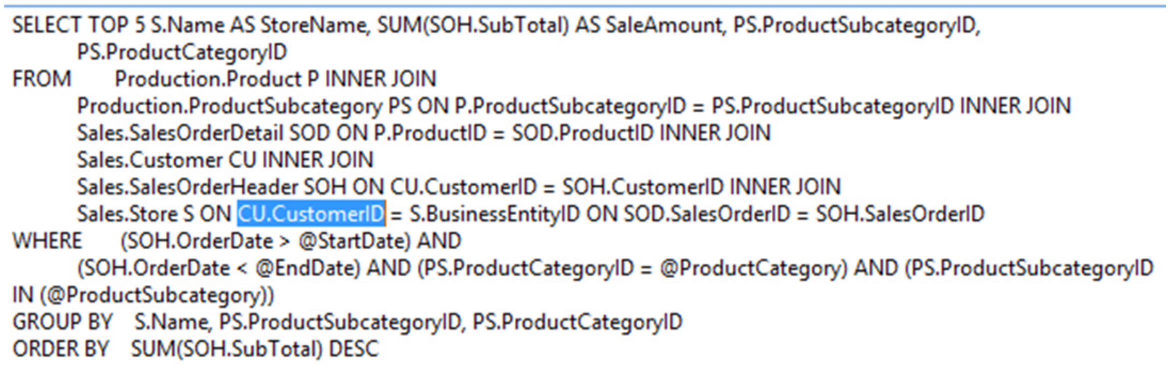
_____You need to change one of the columns that are used in the `SELECT` statement. There are two ways you can make the change:

Change technique 1

_____ Locate the `INNER JOIN Sales.Store S ON CU.CustomerID = S.BusinessEntityID` clause. In Figure 7, the column that you need to change is highlighted.

_____ Change `CU.CustomerID` to `CU.StoreID`. Be sure you change the correct clause, `CU.CustomerID` is also used in the previous `INNER JOIN`.

_____ After making the change, click the **OK** button.



```
SELECT TOP 5 S.Name AS StoreName, SUM(SOH.SubTotal) AS SaleAmount, PS.ProductSubcategoryID,
         PS.ProductCategoryID
FROM     Production.Product P INNER JOIN
         Production.ProductSubcategory PS ON P.ProductSubcategoryID = PS.ProductSubcategoryID INNER JOIN
         Sales.SalesOrderDetail SOD ON P.ProductID = SOD.ProductID INNER JOIN
         Sales.Customer CU INNER JOIN
         Sales.SalesOrderHeader SOH ON CU.CustomerID = SOH.CustomerID INNER JOIN
         Sales.Store S ON CU.CustomerID = S.BusinessEntityID ON SOD.SalesOrderID = SOH.SalesOrderID
WHERE    (SOH.OrderDate > @StartDate) AND
         (SOH.OrderDate < @EndDate) AND (PS.ProductCategoryID = @ProductCategory) AND (PS.ProductSubcategoryID
IN (@ProductSubcategory))
GROUP BY  S.Name, PS.ProductSubcategoryID, PS.ProductCategoryID
ORDER BY  SUM(SOH.SubTotal) DESC
```

ssrs3114

Figure 7: This shows the column name that must be changed.

Change technique 2

_____ Select all of the text (the `SELECT` statement) that is in the **Query Designer**.

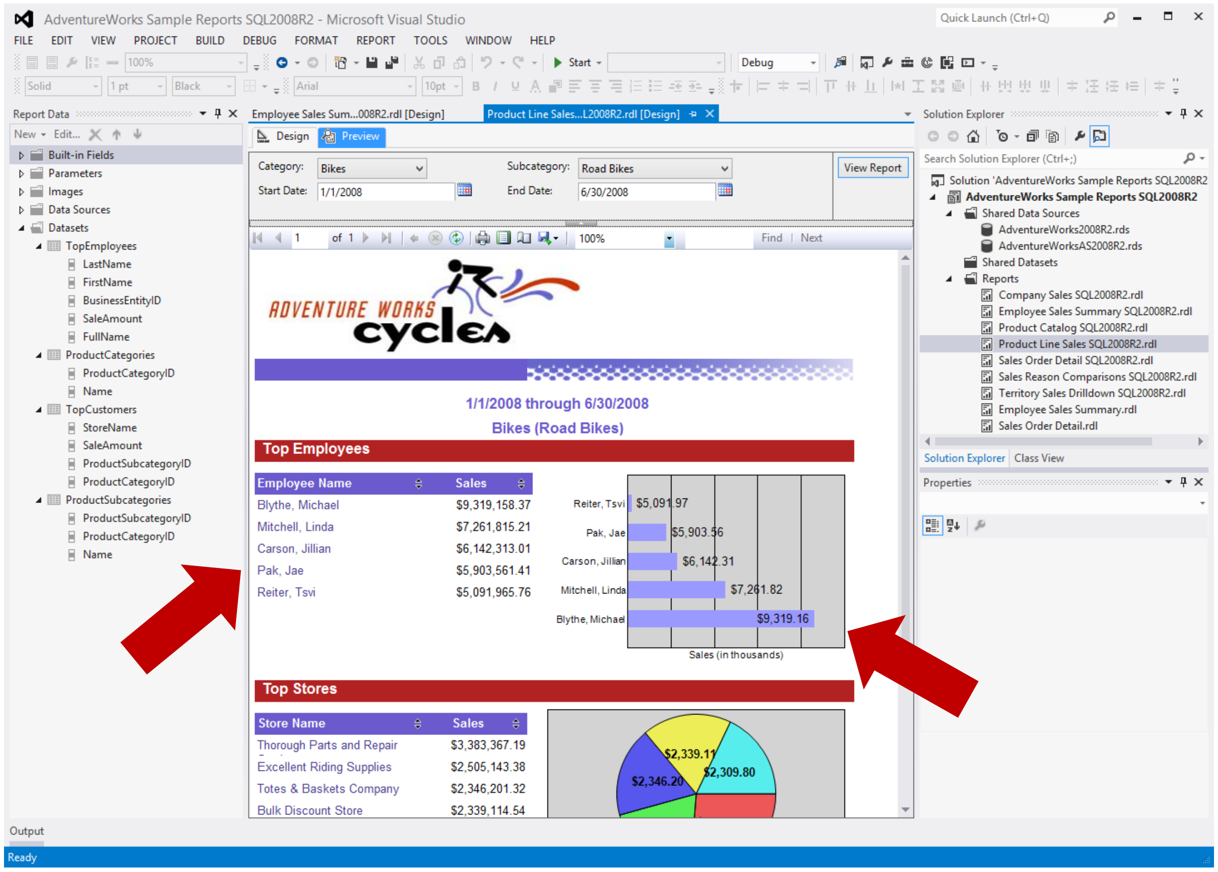
_____ Delete all of the text. The **Query Designer** should be empty.

_____ Open the `ProductLineSales_TopCustomers.sql.txt` file (provided with the course materials). **The correction has already been made in this file.**

_____ Select all of the text that is in the file, copy-and-paste all of the selected text into the **Query Designer**.

_____ After pasting the text, click the **OK** button.

______Click the **Preview** tab. You should see the **Product Line Sales SQL2008R2** report, as shown in Figure 8.


AdventureWorks Sample Reports SQL2008R2 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG FORMAT REPORT TOOLS WINDOW HELP
Start
Debug
Report Data
New Edit...
Built-in Fields
Parameters
Images
Data Sources
Datasets
TopEmployees
LastName
FirstName
BusinessEntityID
SaleAmount
FullName
ProductCategories
ProductCategoryID
Name
TopCustomers
StoreName
SaleAmount
ProductSubcategoryID
ProductCategoryID
ProductSubcategories
ProductSubcategoryID
ProductCategoryID
Name
Employee Sales Sum...008R2.rdl [Design]
Product Line Sales...L2008R2.rdl [Design]
Design
Preview
View Report
Find | Next
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'AdventureWorks Sample Reports SQL2008R2
AdventureWorks Sample Reports SQL2008R2
Shared Data Sources
AdventureWorks2008R2.rds
AdventureWorksAS2008R2.rds
Shared Datasets
Reports
Company Sales SQL2008R2.rdl
Employee Sales Summary SQL2008R2.rdl
Product Catalog SQL2008R2.rdl
Product Line Sales SQL2008R2.rdl
Sales Order Detail SQL2008R2.rdl
Sales Reason Comparisons SQL2008R2.rdl
Territory Sales Drilldown SQL2008R2.rdl
Employee Sales Summary.rdl
Sales Order Detail.rdl
Solution Explorer
Class View
Properties
ADVENTURE WORKS cycles
Sales (in thousands)
Output
Ready

Category: Bikes Subcategory: Road Bikes
Start Date: 1/1/2008 End Date: 6/30/2008

1 of 1 100%

1/1/2008 through 6/30/2008
Bikes (Road Bikes)

Top Employees

Employee Name	Sales
Blythe, Michael	$9,319,158.37
Mitchell, Linda	$7,261,815.21
Carson, Jillian	$6,142,313.01
Pak, Jae	$5,903,561.41
Reiter, Tsvi	$5,091,965.76

Reiter, Tsvi	$5,091.97
Pak, Jae	$5,903.56
Carson, Jillian	$6,142.31
Mitchell, Linda	$7,261.82
Blythe, Michael	$9,319.16

Top Stores

Store Name	Sales
Thorough Parts and Repair	$3,383,367.19
Excellent Riding Supplies	$2,505,143.38
Totes & Baskets Company	$2,346,201.32
Bulk Discount Store	$2,339,114.54

$2,339.11 $2,309.80 $2,346.20

ssrs3121

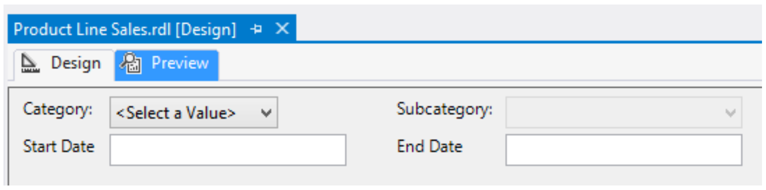
Figure 8: This shows the Product Line Sales report in Preview mode.

______You can click an employee name (pointed to by the arrow on the left) or one of the bars in the graph (pointed to by the arrow on the right). When you click either the name or the graph, the **Employee Sales Summary** report shown in Figure 9 is displayed.

3.2.6.5 Review the interaction of the Category and Subcategory parameters

As you can see, there is an interaction between the **Category** and **Subcategory** parameters. That is, when you select one the **Category** values, the values for the **Subcategory** are dynamically loaded.

To see how this works, first look at Figure 51. This shows what the report parameters section looks like in preview mode, if you did not assign a default value to the **Category** parameter. When there is no **Category** selection, the **Subcategory** drop-down list is disabled. That makes sense, since there is no way for SSRS to know which subcategory items are to be displayed, if there is no category selection.



Category: <Select a Value>
Subcategory:
Start Date
End Date

ssrs3205

Figure 51: If there is not a default value for the Category parameter, the Subcategory drop-down is initially disabled.

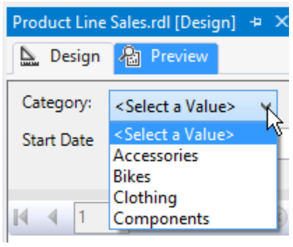
The **Category** list is created from the **ProductCategories** dataset. Here is the `SELECT` statement that fills that dataset (this `SELECT` is in the `ProductLineSales_ProductCategories.sql.txt` file):

```
SELECT DISTINCT ProductCategoryID,
                Name

FROM      Production.ProductCategory

ORDER BY  Name
```

As you can see, that `SELECT` does not have a `WHERE` clause. It simply gets all of the rows in the `ProductCategory` table and returns them in order by the `Name` column. Figure 52 shows the data that is returned from the `SELECT` statement. For this parameter, the `SELECT` statement runs when the report is loaded. The reason why the `SELECT` runs when the report is loaded is because there is nothing else needed to run the `SELECT`: it is not waiting for a substitution variable value for the `WHERE` clause.



Category: <Select a Value>
Start Date

<Select a Value>
Accessories
Bikes
Clothing
Components

ssrs3206

Figure 52: This shows the contents of the Category parameter drop-down list, filled from the ProductCategories dataset.

Now look at the SELECT statement that is used to retrieve the data for the **Subcategories** parameter. That parameter uses the **ProductSubcategories** dataset to get the list of items.

This SELECT statement is in the ProductLineSales_ProductSubcategories.sql.txt file.

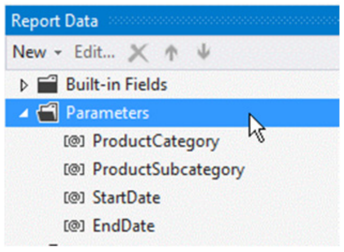
```
SELECT   ProductSubcategoryID,
         ProductCategoryID,
         Name

FROM     Production.ProductSubcategory

WHERE    ProductCategoryID = @ProductCategory
```

The SELECT statement uses a substitution variable named @ProductCategory on the WHERE clause, to limit the rows that are selected. Notice that the substitution variable name is the same as the first parameter name (as defined in the **Parameters** section of the **Report Data** panel), as shown in Figure 53. In SQL Server, a substitution variable starts with the @ character.

> **Side note:** you can only use substitution variables in a WHERE clause. If you want to dynamically substitute values in other parts of a SELECT statement (for example, in the FROM clause), you essentially need to use string concatenation to build the SQL statement, using dynamic SQL. That technique is not covered in this course.



ssrs3207

Figure 53: This shows the parameter names, as defined in the Parameters section of the Report Data panel.

When you select a value in the **Category** parameter drop-down list, the selected value becomes the value of the ProductCategory parameter. When you configured the ProductCategory parameter, you assigned the ProductCategoryID as the value to be assigned to the parameter (see Figure 38 on page 27). The ProductCategoryID is an integer value, not the text value that is displayed to the user.

As soon as the **Category** parameter value is selected, the **ProductSubcategories** dataset can be filled, since there is now a value that can be used for the substitution variable (see the SELECT statement above). The following WHERE clause:

```
WHERE    ProductCategoryID = @ProductCategory
```

limits the row selection to only those rows in the ProductSubcategory table that are in the ProductCategoryID that matches the value selected for the ProductCategory parameter. The SELECT statement runs, and the rows that are retrieved for the selected category are displayed in the **Subcategory** parameter drop-down list.

Notice also that the **Category** parameter is limited to a single selection, but the **Subcategory** parameter allows multiple selections. Later in this lab, you will see how the multiple selections are handled in SQL SELECT statements.