**T SQL**

* 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.*

**Lab 1: SELECT, CAST, TRY\_CAST, ISNULL, CONVERT, COALESCE, CASE, FROM, WHERE**

## 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.

**SELECT Statement:**

######## Select Statements #######

SELECT Name, StandardCost, ListPrice

FROM SalesLT.Product;

SELECT Name,ListPrice, StandardCost,

ListPrice - StandardCost as Sales\_Margin

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(3)) + ': ' + Name AS ProductName

FROM SalesLT.Product;

SELECT CONVERT(varchar(4), ProductID) + '-' + Name as ProductName

From SalesLT.Product;

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

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

**#### HANDLING NULL ######**

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;

**#### Selecting statement with preference #####**

SELECT Name,DiscontinuedDate, SellEndDate, SellStartDate,

COALESCE(DiscontinuedDate, SellEndDate, SellStartDate) AS FirstNonNullDate

FROM SalesLT.Product;

**##### CASE STATEMENTS ######**

SELECT Name,

CASE

WHEN SellEndDate IS NULL THEN 'On Sale'

ELSE 'Discontinued'

END AS SalesStatus

FROM SalesLT.Product;

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 CODE:**

**1. Retrieve customer details**

Familiarize yourself with the Customer table by writing a Transact-SQL query that retrieves 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.

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.

SELECT SalesPerson, ISNULL(Title, '') + FirstName AS CustomerName, Phone

FROM SalesLT.Customer;

**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.

SELECT CAST(CustomerID AS varchar(5)) + ':' + CompanyName as CustomerDetails

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).

SELECT CAST(SalesOrderNumber AS varchar(5)) + '(' + STR(RevisionNumber,7, 1) + ')', CONVERT(VARCHAR(19),OrderDate,102) as OrderDate

FROM SalesLT.SalesOrderHeader;

**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.

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

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.

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.

UPDATE SalesLT.SalesOrderHeader

SET ShipDate = NULL

WHERE SalesOrderID > 71899;

SELECT CAST(SalesOrderID AS varchar(5)) , OrderDate, ShipDate,

CASE

WHEN ShipDate IS NULL THEN 'Awaiting Shipment'

ELSE 'Shipped'

END AS ShippingStatus

FROM SalesLT.SalesOrderHeader;

**Lab 2: DISTINCT, TOP, PERCENT, INDEXING, IN, LIKE**

**################ 222222222222222 ########################**

**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;

SELECT DISTINCT Name, rowguid, modifiedDate

FROM SalesLT.ProductCategory

ORDER BY Name ASC;

**--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;

SELECT TOP 3 Name, ListPrice

FROM SalesLT.Product

ORDER BY ListPrice DESC;

**--Display the first ten products by product number**

SELECT Name, ListPrice, ProductNumber

FROM SalesLT.Product

ORDER BY ProductNumber

OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;

**-- First 10 Products**

SELECT TOP 10 Name, ListPrice, ProductNumber

FROM SalesLT.Product

ORDER BY ProductNumber;

**-- Last 10 Products**

SELECT TOP 10 Name, ListPrice, ProductNumber

FROM SalesLT.Product

ORDER BY ProductNumber DESC;

**-- Last 10 Products by list price**

SELECT TOP 10 Name, ListPrice, ProductNumber

FROM SalesLT.Product

ORDER BY ListPrice DESC;

**--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;

**--Retrieve Heaviest Products**

SELECT TOP 10 PERCENT Name, Weight

FROM SalesLT.Product ORDER BY Weight DESC;

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

SELECT Name, Weight

FROM SalesLT.Product

ORDER BY Weight DESC

OFFSET 10 ROWS FETCH NEXT 100 ROWS ONLY;

**Filtering with predicates:**

--List information about product model 6

SELECT Name, Color, Size

FROM SalesLT.Product

WHERE ProductModelID = 6;

**-- Products begin with M, H and S**

SELECT Name, Weight, Color

FROM SalesLT.Product

WHERE Name LIKE 'M%' or Name LIKE'H%' OR NAME LIKE 'S%'

Order BY Weight DESC

OFFSET 0 ROW FETCH NEXT 30 ROWS ONLY

**--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 , ProductNumber

FROM SalesLT.Product

WHERE ProductNumber LIKE 'FR-\_[0-9][0-9]\_-[0-9][0-9]';

**-- Charecter Match**

SELECT \* FROM SalesLT.ProductModel;

SELECT NAME

From SalesLT.ProductModel

WHERE NAME LIKE '\_\_\_\_-%' OR NAME LIKE '\_\_ %';

**--Find products that have no sell end date**

SELECT Name

FROM SalesLT.Product

WHERE SellEndDate IS NOT NULL;

SELECT NAME , SellEndDate

FROM SalesLT.Product

WHERE SellEndDate IS NULL ;

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

SELECT Name, SellEndDate

FROM SalesLT.Product

WHERE SellEndDate BETWEEN '2006/1/1' AND '2006/12/31';

SELECT \* FROM SalesLT.SalesOrderHeader

WHERE SubTotal BETWEEN 10000 AND 50000;

WHERE Freight BETWEEN 1000 AND 2000;

**--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);

**--Retrieve Specific Products by Product Number**

SELECT ProductNumber, Name, ListPrice

FROM SalesLT.Product

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

--Retrieve Products by Color and Size

SELECT ProductNumber, Name

FROM SalesLT.Product

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

LAB:

**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.

SELECT DISTINCT City, StateProvince FROM SalesLT.Address;

SELECT DISTINCT City FROM SalesLT.Address;

SELECT DISTINCT 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.

**--Top 10 product by weight**

SELECT Name, Weight, ProductNumber

FROM SalesLT.Product

ORDER BY Weight DESC

OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;

SELECT TOP 10 Weight, NAME, ProductNumber

FROM SalesLT.Product

Order by Weight DESC;

**-- Top 10 Percent**

SELECT TOP 10 PERCENT Name, Weight

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.

SELECT Name, Weight, ProductNumber

FROM SalesLT.Product

ORDER BY Weight DESC

OFFSET 10 ROWS FETCH NEXT 100 ROWS ONLY;

**-- Selecting top 100**

SELECT TOP 100 Weight, Name, ProductNumber

FROM SalesLT.Product

ORDER BY Weight DESC;

**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.

SELECT Name, color, size, ProductID, ProductModelID

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'.

SELECT Name, ProductID, ProductNumber, Color, Size

FROM SalesLT.Product

WHERE (Color LIKE 'Black' or Color LIKE 'Red' OR Color LIKE 'White') and (Size LIKE 'S' or Size LIKE 'M');

SELECT ProductNumber, Name, Color, Size

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-'.

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'.

SELECT ProductNumber, Name, ListPrice

FROM SalesLT.Product

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

**Lab 3: JOIN**

################ 33333333333333 ########################

**Inner Joins:**

--Basic inner join

• 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.

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;

SELECT P.Name, P.ProductID, OD.OrderQty, SH.OrderDate

FROM SalesLT.Product as P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

JOIN SalesLT.SalesOrderHeader as SH

ON OD.SalesOrderID = SH.SalesOrderID;

SELECT P.Name As ProductName, C.Name AS Category

FROM SalesLT.Product AS P

INNER JOIN SalesLT.ProductCategory AS C

ON P.ProductCategoryID = C.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.SalesOrderID, oh.SalesOrderNumber,od.SalesOrderDetailID,

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,p.ListPrice , 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 < 0.5 \* p.ListPrice --Note multiple predicates

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

**Outer Joins:**

• 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.

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

• 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.

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

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, c.CustomerID

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;

NOTE: THE Query below doesn’t work; for SalesOrderNumber IS NULL (it can compare 2 fields)

--Return only customers who haven't purchased anything

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

FROM SalesLT.Customer AS c

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID and oh.SalesOrderNumber IS NULL

ORDER BY c.CustomerID;

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

FROM SalesLT.Customer AS c

LEFT OUTER JOIN SalesLT.SalesOrderHeader AS oh

ON c.CustomerID = oh.CustomerID

Where c.CustomerID < 50

**--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:**

• 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.

**--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

ORDER BY FirstName;

SELECT \* FROM SalesLT.Customer;

**Self Joins:**

• 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.

**--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 MAX(EmployeeID) FROM SalesLT.Employee WHERE ManagerID IS NULL)

WHERE ManagerID IS NULL AND EmployeeID <

(SELECT MAX(EmployeeID) FROM SalesLT.Employee WHERE ManagerID IS NULL);

GO

DELETE From SalesLT.Employee

Where EmployeeName = 'adventure-works\david8'

DELETE FROM SalesLT.Employee

WHERE ManagerID = 3

DELETE FROM SalesLT.Employee;

SELECT \* FROM SalesLT.employee;

SELECT \* FROM SalesLT.Rockhurst;

CREATE TABLE SalesLT.Rockhurst

(FacultyID int IDENTITY PRIMARY KEY NOT NULL,

EmployeeName nvarchar(100) NOT NULL,

Age int,

Salary int,

CourseTitle nvarchar(200) NOT NULL,

TimeLog datetime NOT NULL DEFAULT GETDATE(),

Notes nvarchar(300) NULL

);

GO

INSERT INTO SalesLT.Rockhurst ( EmployeeName, Age, Salary, CourseTitle, Timelog, Notes)

VALUES

('Raju', 55, 95000, 'Linear Models', DEFAULT, 'Very Good');

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;

LAB:

**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.

SELECT C.CompanyName, OH.SalesOrderID, OH.TotalDue

FROM SalesLT.Customer AS C

JOIN SalesLT.SalesOrderHeader as OH

ON C.CustomerID = OH.CustomerID;

SELECT C.companyName, OH.SalesOrderID, OH.TotalDue

FROM SalesLT.Customer as C

JOIN SalesLT.SalesOrderHeader as OH

ON C.CustomerID = OH.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

SELECT C.CompanyName, OH.SalesOrderID, OH.TotalDue,

A.AddressLine1 + ISNULL (A.AddressLine2,'') + '-' + A.City + A.StateProvince as ADDRESS

FROM SalesLT.Customer AS C

JOIN SalesLT.SalesOrderHeader as OH

ON C.CustomerID = OH.CustomerID

JOIN SalesLT.CustomerAddress as CA

ON C.CustomerID = CA.CustomerID

Left JOIN SalesLT.Address as A

ON CA.AddressID = A.AddressID;

**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.

SELECT C.CompanyName, C.FirstName + C.LastName as Contact, OH.SalesOrderID, OH.TotalDue

FROM SalesLT.Customer AS C

Left JOIN SalesLT.SalesOrderHeader as OH

ON C.CustomerID = OH.CustomerID

ORDER BY OH.TotalDue 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.

SELECT C.CustomerID, C.CompanyName,ISNULL(C.MiddleName,'') AS MiddleName , C.FirstName, C.LastName, C.Phone, A.AddressID

FROM SalesLT.Customer AS C

LEFT JOIN SalesLT.CustomerAddress AS A

ON C.CustomerID = A.CustomerID

WHERE A.AddressID IS NOT NULL

SELECT C.CustomerID, C.CompanyName, C.FirstName + C.LastName as Contact,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).

-- Products Without 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 OH.SalesOrderID = OD.SalesOrderID

FULL JOIN SalesLT.Product as P

ON OD.ProductID = P.ProductID

WHERE OH.SalesOrderID IS NULL

**-- Products with 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 OH.SalesOrderID = OD.SalesOrderID

FULL JOIN SalesLT.Product as P

ON OD.ProductID = P.ProductID

WHERE OH.SalesOrderID IS NOT NULL

CREATE DATABASE testDB;

SHOW DATABASES;

################ 444444444444444444 ########################

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

• 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:**

**• Use UNION to combine the rowsets returned by mulitple queries.**

-- 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

SELECT \* FROM SalesLT.Employees

SELECT \* FROM SalesLT.Customers

-- Union example # Discards common item

SELECT FirstName, LastName

FROM SalesLT.Employees

UNION

SELECT FirstName, LastName

FROM SalesLT.Customers

ORDER BY LastName;

**-- Includes common item**

SELECT FirstName, LastName

FROM SalesLT.Employees

UNION ALL

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;

**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’.

SELECT C.CompanyName, A.AddressLine1, A.City,

CASE CA.AddressType

WHEN 'Main Office' THEN 'Billing'

ELSE CA.AddressType

END 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’.

SELECT C.CompanyName, A.AddressLine1, A.City,

CASE CA.AddressType

WHEN 'Shipping' THEN 'Shipping'

ELSE CA.AddressType

END 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.

SELECT C.CompanyName, A.AddressLine1, A.City,

CASE CA.AddressType

WHEN 'Main Office' THEN 'Billing'

ELSE CA.AddressType

END 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

SELECT C.CompanyName, A.AddressLine1, A.City,

CASE CA.AddressType

WHEN 'Shipping' THEN 'Shipping'

ELSE CA.AddressType

END 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'

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.

SELECT C.CompanyName, A.AddressLine1, CA.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

EXCEPT

SELECT C.CompanyName, A.AddressLine1, CA.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'

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.

SELECT C.CompanyName, A.AddressLine1, CA.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

INTERSECT

SELECT C.CompanyName, A.AddressLine1, CA.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'

####################### 55555555555555 ####################

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

Functions:

**-- Scalar functions**

SELECT YEAR(SellStartDate) SellStartYear, ProductID, Name

FROM SalesLT.Product

ORDER BY SellStartYear;

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

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

ProductID, Name

FROM SalesLT.Product

ORDER BY SellStartYear;

SELECT DATEDIFF(yy,SellStartDate, GETDATE()) AS 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;

**-** **IF function**

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

FROM SalesLT.Product;

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

FROM SalesLT.Product;

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

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 SalesOrderID, Status, SubTotal, TotalDue,

Rank() OVER(ORDER BY SubTotal DESC) as RankByTotal

FROM SalesLT.SalesOrderHeader AS OH

ORDER BY RankByTotal;

**----- RANK**

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';

SELECT P.ProductID, P.ListPrice, c.Name as ProductCategory

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON P.ProductCategoryID = C.ProductCategoryID

SELECT C.Name as ProductCategory, Count(P.ProductID) as TotalTypes,

AVG(P.ListPrice) as Average

FROM SalesLT.Product AS p

JOIN SalesLT.ProductCategory AS c

ON P.ProductCategoryID = C.ProductCategoryID

GROUP BY C.Name

ORDER BY C.Name

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 5:

1. Retrieve the name and approximate weight of each product

Write a query to return the product ID of each product, together with the product name

formatted as upper case and a column named ApproxWeight with the weight of each product

rounded to the nearest whole unit.

SELECT P.ProductID, UPPER(P.name) as ProductName, ROUND(P.Weight,0) as ApproxWeight

FROM SalesLT.Product as P

2. Retrieve the year and month in which products were first sold

Extend your query to include columns named SellStartYear and SellStartMonth containing

the year and month in which Adventure Works started selling each product.

The month should be displayed as the month name (for example, ‘January’).

SELECT DATEPART(yyyy,SellStartDate) AS SellStartYear,

DATENAME(month, SellStartDate) as SellStartMonth,

P.ProductID, UPPER(P.name) as ProductName,

ROUND(P.Weight,0) as ApproxWeight

FROM SalesLT.Product as P

3. Extract product types from product numbers

Extend your query to include a column named ProductType that contains the leftmost

two characters from the product number.

SELECT LEFT(ProductNumber,2) AS ProductType, DATEPART(yyyy,SellStartDate) AS SellStartYear,

DATENAME(month, SellStartDate) as SellStartMonth,

P.ProductID, UPPER(P.name) as ProductName,

ROUND(P.Weight,0) as ApproxWeight

FROM SalesLT.Product as P

4. Retrieve only products with a numeric size

Extend your query to filter the product returned so that only products with a

numeric size are included.

SELECT Size, LEFT(ProductNumber,2) AS ProductType, DATEPART(yyyy,SellStartDate) AS SellStartYear,

DATENAME(month, SellStartDate) as SellStartMonth,

P.ProductID, UPPER(P.name) as ProductName,

ROUND(P.Weight,0) as ApproxWeight

FROM SalesLT.Product as P

WHERE ISNUMERIC(size) = 1;

**Challenge 2: Rank Customers by Revenue**

The sales manager would like a list of customers ranked by sales.

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

1. Retrieve companies ranked by sales totals

Write a query that returns a list of company names with a ranking of their place in a

list of highest TotalDue values from the SalesOrderHeader table.

SELECT C.CompanyName,

OH.TotalDue as TotalDue,

RANK() OVER(ORDER BY OH.TotalDue DESC) AS RankByDue

FROM SalesLT.Customer as C

JOIN SalesLT.SalesOrderHeader as OH

ON C.CustomerID = OH.CustomerID;

**Challenge 3: Aggregate Product Sales**

The product manager would like aggregated information about product sales.

Tip: Review the documentation for the GROUP BY clause in the Transact-SQL Reference.

1. Retrieve total sales by product

Write a query to retrieve a list of the product names and the total revenue calculated as

the sum of the LineTotal from the SalesLT.SalesOrderDetail table, with the results sorted

in descending order of total revenue.

SELECT P.name as ProductName, SUM(OD.LineTotal) as Revenue,

RANK() OVER(ORDER BY SUM(OD.LineTotal) DESC) AS RankByDue

FROM SalesLT.Product as P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

GROUP BY P.Name

ORDER BY Revenue DESC;

2. Filter the product sales list to include only products that cost over $1,000

Modify the previous query to include sales totals for products that have a list price of

more than $1000.

SELECT P.name as ProductName, SUM(OD.LineTotal) as Revenue,

SUM(OD.LineTotal)/COUNT(OD.LineTotal) as UnitPrice

FROM SalesLT.Product as P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

WHERE OD.LineTotal > 1000

GROUP BY P.Name

ORDER BY Revenue DESC

HAVING name BETWEEN ‘F’ AND ‘L';

3. Filter the product sales groups to include only total sales over $20,000

Modify the previous query to only include only product groups with a total sales value

greater than $20,000.

SELECT P.name as ProductName, SUM(OD.LineTotal) as Revenue,

SUM(OD.LineTotal)/COUNT(OD.LineTotal) as UnitPrice

FROM SalesLT.Product as P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

WHERE OD.LineTotal > 1000

GROUP BY P.Name

HAVING SUM(OD.LineTotal) > 20000

ORDER BY Revenue DESC;

################ 6666666666666666 ########################

**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 >1466.01

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, ProductID

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)

**------ Final Correlated Subquery**

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;

-- Returns highest Line total for each Sales order !!

CREATE FUNCTION SalesLT.udfMaxLineTotal (@SalesOrderID int)

RETURNS TABLE

AS

RETURN

SELECT SalesOrderID, Max(LineTotal) as MaxLineTotal FROM

SalesLT.SalesOrderDetail

Where SalesOrderID = @SalesOrderID

GROUP BY SalesOrderID;

SELECT \* FROM SalesLT.SalesOrderDetail AS SOH

CROSS APPLY SalesLT.udfMaxLineTotal (SOH.SalesOrderID) as MLT

WHERE SOH.LineTotal = MLT.MaxLineTotal

ORDER BY SOH.SalesOrderID;

LAB 6:

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. Tip: Review the documentation for subqueries in Subquery Fundamentals.

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.

-- Scalar Subquery as a part of WHERE condition

SELECT ProductID, Name,ListPrice FROM SalesLT.Product

WHERE ListPrice > (SELECT AVG(UnitPrice) FROM SalesLT.SalesOrderDetail)

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.

-- Multivalued Subquery or Vector Subquery for WHERE Statement

SELECT ProductID, Name,ListPrice FROM SalesLT.Product

WHERE ProductNumber in

(SELECT DISTINCT ProductNumber

FROM SalesLT.Product AS P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

WHERE (P.ListPrice > 100 AND OD.UnitPrice <100))

ORDER BY ProductID;

3. 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.

--Retrieve cost, list price, and average selling price for each product

SELECT P.ProductID, P.StandardCost, P.ListPrice,

(Select Avg(OD.UnitPrice)

FROM SalesLT.SalesOrderDetail AS OD

WHERE OD.ProductID = P.ProductID) as AvgSellingPrice

FROM SalesLT.Product as P

ORDER BY P.ProductID;

4. 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.

SELECT P.ProductID, P.StandardCost, P.ListPrice,

(Select Avg(OD.UnitPrice)

FROM SalesLT.SalesOrderDetail AS OD

WHERE OD.ProductID = P.ProductID) as AvgSellingPrice

FROM SalesLT.Product as P

WHERE P.StandardCost > (Select Avg(OD.UnitPrice)

FROM SalesLT.SalesOrderDetail AS OD

WHERE OD.ProductID = P.ProductID)

ORDER BY P.ProductID;

Challenge 2: Retrieve Customer Information

The AdventureWorksLT database includes a table-valued user-defined function named

dbo.ufnGetCustomerInformation. You must use this function to retrieve details of

customers based on customer ID values retrieved from tables in the database. Tip:

Review the documentation for the APPLY operator in Using APPLY.

1. Retrieve customer information for all sales orders

Retrieve the sales order ID, customer ID, first name, last name, and total due for all

sales orders from the SalesLT.SalesOrderHeader table and the dbo.ufnGetCustomerInformation

function.

2. Retrieve customer address information

Retrieve the customer ID, first name, last name, address line 1 and city for all

customers from the SalesLT.Address and SalesLT.CustomerAddress tables, and the

dbo.ufnGetCustomerInformation function.

**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

SELECT \* FROM SalesLT.vCustomerAddress

**-- Query the view**

SELECT CustomerID, City

FROM SalesLT.vCustomerAddress

CREATE VIEW SalesLT.vProductOrders

AS

SELECT P.ProductID, Name, OD.OrderQty, OD.UnitPrice

FROM SalesLT.Product as P

JOIN SalesLT.SalesOrderDetail as OD

ON P.ProductID = OD.ProductID

SELECT \* FROM SalesLT.vProductOrders;

SELECT \* 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 - Table valued function**

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 : Common table expression**

--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 – INSERT, UPDATE AND Delete

#######################################################################

**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'),

('adventure-works\josé2', 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

(11, '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;

## SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE table\_name;

# **SQL ALTER TABLE Statement**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

## ALTER TABLE - ADD Column

ALTER TABLE table\_name  
ADD column\_name datatype;

**Example:**

ALTER TABLE Persons  
ADD DateOfBirth date;

## ALTER TABLE - DROP COLUMN

ALTER TABLE table\_name  
DROP COLUMN column\_name;

Example:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

**ALTER TABLE - ALTER/MODIFY COLUMN**

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype;

Example:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

# **SQL NOT NULL Constraint**

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);

**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