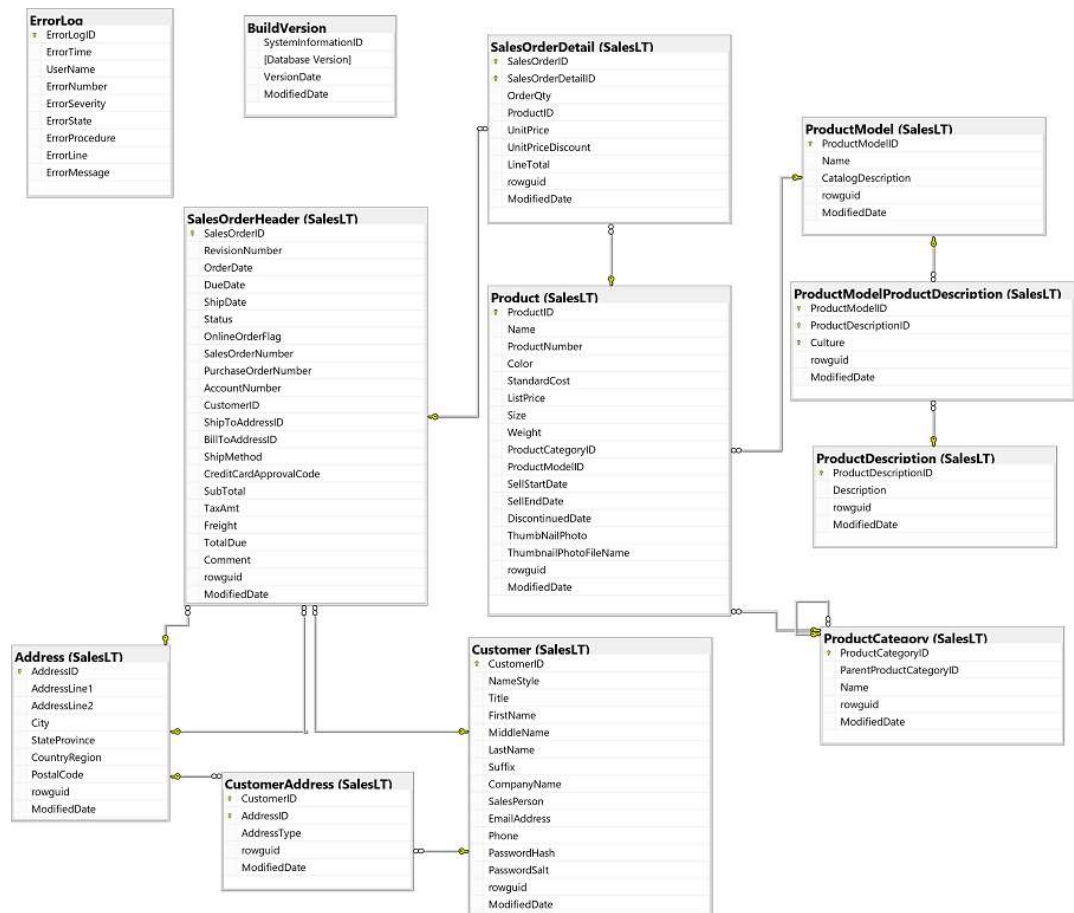


Sort and Filter Query Results

In this lab, you'll use the Transact-SQL **SELECT** statement to query and filter data in the **adventureworks** database. For your reference, the following diagram shows the tables in the database (you may need to resize the pane to see them clearly).



Note: If you're familiar with the standard **AdventureWorks** sample database, you may notice that in this lab we are using a simplified version that makes it easier to focus on learning Transact-SQL syntax.

Sort results using the ORDER BY clause

It's often useful to sort query results into a particular order.

1. Modify the existing query to return the **Name** and **ListPrice** of all products:


Code	Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product;</pre>	

2. Run the query and note that the results are presented in no particular order.
3. Modify the query to add an **ORDER BY** clause that sorts the results by **Name**, as shown here:

Code	Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY Name;</pre>	


4. Run the query and review the results. This time the products are listed in alphabetical order by **Name**.

5. Modify the query as shown below to sort the results by **ListPrice**.

Code	 Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice;</pre>	

6. Run the query and note that the results are listed in ascending order of **ListPrice**. By default, the **ORDER BY** clause applies an ascending sort order to the specified field.

7. Modify the query as shown below to sort the results into descending order of **ListPrice**.

Code	 Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC;</pre>	

8. Run the query and note that the results now show the most expensive items first.

9. Modify the query as shown below to sort the results into descending order of **ListPrice**, and then into ascending order of **Name**.


Code	 Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC, Name ASC;</pre>	

10. Run the query and review the results. Note that they are sorted into descending order of **ListPrice**, and each set of products with the same price is sorted in ascending order of **Name**.

Restrict results using TOP


Sometimes you only want to return a specific number of rows. For example, you might want to find the twenty most expensive products.

1. Modify the existing query to return the **Name** and **ListPrice** of all products:

Code	 Copy
<pre>SELECT TOP 20 Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC;</pre>	


2. Run the query and note that the results contain the first twenty products in descending order of **ListPrice**. Typically, you include an **ORDER BY** clause when using the **TOP** parameter; otherwise the query just returns the first specified number of rows in an arbitrary order.

3. Modify the query to add the **WITH TIES** parameter as shown here, and re-run it.

Code	 Copy
<pre>SELECT TOP 20 WITH TIES Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC;</pre>	

4. This time, there are 21 rows in the results, because there are multiple products that share the same price, one of which wasn't included when ties were ignored by the previous query.

5. Modify the query to add the **PERCENT** parameter as shown here, and re-run it.


Code	 Copy
<pre>SELECT TOP 20 PERCENT WITH TIES Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC;</pre>	

6. Note that this time the results contain the 20% most expensive products.


Retrieve pages of results with OFFSET and FETCH

User interfaces and reports often present large volumes of data as pages, you make them easier to navigate on a screen.

1. Modify the existing query to return product **Name** and **ListPrice** values:

Code	 Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY Name OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;</pre>	


2. Run the query and note the effect of the **OFFSET** and **FETCH** parameters of the **ORDER BY** clause. The results start at the 0 position (the beginning of the result set) and include only the next 10 rows, essentially defining the first page of results with 10 rows per page.
3. Modify the query as shown here, and run it to retrieve the next page of results.

Code	 Copy
<pre>SELECT Name, ListPrice FROM SalesLT.Product ORDER BY Name OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;</pre>	

Use the ALL and DISTINCT options

Often, multiple rows in a table may contain the same values for a given subset of fields. For example, a table of products might contain a **Color** field that identifies the color of a given product. It's not unreasonable to assume that there may be multiple products of the same color. Similarly, the table might contain a **Size** field; and again it's not unreasonable to assume that there may be multiple products of the same size; or even multiple products with the same combination of size and color.

1. Start Azure Data Studio, and create a new query (you can do this from the **File** menu or on the *welcome* page).
2. In the new **SQLQuery_...** pane, use the **Connect** button to connect the query to the **AdventureWorks** saved connection.
3. In the query editor, enter the following code:

Code	 Copy
<pre>SELECT Color FROM SalesLT.Product;</pre>	

4. Use the **Run** button to run the query, and after a few seconds, review the results, which includes the color of each product in the table.
5. Modify the query as follows, and re-run it.

	
--	---

Code Copy

```
SELECT ALL Color
FROM SalesLT.Product;
```

The results should be the same as before. The **ALL** parameter is the default behavior, and is applied implicitly to return a row for every record that meets the query criteria.

6. Modify the query to replace **ALL** with **DISTINCT**, as shown here:

Code Copy

```
SELECT DISTINCT Color
FROM SalesLT.Product;
```

7. Run the modified query and note that the results include one row for each unique **Color** value. This ability to remove duplicates from the results can often be useful - for example to retrieve values in order to populate a drop-down list of color options in a user interface.

8. Modify the query to add the **Size** field as shown here:

Code Copy

```
SELECT DISTINCT Color, Size
FROM SalesLT.Product;
```

9. Run the modified query and note that it returns each unique combination of color and size.

Filter results with the WHERE clause

Most queries for application development or reporting involve filtering the data to match specified criteria. You typically apply filtering criteria as a predicate in a **WHERE** clause of a query.

1. In Azure Data Studio, replace the existing query with the following code:

Code Copy

```
SELECT Name, Color, Size
FROM SalesLT.Product
WHERE ProductModelID = 6
ORDER BY Name;
```

[Challenge 1:
Retrieve data for
transportation
reports](#)

[Challenge 2:
Retrieve product
data](#)

[Challenge 1](#)

[Challenge 2](#)

2. Run the query and review the results, which contain the **Name**, **Color**, and **Size** for each product with a **ProductModelID** value of 6 (this is the ID for the *HL Road Frame* product model, of which there are multiple variants).

3. Replace the query with the following code, which uses the *not equal* (<>) operator, and run it.

Code Copy

```
SELECT Name, Color, Size
FROM SalesLT.Product
WHERE ProductModelID <> 6
ORDER BY Name;
```

4. Review the results, noting that they contain all products with a **ProductModelID** other than 6.

5. Replace the query with the following code, and run it.

Code Copy


```

SELECT Name, ListPrice
FROM SalesLT.Product
WHERE ListPrice > 1000.00
ORDER BY ListPrice;

```

6. Review the results, noting that they contain all products with a **ListPrice** greater than 1000.00.


7. Modify the query as follows, and run it.

Code	 Copy
<pre> SELECT Name, ListPrice FROM SalesLT.Product WHERE Name LIKE 'HL Road Frame %'; </pre>	

8. Review the results, noting that the **LIKE** operator enables you to match string patterns. The % character in the predicate is a wildcard for one or more characters, so the query returns all rows where the **Name** is *HL Road Frame* followed by any string.

The **LIKE** operator can be used to define complex pattern matches based on regular expressions, which can be useful when dealing with string data that follows a predictable pattern.


9. Modify the query as follows, and run it.

Code	 Copy
<pre> SELECT Name, ListPrice FROM SalesLT.Product WHERE ProductNumber LIKE 'FR-_[0-9][0-9]_[0-9][0-9]'; </pre>	

10. Review the results. This time the results include products with a **ProductNumber** that matches patterns similar to *FR-xNNx-NN* (in which x is a letter and N is a numeral).


Tip: To learn more about pattern-matching with the **LIKE** operator, see the [Transact-SQL documentation](#).

11. Modify the query as follows, and run it.

Code	 Copy
<pre> SELECT Name, ListPrice FROM SalesLT.Product WHERE SellEndDate IS NOT NULL; </pre>	

12. Note that to filter based on *NULL* values you must use **IS NULL** (or **IS NOT NULL**) you cannot compare a *NULL* value using the = operator.

13. Now try the following query, which uses the **BETWEEN** operator to define a filter based on values within a defined range.

Code	 Copy
<pre> SELECT Name FROM SalesLT.Product WHERE SellEndDate BETWEEN '2006/1/1' AND '2006/12/31'; </pre>	

14. Review the results, which contain products that the company stopped selling in 2006.

15. Run the following query, which retrieves products with a **ProductCategoryID** value that is in a specified list.

	
--	---

Code Copy

```
SELECT ProductCategoryID, Name, ListPrice
FROM SalesLT.Product
WHERE ProductCategoryID IN (5,6,7);
```

16. Now try the following query, which uses the **AND** operator to combine two criteria.

Code Copy

```
SELECT ProductCategoryID, Name, ListPrice, SellEndDate
FROM SalesLT.Product
WHERE ProductCategoryID IN (5,6,7) AND SellEndDate IS NULL;
```


17. Try the following query, which filters the results to include rows that match one (or both) of two criteria.

Code Copy

```
SELECT Name, ProductCategoryID, ProductNumber
FROM SalesLT.Product
WHERE ProductNumber LIKE 'FR%' OR ProductCategoryID IN (5,6,7);
```

Challenges

Now that you've seen some examples of filtering and sorting data, it's your chance to put what you've learned into practice.

 **Tip:** Try to determine the appropriate queries for yourself. If you get stuck, suggested answers are provided at the end of this lab.

Challenge 1: Retrieve data for transportation reports

The logistics manager at Adventure Works has asked you to generate some reports containing details of the company's customers to help to reduce transportation costs.

1. Retrieve a list of cities
 - Initially, you need to produce a list of all of your customers' locations. Write a Transact-SQL query that queries the **SalesLT.Address** table and retrieves the values for **City** and **StateProvince**, removing duplicates and sorted in ascending order of city.
2. Retrieve the heaviest products
 - Transportation costs are increasing and you need to identify the heaviest products. Retrieve the names of the top ten percent of products by weight.

Challenge 2: Retrieve product data

The Production Manager at Adventure Works would like you to create some reports listing details of the products that you sell.

1. Retrieve product details for product model 1
 - Initially, you need to find the names, colors, and sizes of the products with a product model ID 1.
2. Filter products by color and size
 - Retrieve the product number and name of the products that have a color of *black*, *red*, or *white* and a size of *S* or *M*.
3. Filter products by product number


- Retrieve the product number, name, and list price of products whose product number begins *BK*-
4. Retrieve specific products by product number
- Modify your previous query to retrieve the product number, name, and list price of products whose product number begins *BK*- followed by any character other than *R*, and ends with a - followed by any two numerals.

Challenge Solutions


This section contains suggested solutions for the challenge queries.

Challenge 1

1. Retrieve a list of cities:


Code	 Copy
<pre>SELECT DISTINCT City, StateProvince FROM SalesLT.Address ORDER BY City</pre>	

2. Retrieve the heaviest products:


Code	 Copy
<pre>SELECT TOP 10 PERCENT WITH TIES Name FROM SalesLT.Product ORDER BY Weight DESC;</pre>	

Challenge 2


1. Retrieve product details for product model 1:

Code	 Copy
<pre>SELECT Name, Color, Size FROM SalesLT.Product WHERE ProductModelID = 1;</pre>	

2. Filter products by color and size:

Code	 Copy
<pre>SELECT ProductNumber, Name FROM SalesLT.Product WHERE Color IN ('Black', 'Red', 'White') AND Size IN ('S', 'M');</pre>	

3. Filter products by product number:

Code	 Copy
<pre>SELECT ProductNumber, Name, ListPrice FROM SalesLT.Product WHERE ProductNumber LIKE 'BK-%';</pre>	

4. Retrieve specific products by product number:

Code	 Copy
------	--

```
SELECT ProductNumber, Name, ListPrice
FROM SalesLT.Product
WHERE ProductNumber LIKE 'BK-[^R]%-[0-9][0-9]';
```