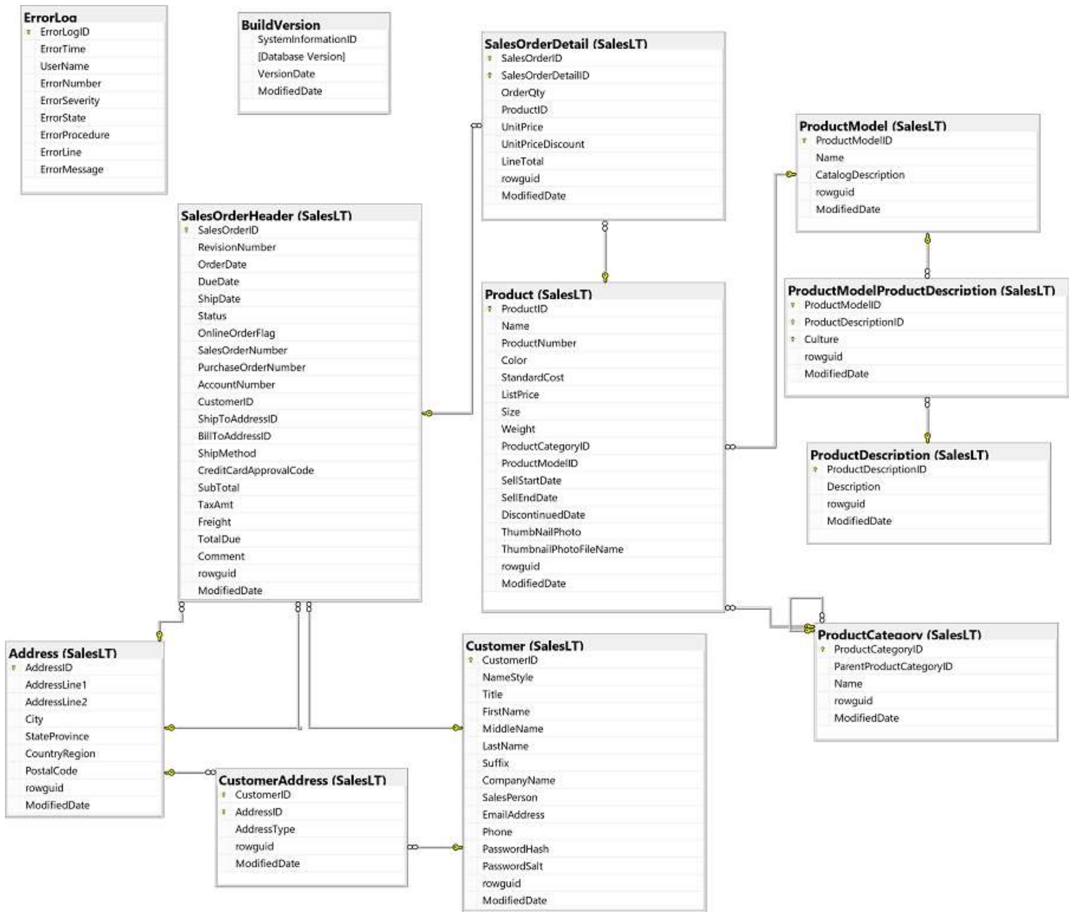
Use Subqueries

In this lab, you'll use subqueries to retrieve data from tables 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.

Use simple subqueries

A subquery is a query that is nested within another query. The subquery is often referred to as the *inner* query, and the query within which it is nested is referred to as the *outer* query.

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

```
SELECT MAX(UnitPrice)
FROM SalesLT.SalesOrderDetail;
```

4. Use the ▸ **Run** button to run the query, and and after a few seconds, review the results, which consists of the maximum **UnitPrice** in the **SalesLT.SalesOrderDetail** (the highest price for which any individual product has been sold).

Challenge 2

5. Modify the query as follows to use the query you just ran as a subquery in an outer query that retrieves products with a **ListPrice** higher than the maximum selling price.

Code Copy

```
SELECT Name, ListPrice
FROM SalesLT.Product
WHERE ListPrice >
    (SELECT MAX(UnitPrice)
     FROM SalesLT.SalesOrderDetail);
```

6. Run the query and review the results, which include all products that have a **listPrice** that is higher than the maximum price for which any product has been sold.

7. Replace the existing query with the following code:

Code Copy

```
SELECT DISTINCT ProductID
FROM SalesLT.SalesOrderDetail
WHERE OrderQty >= 20;
```

8. Run the query and note that it returns the **ProductID** for each product that has been ordered in quantities of 20 or more.

9. Modify the query as follows to use it in a subquery that finds the names of the products that have been ordered in quantities of 20 or more.

Code Copy

```
SELECT Name FROM SalesLT.Product
WHERE ProductID IN
    (SELECT DISTINCT ProductID
     FROM SalesLT.SalesOrderDetail
     WHERE OrderQty >= 20);
```

10. Run the query and note that it returns the product names.

11. Replace the query with the following code:

Code Copy

```
SELECT DISTINCT Name
FROM SalesLT.Product AS p
JOIN SalesLT.SalesOrderDetail AS o
    ON p.ProductID = o.ProductID
WHERE OrderQty >= 20;
```

12. Run the query and note that it returns the same results. Often you can achieve the same outcome with a subquery or a join, and often a subquery approach can be more easily interpreted by a developer looking at the code than a complex join query because the operation can be broken down into discrete components. In most cases, the performance of equivalent join or subquery operations is similar, but in some cases where existence checks need to be performed, joins perform better.

Use correlated subqueries

So far, the subqueries we've used have been independent of the outer query. In some cases, you might need to use an inner subquery that references a value in the outer query. Conceptually, the inner query runs once for each row returned by the outer query (which is why correlated subqueries are sometimes referred to as *repeating subqueries*).

1. Replace the existing query with the following code:

Code Copy

```
SELECT od.SalesOrderID, od.ProductID, od.OrderQty
FROM SalesLT.SalesOrderDetail AS od
ORDER BY od.ProductID;
```

2. Run the query and note that the results contain the order ID, product ID, and quantity for each sale of a product.

3. Modify the query as follows to filter it using a subquery in the **WHERE** clause that retrieves the maximum purchased quantity for each product retrieved by the outer query. Note that the inner query references a table alias that is declared in the outer query.

Code Copy

```
SELECT od.SalesOrderID, od.ProductID, od.OrderQty
FROM SalesLT.SalesOrderDetail AS od
WHERE od.OrderQty =
    (SELECT MAX(OrderQty)
     FROM SalesLT.SalesOrderDetail AS d
     WHERE od.ProductID = d.ProductID)
ORDER BY od.ProductID;
```

4. Run the query and review the results, which should only contain product order records for which the quantity ordered is the maximum ordered for that product.

5. Replace the query with the following code:

Code Copy

```
SELECT o.SalesOrderID, o.OrderDate, o.CustomerID
FROM SalesLT.SalesOrderHeader AS o
ORDER BY o.SalesOrderID;
```

6. Run the query and note that it returns the order ID, order date, and customer ID for each order that has been placed.

7. Modify the query as follows to retrieve the company name for each customer using a correlated subquery in the **SELECT** clause.

Code Copy

```
SELECT o.SalesOrderID, o.OrderDate,
      (SELECT CompanyName
      FROM SalesLT.Customer AS c
      WHERE c.CustomerID = o.CustomerID) AS CompanyName
FROM SalesLT.SalesOrderHeader AS o
ORDER BY o.SalesOrderID;
```

8. Run the query, and verify that the company name is returned for each customer found by the outer query.

Challenges

Now it's your opportunity to try using subqueries to retrieve data.

> **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 product price information

Adventure Works products each have a standard cost price that indicates the cost of manufacturing the product, and a list price that indicates the recommended selling price for the product. This data is stored in the **SalesLT.Product** table. Whenever a product is ordered, the actual unit price at which it was sold is also recorded in the **SalesLT.SalesOrderDetail** table. You must use subqueries to compare the cost and list prices for each product with the unit prices charged in each sale.

1. Retrieve products whose list price is higher than the average unit price.
    - Retrieve the product ID, name, and list price for each product where the list price is higher than the average unit price for all products that have been sold.
    - **Tip**: Use the **AVG** function to retrieve an average value.
2. Retrieve Products with a list price of 100 or more that have been sold for less than 100.
    - Retrieve the product ID, name, and list price for each product where the list price is 100 or more, and the product has been sold for less than 100.

Challenge 2: Analyze profitability

The standard cost of a product and the unit price at which it is sold determine its profitability. You must use correlated subqueries to compare the cost and average selling price for each product.

1. Retrieve the cost, list price, and average selling price for each product
    - Retrieve the product ID, name, cost, and list price for each product along with the average unit price for which that product has been sold.
2. Retrieve products that have an average selling price that is lower than the cost.
    - Filter your previous query to include only products where the cost price is higher than the average selling price.

Challenge Solutions

This section contains suggested solutions for the challenge queries.

Challenge 1

1. Retrieve products whose list price is higher than the average unit price:

Code Copy

```
SELECT ProductID, Name, ListPrice
FROM SalesLT.Product
WHERE ListPrice >
    (SELECT AVG(UnitPrice)
     FROM SalesLT.SalesOrderDetail)
ORDER BY ProductID;
```

2. Retrieve Products with a list price of 100 or more that have been sold for less than 100:

Code Copy

```
SELECT ProductID, Name, ListPrice
FROM SalesLT.Product
WHERE ProductID IN
    (SELECT ProductID
     FROM SalesLT.SalesOrderDetail
     WHERE UnitPrice < 100.00)
AND ListPrice >= 100.00
ORDER BY ProductID;
```

Challenge 2

1. Retrieve the cost, list price, and average selling price for each product:

Code Copy

```
SELECT p.ProductID, p.Name, p.StandardCost, p.ListPrice,
    (SELECT AVG(o.UnitPrice)
     FROM SalesLT.SalesOrderDetail AS o
     WHERE p.ProductID = o.ProductID) AS AvgSellingPrice
FROM SalesLT.Product AS p
ORDER BY p.ProductID;
```

2. Retrieve products that have an average selling price that is lower than the cost:

Code Copy

```
SELECT p.ProductID, p.Name, p.StandardCost, p.ListPrice,
    (SELECT AVG(o.UnitPrice)
    FROM SalesLT.SalesOrderDetail AS o
    WHERE p.ProductID = o.ProductID) AS AvgSellingPrice
FROM SalesLT.Product AS p
WHERE StandardCost >
    (SELECT AVG(od.UnitPrice)
     FROM SalesLT.SalesOrderDetail AS od
     WHERE p.ProductID = od.ProductID)
ORDER BY p.ProductID;
```