**SECTION 1: Use inner joins**

An inner join is used to find related data in two tables.

Task 1: You need to retrieve data about a product and its category from the **SalesLT.Product** and **SalesLT.ProductCategory** tables.

You can find the relevant product category record for a product based on its **ProductSubCategoryID** field which is a foreign-key in the product table that matches a primary key in the product category table.

1.      Start a new query editor. Open a new query file

2.      Connect the query to the **Sales LT** saved connection.

3.      In the query editor, enter the following code:

SELECT   SalesLT.Product.Name AS ProductName,

                 SalesLT.ProductCategory.Name AS Category

FROM SalesLT.Product

INNER JOIN SalesLT.ProductCategory

     ON SalesLT.Product.ProductCategoryID = SalesLT.ProductCategory.ProductCategoryID;

4.      Run the query, and after a few seconds, review the results.

5.      Because the query uses an **INNER** join, any products that do not have corresponding categories, and any categories that contain no products are omitted from the results.

6.      Modify the query as follows to remove the **INNER** keyword, and re-run it.

SELECT SalesLT.Product.Name AS ProductName, SalesLT.ProductCategory.Name AS Category

FROM SalesLT.Product

JOIN SalesLT.ProductCategory

  ON SalesLT.Product.ProductCategoryID = SalesLT.ProductCategory.ProductCategoryID;

The results should be the same as before. **INNER** joins are the default kind of join.

7.      Modify the query to assign aliases to the tables in the **JOIN** clause, as shown here:

SELECT p.Name AS ProductName, c.Name AS Category

FROM SalesLT.Product AS p

      JOIN SalesLT.ProductCategory AS c

      ON p.ProductSubCategoryID = c.ProductCategoryID;

8.      Run the modified query and confirm that it returns the same results as before. The use of table aliases can greatly simplify a query, particularly when multiple joins must be used.

9.      Replace the query with the following code, which retrieves sales order data from the **SalesLT.SalesOrderHeader**, **SalesLT.SalesOrderDetail**, and **SalesLT.Product** tables:

SELECT

                         oh.OrderDate,

                         oh.SalesOrderNumber,

                         p.Name AS ProductName,

                         od.OrderQty,

                         od.UnitPrice,

                         od.LineTotal

FROM SalesLT.SalesOrderHeader AS oh

                         JOIN SalesLT.SalesOrderDetail AS od

                             ON od.SalesOrderID = oh.SalesOrderID

JOIN SalesLT.Product AS p

    ON od.ProductID = p.ProductID

        ORDER BY oh.OrderDate, oh.SalesOrderID, od.SalesOrderDetailID;

10.  Run the modified query and note that it returns data from all three tables.

**SECTION 2: Use outer joins**

An outer join is used to retrieve all rows from one table, and any corresponding rows from a related table. In cases where a row in the outer table has no corresponding rows in the related table, *NULL* values are returned for the related table fields. For example, suppose you want to retrieve a list of all customers and any orders they have placed, including customers who have registered but never placed an order.

1.      Create a new query with the following code:

SELECT

        c.AccountNumber,

        c.CustomerType,

        oh.SalesOrderNumber

FROM SalesLT.Customer AS c

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

       ON c.CustomerID = oh.CustomerID

                           ORDER BY c.CustomerID;

2.      Run the query and note that the results contain data for every customer. If a customer has placed an order, the order number is shown. Customers who have registered but not placed an order are shown with a *NULL* order number.

Note the use of the **LEFT** keyword. This identifies which of the tables in the join is the *outer* table (the one from which all rows should be preserved). In this case, the join is between the **Customer** and **SalesOrderHeader** tables, so a **LEFT** join designates **Customer** as the outer table.

3.      Replace the query with the following one, which uses outer joins to retrieve data from three tables.

SELECT p.Name As ProductName, oh.SalesOrderNumber

FROM SalesLT.Product AS p

LEFT JOIN SalesLT.SalesOrderDetail AS od

    ON p.ProductID = od.ProductID

LEFT JOIN SalesLT.SalesOrderHeader AS oh

    ON od.SalesOrderID = oh.SalesOrderID

ORDER BY p.ProductID;

4.      Run the query and review the results, which include product names, categories, and sales order numbers.

**SECTION 3: Use cross joins (Optional)**

A *cross* join matches all possible combinations of rows from the tables being joined. In practice, it's rarely used; but there are some specialized cases where it is useful.

1.      Replace the existing query with the following code:

SELECT p.Name, c.TerritoryID, c.CustomerType

FROM SalesLT.Product AS p

CROSS JOIN SalesLT.Customer AS c;

2.      Run the query and note that the results contain a row for every product and customer combination (which might be used to create a mailing campaign in which an indivdual advertisement for each product is emailed to each customer - a strategy that may not endear the company to its customers!).

**Challenges**

Now that you've seen some examples of joins, it's your turn to try retrieving data from multiple tables for yourself.

**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: Generate invoice reports**

Adventure Works Cycles sells directly to retailers, who must be invoiced for their orders. You have been tasked with writing a query to generate a list of invoices to be sent to customers.

1. Retrieve customer orders
   * As an initial step towards generating the invoice report, write a query that returns the Account number from the **SalesLT.Customer** table, and the sales order ID and total due from the **SalesLT.SalesOrderHeader** table.

1. Retrieve customer orders with addresses
   * Extend your customer orders query to include the Address for each customer, including the address (combine Address Line 1 and 2), city, and postal code.
   * **Tip**: Note that each customer can have multiple addressees in the **SalesLT.Address** table, so the database developer has created the **SalesLT.CustomerAddress** table to enable a many-to-many relationship between customers and addresses. Your query will need to include both of these tables

**Challenge 2: Retrieve customer data**

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

1. Retrieve a list of customers with no address
   * A sales employee has noticed that Adventure Works does not have address information for all customers. You must write a query that returns a list of customer IDs, Account number, Territory ID and types of customers for customers with no address stored in the database.
2. Retrieve a list of all customers and their sales orders
   * The sales manager wants a list of all customer companies account number starting with AW and the Purchase order number, showing the sales order ID. Customers who have not placed any orders should be excluded where Sales order ID and total due are NULL.
   * The sales manager wants a column TaxFreightAmt that sums the freight and tax amount and total due for each order they have placed.
   * Finally display the Order Date in the “yyyy – mm - dd” format in the order of most recent order

**Challenge 3: Create a product catalog**

The marketing team has asked you to retrieve data for a new product catalog.

1. Retrieve product information by category
   * The product catalog will list products by parent category and subcategory, so you must write a query that retrieves the parent category name, subcategory name, and product name fields for the catalog.