks for this exercise are as follows:

1. Create a report that selects only the rows in the dbo.Orders table which were placed by either Alfreds Futterkiste (CustomerID 'ALFKI'), Ernst Handle ('ERNSH') or Simon's bistro ('SIMOB').
2. Modify the report so that it only returns orders for those customers placed between the 1st August and the 31st August 1997.
3. Ensure that the report uses IN and BETWEEN operators.

Task 1: Create a report that only returns orders for some customers

1. Create a new query and save it with a name of "CustomerComplaints.sql".
2. Write a report that uses the Northwind database and displays the OrderID, CustomerID and OrderDate columns from the dbo.Orders table.
3. Limit the results to only those orders with a CustomerID of either 'ALFKI', 'ERNSH' or 'SIMOB'.
4. Remember that string literals need to be enclosed in 'single quotes'.
5. Execute the query and verify that it returns 43 rows.

Task 2: Write a report that returns orders for some customers in August 1997

1. Modify your existing report.
2. Further limit the results of the query so that it only returns orders with an OrderDate greater than or equal to the 1st August 1997 and an OrderDate of less than or equal to the 31st August 1997.
3. Remember that date literals, like string literals, need to be enclosed in 'single quotes'.
4. Execute the query and verify that there are now just 3 rows – OrderIDs 10633, 10642 and 10643.

   NOTE: it is quite likely that you'll get an error at this point! The reason for this being that SQL Server is an American product and the American date format is MDY rather than DMY. There is no 31st month of the year!

   There are several ways to resolve this issue but the simplest is either to spell out the month – i.e. *'31 Aug 1997'* or to use YMD format – *'1997/08/31'*. Remember to do this for both the start date and the end date otherwise you might be looking for orders between the 8th January and the 31st August!
5. Make sure the query returns the three rows we want. 37 rows and 73 rows are wrong for different reasons.

Task 3: Ensure that the query uses IN and BETWEEN

1. Modify your existing report.
2. If it doesn't already, ensure that the report uses IN for the CustomerIDs and BETWEEN for the OrderDates.
3. Notice that the query is much easier to read and doesn't require so many parentheses.

4. Execute the query and verify that it still returns 3 rows.

Exercise 3: String comparisons

Northwind Traders Marketing department has asked you to produce a list of all your customer contacts who are involved in the sales function.

In this exercise, you will use the LIKE operator and wildcard characters.

The main tasks for this exercise are as follows:

1. Create a report that selects rows from the dbo.Customers table where the Contact's job description starts with the word 'Sales'.
2. Modify the report so that it only returns contact details for anyone with the word 'sales' anywhere in their job description.
3. Experiment with case sensitivity.

Task 1: Create a report that only returns contact details for some customers

1. Create a new query and save it with a name of "SalesContacts.sql".
2. Write a report that uses the Northwind database and selects the CustomerID, CompanyName, ContactName, ContactTitle and Phone columns from the dbo.Customers table.
3. Execute the query and verify that it returns 91 rows.
4. Now add a WHERE clause which looks for a ContactTitle that begins with the word 'sales'.
5. You should retrieve 40 rows.

Task 2: Use leading and trailing wildcards

1. Modify the existing report.
2. Modify the LIKE comparison so that it looks for Contacts where the word 'sales' appears anywhere in their ContactTitle column.
3. You're expecting 43 rows now – there are 3 customer contacts whose job title is "Assistant Sales Agent".

Task 3: Investigate case sensitivity in SQL Server

1. Modify the existing report.
2. Try looking for SALES, or Sales, or sales, or even SaLeS.
3. You should still see the same 43 rows – by default, SQL Server is configured to be case insensitive.

Exercise 4: NULL comparisons

Northwind Traders Marketing department wants to update all their contact details to ensure that they have fax numbers for everyone. You have been asked to produce a list of all your customer contacts that don't have a Fax number listed.

In this exercise, you will use the IS NULL operator.

The main tasks for this exercise are as follows:

1. Create a report that selects rows from the dbo.Customers table where the Fax column has a *NULL* value.

Task 1: Create a report that only lists customers with no fax number listed

1. Create a new query and save it with a name of "FaxlessContacts.sql".
2. Write a report that uses the Northwind database and selects the CustomerID, CompanyName, ContactName, ContactTitle, Phone and Fax columns from the dbo.Customers table.
3. Execute the query and verify that it returns 91 rows.
4. Now add a WHERE clause which looks for *null* values in the Fax column.
5. You should retrieve 22 rows.

(Optional) Exercise 5: Filtering on expressions

You have been asked to produce a report on products whose future stock value is greater than 2000. Re-use the final query from Lab 2 Exercise 1 Task 3 but add an appropriate WHERE clause. You should get 14 rows.

Optional Extras:

1. Modify the expensive products query from Exercise 1 so that it looks for current products in both category 1 and category 8.
2. Modify the problem orders query from Exercise 2 so that it looks at a bigger range of dates but also limits the results to only certain EmployeeIDs (say 1,3,5,7 and 9).
3. Try using an "= NULL" comparison instead of an IS NULL in the query from exercise 4.
4. For extra credit, try and get it to return some rows!!!

Completed Scripts

Exercise 1 Task 1:

```
USE Northwind

SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0
```

Exercise 1 Task 2:

```
SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0 AND CategoryID = 1
```

Exercise 1 Task 3:

```
SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0 AND CategoryID = 1 AND UnitPrice >= 40
```

Exercise 2 Task 1:

```
USE Northwind

SELECT
  OrderID, CustomerID, OrderDate
FROM
  dbo.Orders
WHERE
  CustomerID = 'ALFKI' OR CustomerID = 'ERNSH'
OR CustomerID = 'SIMOB'
```

Exercise 2 Task 2:

```
SELECT
  OrderID, CustomerID, OrderDate
FROM
  dbo.Orders
WHERE
  (CustomerID = 'ALFKI' OR CustomerID = 'ERNSH'
   OR CustomerID = 'SIMOB')
AND
  (OrderDate >= '1 Aug 1997' AND OrderDate <= '31 Aug 1997')
```

Exercise 2 Task 3:

```
SELECT
  OrderID, CustomerID, OrderDate
FROM
  dbo.Orders
WHERE
  CustomerID IN ('ALFKI', 'ERNSH', 'SIMOB')
AND
  OrderDate BETWEEN '1 Aug 1997' AND '31 Aug 1997'
```

Exercise 3 Task 1:

```
USE Northwind

SELECT
  CustomerID, CompanyName, ContactName, ContactTitle, Phone
FROM
  dbo.Customers
WHERE
  ContactTitle LIKE 'sales%'
```

Exercise 3 Task 2:

```
SELECT
  CustomerID, CompanyName, ContactName, ContactTitle, Phone
FROM
  dbo.Customers
WHERE
  ContactTitle LIKE '%sales%'
```

Exercise 4 Task 1:

```
SELECT
  CustomerID, CompanyName, ContactName, ContactTitle, Phone, Fax
FROM
  dbo.Customers
WHERE
  Fax IS NULL
```

Exercise 5:

```
SELECT
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
WHERE
  (UnitsInStock + UnitsOnOrder) * UnitPrice > 2000
```

Exercise 4 Sorting rows

Overview

In this lab, you will work with the dbo.Products table to produce a list of products sorted by category and by unit price within those categories.

In the second exercise you will revisit an earlier product query to sort the output.

The third exercise asks you to produce a report of unique countries from the dbo.Customers table.

The fourth exercise involves listing the ten most expensive products.

The final, optional, exercise involves listing the ten products with the highest value of current stock.

At the end of this chapter is a “completed scripts” section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- write a query that sorts on a single column
- write a query that sorts on multiple columns
- write a query that sorts on a calculated column
- write a query that selects unique values
- write a query that returns only a specified number of rows

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Administrator.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Basic ORDER BY clauses

Northwind Traders are attempting to learn more about their product pricing. You have been asked to produce a list of current products, sorted by category and then by unit price in reverse order.

In this exercise, you will use basic ORDER BY clauses.

The main tasks for this exercise are as follows:

1. Create a report that selects only the rows in the dbo.Products table which have a value of 0 (zero) in the Discontinued column.
2. Modify the report so that it only returns non-discontinued Products sorted by their CategoryIDs.
3. Modify the report to include an additional sort that sorts on UnitPrice from highest to lowest.

Task 1: Create a report that only returns current products

1. Create a new query and save it with a name of "CurrentProducts.sql".
2. Write a report that uses the Northwind database and displays the ProductID, ProductName, CategoryID, Discontinued and UnitPrice columns from the dbo.Products table and only selects rows with a Discontinued value of 0.
3. Execute the query and verify that it returns 69 rows.

Task 2: Write a report that sorts current products by category

1. Modify your existing report.
2. Add an ORDER BY clause to the query that sorts the results based on the CategoryID column.
3. Execute the query and verify that it returns the 69 rows but that all the products with a CategoryID of 1 are together, all those with CategoryID 2 are together, *etc*.

Task 3: Write a report that sorts current products by category and unit price

1. Modify your existing report.
2. Add an additional statement to the ORDER BY clause that sorts the results in reverse order on UnitPrice.
3. Execute the query and verify that your 69 rows are still sorted by categoryid but that within those categories the most expensive products are displayed first.

Exercise 2: Sorting on calculated columns

Northwind Traders are still trying to get a handle on their stock levels. They are really pleased with a stock report you wrote earlier but now that they know you can sort data, they'd like you to make it easier to see which products have the highest value of future stock.

In this exercise, you will work on an earlier query and add an ORDER BY clause to it.

The main tasks for this exercise are as follows:

1. Re-use, or copy and paste, or re-type the StockList query from Lab 2 Exercise 1 Task 3.
2. Modify the report so that it sorts the results in reverse order of future stock value.

Task 1: Re-use an existing query

1. Create a copy of the script file “StockList.sql”.
2. Rename the new file “SortedStockList.sql”.

   NOTE: If you can’t find the query file, the SQL is reproduced below:

```
SELECT
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
```

3. Open the query in SSMS.
4. Execute the query and verify that it returns 77 rows.

Task 2: Write a report that sorts on a calculated column

1. Modify the existing report.
2. Add an order by clause that sorts on the value of the calculated “FutureStockValue” column, in reverse order.

   NOTE: There are three different ways of achieving this.
3. Execute the query and verify that the first product returned is Cote de Blaye, product id 38.

Exercise 3: SELECT DISTINCT

You have been asked to produce a list of the unique countries in which Northwind Traders has customers.

In this exercise, you will use the DISTINCT modifier on a SELECT statement.

The main tasks for this exercise are as follows:

1. Create a report that selects the Country column from the dbo.Customers table.
2. Modify the report so that it only returns a single row for duplicate country names.

Task 1: Create a report that selects the customer’s countries

1. Create a new query and save it with a name of “CustomerCountries.sql”.

2. Write a report that uses the Northwind database and selects the Country column from the dbo.Customers table.
3. Execute the query and verify that it returns 91 rows.

Task 2: Select distinct rows

1. Modify the existing report.
2. Modify the SELECT statement so that it looks for distinct values only.
3. You're expecting 21 rows now. Notice that the countries are sorted alphabetically as well.

Exercise 4: SELECT TOP

You have been asked to produce a list of all the top ten most expensive products based on unit price.

In this exercise, you will use the TOP modifier.

The main task for this exercise is as follows:

1. Create a report that selects the top ten rows from the dbo.Products table sorted on reverse value of unit price.

Task 1: Create a report of the top ten most expensive products

1. Create a new query and save it with a name of "MostExpensiveProducts.sql".
2. Write a report that uses the Northwind database and selects the ProductID, ProductName and UnitPrice columns from the dbo.Products table.
3. Order the results in descending order of UnitPrice.
4. Restrict the results to the first ten results
5. You should retrieve 10 rows. Cote de Blaye, unsurprisingly, should be the first.

(Optional) Exercise 5: More TOPping

You have been asked to modify a copy of the SortedStockList report from Exercise 2 Task 2 of this lab, and select only the top ten products based on their current stock value.

Completed Scripts

Exercise 1 Task 1:

```
USE Northwind

SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0
```

Exercise 1 Task 2:

```
SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0
ORDER BY
  CategoryID
```

Exercise 1 Task 3:

```
SELECT
  ProductID, ProductName, CategoryID, Discontinued, UnitPrice
FROM
  dbo.Products
WHERE
  Discontinued = 0
ORDER BY
  CategoryID,
  UnitPrice DESC
```

Exercise 2 Task 1:

```
USE Northwind

SELECT
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
```

Exercise 2 Task 2:

```
SELECT
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
ORDER BY
  FutureStockValue DESC
```

Exercise 3 Task 1:

```
USE Northwind

SELECT
  Country
FROM
  dbo.Customers
```

Exercise 3 Task 2:

```
SELECT DISTINCT
  Country
FROM
  dbo.Customers
```

Exercise 4:

```
USE Northwind

SELECT TOP 10
  ProductID, ProductName, UnitPrice
FROM
  dbo.Products
ORDER BY
  UnitPrice DESC
```

Exercise 5:

```
SELECT TOP 10
  ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder,
  UnitPrice * UnitsInStock AS CurrentStockValue,
  (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM
  dbo.Products
ORDER BY
  CurrentStockValue DESC
```

Exercise 5 Grouping and aggregating

Overview

In this lab, you will work with the dbo.Orders table to find out general information about the orders on the system.

In the second exercise, you will start to investigate individual customers' ordering habits.

In the third exercise you will work with the dbo.[Order Details] table to start drilling down into exactly what products were ordered by customers and filtering the results of aggregated queries.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- write a query that calculates single aggregates
- write a query that calculates aggregates for groups of information
- write a query that aggregates, groups and filters on calculated values

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Administrator.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Basic aggregates

You want to learn more about the data stored in the dbo.Orders table, specifically those placed by Nancy Davolio.

In this exercise, you will use basic aggregates.

The main tasks for this exercise are as follows:

1. Create a report that selects a count of the orders.
2. Modify the report so that it also calculates the earliest and latest dates on which orders were placed.
3. Modify the report to only calculate the values for Nancy's orders (employee id 1).

Task 1: Create a report that selects a count of rows

4. Create a new query and save it with a name of “OrderAnalysis.sql”.
5. Write a report that uses the Northwind database and displays a count of all the rows in the dbo.Orders table. Alias the count to call it “NumberOfOrders”.
6. Execute the query and verify that the result set contains a single value of 830.

Task 2: Write a report that selects maximums and minimums

1. Modify your existing report.
2. Add an aggregate to the select list that calculates the minimum OrderDate value. Alias it as “EarliestOrder”.
3. Add another aggregate that calculates the maximum OrderDate value. Alias it as “LatestOrder”.
4. Execute the query and verify that the answers are 830, 4th July 1996 and 6th May 1998.

Task 3: Write a report that selects aggregates for only one employee

1. Modify your existing report.
2. Add a WHERE clause to your query so that it only includes rows with an EmployeeID equal to 1.
3. Execute the query and verify that the answers are now 123, 17th July 1996 and 6th May 1998.

Exercise 2: Grouping aggregates

You have been asked to create a report on the number of orders placed by each customer. The report is to be sorted by the number of orders placed, from highest to lowest.

In this exercise, you will use the GROUP BY clause and an ORDER BY.

The main tasks for this exercise are as follows:

1. Create a report that counts the number of OrderIDs in the dbo.Orders table.
2. Modify the report so that it includes the CustomerID and groups the results on that column.
3. Modify the report to sort in reverse order on the number of orders placed.

Task 1: Create a query that counts orders

1. Create a new query and save it with a name of “CustomerOrders.sql”.
2. Write a report that uses the Northwind database and displays a count of all the OrderIDs in the dbo.Orders table. Alias the aggregate as “NumberOfOrders”.

3. Execute the query and verify that it returns a value of 830.

Task 2: Write a report that groups orders based on the customer's ID

1. Modify the existing report.
2. Add the CustomerID column to the report's select list.

   NOTE: At this point, the query won't work.
3. Add a GROUP BY clause to the report that groups the results based on the CustomerID column.
4. Execute the query and verify that it returns 89 rows, the top one being ALFKI with a NumberOfOrders of 6.

Task 3: Write a report that sorts order counts in descending order

1. Modify the existing report.
2. Add an ORDER BY clause to the report that sorts on the NumberOfOrders column in descending order.
3. Execute the query and verify that it returns 89 rows, the top one now being SAVEA with a count of 31.

Exercise 3: Aggregating calculated values and filtering aggregates

Northwind Traders are trying to see exactly how much customers are spending on products.

In this exercise, you will use aggregate functions on calculated columns.

The main tasks for this exercise are as follows:

1. Create a report that sums the Quantity column of the dbo.[Order Details] table, grouped by product ids.
2. Modify the report so that it sums the Quantity times the UnitPrice and sorts the results in descending order of those values.
3. Modify the report to filter the results to only include those products with a total value of less than or equal to 5000.

Task 1: Create a report that sums quantities of products sold

1. Create a new query and save it with a name of "ProductSales.sql".
2. Write a report that uses the Northwind database and selects the ProductID and the sum of the Quantity columns from the dbo.[Order Details] table. Alias the aggregate column as "TotalSold"
3. Group the results on the ProductID column.
4. Execute the query and verify that it returns 77 rows, the first row being product id 23, with a TotalSold of 580.

Task 2: Create a report that sums a calculation

1. Modify the existing report.
2. Modify the SUM aggregate so that it adds up the value of the Quantity column multiplied by the UnitPrice column for each product. Change the alias name to “TotalValue”.
3. Sort the results on the TotalValue column, in descending order.
4. Execute the query and verify that the top-selling product is productid 38, with a total sales value of 149984.20.

Task 3: Create a report that filters aggregate values

1. Modify the existing report.
2. Add a HAVING clause to the report to only return the rows with a TotalValue of less than or equal to 5000.
3. REMEMBER: just like with a WHERE clause, the actual column named TotalValue doesn’t exist yet in your HAVING, so you’ll need to re-use the calculation.
4. Execute the query and verify that you now see only 16 rows, the first of which is product 23 with a value of 4840.20.

Completed Scripts

Exercise 1 Task 1:

```
USE Northwind

SELECT COUNT(*) as NumberOfOrders
FROM  dbo.Orders
```

Exercise 1 Task 2:

```
SELECT
  COUNT(*) as NumberOfOrders,
  MIN(OrderDate) as EarliestOrder,
  MAX(OrderDate) as LatestOrder
FROM  dbo.Orders
```

Exercise 1 Task 3:

```
SELECT COUNT(*) as NumberOfOrders,
  MIN(OrderDate) as EarliestOrder, MAX(OrderDate) as LatestOrder
FROM  dbo.Orders
WHERE EmployeeID = 1
```

Exercise 2 Task 1:

```
USE Northwind

SELECT COUNT(OrderID) AS NumberOfOrders
FROM dbo.Orders
```

Exercise 2 Task 2:

```
SELECT CustomerID, COUNT(OrderID) AS NumberOfOrders
FROM  dbo.Orders
GROUP BY CustomerID
```

Exercise 2 Task 3:

```
SELECT CustomerID, COUNT(OrderID) AS NumberOfOrders
FROM  dbo.Orders
GROUP BY CustomerID
ORDER BY NumberOfOrders DESC
```

Exercise 3 Task 1:

```
USE Northwind

SELECT ProductID, SUM(Quantity) AS TotalSold
FROM  dbo.[Order Details]
GROUP BY ProductID
```

Exercise 3 Task 2:

```
SELECT ProductID, SUM(Quantity * UnitPrice) AS TotalValue
FROM  dbo.[Order Details]
GROUP BY ProductID
ORDER BY TotalValue DESC
```

Exercise 3 Task 3:

```
SELECT ProductID, SUM(Quantity * UnitPrice) AS TotalValue
FROM  dbo.[Order Details]
GROUP BY ProductID
HAVING SUM(Quantity * UnitPrice) <= 5000
ORDER BY TotalValue DESC
```

Exercise 6 Using multiple tables

Overview

In this lab, you will work with the dbo.Orders, dbo.Customers, dbo.[Order Details] and dbo.Products tables to find out detailed information about the orders on the system.

In the second exercise, you will investigate individual customers' ordering patterns.

In the third exercise you will work with the dbo.Suppliers, dbo.Customers and dbo.Employees table to produce a company-wide telephone directory.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- write a query that uses an inner join to select rows from two tables
- write a query that filters joined data
- write a query that uses an inner join to select rows from more than two tables
- write a query that investigates the difference between left and right outer joins
- write a query that combines the results of multiple other queries

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Admin.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Join fundamentals

You want to write a report about UK-based customers and the orders that they have placed.

In this exercise, you will use inner joins, the fundamental type of join in SQL.

The main tasks for this exercise are as follows:

1. Create a report that selects the customer id, city and company and contact names from the Customers table and the order number and order date from the Orders table.

2. Modify the report so that it only picks up orders placed by UK Customers and sorts the results by customer and order date
3. Modify the report so that it also includes details about the name of the product that was ordered, and how many units of each product were ordered.

Task 1: Create a report that selects rows from the Customers and Orders tables

1. Create a new query and save it with a name of "UKCustomerOrders.sql".
2. Write a report that uses the Northwind database and selects the CustomerID, CompanyName, ContactName and City columns from the dbo.Customers table. Alias the table as 'c'.
3. Execute the query and verify that 91 rows are returned.
4. Modify the query to join rows from the dbo.Orders table that have the same value in the CustomerID column. Alias Orders as 'o'.
5. Add the OrderID and OrderDate columns from the Orders table to the select list.
6. Execute the query and verify that 830 rows are returned.

Task 2: Write a query that filters and sorts the results

1. Modify your existing report.
2. Add a where clause to limit the rows returned to those Customers whose Country column is equal to UK.
3. Execute the query and verify that 56 rows are returned, all of whose City columns are British cities.
4. Add an order by clause to ensure that results are sorted on CompanyName then OrderDate
5. Execute the query and verify that 56 rows are returned, the first one is for AROUT from November 1996 and the last one is for SEVES from February 1998.

Task 3: Write a query that includes rows from more than two tables

1. Make a copy of your existing query.
2. Join the dbo.[Order Details] table aliased as 'od'. Use the OrderID column in Orders and Order Details to find matching rows.
3. Add the ProductID and Quantity columns to the select list. You can also remove the City column from the earlier queries – that was just to add a visual verification that you were only getting UK customers.
4. Execute the query and verify that 135 rows are returned – we're getting more rows back because we're selecting every row down at the Order Detail level now.

5. Join yet another table – the Products table. Alias it as 'p' and use the ProductID column from Order Details to find the matching rows. Include the ProductID and ProductName columns in your select list.
6. Execute the query and verify that the first order is for 25 lots of Guarana Fantastica and the last one is for 20 Scottish Longbreads.
7. [Extra Credit] limit the results to only include products in the Seafood category and display the category name as well. (18 rows)

Exercise 2: Investigate outer joins

In this exercise you will investigate left and right outer joins. You will also revisit aggregation from an earlier exercise.

The main tasks for this exercise are as follows:

1. Create a query that gets a count of Customers.
2. Create a query that selects the Company Name from Customers and the number of orders for those customers from the Orders table.
3. Modify the report to retrieve all customers, regardless of whether they have placed orders and add a column showing the minimum order date to the select list.

Task 1: Create a query that counts customers

1. Create a new query and save it with a name of "CustomerOrderAnalysis.sql".
2. Write a report that uses the Northwind database and displays a count of all the rows in the dbo.Customers table.
3. Execute the query and make a note of the result.

Task 2: Write a report that groups orders based on the customer's name

[NOTE: This is similar to a task in an earlier lab]

1. Comment out or delete the Customer count query.
2. Write a new query that selects the CompanyName column from the Customers table and a count of OrderID columns from the Orders. Alias the count column as 'NumOrders'. You will need a group by clause.
3. Add an order by clause to sort the records in number of orders placed.
4. Execute the query and make a note of the number of rows returned. You will see that it differs from the number of customers. Take a few moments to think about why that might be. HINT: what is the lowest count of orders?

Task 3: Write a report that uses left and right outer joins

1. Modify the existing report.
2. Change the query so that it selects all of the rows from the Customers table and those rows from the Orders table where there is a match.

3. Execute the query and verify that it now returns 91 rows, the top two now being FISSA and Paris Specialités, both with a count of 0.
4. Change the left outer join to a right outer join and confirm that you're back to only seeing 89 rows. In this case, we get the same results as for an inner join.
5. Change the order in which the tables are listed so that the right outer join returns 91 rows.
6. Finally, add another column to the query that selects the minimum order date from the Orders table and alias it as 'MinDate'.
7. Execute the query and note that we have a couple of null values for the inactive customers. The COUNT aggregate was able to come up with a total of zero for those customers but the MIN has no values to work on so it must return null. In an actual report, you might want to tidy that up so that it shows some text instead.

Exercise 3: Create a telephone directory

Northwind Traders want to create a centralised telephone directory for all suppliers, customers and employees.

In this exercise, you will use the UNION operator to combine results.

The main tasks for this exercise are as follows:

1. Create a report that retrieves the company name, contact name and telephone number of all customers.
2. In the same query, create a report that retrieves the same columns for all suppliers.
3. In the same query, create a report that retrieves the full name and extension number for all employees.
4. Combine the results using a UNION operator.

Task 1: Create a report that retrieves Customer contact details

1. Create a new query and save it with a name of "ContactDirectory.sql".
2. Write a report that uses the Northwind database and selects the CompanyName, ContactName and Phone columns from the dbo.Customers table.
3. Execute the query and verify that it returns 91 rows.

Task 2: Create a report that retrieves Supplier contact details

1. In the same script file, add another query that selects the same three columns from the Suppliers table.
2. Select only that part of the script and execute the query. Verify that 29 rows are returned.

Task 3: Create a report that retrieves Employee contact details

1. In the same script file, add another query that selects rows from the Employees table
2. In the select list for the query, add a column that concatenates the FirstName column to a space character and the LastName column (building up a string containing the employee's full name).
3. Add the Extension column as well.
4. Execute the query and verify that you retrieve 9 rows of two columns – one containing the full name and one containing the extension.

Task 4: Combine the results using the UNION operator

1. In the same script file, between the customers query and the suppliers query, add the UNION keyword.
2. Select both parts of the script and execute. Verify that you get 120 rows back. Scroll through the results and note that they are sorted alphabetically by company name.
3. Now add the UNION keyword between the suppliers query and the employees query and attempt to run the whole script.
4. You get an error, because the employees query only has two columns in it whilst the other two have three columns.
5. You need to add a company name column to the employee's part of the query. Now, the Employees table doesn't have a company name column but that's not a problem because we actually know what company all of the employees work for – Northwind!
6. At the start of the select list for employees, add the string 'Northwind Traders'.
7. Run the whole query again and verify that 129 rows are returned. Scroll through the list and note that the Northwind Traders employees are all floating around in the middle of the results.
8. Change the UNIONs to UNION ALL and rerun the query. This time all the employees are right at the very end. Also, although you probably won't have noticed, the query has run considerably quicker than before as it hasn't tried to remove duplicates by sorting first.
9. Feel free to add an order by clause if you want to.

Completed Scripts

Exercise 1 Task 1:

```
SELECT c.CustomerID, c.CompanyName, c.ContactName,
       c.City, o.OrderID, o.OrderDate
FROM   dbo.Customers AS c
JOIN   dbo.orders AS o
ON     c.CustomerID = o.CustomerID
```

Exercise 1 Task 2:

```
SELECT   c.CustomerID, c.CompanyName, c.ContactName,
         c.City, o.OrderID, o.OrderDate
FROM     dbo.Customers AS c
JOIN     dbo.orders AS o
ON       c.CustomerID = o.CustomerID
WHERE    c.Country = 'UK'
ORDER BY c.CompanyName, o.OrderDate
```

Exercise 1 Task 3:

```
SELECT   c.CustomerID, c.CompanyName, c.ContactName,
         o.OrderID, o.OrderDate, od.ProductID,
         p.ProductName, od.Quantity
FROM     dbo.Customers AS c
JOIN     dbo.orders AS o
ON       c.CustomerID = o.CustomerID
JOIN     dbo.[Order Details] AS od
ON       o.OrderID = od.OrderID
JOIN     dbo.Products AS p
ON       od.ProductID = p.ProductID
WHERE    c.Country = 'UK'
ORDER BY c.CompanyName, o.OrderDate
```

Exercise 2 Task 1:

```
USE Northwind

SELECT COUNT(*) FROM dbo.Customers
```

Exercise 2 Task 2:

```
SELECT   c.CompanyName, COUNT(o.OrderID) AS NumOrders
FROM     dbo.Customers AS c
JOIN     dbo.Orders AS o
ON       o.CustomerID = c.CustomerID
GROUP BY c.CompanyName
ORDER BY NumOrders
```

Exercise 2 Task 3:

```
SELECT
     c.CompanyName, COUNT(o.OrderID) AS NumOrders,
     MIN(o.OrderDate) AS MinDate
FROM
     dbo.Orders AS o
RIGHT OUTER JOIN
     dbo.Customers AS c
ON
     o.CustomerID = c.CustomerID
GROUP BY
     c.CompanyName
ORDER BY
     NumOrders
```

Exercise 3 Task 1:

```
USE Northwind

SELECT CompanyName,   ContactName, Phone
FROM  dbo.Customers
```

Exercise 3 Task 2:

```
SELECT CompanyName,   ContactName, Phone
FROM  dbo.Suppliers
```

Exercise 3 Task 3:

```
SELECT FirstName + ' ' + LastName,    Extension
FROM   dbo.Employees
```

Exercise 3 Task 4:

```
SELECT CompanyName, ContactName, Phone
FROM   dbo.Customers

UNION ALL

SELECT CompanyName, ContactName, Phone
FROM   dbo.Suppliers

UNION ALL

SELECT 'Northwind Traders',
       FirstName + ' ' + LastName,    Extension
FROM   dbo.Employees
```

Exercise 7 Views and Stored Procedures

Overview

In this lab, you will use an existing view to see just how much it simplifies a complex query.

In the second exercise, you will create your own view.

In the third exercise you will use existing stored procedures to retrieve data.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- use a predefined view
- write and use a simple view
- execute predefined stored procedures

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Student.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: Use a predefined view

You want to write a query that retrieves full information about orders, including the name of the employee who placed the order, the name of the product that was ordered, details about the company name the order was for, which address the order was to be shipped to, calculations about the order total after applying discounts and much more. You will use a view to do this.

The main tasks for this exercise are as follows:

1. Examine a prewritten query and note its complexity.
2. Use a view that retrieves the same data.

Task 1: Examine a complex query

1. Open the script file called 'Ex01Task01.sql' from the sample solutions folder.
2. Execute the query and verify that 2155 rows are returned.
3. Add a where clause to filter the results only for UK customers (it's the Country column within the Customers – it's in there somewhere!). You'll

need to ensure that you use the customers table's alias because there are at least two Country columns in the tables the query uses.

4. Confirm that 135 rows are returned
5. Now imagine having to rewrite that query every time you wanted full details about an order or filter or sort it in a different way!

Task 2: Use a view

1. Create a new query called "UseView.sql".
2. Write a select statement that retrieves every row and column from the dbo.Invoices table.
3. Execute the query and verify that 2155 rows are returned – the same as the previous complex query. The Invoices "table" is actually a view, which is a stored version of the query you just looked at.
4. Add a where clause to filter the results only for UK customers. You don't need to worry about which table it's coming from this time, because as far as the view is concerned, there's only the one Country column.
5. Experiment with sorting and filtering options and only selecting some of the columns.

Exercise 2: Create a telephone directory view

In an earlier lab, you may have created a query to generate a combined contact directory for Northwind Traders. In this exercise, rather than having it stored in a script file, you will store it in the database as a view.

The main tasks for this exercise are as follows:

1. Create (or reuse) a report that retrieves the company name, contact name and telephone number of all customers, suppliers and employees.
2. Store that report as a view in the database.
3. Use the view in another query.

Task 1: Create a report that retrieves Customer contact details

1. Create a new query and save it with a name of "DirectoryView.sql".
2. If you completed an earlier lab exercise on UNIONs, copy the directory query into your new script. If not, copy the content of Ex02Starter.txt from this lab's starter folder.
3. Execute the query and verify that it returns 129 rows.

Task 2: Create a new view in the database

1. Immediately before the first SELECT statement of the contact directory query, add the appropriate commands to create a new view called 'dbo.ContactDirectory'. If you have a 'USE Northwind' at the start of your script, you might need to put a "GO" statement between that and your create statement.

2. Run the script and confirm that you get a message saying “command(s) completed successfully”.

Task 3: Create a report that uses your new view

1. Create a new script file called “UseMyView.sql”.
2. Write a query that selects all columns from the ContactDirectory view. Don’t worry about any red-squiggly lines – that just means that SSMS hasn’t caught up with the fact that there’s a new view. Make sure it retrieves 129 rows. If you get an error message saying “invalid object name” then you might have a problem…
3. Add a where clause to only select those contacts with a contact name that starts with the letter ‘a’. You’ll need to use a like operator.
4. Execute the query and verify that 14 contacts are returned, two of which are Northwind employees.

Exercise 3: Use existing stored procedures to retrieve rows

Northwind Traders already have a number of stored procedures defined. In this exercise you will get some practice executing them

The main tasks for this exercise are as follows:

1. Execute a stored procedure.
2. Execute a stored procedure using named parameters

Task 1: Execute the CustOrderHist stored procedure

1. Create a new query and save it with a name of “OrdHist.sql”.
2. Execute the dbo.CustOrderHist stored procedure, passing it a parameter of ‘ALFKI’
3. Verify that 11 rows are returned.
4. Try it with a couple of different CustomerID parameters to make sure that it returns different values for different customers.

Task 2: Execute the SalesByCategory stored procedure

1. Create a new query and save it with a name of “CatSales.sql”.
2. Execute the dbo.SalesByCategory stored procedure 3 times, passing it a category name of ‘Seafood’ each time and a year of 1996, 1997 then 1998.
3. Verify that 12 rows are returned each time but the Total Purchase values change. This procedure calculates total sales by category by year (so perhaps it’s badly-named…)
4. Remove the second parameter and execute the proc again. It still works and retrieves the sales from 1998 as a default.
5. Remove the category name parameter and execute. It fails.

6. Let's try it another way – year first then category. It doesn't fail, but doesn't give us back any meaningful results. It's actually looking for a category called "199x", which doesn't exist.
7. One final experiment. Use named parameters. Leave them the "wrong way round" but before the year, type "@OrdYear = "; and before the category, type "@CategoryName = ". It should work correctly again.

Completed Scripts

Exercise 1 Task 1:

```
… WHERE c.Country = 'UK'
```

Exercise 1 Task 2:

```
SELECT * FROM dbo.Invoices
WHERE Country = 'UK'
```

Exercise 2 Task 2:

```
CREATE VIEW dbo.ContactDirectory
AS

SELECT CompanyName, ContactName, Phone
FROM   dbo.Customers
UNION ALL
SELECT CompanyName, ContactName, Phone
FROM   dbo.Suppliers
UNION ALL
SELECT 'Northwind Traders',
       FirstName + ' ' + LastName,    Extension
FROM   dbo.Employees
```

Exercise 2 Task 3:

```
SELECT * FROM dbo.ContactDirectory
WHERE ContactName LIKE 'A%'
```

Exercise 3 Task 1:

```
EXEC dbo.CustOrderHist 'ALFKI'
```

Exercise 3 Task 2:

```
EXEC dbo.SalesByCategory
     @OrdYear = 1997, @CategoryName = 'Seafood'
```

Exercise 8 Common Functions

Overview

In this lab, you will work with existing queries that might be familiar from earlier labs and enhance them with the use of some common functions.

In the first exercise, you will include some string manipulation in the Contact Directory to enable you to sort it by the contact's last name.

In the second exercise you will use some date functions to work with orders from a given year.

In the third exercise you will add some default text to an outer-joined query when there are no matches on the right-hand side.

Finally, you will add some date formatting to the same query.

At the end of this chapter is a "completed scripts" section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- use string functions
- use date functions
- use null functions
- format dates

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Admin.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: String functions

You want to modify the ContactDirectory query so that you can sort it based on the contact's last name. For employees, this is not a problem as their last names and first names are stored in two separate columns, but for customers and suppliers, you need to start manipulating strings.

The main tasks for this exercise are as follows:

1. Add a column to an existing query that calculates the number of characters in the contact name's first name (by finding the first space).

2. Use that calculation (twice) to get the first name and the last name out of the query. Repeat for the suppliers table and modify the employees query so that it uses separate first and last names.
3. Sort the union query by last name.

Task 1: Add a column to an existing query

1. Open the script file called 'Ex01.sql' from the lab's starter folder. Save it as 'ImprovedDirectory.sql'
2. Just focusing on the Customers part of the query for now (in fact you might want to comment the rest out), add another column to the select list that uses the CharIndex function to find the first occurrence of a space character (' ') in the ContactName column.
3. Run that part of the query and confirm that that calculation is returning the right numbers (it will be one more than the length of the contact's first name).

Task 2: Calculate the first and last names based on the length of the first name.

1. Add a column that uses the Left function, passing it the contactname and (a copy of) the calculation from task 1 to retrieve the contact's first name only.
2. Alias the column as 'FirstName'.
3. Add another column that uses the Substring function, passing it the contactname, (a copy of) the calculation from task 1, and the value 50, to retrieve the contact's last name only. The value 50 just says "take all of the rest of the string". You might want to add 1 to the value returned by the CharIndex function to remove a leading space, but it's not going to stop things from working.
4. Alias the new column as 'LastName'.
5. Ensure that the Customers query only has four columns – CompanyName, FirstName, LastName and Phone and duplicate it for the Suppliers table.
6. Finally modify the Employees part of the query so that it selects first and last names as two separate columns.
7. Run the whole query and confirm that it returns 129 rows, with 4 columns.

Task 3: Sort the results on last name.

1. Add an order by clause that sorts the results of the union on last name.
2. Execute and confirm that Mr Acconti is top of list and Mr Yorres is last.

Exercise 2: Work with date functions

In an earlier lab, you may have created a query to get the count of orders for customers using outer joins. The rest of this lab will tweak the output from that lab.

The main tasks for this exercise are as follows:

1. Modify an existing query so that it only retrieves information about orders placed in a specified year.

Task 1: Create a query that filters by year

1. Open the script file called 'Ex02.sql' from the lab's start folder and save it as "ImprovedOrderAnalysis.sql".
2. Add a where clause that uses the Year function to select only those rows whose order date was placed in 1996.
3. Execute the query and verify that it returns 67 rows. You might be expecting it to return 91 rows because of the left join but the where clause is saying "if they've never placed orders, I'm not interested!"

Exercise 3: Work with ISNULL and CONVERT

The main tasks for this exercise are as follows:

1. Modify an existing query so that it outputs an informational message rather than the word "null".
2. Modify the same query so that it formats the dates.

Task 1: Create a query that outputs an informational message

1. Open the script file called 'Ex03.sql' from the lab's start folder and save it as "FormattedOrderAnalysis.sql".
2. Wrap the MIN aggregate call in an ISNULL function call that returns the text 'None placed' when the value is null.
3. Attempt to execute the query and note that it fails – you are currently trying to output text in a datetime column.

Task 2: Use the convert function to format a datetime column

1. Work with the existing query.
2. Wrap the MIN aggregate call in a Convert function call that converts the OrderDate column into a varchar(20) using format code 106 (or pick another style). So the outer-most function should be IsNull, the Convert, then Min right in the middle). It might take some fiddling to get the brackets all to match up.
3. Execute the query and bask in the beauty of your formatted query that returns 91 rows.

Completed Scripts

Exercise 1 Task 1:

```
SELECT
      CompanyName,
      CHARINDEX(' ', ContactName),
      ContactName,
      Phone
FROM
      dbo.Customers
```

Exercise 1 Task 2:

```
SELECT CompanyName,
       LEFT(ContactName, CHARINDEX(' ', ContactName))
         AS FirstName,
       SUBSTRING(ContactName,
         CHARINDEX(' ', ContactName) + 1, 50)
         AS LastName,
        Phone
FROM   dbo.Customers
UNION ALL
SELECT CompanyName,
       LEFT(ContactName, CHARINDEX(' ', ContactName))
         AS FirstName,
       SUBSTRING(ContactName,
         CHARINDEX(' ', ContactName) + 1, 50)
         AS LastName,
        Phone
FROM   dbo.Suppliers
UNION ALL
SELECT 'Northwind Traders', FirstName,
       LastName, Extension
FROM   dbo.Employees
```

Exercise 1 Task 3:

```
…
ORDER BY LastName
```

Exercise 2 Task 1:

```
SELECT  c.CompanyName, COUNT(o.OrderID) AS NumOrders,
        MIN(o.OrderDate) AS MinDate
FROM      dbo.Customers AS c
LEFT JOIN dbo.Orders AS o
ON        o.CustomerID = c.CustomerID
WHERE     YEAR(o.OrderDate) = 1996
GROUP BY  c.CompanyName
ORDER BY  NumOrders
```

Exercise 3 Task 1:

```
SELECT  c.CompanyName, COUNT(o.OrderID) AS NumOrders,
       ISNULL(MIN(o.OrderDate), 'None placed') AS MinDate
FROM      dbo.Customers AS c
LEFT JOIN dbo.Orders AS o
ON        o.CustomerID = c.CustomerID
WHERE     YEAR(o.OrderDate) = 1996
GROUP BY  c.CompanyName
ORDER BY  NumOrders
```

Exercise 3 Task 2:

```
SELECT  c.CompanyName, COUNT(o.OrderID) AS NumOrders,
       ISNULL(
         CONVERT(
           VARCHAR(20),
           MIN(o.OrderDate),
           106),
         'None placed') AS MinDate
FROM      dbo.Customers AS c
LEFT JOIN dbo.Orders AS o
ON        o.CustomerID = c.CustomerID
WHERE     YEAR(o.OrderDate) = 1996
GROUP BY  c.CompanyName
ORDER BY  NumOrders
```

Exercise 9 Table manipulation

Overview

In this lab, you will work to create, alter and drop a table, and insert, delete and update rows of data within the new table.

At the end of this chapter is a “completed scripts” section which tells you what to type at each stage if you are really stuck.

Objectives

At the end of this lab, you will be able to:

- Create a new table
- Alter the structure of a table
- Remove an existing table
- Add new rows to a table
- Remove rows from an existing table
- Update rows in a table

Setup: Launch SQL Server Management Studio (if necessary)

1. Launch the virtual machine.
2. Log on as Admin.
3. Launch SQL Server Management Studio.
4. Connect to the server.

Exercise 1: String functions

You want to create a new table within a new database. The database must be created first.

The main tasks for this exercise are as follows:

1. Add a new database to the SQL instance.
2. Add a table to the database using the specified design.
3. Insert data into the new table.

Task 1: Add a new database

1. Start a new script file. Save it as ‘Data Manipulation.sql’
2. Enter and execute the code below to add a new database to the SQL Instance.

```
CREATE DATABASE NewDB
GO

USE NewDB
GO
```

3. Using the object explorer, refresh the Databases folder to list the database.
4. Expand the NewDB database to show the objects types.
5. Expand the Tables folder. There are no user tables within the database yet. A table will be added in the next exercise.

Task 2: Add a table

1. In the query window, enter an execute the following code to create a table.

```
CREATE TABLE dbo.Payments(
   RowID INT IDENTITY(1,1) PRIMARY KEY,
   PaymentRef INT,
   CustomerRef VARCHAR(10),
   PaymentDT DATETIME,
   PaymentAmount MONEY
)
GO
```

2. Refresh the Tables folder within the NewDB. The dbo.payments table should show.
3. In the query window, enter and execute the following code to review the rows of data within the table.

```
SELECT * FROM dbo.Payments
GO
```

4. There are no rows within the data so the result should be an empty table.

Task 2: Add data to the new table

1. In the query window, enter an execute the following code to add rows.

```
INSERT INTO dbo.Payments VALUES     (1001,'CR001','2019/10/01
04:50:00',900.00)
INSERT INTO dbo.Payments VALUES     (1002,'CR002','2019/10/01
07:13:00',234.00)
INSERT INTO dbo.Payments VALUES     (1003,'CR003','2019/10/01
08:59:00',352.00)
INSERT INTO dbo.Payments VALUES     (1004,'CR001','2019/10/01
09:12:00',617.00)
INSERT INTO dbo.Payments VALUES     (1005,'CR004','2019/10/01
09:16:00',778.00)
INSERT INTO dbo.Payments VALUES     (0,'ERROR','1753/01/01
00:00:00',0.00)
GO
```

2. In the query window, enter and execute the following code to review the rows of data within the table.

```
SELECT * FROM dbo.Payments
GO
```

3. The query should return 6 rows.
4. In the query window, enter and execute the following code to summarize the rows of data by payment date.

```
SELECT PaymentDT , count(*) AS NumberRows,
  sum(PaymentAmount) AS TotalPaid

  FROM dbo.Payments

  GROUP BY PaymentDT

GO
```

5. The query does not summarize by date due to the column holding the time as well.
6. Keep the existing query for the next exercise.

Exercise 2: Adding columns

In an earlier lab, you created and added data to a table.

The main tasks for this exercise are as follows:

1. Alter the design of the table to add a column only holding the date of the payment.
2. Update the existing rows to hold the payment date.
3. Re-execute the queries to summarize by date only.

Task 1: Create a query that filters by year

1. In the query window, enter and execute the following code to add a new column to hold the PaymentDate.

```
ALTER TABLE dbo.Payments

  ADD PaymentDate DATE

GO
```

2. Open the dbo.Payments table using the object explorer. Open the Columns folder to review the table design now that the PaymentDate column has been added.
3. Enter and execute the following code to review the content of the table. Notice that the last column PaymentDate holds NULL for all rows.

```
SELECT * FROM dbo.Payments

GO
```

4. Enter and execute the following code to update all rows to fill the PaymentDate column calculated from the PatmentDT column

```
UPDATE dbo.Payments

   SET PaymentDate = CONVERT(DATE, PaymentDT)

GO
```

5. Re-execute the SELECT queries to review the PaymentDate column contents. All rows will have a date in the PaymentDate column.

```
SELECT * FROM dbo.Payments

GO

SELECT PaymentDate , count(*) AS NumberRows,
sum(PaymentAmount) AS TotalPaid

   FROM dbo.Payments

   GROUP BY PaymentDate

GO
```

Exercise 3: Cleaning the data

The main tasks for this exercise are as follows:

1. Modify the table content to remove all rows with PaymentRef as 0.
2. Modify the same query so that it formats the dates.

Task 1: Remove rows with errors

1. Enter and execute the following code to remove rows with PaymentRef set to 0.

```
DELETE FROM dbo.Payments

   WHERE PaymentRef = 0

GO
```

2. Enter and execute the following review the content of the Payments table.

```
SELECT * FROM dbo.Payments

GO

SELECT PaymentDate , count(*) AS NumberRows,
sum(PaymentAmount) AS TotalPaid

   FROM dbo.Payments

   GROUP BY PaymentDate

GO
```

Task 2: Removing the table

1. When the table is no longer required then it can be removed. Enter and execute the following code to remove the table.

```
DROP TABLE dbo.Payments

GO
```

2. Refresh the Tables folder in the object explorer to confirm that the table has been removed.