

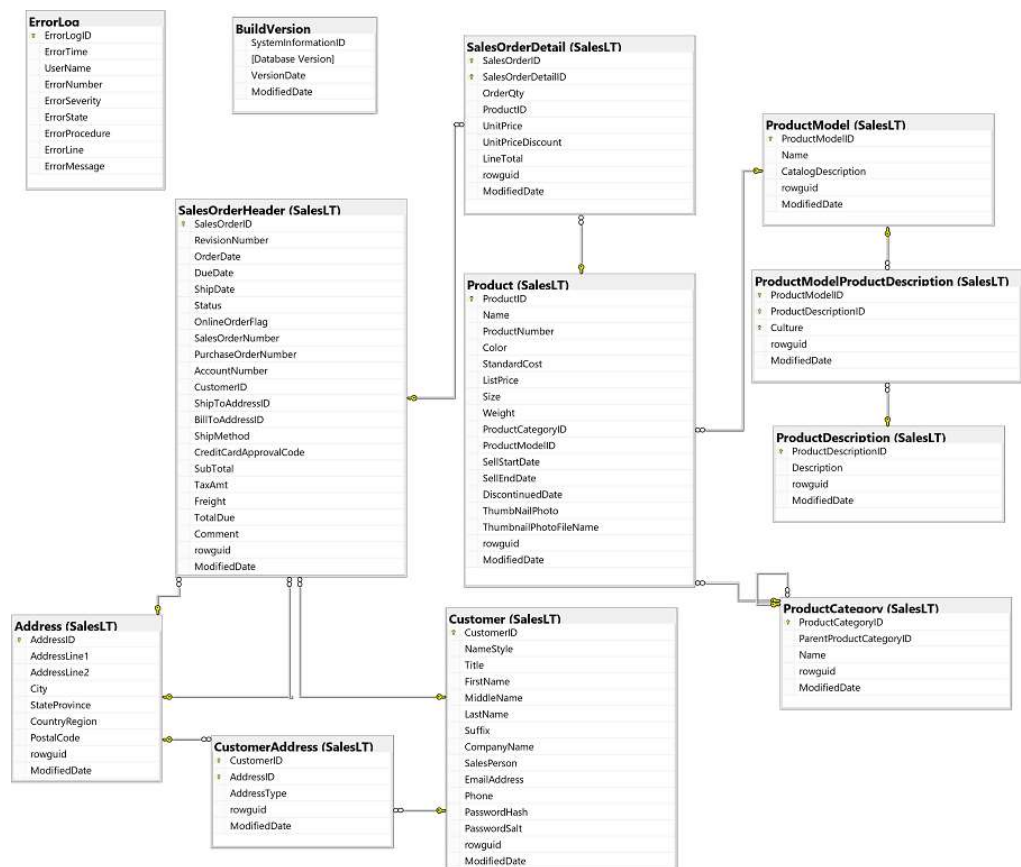
Get Started with Transact-SQL

In this lab, you will use some basic SELECT queries to retrieve data from the **AdventureWorks** database.

Explore the *AdventureWorks* database

We'll use the **AdventureWorks** database in this lab, so let's start by exploring it in Azure Data Studio.

1. Start Azure Data Studio, and in the **Connections** tab, select the **AdventureWorks** connection by clicking on the arrow just to the left of the name. This will connect to the SQL Server instance and show the objects in the **AdventureWorks** database.
2. Expand the **Tables** folder to see the tables that are defined in the database. Note that there are a few tables in the **dbo** schema, but most of the tables are defined in a schema named **SalesLT**.
3. Expand the **SalesLT.Product** table and then expand its **Columns** folder to see the columns in this table. Each column has a name, a data type, an indication of whether it can contain *null* values, and in some cases an indication that the column is used as a primary key (PK) or foreign key (FK).
4. Right-click the **SalesLT.Product** table and use the **Select Top 1000** option to create and run a new query script that retrieves the first 1000 rows from the table.
5. Review the query results, which consist of 1000 rows - each row representing a product that is sold by the fictitious *Adventure Works Cycles* company.
6. Close the **SQLQuery_1** pane that contains the query and its results.
7. Explore the other tables in the database, which contain information about product details, customers, and sales orders. The tables are related through primary and foreign keys, as shown here (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.


Use SELECT queries to retrieve data

Now that you've had a chance to explore the **AdventureWorks** database, it's time to dig a little deeper into the product data it contains by querying the **Product** table.


1. In Azure Data Studio, create a new query (you can do this from the **File** menu or on the *welcome* page).
2. In the new **SQLQuery_...** pane, ensure that the **AdventureWorks** database is selected at the top of the query pane. If not, 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 * FROM SalesLT.Product;</pre>	


4. Use the **Run** button to run the query, and after a few seconds, review the results, which includes all columns for all products.
5. In the query editor, modify the query as follows:

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


6. Use the **Run** button to re-run the query, and after a few seconds, review the results, which this time include only the **Name**, **StandardCost**, and **ListPrice** columns for all products.
7. Modify the query as shown below to include an expression that results in a calculated column, and then re-run the query:

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

8. Note that the results this time include the **Name** column and an unnamed column containing the result of subtracting the **StandardCost** from the **ListPrice**.
9. Modify the query as shown below to assign names to the columns in the results, and then re-run the query.

Code	 Copy
<pre>SELECT Name AS ProductName, ListPrice - StandardCost AS Markup FROM SalesLT.Product;</pre>	


10. Note that the results now include columns named **ProductName** and **Markup**. The **AS** keyword has been used to assign an *alias* for each column in the results.
11. Replace the existing query with the following code, which also includes an expression that produces a calculated column in the results:

Code	 Copy
<pre>SELECT ProductNumber, Color, Size, Color + ', ' + Size AS ProductDetails FROM SalesLT.Product;</pre>	

12. Run the query, and note that the **+** operator in the calculated **ProductDetails** column is used to *concatenate* the **Color** and **Size** column values (with a literal comma between them). The behavior of this operator is determined by the data types of the columns - had they been numeric values, the **+** operator would have *added* them. Note also that some results are *NULL* - we'll explore NULL values later in this lab.


As you just saw, columns in a table are defined as specific data types, which affects the operations you can perform on them.

1. Replace the existing query with the following code, and run it:

Code	 Copy
<pre>SELECT ProductID + ': ' + Name AS ProductName FROM SalesLT.Product;</pre>	


2. Note that this query returns an error. The **+** operator can be used to *concatenate* text-based values, or *add* numeric values; but in this case there's one numeric value (**ProductID**) and one text-based value (**Name**), so it's unclear how the operator should be applied.

3. Modify the query as follows, and re-run it:

Code	 Copy
<pre>SELECT CAST(ProductID AS varchar(5)) + ': ' + Name AS ProductName FROM SalesLT.Product;</pre>	


4. Note that the effect of the **CAST** function is to change the numeric **ProductID** column into a **varchar** (variable-length character data) value that can be concatenated with other text-based values.

5. Modify the query to replace the **CAST** function with a **CONVERT** function as shown below, and then re-run it:


Code	 Copy
<pre>SELECT CONVERT(varchar(5), ProductID) + ': ' + Name AS ProductName FROM SalesLT.Product;</pre>	

6. Note that the results of using **CONVERT** are the same as for **CAST**. The **CAST** function is an ANSI standard part of the SQL language that is available in most database systems, while **CONVERT** is a SQL Server specific function.

7. Another key difference between the two functions is that **CONVERT** includes an additional parameter that can be useful for formatting date and time values when converting them to text-based data. For example, replace the existing query with the following code and run it.


Code	 Copy
<pre>SELECT SellStartDate, CONVERT(nvarchar(30), SellStartDate) AS ConvertedDate, CONVERT(nvarchar(30), SellStartDate, 126) AS ISO8601FormatDate FROM SalesLT.Product;</pre>	

8. Replace the existing query with the following code, and run it.

Code	 Copy
<pre>SELECT Name, CAST(Size AS Integer) AS NumericSize FROM SalesLT.Product;</pre>	

9. Note that an error is returned because some **Size** values are not numeric (for example, some item sizes are indicated as *S*, *M*, or *L*).

10. Modify the query to use a **TRY_CAST** function, as shown here.

Code	 Copy
------	--

[Challenge 1:
Retrieve
customer data](#)

[Challenge 2:](#)

[Retrieve customer order data](#)

[Challenge 3: Retrieve customer contact details](#)

[Challenge 1](#)

[Challenge 2](#)

[Challenge 3:](#)


```
SELECT Name, TRY_CAST(Size AS Integer) AS NumericSize
FROM SalesLT.Product;
```

11. Run the query and note that the numeric **Size** values are converted successfully to integers, but that non-numeric sizes are returned as **NULL**.

Handle NULL values

We've seen some examples of queries that return **NULL** values. **NULL** is generally used to denote a value that is *unknown*. Note that this is not the same as saying the value is *none* - that would imply that you *know* that the value is zero or an empty string!


1. Modify the existing query as shown here:

Code	 Copy
<pre>SELECT Name, ISNULL(TRY_CAST(Size AS Integer),0) AS NumericSize FROM SalesLT.Product;</pre>	

2. Run the query and view the results. Note that the **ISNULL** function replaces **NULL** values with the specified value, so in this case, sizes that are not numeric (and therefore can't be converted to integers) are returned as **0**.


In this example, the **ISNULL** function is applied to the output of the inner **TRY_CAST** function, but you can also use it to deal with **NULL** values in the source table.

3. Replace the query with the following code to handle **NULL** values for **Color** and **Size** values in the source table:

Code	 Copy
<pre>SELECT ProductNumber, ISNULL(Color, '') + ', ' + ISNULL(Size, '') AS ProductDetails FROM SalesLT.Product;</pre>	


The **ISNULL** function replaces **NULL** values with a specified literal value. Sometimes, you may want to achieve the opposite result by replacing an explicit value with **NULL**. To do this, you can use the **NULLIF** function.

4. Try the following query, which replaces the **Color** value "Multi" to **NULL**.

Code	 Copy
<pre>SELECT Name, NULLIF(Color, 'Multi') AS SingleColor FROM SalesLT.Product;</pre>	

In some scenarios, you might want to compare multiple columns and find the first one that isn't **NULL**. For example, suppose you want to track the status of a product's availability based on the dates recorded when it was first offered for sale or removed from sale. A product that is currently available will have a **SellStartDate**, but the **SellEndDate** value will be **NULL**. When a product is no longer sold, a date is entered in its **SellEndDate** column. To find the first non-**NULL** column, you can use the **COALESCE** function.


5. Use the following query to find the first non-**NULL** date for product selling status.

Code	 Copy
<pre>SELECT Name, COALESCE(SellEndDate, SellStartDate) AS StatusLastUpdated FROM SalesLT.Product;</pre>	

The previous query returns the last date on which the product selling status was updated, but doesn't actually tell us the sales status itself. To determine that, we'll need to check the dates to see if the **SellEndDate** is **NULL**. To do this, you can use a **CASE** expression in the **SELECT** clause to check for **NULL SellEndDate** values. The **CASE** expression has two variants: a *simple CASE* what evaluates a specific column or value, or a *searched CASE* that evaluates one or more expressions.

In this example, or **CASE** expression must determine if the **SellEndDate** column is **NULL**. Typically, when you are trying to check the value of a column you can use the **=** operator; for example the predicate **SellEndDate = '01/01/2005'** returns **True** if the **SellEndDate** value is **01/01/2005**, and **False** otherwise. However, when dealing with **NULL** values, the default behavior may not be what you expect. Remember that **NULL** actually means *unknown*, so using the **=** operator to compare two unknown values always results in a value of **NULL** - semantically, it's impossible to know if one unknown value is the same as another. To check to see if a value is **NULL**, you must use the **IS NULL** predicate; and conversely to check that a value is not **NULL** you can use the **IS NOT NULL** predicate.

6. Run the following query, which includes *searched CASE* that uses an **IS NULL** expression to check for **NULL SellEndDate** values.

Code	 Copy
<pre>SELECT Name, CASE WHEN SellEndDate IS NULL THEN 'Currently for sale' ELSE 'No longer available' END AS SalesStatus FROM SalesLT.Product;</pre>	

The previous query used a *searched CASE* expression, which begins with a **CASE** keyword, and includes one or more **WHEN...THEN** expressions with the values and predicates to be checked. An **ELSE** expression provides a value to use if none of the **WHEN** conditions are matched, and the **END** keyword denotes the end of the **CASE** expression, which is aliased to a column name for the result using an **AS** expression.

In some queries, it's more appropriate to use a *simple CASE* expression that applies multiple **WHERE...THEN** predicates to the same value.


7. Run the following query to see an example of a *simple CASE* expression that produced different results depending on the **Size** column value.

Code	 Copy
<pre>SELECT Name, CASE Size WHEN 'S' THEN 'Small' WHEN 'M' THEN 'Medium' WHEN 'L' THEN 'Large' WHEN 'XL' THEN 'Extra-Large' ELSE ISNULL(Size, 'n/a') END AS ProductSize FROM SalesLT.Product;</pre>	

8. Review the query results and note that the **ProductSize** column contains the text-based description of the size for *S*, *M*, *L*, and *XL* sizes; the measurement value for numeric sizes, and *n/a* for any other sizes values.

Challenges

Now that you've seen some examples of **SELECT** statements that retrieve data from a table, it's time to try to compose some queries of your own.

 **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 customer data

Adventure Works Cycles sells directly to retailers, who then sell products to consumers. Each retailer that is an Adventure Works customer has provided a named contact for all communication from Adventure Works. The sales manager at Adventure Works has asked you to generate some reports containing details of the company's customers to support a direct sales campaign.

1. Retrieve customer details
 - Familiarize yourself with the **SalesLT.Customer** table by writing a Transact-SQL query that retrieves all columns for all customers.
2. Retrieve customer name data
 - Create a list of all customer contact names that includes the title, first name, middle name (if any), last name, and suffix (if any) of all customers.
3. Retrieve customer names and phone numbers
 - Each customer has an assigned salesperson. You must write a query to create a call sheet that lists:
 - The salesperson
 - A column named **CustomerName** that displays how the customer contact should be greeted (for example, *Mr Smith*)
 - The customer's phone number.

Challenge 2: Retrieve customer order data

As you continue to work with the Adventure Works customer data, you must create queries for reports that have been requested by the sales team.


1. Retrieve a list of customer companies
 - You have been asked to provide a list of all customer companies in the format *Customer ID : Company Name* - for example, *78: Preferred Bikes*.
2. Retrieve a list of sales order revisions
 - The **SalesLT.SalesOrderHeader** table contains records of sales orders. You have been asked to retrieve data for a report that shows:
 - The sales order number and revision number in the format *()* - for example *SO71774 (2)*.
 - The order date converted to ANSI standard *102* format (*yyyy.mm.dd* - for example *2015.01.31*).

Challenge 3: Retrieve customer contact details

Some records in the database include missing or unknown values that are returned as NULL. You must create some queries that handle these NULL values appropriately.

1. Retrieve customer contact names with middle names if known
 - You have been asked to write a query that returns a list of customer names. The list must consist of a single column in the format *first last* (for example *Keith Harris*) if the middle name is unknown, or *first middle last* (for example *Jane M. Gates*) if a middle name is known.
2. Retrieve primary contact details
 - Customers may provide Adventure Works with an email address, a phone number, or both. If an email address is available, then it should be used as the primary contact method; if not, then the phone number should be used. You must write a query that returns a list of customer IDs in one column, and a second column named **PrimaryContact** that contains the email address if known, and otherwise the phone number.


IMPORTANT: In the sample data provided, there are no customer records without an email address. Therefore, to verify that your query works as expected, run the following **UPDATE** statement to remove some existing email addresses before creating your query:

Code	 Copy
<pre>UPDATE SalesLT.Customer SET EmailAddress = NULL WHERE CustomerID % 7 = 1;</pre>	

3. Retrieve shipping status

- o You have been asked to create a query that returns a list of sales order IDs and order dates with a column named **ShippingStatus** that contains the text *Shipped* for orders with a known ship date, and *Awaiting Shipment* for orders with no ship date.

IMPORTANT: In the sample data provided, there are no sales order header records without a ship date. Therefore, to verify that your query works as expected, run the following UPDATE statement to remove some existing ship dates before creating your query.

Code	 Copy
<pre>UPDATE SalesLT.SalesOrderHeader SET ShipDate = NULL WHERE SalesOrderID > 71899;</pre>	

Challenge Solutions


This section contains suggested solutions for the challenge queries.

Challenge 1


1. Retrieve customer details:

Code	 Copy
<pre>SELECT * FROM SalesLT.Customer;</pre>	

2. Retrieve customer name data:


Code	 Copy
<pre>SELECT Title, FirstName, MiddleName, LastName, Suffix FROM SalesLT.Customer;</pre>	

3. Retrieve customer names and phone numbers:

Code	 Copy
<pre>SELECT Salesperson, Title + ' ' + LastName AS CustomerName, Phone FROM SalesLT.Customer;</pre>	

Challenge 2

1. Retrieve a list of customer companies:

Code	 Copy
------	--

```
SELECT CAST(CustomerID AS varchar) + ': ' + CompanyName AS CustomerCompany
FROM SalesLT.Customer;
```

2. Retrieve a list of sales order revisions:

Code

 Copy

```
SELECT SalesOrderNumber + ' (' + STR(RevisionNumber, 1) + ')' AS OrderRevision,
       CONVERT(nvarchar(30), OrderDate, 102) AS OrderDate
FROM SalesLT.SalesOrderHeader;
```

Challenge 3:

1. Retrieve customer contact names with middle names if known:

Code

 Copy

```
SELECT FirstName + ' ' + ISNULL(MiddleName + ' ', '') + LastName AS CustomerName
FROM SalesLT.Customer;
```

2. Retrieve primary contact details:

Code

 Copy

```
SELECT CustomerID, COALESCE(EmailAddress, Phone) AS PrimaryContact
FROM SalesLT.Customer;
```

3. Retrieve shipping status:

Code

 Copy

```
SELECT SalesOrderID, OrderDate,
       CASE
         WHEN ShipDate IS NULL THEN 'Awaiting Shipment'
         ELSE 'Shipped'
       END AS ShippingStatus
FROM SalesLT.SalesOrderHeader;
```