Key Points

* Transact-SQL is the language used to query data in Microsoft SQL Server and Azure SQL Database.
* Data is stored in tables, which may be related to one another through common key fields.
* Objects in a database are organized into schemas.
* The fully qualified naming syntax for an object is *server\_name*.*database\_name*.*schema\_name*.*object\_name*, but in most cases you can abbreviate this to *schema\_name.object\_name.*

## Key Points – SELECT Statement

* Use the SELECT statement to retrieve a rowset of data from tables and views in a database.
* SELECT statements are written with the following clauses: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY. However, the query engine processes the clauses in this order: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY.
* In the SELECT clause, you can use \* to return all columns, but generally you should specify explicit columns.
* You can specify expressions in the SELECT clause to return the results of calculations.
* You can use the AS keyword to specify aliases for columns in the rowset returned by the SELECT statement.

## Key Points – DATA Types and conversions

* Transact-SQL supports a wide range of data types, which can be broadly categorized as **exact numeric**,**approximate numeric**, **character**, **date/time**, **binary**, and **other** (which includes specialized data types for handling data such as XML and spatial data).
* Some data types are compatible, and values can be implicitly converted between them. Conversion between other data types requires the use of explicit conversion functions.

## In This Section

[CAST and CONVERT (Transact-SQL)](https://msdn.microsoft.com/en-us/library/ms187928.aspx)

[PARSE (Transact-SQL)](https://msdn.microsoft.com/en-us/library/hh213316.aspx)

[TRY\_CAST (Transact-SQL)](https://msdn.microsoft.com/en-us/library/hh974669.aspx)

[TRY\_CONVERT (Transact-SQL)](https://msdn.microsoft.com/en-us/library/hh230993.aspx)

[TRY\_PARSE (Transact-SQL)](https://msdn.microsoft.com/en-us/library/hh213126.aspx)

## Key Points – NULL Values

* NULL is used to indicate an unknown or missing value. NULL is **not** equivalent to zero or an empty string.
* Arithmetic or string concatenation operations involving one or more NULL operands return NULL. For example, 12 + NULL = NULL.
* If you need to compare a value to NULL, use the **IS** operator instead of the **=** operator.
* The **ISNULL** function returns a specified alternative value for NULL columns and variables.
* The **NULLIF** function returns NULL when a column or variable contains a specified value.
* The **COALESCE** function returns the first non-NULL value in a specified list of columns or variables).

Demo:

**Select Statement:**

SELECT Name, StandardCost, ListPrice

FROM SalesLT.Product;

SELECT Name, ListPrice - StandardCost

FROM SalesLT.Product;

SELECT Name, ListPrice - StandardCost AS Markup

FROM SalesLT.Product;

SELECT ProductNumber, Color, Size, Color + ', ' + Size AS ProductDetails

FROM SalesLT.Product;

SELECT ProductID + ': ' + Name

FROM SalesLT.Product;

**Converting Data types:**

SELECT CAST(ProductID AS varchar(5)) + ': ' + Name AS ProductName

FROM SalesLT.Product;

SELECT CONVERT(varchar(5), ProductID) + ': ' + Name AS ProductName

FROM SalesLT.Product;

SELECT SellStartDate,

CONVERT(nvarchar(30), SellStartDate) AS ConvertedDate,

CONVERT(nvarchar(30), SellStartDate, 126) AS ISO8601FormatDate

FROM SalesLT.Product;

SELECT Name, CAST (Size AS Integer) AS NumericSize

FROM SalesLT.Product; --(note error - some sizes are incompatible)

SELECT Name, TRY\_CAST (Size AS Integer) AS NumericSize

FROM SalesLT.Product; --(note incompatible sizes are returned as NULL)

**Null and Expressions:**

SELECT Name, ISNULL(TRY\_CAST(Size AS Integer),0) AS NumericSize

FROM SalesLT.Product;

SELECT ProductNumber, ISNULL(Color, '') + ', ' + ISNULL(Size, '') AS ProductDetails

FROM SalesLT.Product;

SELECT Name, NULLIF(Color, 'Multi') AS SingleColor

FROM SalesLT.Product;

SELECT Name, COALESCE(DiscontinuedDate, SellEndDate, SellStartDate) AS FirstNonNullDate

FROM SalesLT.Product;

--Searched case

SELECT Name,

CASE

WHEN SellEndDate IS NULL THEN 'On Sale'

ELSE 'Discontinued'

END AS SalesStatus

FROM SalesLT.Product;

--Simple case

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;

Lab 01

SELECT, CAST, CONVERT, ISNULL, UPDATE, COALESCE

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 Customer table by writing a Transact-SQL query that retrieves all columns for all customers.

-- Display all columns for all customers

SELECT \* FROM SalesLT.Customer;

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.

-- Display customer name fields

SELECT Title, FirstName, MiddleName, LastName, Suffix

FROM SalesLT.Customer;

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.

-- Display title and last name with phone number

SELECT Salesperson, Title + ' ' + LastName AS CustomerName, Phone

FROM SalesLT.Customer;

Challenge 2: Retrieve Customer and Sales 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.

**Tip**: Review the documentation for Conversion Functions in the Transact-SQL Reference.

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**.

-- Customer Companies

SELECT CAST(CustomerID AS varchar) + ': ' + CompanyName AS CustomerCompany

FROM SalesLT.Customer;

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 <*Order Number*> (<*Revision*>) – for example **SO71774 (2)**.

The order date converted to ANSI standard format (yyyy.mm.dd – for example **2015.01.31**).

--Sales Order Revisions

SELECT SalesOrderNumber + ' (' + STR(RevisionNumber, 1) + ')' AS OrderRevision,

CONVERT(nvarchar(30), OrderDate, 102) AS OrderDate

FROM SalesLT.SalesOrderHeader;

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 fields appropriately.

**Tip**: Review the documentation for the ISNULL function and Expressions in the Transact-SQL Reference.

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 field in the format <*first name*> <*last name*> (for example **Keith Harris**) if the middle name is unknown, or <*first name*> <*middle name*> <*last name*> (for example **Jane M. Gates**) if a middle name is stored in the database.

-- Get middle names if known

SELECT FirstName + ' ' + ISNULL(MiddleName + ' ', '')+ LastName AS CustomerName

FROM SalesLT.Customer;

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 in **AdventureWorksLT**, 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 (don’t worry, you’ll learn about UPDATE statements later in the course).

UPDATE SalesLT.Customer

SET EmailAddress = NULL

WHERE CustomerID % 7 = 1;

-- Get primary contact details

UPDATE SalesLT.Customer

SET EmailAddress = NULL

WHERE CustomerID % 7 = 1;

SELECT CustomerID, COALESCE(EmailAddress, Phone) AS PrimaryContact

FROM SalesLT.Customer;

3. Retrieve shipping status

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 in **AdventureWorksLT**, 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 (don’t worry, you’ll learn about UPDATE statements later in the course).

UPDATE SalesLT.SalesOrderHeader

SET ShipDate = NULL

WHERE SalesOrderID > 71899;

-- Get shipping status

UPDATE SalesLT.SalesOrderHeader

SET ShipDate = NULL

WHERE SalesOrderID > 71899;

SELECT SalesOrderID, OrderDate,

CASE

WHEN ShipDate IS NULL THEN 'Awaiting Shipment'

ELSE 'Shipped'

END AS ShippingStatus

FROM SalesLT.SalesOrderHeader;

**Lab 2:**

DISTINCT, TOP, PERCENT, IN, LIKE,

**Eliminating duplicates and sorting:**

--Display a list of product colors

SELECT Color FROM SalesLT.Product;

--Display a list of product colors with the word 'None' if the value is null

SELECT DISTINCT ISNULL(Color, 'None') AS Color FROM SalesLT.Product;

--Display a list of product colors with the word 'None' if the value is null sorted by color

SELECT DISTINCT ISNULL(Color, 'None') AS Color FROM SalesLT.Product ORDER BY Color;

--Display a list of product colors with the word 'None' if the value is null and a dash if the size is null sorted by color

SELECT DISTINCT ISNULL(Color, 'None') AS Color, ISNULL(Size, '-') AS Size FROM SalesLT.Product ORDER BY Color;

--Display the top 100 products by list price

SELECT TOP 100 Name, ListPrice FROM SalesLT.Product ORDER BY ListPrice DESC;

--Display the first ten products by product number

SELECT Name, ListPrice FROM SalesLT.Product ORDER BY ProductNumber OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;

--Display the next ten products by product number

SELECT Name, ListPrice FROM SalesLT.Product ORDER BY ProductNumber OFFSET 10 ROWS FETCH FIRST 10 ROW ONLY;

**Filtering with predicates:**

--List information about product model 6

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

--List information about products that have a product number beginning FR

SELECT productnumber,Name, ListPrice FROM SalesLT.Product WHERE ProductNumber LIKE 'FR%';

--Filter the previous query to ensure that the product number contains two sets of two didgets

SELECT Name, ListPrice FROM SalesLT.Product WHERE ProductNumber LIKE 'FR-\_[0-9][0-9]\_-[0-9][0-9]';

--Find products that have no sell end date

SELECT Name FROM SalesLT.Product WHERE SellEndDate IS NOT NULL;

--Find products that have a sell end date in 2006

SELECT Name FROM SalesLT.Product WHERE SellEndDate BETWEEN '2006/1/1' AND '2006/12/31';

--Find products that have a category ID of 5, 6, or 7.

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

--Find products that have a category ID of 5, 6, or 7 and have a sell end date

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

--Select products that have a category ID of 5, 6, or 7 and a product number that begins FR

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

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.

**Tip**: Review the documentation for the SELECT and ORDER BY clauses in the Transact-SQL Reference.

1. Retrieve a list of cities

Initially, you need to produce a list of all of you customers' locations. Write a Transact-SQL query that queries the Address table and retrieves all values for **City** and **StateProvince**, removing duplicates.

--Retrieve City List

SELECT DISTINCT City, StateProvince

FROM SalesLT.Address

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.

--Retrieve Heaviest Products

SELECT TOP 10 PERCENT Name FROM SalesLT.Product ORDER BY Weight DESC;

3. Retrieve the heaviest 100 products not including the heaviest ten

The heaviest ten products are transported by a specialist carrier, therefore you need to modify the previous query to list the heaviest 100 products not including the heaviest ten.

--Retrieve the Heaviest 100 Products Not Including the Heaviest Ten

SELECT Name FROM SalesLT.Product ORDER BY Weight DESC

OFFSET 10 ROWS FETCH NEXT 100 ROWS ONLY;

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.

**Tip**: Review the documentation for the WHERE and LIKE keywords in the Transact-SQL Reference.

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**.

--Retrieve Product Details

SELECT Name, Color, Size

FROM SalesLT.Product

WHERE ProductModelID = 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'.

--Retrieve Products by Color and Size

SELECT ProductNumber, Name

FROM SalesLT.Product

WHERE Color IN ('Black','Red','White') and Size IN ('S','M');

3. Filter products by product number

Retrieve the product number, name, and list price of products whose product number begins 'BK-'.

--Retrieve Products by Product Number

SELECT ProductNumber, Name, ListPrice

FROM SalesLT.Product

WHERE ProductNumber LIKE '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.

--Retrieve Specific Products by Product Number

SELECT ProductNumber, Name, ListPrice

FROM SalesLT.Product

WHERE ProductNumber LIKE 'BK-[^R]%-[0-9][0-9]';

**Lab 3: JOIN**

## Key Points

* **Inner joins** return only rows where a match can be found in both tables.
* **Inner joins** that match rows based on columns containing the same value in both tables are sometimes referred to as equi-joins.
* Use a **Left Outer Join** to include all rows from the first table and values from matched rows in the second table. Columns in the second table for which no matching rows exist are populated with NULLs.
* Use a **Right Outer Join** to include all rows from the second table and values from matched rows in thefirst table. Columns in the first table for which no matching rows exist are populated with NULLs.
* Use a **Full Outer Join** to include all rows from the first and second tables. Columns in the either table for which no matching rows exist are populated with NULLs.
* A cross join returns a **Cartesian product** that includes every combination of the selected columns from both tables.
* While not commonly used in typical application processing, cross joins can be useful in some specialized scenarios - such as generating test data.
* A **self-join** is an inner, outer, or cross join that matches rows in a table to other rows in the same table.
* When defining a **self-join**, you must specify an alias for at least one instance of the table being joined.

**Inner Joins:**

--Basic inner join

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;

-- Table aliases

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

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory As c

ON p.ProductCategoryID = c.ProductCategoryID;

-- Joining more than 2 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;

-- Multiple join predicates

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 AND od.UnitPrice = p.ListPrice --Note multiple predicates

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

**Outer Joins:**

--Get all customers, with sales orders for those who've bought anything

SELECT c.FirstName, c.LastName, oh.SalesOrderNumber

FROM SalesLT.Customer AS c

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID

ORDER BY c.CustomerID;

--Return only customers who haven't purchased anything

SELECT c.FirstName, c.LastName, oh.SalesOrderNumber

FROM SalesLT.Customer AS c

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID

WHERE oh.SalesOrderNumber IS NULL

ORDER BY c.CustomerID;

--More than 2 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 --Additional tables added to the right must also use a left join

ON od.SalesOrderID = oh.SalesOrderID

ORDER BY p.ProductID;

SELECT p.Name As ProductName, c.Name AS Category, oh.SalesOrderNumber

FROM SalesLT.Product AS p

LEFT OUTER JOIN SalesLT.SalesOrderDetail AS od

ON p.ProductID = od.ProductID

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

ON od.SalesOrderID = oh.SalesOrderID

INNER JOIN SalesLT.ProductCategory AS c --Added to the left, so can use inner join

ON p.ProductCategoryID = c.ProductCategoryID

ORDER BY p.ProductID;

**Cross Joins:**

--Call each customer once per product

SELECT p.Name, c.FirstName, c.LastName, c.Phone

FROM SalesLT.Product as p

CROSS JOIN SalesLT.Customer as c;

**Self Joins:**

--note there's no employee table, so we'll create one for this example

CREATE TABLE SalesLT.Employee

(EmployeeID int IDENTITY PRIMARY KEY,

EmployeeName nvarchar(256),

ManagerID int);

GO

-- Get salesperson from Customer table and generate managers

INSERT INTO SalesLT.Employee (EmployeeName, ManagerID)

SELECT DISTINCT Salesperson, NULLIF(CAST(RIGHT(SalesPerson, 1) as INT), 0)

FROM SalesLT.Customer;

GO

UPDATE SalesLT.Employee

SET ManagerID = (SELECT MIN(EmployeeID) FROM SalesLT.Employee WHERE ManagerID IS NULL)

WHERE ManagerID IS NULL

AND EmployeeID > (SELECT MIN(EmployeeID) FROM SalesLT.Employee WHERE ManagerID IS NULL);

GO

-- Here's the actual self-join demo

SELECT e.EmployeeName, m.EmployeeName AS ManagerName

FROM SalesLT.Employee AS e

LEFT JOIN SalesLT.Employee AS m

ON e.ManagerID = m.EmployeeID

ORDER BY e.ManagerID;

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.

**Tip**: Review the documentation for the FROM clause in the Transact-SQL Reference.

1. Retrieve customer orders

As an initial step towards generating the invoice report, write a query that returns the company name from the **SalesLT.Customer** table, and the sales order ID and total due from the **SalesLT.SalesOrderHeader** table.

-- Customer Orders

SELECT c.CompanyName, oh.SalesOrderID, oh.TotalDue

FROM SalesLT.Customer AS c

JOIN SalesLT.SalesOrderHeader AS oh

ON oh.CustomerID = c.CustomerID;

2. Retrieve customer orders with addresses

Extend your customer orders query to include the *Main Office* address for each customer, including the full street address, city, state or province, postal code, and country or region

**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, and should filter the join to **SalesLT.CustomerAddress** so that only *Main Office* addresses are included.

-- Customer Orders with Addresses

SELECT c.CompanyName, a.AddressLine1, ISNULL(a.AddressLine2, '') AS AddressLine2,

a.City, a.StateProvince, a.PostalCode, a.CountryRegion, oh.SalesOrderID, oh.TotalDue

FROM SalesLT.Customer AS c

JOIN SalesLT.SalesOrderHeader AS oh

ON oh.CustomerID = c.CustomerID

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID AND AddressType = 'Main Office'

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID;

Challenge 2: Retrieve Sales 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 all customers and their orders

The sales manager wants a list of all customer companies and their contacts (first name and last name), showing the sales order ID and total due for each order they have placed. Customers who have not placed any orders should be included at the bottom of the list with NULL values for the order ID and total due.

-- All customers and their orders

SELECT c.CompanyName, c.FirstName, c.LastName, oh.SalesOrderID, oh.TotalDue

FROM SalesLT.Customer AS c

LEFT JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID

ORDER BY oh.SalesOrderID DESC;

2. 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, company names, contact names (first name and last name), and phone numbers for customers with no address stored in the database.

-- Customers with no address

SELECT c.CompanyName, c.FirstName, c.LastName, c.Phone

FROM SalesLT.Customer AS c

LEFT JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

WHERE ca.AddressID IS NULL;

3. Retrieve a list of customers and products without orders

Some customers have never placed orders, and some products have never been ordered. Create a query that returns a column of customer IDs for customers who have never placed an order, and a column of product IDs for products that have never been ordered. Each row with a customer ID should have a NULL product ID (because the customer has never ordered a product) and each row with a product ID should have a NULL customer ID (because the product has never been ordered by a customer).

-- Customers and products for which there are no orders

SELECT c.CustomerID, p.ProductID

FROM SalesLT.Customer AS c

FULL JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID

FULL JOIN SalesLT.SalesOrderDetail AS od

ON od.SalesOrderID = oh.SalesOrderID

FULL JOIN SalesLT.Product AS p

ON p.ProductID = od.ProductID

WHERE oh.SalesOrderID IS NULL

ORDER BY ProductID, CustomerID;

**Lab 4: SET Operators – UNION ALL, INTERSECT, EXCEPT**

* Use UNION to combine the rowsets returned by mulitple queries.
* Each unioned query must return the same number of columns with compatible data types.
* By default, UNION eliminates duplicate rows. Specify the ALL option to include duplicates (or to avoid the overhead of checking for duplicates when you know in advance that there are none).

**UNION:**

-- Setup

CREATE VIEW [SalesLT].[Customers]

as

select distinct firstname,lastname

from saleslt.customer

where lastname >='m'

or customerid=3;

GO

CREATE VIEW [SalesLT].[Employees]

as

select distinct firstname,lastname

from saleslt.customer

where lastname <='m'

or customerid=3;

GO

-- Union example

SELECT FirstName, LastName

FROM SalesLT.Employees

UNION

SELECT FirstName, LastName

FROM SalesLT.Customers

ORDER BY LastName;

**INTERSECT:**

SELECT FirstName, LastName

FROM SalesLT.Customers

INTERSECT

SELECT FirstName, LastName

FROM SalesLT.Employees;

**EXCEPT:**

SELECT FirstName, LastName

FROM SalesLT.Customers

EXCEPT

SELECT FirstName, LastName

FROM SalesLT.Employees;

Challenge 1: Retrieve Customer Addresses

Customers can have two kinds of address: a main office address and a shipping address. The accounts department want to ensure that the main office address is always used for billing, and have asked you to write a query that clearly identifies the different types of address for each customer.

**Tip**: Review the documentation for the UNION operator in the Transact-SQL Reference.

1. Retrieve billing addresses

Write a query that retrieves the company name, first line of the street address, city, and a column named **AddressType** with the value ‘Billing’ for customers where the address type in the **SalesLT.CustomerAddress** table is ‘Main Office’.

-- Billing addresses

SELECT c.CompanyName, a.AddressLine1, a.City, 'Billing' AS AddressType

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Main Office';

2. Retrieve shipping addresses

Write a similar query that retrieves the company name, first line of the street address, city, and a column named **AddressType** with the value ‘Shipping’ for customers where the address type in the **SalesLT.CustomerAddress** table is ‘Shipping’.

-- Shipping addresses

SELECT c.CompanyName, a.AddressLine1, a.City, 'Shipping' AS AddressType

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Shipping';

3. Combine billing and shipping addresses

Combine the results returned by the two queries to create a list of all customer addresses that is sorted by company name and then address type.

-- All customer addresses

SELECT c.CompanyName, a.AddressLine1, a.City, 'Billing' AS AddressType

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Main Office'

UNION ALL

SELECT c.CompanyName, a.AddressLine1, a.City, 'Shipping' AS AddressType

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Shipping'

ORDER BY c.CompanyName, AddressType;

Challenge 2: Filter Customer Addresses

You have created a master list of all customer addresses, but now you have been asked to create filtered lists that show which customers have only a main office address, and which customers have both a main office and a shipping address.

**Tip**: Review the documentation for the EXCEPT and INTERSECT operators in the Transact-SQL Reference.

1. Retrieve customers with only a main office address

Write a query that returns the company name of each company that appears in a table of customers with a ‘Main Office’ address, but not in a table of customers with a ‘Shipping’ address.

-- Customers with only a main office address

SELECT c.CompanyName

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Main Office'

EXCEPT

SELECT c.CompanyName

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Shipping'

ORDER BY c.CompanyName;

2. Retrieve only customers with both a main office address and a shipping address

Write a query that returns the company name of each company that appears in a table of customers with a ‘Main Office’ address, and also in a table of customers with a ‘Shipping’ address.

-- Only customers with both a main office and a shipping address

SELECT c.CompanyName

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Main Office'

INTERSECT

SELECT c.CompanyName

FROM SalesLT.Customer AS c

JOIN SalesLT.CustomerAddress AS ca

ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address AS a

ON ca.AddressID = a.AddressID

WHERE ca.AddressType = 'Shipping'

ORDER BY c.CompanyName;

**Lab 5: Using function and aggregating data**

## Key PointS

* Scalar functions return a single value based on zero or more input parameters.
* Logical functions return Boolean values (true or false) based on an expression or column value.
* Window functions are used to rank rows across partitions or "windows". Window functions include RANK, DENSE\_RANK, NTILE, and ROW\_NUMBER.
* Aggregate functions are used to provide summary values for mulitple rows - for example, the total cost of products or the maximum number of items in an order. Commonly used aggregate functions include SUM, COUNT, MIN, MAX, and AVG.

**Functions:**

-- Scalar functions

SELECT YEAR(SellStartDate) SellStartYear, ProductID, Name

FROM SalesLT.Product

ORDER BY SellStartYear;

SELECT YEAR(SellStartDate) SellStartYear, DATENAME(mm,SellStartDate) SellStartMonth,

DAY(SellStartDate) SellStartDay, DATENAME(dw, SellStartDate) SellStartWeekday,

ProductID, Name

FROM SalesLT.Product

ORDER BY SellStartYear;

SELECT DATEDIFF(yy,SellStartDate, GETDATE()) YearsSold, ProductID, Name

FROM SalesLT.Product

ORDER BY ProductID;

SELECT UPPER(Name) AS ProductName

FROM SalesLT.Product;

SELECT CONCAT(FirstName + ' ', LastName) AS FullName

FROM SalesLT.Customer;

SELECT Name, ProductNumber, LEFT(ProductNumber, 2) AS ProductType

FROM SalesLT.Product;

SELECT Name, ProductNumber, LEFT(ProductNumber, 2) AS ProductType,

SUBSTRING(ProductNumber,CHARINDEX('-', ProductNumber) + 1, 4) AS ModelCode,

SUBSTRING(ProductNumber, LEN(ProductNumber) - CHARINDEX('-', REVERSE(RIGHT(ProductNumber, 3))) + 2, 2) AS SizeCode

FROM SalesLT.Product;

-- Logical functions

SELECT Name, Size AS NumericSize

FROM SalesLT.Product

WHERE ISNUMERIC(Size) = 1;

SELECT Name, IIF(ProductCategoryID IN (5,6,7), 'Bike', 'Other') ProductType

FROM SalesLT.Product;

SELECT Name, IIF(ISNUMERIC(Size) = 1, 'Numeric', 'Non-Numeric') SizeType

FROM SalesLT.Product;

SELECT prd.Name AS ProductName, cat.Name AS Category,

CHOOSE (cat.ParentProductCategoryID, 'Bikes','Components','Clothing','Accessories') AS ProductType

FROM SalesLT.Product AS prd

JOIN SalesLT.ProductCategory AS cat

ON prd.ProductCategoryID = cat.ProductCategoryID;

-- Window functions

SELECT TOP(100) ProductID, Name, ListPrice,

RANK() OVER(ORDER BY ListPrice DESC) AS RankByPrice

FROM SalesLT.Product AS p

ORDER BY RankByPrice;

SELECT c.Name AS Category, p.Name AS Product, ListPrice,

RANK() OVER(PARTITION BY c.Name ORDER BY ListPrice DESC) AS RankByPrice

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON p.ProductCategoryID = c.ProductcategoryID

ORDER BY Category, RankByPrice;

-- Aggregate Functions

SELECT COUNT(\*) AS Products, COUNT(DISTINCT ProductCategoryID) AS Categories, AVG(ListPrice) AS AveragePrice

FROM SalesLT.Product;

SELECT COUNT(p.ProductID) BikeModels, AVG(p.ListPrice) AveragePrice

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON p.ProductCategoryID = c.ProductCategoryID

WHERE c.Name LIKE '%Bikes';

**GROUP BY:**

SELECT Salesperson, COUNT(CustomerID) Customers

FROM SalesLT.Customer

GROUP BY Salesperson

ORDER BY Salesperson;

SELECT c.Name AS Category, COUNT(p.ProductID) AS Products

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON p.ProductCategoryID = c.ProductCategoryID

GROUP BY c.Name

ORDER BY Category;

SELECT c.Salesperson, SUM(oh.SubTotal) SalesRevenue

FROM SalesLT.Customer c

JOIN SalesLT.SalesOrderHeader oh

ON c.CustomerID = oh.CustomerID

GROUP BY c.Salesperson

ORDER BY SalesRevenue DESC;

SELECT c.Salesperson, ISNULL(SUM(oh.SubTotal), 0.00) SalesRevenue

FROM SalesLT.Customer c

LEFT JOIN SalesLT.SalesOrderHeader oh

ON c.CustomerID = oh.CustomerID

GROUP BY c.Salesperson

ORDER BY SalesRevenue DESC;

SELECT c.Salesperson, CONCAT(c.FirstName +' ', c.LastName) AS Customer, ISNULL(SUM(oh.SubTotal), 0.00) SalesRevenue

FROM SalesLT.Customer c

LEFT JOIN SalesLT.SalesOrderHeader oh

ON c.CustomerID = oh.CustomerID

GROUP BY c.Salesperson, CONCAT(c.FirstName +' ', c.LastName)

ORDER BY SalesRevenue DESC, Customer;

**HAVING**

-- Try to find salespeople with over 150 customers (fails with error)

SELECT Salesperson, COUNT(CustomerID) Customers

FROM SalesLT.Customer

WHERE COUNT(CustomerID) > 100

GROUP BY Salesperson

ORDER BY Salesperson;

--Need to use HAVING clause to filter based on aggregate

SELECT Salesperson, COUNT(CustomerID) Customers

FROM SalesLT.Customer

GROUP BY Salesperson

HAVING COUNT(CustomerID) > 100

ORDER BY Salesperson;

**LAB 6:**

Key Points

* Subqueries are Transact-SQL queries nested within an outer query.
* Scalar subqueries return a single value.
* Multi-valued subqueries return a single-column rowset.

**SCALAR SUBQuery**

--Display a list of products whose list price is higher than the highest unit price of items that have sold

SELECT MAX(UnitPrice) FROM SalesLT.SalesOrderDetail

SELECT \* from SalesLT.Product

WHERE ListPrice >

SELECT \* from SalesLT.Product

WHERE ListPrice >

(SELECT MAX(UnitPrice) FROM SalesLT.SalesOrderDetail)

**Multivalued subquery:**

--List products that have an order quantity greater than 20

SELECT Name FROM SalesLT.Product

WHERE ProductID IN

(SELECT ProductID from SalesLT.SalesOrderDetail

WHERE OrderQty>20)

SELECT Name

FROM SalesLT.Product P

JOIN SalesLT.SalesOrderDetail SOD

ON P.ProductID=SOD.ProductID

WHERE OrderQty>20

## Key Points

* Correlated subqueries reference objects in the outer query.

**Correlated Subquery:**

--For each customer list all sales on the last day that they made a sale

SELECT CustomerID, SalesOrderID, OrderDate

FROM SalesLT.SalesOrderHeader AS SO1

ORDER BY CustomerID,OrderDate

SELECT CustomerID, SalesOrderID, OrderDate

FROM SalesLT.SalesOrderHeader AS SO1

WHERE orderdate =

(SELECT MAX(orderdate)

FROM SalesLT.SalesOrderHeader)

SELECT CustomerID, SalesOrderID, OrderDate

FROM SalesLT.SalesOrderHeader AS SO1

WHERE orderdate =

(SELECT MAX(orderdate)

FROM SalesLT.SalesOrderHeader AS SO2

WHERE SO2.CustomerID = SO1.CustomerID)

ORDER BY CustomerID

## Key Points

* The APPLY operator enables you to execute a table-valued function for each row in a rowset returned by a SELECT statement. Conceptually, this approach is similar to a correlated subquery.
* CROSS APPLY returns matching rows, similar to an inner join. OUTER APPLY returns all rows in the original SELECT query results with NULL values for rows where no match was found.

**Cross Apply:**

-- Setup

CREATE FUNCTION SalesLT.udfMaxUnitPrice (@SalesOrderID int)

RETURNS TABLE

AS

RETURN

SELECT SalesOrderID,Max(UnitPrice) as MaxUnitPrice FROM

SalesLT.SalesOrderDetail

WHERE SalesOrderID=@SalesOrderID

GROUP BY SalesOrderID;

--Display the sales order details for items that are equal to

-- the maximum unit price for that sales order

SELECT \* FROM SalesLT.SalesOrderDetail AS SOH

CROSS APPLY SalesLT.udfMaxUnitPrice(SOH.SalesOrderID) AS MUP

WHERE SOH.UnitPrice=MUP.MaxUnitPrice

ORDER BY SOH.SalesOrderID;

**LAB 7: View,** **Temp Tables and Variables**

## Key Points

* Views are database objects that encapsulate SELECT queries.
* You can query a view in the same way as a table, however there are some considerations for updating them.

**01- Views:**

-- Create a view

CREATE VIEW SalesLT.vCustomerAddress

AS

SELECT C.CustomerID, FirstName, LastName, AddressLine1, City, StateProvince

FROM

SalesLT.Customer C JOIN SalesLT.CustomerAddress CA

ON C.CustomerID=CA.CustomerID

JOIN SalesLT.Address A

ON CA.AddressID=A.AddressID

-- Query the view

SELECT CustomerID, City

FROM SalesLT.vCustomerAddress

-- Join the view to a table

SELECT c.StateProvince, c.City, ISNULL(SUM(s.TotalDue), 0.00) AS Revenue

FROM SalesLT.vCustomerAddress AS c

LEFT JOIN SalesLT.SalesOrderHeader AS s

ON s.CustomerID = c.CustomerID

GROUP BY c.StateProvince, c.City

ORDER BY c.StateProvince, Revenue DESC;

**02-Temp Tables and Variables**

-- Temporary table

CREATE TABLE #Colors

(Color varchar(15));

INSERT INTO #Colors

SELECT DISTINCT Color FROM SalesLT.Product;

SELECT \* FROM #Colors;

-- Table variable

DECLARE @Colors AS TABLE (Color varchar(15));

INSERT INTO @Colors

SELECT DISTINCT Color FROM SalesLT.Product;

SELECT \* FROM @Colors;

-- New batch

SELECT \* FROM #Colors;

SELECT \* FROM @Colors; -- now out of scope

**03-TVFs**

CREATE FUNCTION SalesLT.udfCustomersByCity

(@City AS VARCHAR(20))

RETURNS TABLE

AS

RETURN

(SELECT C.CustomerID, FirstName, LastName, AddressLine1, City, StateProvince

FROM SalesLT.Customer C JOIN SalesLT.CustomerAddress CA

ON C.CustomerID=CA.CustomerID

JOIN SalesLT.Address A ON CA.AddressID=A.AddressID

WHERE City=@City);

SELECT \* FROM SalesLT.udfCustomersByCity('Bellevue')

**04-Derived Tables**

SELECT Category, COUNT(ProductID) AS Products

FROM

(SELECT p.ProductID, p.Name AS Product, c.Name AS Category

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON p.ProductCategoryID = c.ProductCategoryID) AS ProdCats

GROUP BY Category

ORDER BY Category;

**05- CTEs**

--Using a CTE

WITH ProductsByCategory (ProductID, ProductName, Category)

AS

(

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

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON p.ProductCategoryID = c.ProductCategoryID

)

SELECT Category, COUNT(ProductID) AS Products

FROM ProductsByCategory

GROUP BY Category

ORDER BY Category;

-- Recursive CTE

SELECT \* FROM SalesLT.Employee

-- Using the CTE to perform recursion

WITH OrgReport (ManagerID, EmployeeID, EmployeeName, Level)

AS

(

-- Anchor query

SELECT e.ManagerID, e.EmployeeID, EmployeeName, 0

FROM SalesLT.Employee AS e

WHERE ManagerID IS NULL

UNION ALL

-- Recursive query

SELECT e.ManagerID, e.EmployeeID, e.EmployeeName, Level + 1

FROM SalesLT.Employee AS e

INNER JOIN OrgReport AS o ON e.ManagerID = o.EmployeeID

)

SELECT \* FROM OrgReport

OPTION (MAXRECURSION 3);

**01-Grouping Sets**

SELECT cat.ParentProductCategoryName, cat.ProductCategoryName, count(prd.ProductID) AS Products

FROM SalesLT.vGetAllCategories as cat

LEFT JOIN SalesLT.Product AS prd

ON prd.ProductCategoryID = cat.ProductcategoryID

GROUP BY cat.ParentProductCategoryName, cat.ProductCategoryName

--GROUP BY GROUPING SETS(cat.ParentProductCategoryName, cat.ProductCategoryName, ())

--GROUP BY ROLLUP (cat.ParentProductCategoryName, cat.ProductCategoryName)

--GROUP BY CUBE (cat.ParentProductCategoryName, cat.ProductCategoryName)

ORDER BY cat.ParentProductCategoryName, cat.ProductCategoryName;

**02-Pivot**

SELECT \* FROM

(SELECT P.ProductID, PC.Name,ISNULL(P.Color, 'Uncolored') AS Color

FROM saleslt.productcategory AS PC

JOIN SalesLT.Product AS P

ON PC.ProductCategoryID=P.ProductCategoryID

) AS PPC

PIVOT(COUNT(ProductID) FOR Color IN([Red],[Blue],[Black],[Silver],[Yellow],[Grey], [Multi], [Uncolored])) as pvt

ORDER BY Name;

-- Unpivot

CREATE TABLE #ProductColorPivot

(Name varchar(50), Red int, Blue int, Black int, Silver int, Yellow int, Grey int , multi int, uncolored int);

INSERT INTO #ProductColorPivot

SELECT \* FROM

(SELECT P.ProductID, PC.Name,ISNULL(P.Color, 'Uncolored') AS Color

FROM saleslt.productcategory AS PC

JOIN SalesLT.Product AS P

ON PC.ProductCategoryID=P.ProductCategoryID

) AS PPC

PIVOT(COUNT(ProductID) FOR Color IN([Red],[Blue],[Black],[Silver],[Yellow],[Grey], [Multi], [Uncolored])) as pvt

ORDER BY Name;

SELECT Name, Color, ProductCount

FROM

(SELECT Name,

[Red],[Blue],[Black],[Silver],[Yellow],[Grey], [Multi], [Uncolored]

FROM #ProductColorPivot) pcp

UNPIVOT

(ProductCount FOR Color IN ([Red],[Blue],[Black],[Silver],[Yellow],[Grey], [Multi], [Uncolored])

) AS ProductCounts

-- Unpivot

CREATE TABLE #SalesByQuarter

(ProductID int,

Q1 money,

Q2 money,

Q3 money,

Q4 money);

INSERT INTO #SalesByQuarter

VALUES

(1, 19999.00, 21567.00, 23340.00, 25876.00),

(2, 10997.00, 12465.00, 13367.00, 14365.00),

(3, 21900.00, 21999.00, 23376.00, 23676.00);

SELECT \* FROM #SalesByQuarter;

SELECT ProductID, Period, Revenue

FROM

(SELECT ProductID,

Q1, Q2, Q3, Q4

FROM #SalesByQuarter) sbq

UNPIVOT

(Revenue FOR Period IN (Q1, Q2, Q3, Q4)

) AS RevenueReport

**Lab 09 - Raising Errors**

**01-Inserting Data**

-- Create a table for the demo

CREATE TABLE SalesLT.CallLog

(

CallID int IDENTITY PRIMARY KEY NOT NULL,

CallTime datetime NOT NULL DEFAULT GETDATE(),

SalesPerson nvarchar(256) NOT NULL,

CustomerID int NOT NULL REFERENCES SalesLT.Customer(CustomerID),

PhoneNumber nvarchar(25) NOT NULL,

Notes nvarchar(max) NULL

);

GO

-- Insert a row

INSERT INTO SalesLT.CallLog

VALUES

('2015-01-01T12:30:00', 'adventure-works\pamela0', 1, '245-555-0173', 'Returning call re: enquiry about delivery');

SELECT \* FROM SalesLT.CallLog;

-- Insert defaults and nulls

INSERT INTO SalesLT.CallLog

VALUES

(DEFAULT, 'adventure-works\david8', 2, '170-555-0127', NULL);

SELECT \* FROM SalesLT.CallLog;

-- Insert a row with explicit columns

INSERT INTO SalesLT.CallLog (SalesPerson, CustomerID, PhoneNumber)

VALUES

('adventure-works\jillian0', 3, '279-555-0130');

SELECT \* FROM SalesLT.CallLog;

-- Insert multiple rows

INSERT INTO SalesLT.CallLog

VALUES

(DATEADD(mi,-2, GETDATE()), 'adventure-works\jillian0', 4, '710-555-0173', NULL),

(DEFAULT, 'adventure-works\shu0', 5, '828-555-0186', 'Called to arrange deliver of order 10987');

SELECT \* FROM SalesLT.CallLog;

-- Insert the results of a query

INSERT INTO SalesLT.CallLog (SalesPerson, CustomerID, PhoneNumber, Notes)

SELECT SalesPerson, CustomerID, Phone, 'Sales promotion call'

FROM SalesLT.Customer

WHERE CompanyName = 'Big-Time Bike Store';

SELECT \* FROM SalesLT.CallLog;

-- Retrieving inserted identity

INSERT INTO SalesLT.CallLog (SalesPerson, CustomerID, PhoneNumber)

VALUES

('adventure-works\josé1', 10, '150-555-0127');

SELECT SCOPE\_IDENTITY();

SELECT \* FROM SalesLT.CallLog;

--Overriding Identity

SET IDENTITY\_INSERT SalesLT.CallLog ON;

INSERT INTO SalesLT.CallLog (CallID, SalesPerson, CustomerID, PhoneNumber)

VALUES

(9, 'adventure-works\josé1', 11, '926-555-0159');

SET IDENTITY\_INSERT SalesLT.CallLog OFF;

SELECT \* FROM SalesLT.CallLog;

**02-Updating and Deleting**

-- Update a table

UPDATE SalesLT.CallLog

SET Notes = 'No notes'

WHERE Notes IS NULL;

SELECT \* FROM SalesLT.CallLog;

-- Update multiple columns

UPDATE SalesLT.CallLog

SET SalesPerson = '', PhoneNumber = ''

SELECT \* FROM SalesLT.CallLog;

-- Update from results of a query

UPDATE SalesLT.CallLog

SET SalesPerson = c.SalesPerson, PhoneNumber = c.Phone

FROM SalesLT.Customer AS c

WHERE c.CustomerID = SalesLT.CallLog.CustomerID;

SELECT \* FROM SalesLT.CallLog;

-- Delete rows

DELETE FROM SalesLT.CallLog

WHERE CallTime < DATEADD(dd, -7, GETDATE());

SELECT \* FROM SalesLT.CallLog;

-- Truncate the table

TRUNCATE TABLE SalesLT.CallLog;

SELECT \* FROM SalesLT.CallLog;

**Lab 10 - Raising Errors**

**0 – Setup**

IF OBJECT\_ID('SalesLT.DemoTable') IS NOT NULL

BEGIN

DROP TABLE SalesLT.DemoTable

END

GO

CREATE TABLE SalesLT.DemoTable

(ID INT IDENTITY(1,1),

Description Varchar(20),

CONSTRAINT [PK\_DemoTable] PRIMARY KEY CLUSTERED(ID)

)

GO

**1 – Variables**

--Search by city using a variable

DECLARE @City VARCHAR(20)='Toronto'

Set @City='Bellevue'

Select FirstName +' '+LastName as [Name],AddressLine1 as Address,City

FROM SalesLT.Customer as C

JOIN SalesLT.CustomerAddress as CA

ON C.CustomerID=CA.CustomerID

JOIN SalesLT.Address as A

ON CA.AddressID=A.AddressID

WHERE City=@City

--Use a variable as an output

DECLARE @Result money

SELECT @Result=MAX(TotalDue)

FROM SalesLT.SalesOrderHeader

PRINT @Result

**2 - If Else**

--Simple logical test

If 'Yes'='Yes'

Print 'True'

--Change code based on a condition

UPDATE SalesLT.Product

SET DiscontinuedDate=getdate()

WHERE ProductID=1;

IF @@ROWCOUNT<1

BEGIN

PRINT 'Product was not found'

END

ELSE

BEGIN

PRINT 'Product Updated'

END

**3 – While**

DECLARE @Counter int=1

WHILE @Counter <=5

BEGIN

INSERT SalesLT.DemoTable(Description)

VALUES ('ROW '+CONVERT(varchar(5),@Counter))

SET @Counter=@Counter+1

END

SELECT Description FROM SalesLT.DemoTable

--Testing for existing values

DECLARE @Counter int=1

DECLARE @Description int

SELECT @Description=MAX(ID)

FROM SalesLT.DemoTable

WHILE @Counter <5

BEGIN

INSERT SalesLT.DemoTable(Description)

VALUES ('ROW '+CONVERT(varchar(5),@Description))

SET @Description=@Description+1

SET @Counter=@Counter+1

END

SELECT Description FROM SalesLT.DemoTable

**4-Stored Procedure**

-- Create a stored procedure

CREATE PROCEDURE SalesLT.GetProductsByCategory (@CategoryID INT = NULL)

AS

IF @CategoryID IS NULL

SELECT ProductID, Name, Color, Size, ListPrice

FROM SalesLT.Product

ELSE

SELECT ProductID, Name, Color, Size, ListPrice

FROM SalesLT.Product

WHERE ProductCategoryID = @CategoryID;

-- Execute the procedure without a parameter

EXEC SalesLT.GetProductsByCategory

-- Execute the procedure with a parameter

EXEC SalesLT.GetProductsByCategory 6

**Lab 11 - Raising Errors**

**01-Raising Errors**

-- View a system error

INSERT INTO SalesLT.SalesOrderDetail (SalesOrderID, OrderQty, ProductID, UnitPrice, UnitPriceDiscount)

VALUES

(100000, 1, 680, 1431.50, 0.00);

-- Raise an error with RAISERROR

UPDATE SalesLT.Product

SET DiscontinuedDate = GETDATE()

WHERE ProductID = 0;

IF @@ROWCOUNT < 1

RAISERROR('The product was not found - no products have been updated', 16, 0);

-- Raise an error with THROW

UPDATE SalesLT.Product

SET DiscontinuedDate = GETDATE()

WHERE ProductID = 0;

IF @@ROWCOUNT < 1

THROW 50001, 'The product was not found - no products have been updated', 0;

**02-Handling Errors**

-- catch an error

BEGIN TRY

UPDATE SalesLT.Product

SET ProductNumber = ProductID / ISNULL(Weight, 0);

END TRY

BEGIN CATCH

PRINT 'The following error occurred:';

PRINT ERROR\_MESSAGE();

END CATCH;

-- Catch and rethrow

BEGIN TRY

UPDATE SalesLT.Product

SET ProductNumber = ProductID / ISNULL(Weight, 0);

END TRY

BEGIN CATCH

PRINT 'The following error occurred:';

PRINT ERROR\_MESSAGE();

THROW;

END CATCH;

-- Catch, log, and throw a custom error

BEGIN TRY

UPDATE SalesLT.Product

SET ProductNumber = ProductID / ISNULL(Weight, 0);

END TRY

BEGIN CATCH

DECLARE @ErrorLogID as int, @ErrorMsg AS varchar(250);

EXECUTE dbo.uspLogError @ErrorLogID OUTPUT;

SET @ErrorMsg = 'The update failed because of an error. View error #'

+ CAST(@ErrorLogID AS varchar)

+ ' in the error log for details.';

THROW 50001, @ErrorMsg, 0;

END CATCH;

-- View the error log

SELECT \* FROM dbo.ErrorLog;

**03-Transactions**

-- No transaction

BEGIN TRY

INSERT INTO SalesLT.SalesOrderHeader (DueDate, CustomerID, ShipMethod)

VALUES

(DATEADD(dd, 7, GETDATE()), 1, 'STD DELIVERY');

DECLARE @SalesOrderID int = SCOPE\_IDENTITY();

INSERT INTO SalesLT.SalesOrderDetail (SalesOrderID, OrderQty, ProductID, UnitPrice, UnitPriceDiscount)

VALUES

(@SalesOrderID, 1, 99999, 1431.50, 0.00);

END TRY

BEGIN CATCH

PRINT ERROR\_MESSAGE();

END CATCH;

-- View orphaned orders

SELECT h.SalesOrderID, h.DueDate, h.CustomerID, h.ShipMethod, d.SalesOrderDetailID

FROM SalesLT.SalesOrderHeader AS h

LEFT JOIN SalesLT.SalesOrderDetail AS d

ON d.SalesOrderID = h.SalesOrderID

WHERE D.SalesOrderDetailID IS NULL;

-- Manually delete orphaned record

DELETE FROM SalesLT.SalesOrderHeader

WHERE SalesOrderID = SCOPE\_IDENTITY();

-- Use a transaction

BEGIN TRY

BEGIN TRANSACTION

INSERT INTO SalesLT.SalesOrderHeader (DueDate, CustomerID, ShipMethod)

VALUES

(DATEADD(dd, 7, GETDATE()), 1, 'STD DELIVERY');

DECLARE @SalesOrderID int = SCOPE\_IDENTITY();

INSERT INTO SalesLT.SalesOrderDetail (SalesOrderID, OrderQty, ProductID, UnitPrice, UnitPriceDiscount)

VALUES

(@SalesOrderID, 1, 99999, 1431.50, 0.00);

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

BEGIN

PRINT XACT\_STATE();

ROLLBACK TRANSACTION;

END

PRINT ERROR\_MESSAGE();

THROW 50001,'An insert failed. The transaction was cancelled.', 0;

END CATCH;

-- Check for orphaned orders

SELECT h.SalesOrderID, h.DueDate, h.CustomerID, h.ShipMethod, d.SalesOrderDetailID

FROM SalesLT.SalesOrderHeader AS h

LEFT JOIN SalesLT.SalesOrderDetail AS d

ON d.SalesOrderID = h.SalesOrderID

WHERE D.SalesOrderDetailID IS NULL

-- Use XACT\_ABORT

SET XACT\_ABORT ON;

BEGIN TRY

BEGIN TRANSACTION

INSERT INTO SalesLT.SalesOrderHeader (DueDate, CustomerID, ShipMethod)

VALUES

(DATEADD(dd, 7, GETDATE()), 1, 'STD DELIVERY');

DECLARE @SalesOrderID int = SCOPE\_IDENTITY();

INSERT INTO SalesLT.SalesOrderDetail (SalesOrderID, OrderQty, ProductID, UnitPrice, UnitPriceDiscount)

VALUES

(@SalesOrderID, 1, 99999, 1431.50, 0.00);

COMMIT TRANSACTION

END TRY

BEGIN CATCH

PRINT ERROR\_MESSAGE();

THROW 50001,'An insert failed. The transaction was cancelled.', 0;

END CATCH;

SET XACT\_ABORT OFF;

-- Check for orphaned orders

SELECT h.SalesOrderID, h.DueDate, h.CustomerID, h.ShipMethod, d.SalesOrderDetailID

FROM SalesLT.SalesOrderHeader AS h

LEFT JOIN SalesLT.SalesOrderDetail AS d

ON d.SalesOrderID = h.SalesOrderID

WHERE D.SalesOrderDetailID IS NULL