T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ

COM5041 - DATABASE DESIGN AND DEVELOPMENT

LAB 06 – Advanced SQL

After completing this Lab, you will be able to

- More Complex SQL Retrieval Queries
- Use the JOIN operator to return data from multiple tables
- Perform aggregating, grouping, and windowing
- Write the Nested Queries in SQL
- Limit the data returned in your result set

PROCEDURE 1 - Using the JOIN operator to return data from multiple tables

It is highly unlikely that your queries will reference just one table most of the time, you can be required to return data from multiple tables. To do this, you use the JOIN operator. While there are several types of JOINs, you will learn on the three most used:

- INNER
- LEFT OUTER
- RIGHT OUTER

Step 1 - INNER JOIN is an equality match between two or more tables. For example, assume you have a table that contains products and another that contains sales, and you want to find only the products that have been sold. Basically, you are looking for the intersection of the two tables on some value Figure 1 illustrates the intersection. The shaded section of Figure 1 depicts the rows that will be returned from a query that would join the Sales and Product tables.

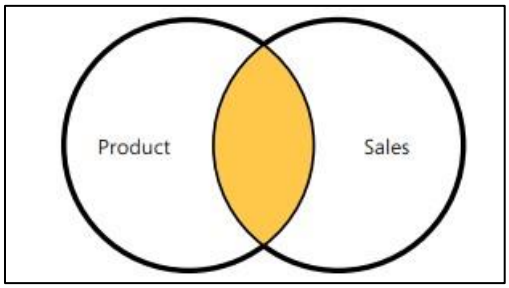

Figure 1: An INNER JOIN intersection.

The INNER JOIN keywords have been included, which allows you to specify a second table in the query. The INNER JOIN or any JOIN must be coupled with the ON keyword. In the ON clause, you specify which column or columns will be used to connect (JOIN) the two tables. The key to successfully joining any two tables is to identify their intersecting data, which is commonly aligned across primary key and foreign key relationships.

- The table in the FROM clause should include a column with values that exist in the table you plan on joining. In this case, you would like to include OrderQty and UnitPrice in the result set. To accomplish this, you must reference a second table in the query, as illustrated in the following query:

```
USE AdventureWorks2019;
SELECT
p.ProductID,
p.Name AS ProductName,
sd.OrderQty,
sd.UnitPrice
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
```

** Execute the query and review the results.

Results | Messages

	ProductID	ProductName	OrderQty	UnitPrice
1	776	Mountain-100 Black, 42	1	2024,994
2	777	Mountain-100 Black, 44	3	2024,994
3	778	Mountain-100 Black, 48	1	2024,994
4	771	Mountain-100 Silver, 38	1	2039,994
5	772	Mountain-100 Silver, 42	1	2039,994
6	773	Mountain-100 Silver, 44	2	2039,994
7	774	Mountain-100 Silver, 48	1	2039,994
8	714	Long-Sleeve Logo Jersey, M	3	28,8404

Step 2 - There are two basic types of OUTER JOINs: LEFT and RIGHT. They both provide very similar functionality, but there is a slight difference that depends on the order of the tables in the query. The LEFT JOIN keyword returns all records from the left table (Product), and the matching records from the right table (Sales). The result is 0 records from the right side if there is no match. Figure 2 illustrates a LEFT OUTER JOIN.

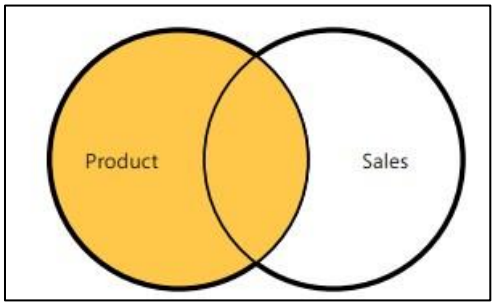

Figure 2: A LEFT OUTER JOIN intersection.

- If you want to retrieve a list of products regardless of their existence in the Sales.SalesOrderDetail table, a LEFT OUTER JOIN should be your choice.

```
SELECT
p.ProductID,
sd.ProductID,
p.Name AS ProductName,
sd.OrderQty,
sd.UnitPrice
FROM Production.Product AS p
LEFT OUTER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
```

** Execute the query and review the results.

Results | Messages

	ProductID	ProductID	ProductName	OrderQty	UnitPrice
12...	712	712	AWC Logo Cap	1	8,99
12...	878	878	Fender Set - Mountain	1	21,98
12...	879	879	All-Purpose Bike Stand	1	159,00
12...	712	712	AWC Logo Cap	1	8,99
12...	355	NULL	Guide Pulley	NULL	NULL
12...	378	NULL	Hex Nut 17	NULL	NULL
12...	401	NULL	HL Hub	NULL	NULL

Note: If you scroll down the result set, you should start to see NULL values in those columns that are part of the Sales.SalesOrderDetail table. This is a direct result of using an OUTER JOIN. Recall the shaded area of Figure 2 illustrating the result set of a LEFT OUTER JOIN. The rows returned by the query are what you should expect. Not only are the products associated with sales returned, but so are those without sales

Step 3- The RIGHT JOIN keyword returns all records from the right table (Sales), and the matching records from the left table (Product). The result is 0 records from the left side if there is no match. Figure 3 illustrates a RIGHT OUTER JOIN.

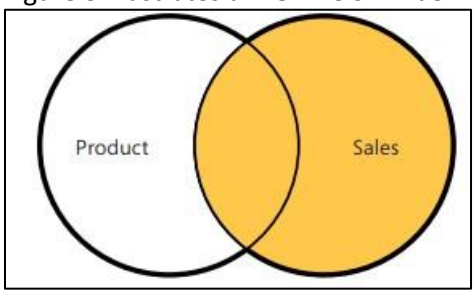

Figure 3: A RIGHT OUTER JOIN intersection.

- If you are trying to retrieve all sales, whether or not they are associated with a product, you should choose a RIGHT OUTER JOIN.

```
SELECT
p.ProductID,
sd.ProductID,
p.Name AS ProductName,
sd.OrderQty,
sd.UnitPrice
FROM Production.Product AS p
RIGHT OUTER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
```

** Execute the query and review the results.

Results | Messages

	ProductID	ProductID	ProductName	OrderQty	UnitPrice
1	776	776	Mountain-100 Black, 42	1	2024,994
2	777	777	Mountain-100 Black, 44	3	2024,994
3	778	778	Mountain-100 Black, 48	1	2024,994
4	771	771	Mountain-100 Silver, 38	1	2039,994
5	772	772	Mountain-100 Silver, 42	1	2039,994
6	773	773	Mountain-100 Silver, 44	2	2039,994
7	774	774	Mountain-100 Silver, 48	1	2039,994
8	714	714	Long-Sleeve Logo Jersey, M	3	28,8404
9	716	716	Long-Sleeve Logo Jersey, XL	1	28,8404
10	709	709	Mountain Bike Socks, M	6	5,70
11	712	712	AWC Logo Cap	2	5,1865
12	711	711	Sport-100 Helmet, Blue	4	20,1865

PROCEDURE 2 - Perform aggregating, grouping, and windowing

Step 1 - The most common aggregation that is performed is summation, which adds all the values of a given data set. The summation of data is supported through T-SQL's SUM function: SUM([ALL | DISTINCT] expression)

The ALL keyword will apply the aggregation to every value in the result, while DISTINCT will aggregate only the unique values. The ALL keyword is used by default; therefore, you do not need to specify it as part of the query. The supplied expression must be a numeric data type that can be a constant, variable, column, or function.

- This query presents the total due for all purchase orders that have been placed.

```
USE AdventureWorks2019;
SELECT
SUM(poh.TotalDue) AS TotalDue
FROM Purchasing.PurchaseOrderHeader poh
```

**Execute the query and review the results.

Results | Messages

	TotalDue
1	70479332.6383

- Add three new aggregations to the query that average the total due and count the number of employees in two different ways:

```
USE AdventureWorks2019;
SELECT
SUM(poh.TotalDue) AS [Total Due],
AVG(poh.TotalDue) AS [Average Total Due],
COUNT(poh.EmployeeID) [Number Of Employees],
COUNT(DISTINCT poh.EmployeeID) [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
```

** Execute the query and review the results.

Results | Messages

	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	70479332.6383	17567.1317	4012	12

- Average Total Due uses the AVG function to calculate the average over the entire result set. It sums the total due, counts the number of rows, divides the two values, and then returns the average.
- Number Of Employees uses the COUNT function to every employee in the result, including duplicate values of the supplied column, EmployeeID.
- Distinct Number Of Employees uses the COUNT function but includes the DISTINCT keyword to ensure that duplicates are ignored.

Step 2 - Using T-SQL syntax, you can aggregate the data with built-in aggregate scalar functions and group the data using the GROUP BY keyword to perform these operations.

- By including the GROUP BY clause with the Name column from the ShippingMethod table, the query was able to provide aggregations for each individual shipping method that was used.

```
USE AdventureWorks2019;
SELECT
sm.Name AS ShippingMethod,
SUM(poh.TotalDue) AS [Total Due],
AVG(poh.TotalDue) AS [Average Total Due],
COUNT(poh.EmployeeID) [Number Of Employees],
COUNT(DISTINCT poh.EmployeeID) [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
ON poh.ShipMethodID = sm.ShipMethodID
GROUP BY sm.Name
```

** Execute the query and review the results.

Results | Messages

	ShippingMethod	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	XRQ - TRUCK GROUND	3330909,2897	5655,194	589	12
2	ZY - EXPRESS	14874601,7677	22709,3156	655	12
3	OVERSEAS - DELUXE	8002938,997	50018,3687	160	12
4	OVERNIGHT J-FAST	11965191,1871	11027,8259	1085	12
5	CARGO TRANSPORT 5	32305691,3968	21211,8787	1523	12

- Add an expression that derives the year from the OrderDate column in the PurchaseOrderHeader table:

```
USE AdventureWorks2019;
SELECT
sm.Name AS ShippingMethod,
YEAR(poh.OrderDate) AS OrderYear,
SUM(poh.TotalDue) AS [Total Due],
AVG(poh.TotalDue) AS [Average Total Due],
COUNT(poh.EmployeeID) AS [Number Of Employees],
COUNT(DISTINCT poh.EmployeeID) AS [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
ON poh.ShipMethodID = sm.ShipMethodID
GROUP BY
sm.Name,
YEAR(poh.OrderDate)
```

** Execute the query and review the results.

Results | Messages

	ShippingMethod	OrderYear	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	XRQ - TRUCK GROUND	2011	54219,7589	13554,9397	4	4
2	XRQ - TRUCK GROUND	2012	184006,9487	4842,2881	38	12
3	XRQ - TRUCK GROUND	2013	1153143,1232	6037,3985	191	12
4	XRQ - TRUCK GROUND	2014	1939539,4589	5448,1445	356	12
5	ZY - EXPRESS	2011	159651,6647	31930,3329	5	5
6	ZY - EXPRESS	2012	978723,4344	23302,9389	42	12
7	ZY - EXPRESS	2013	4563443,6692	21834,6587	209	12
8	ZY - EXPRESS	2014	9172782,9994	22989,431	399	12
9	OVERSEAS - DELUXE	2011	81233,7049	27077,9016	3	3
10	OVERSEAS - DELUXE	2012	326145,231	36238,359	9	5
11	OVERSEAS - DELUXE	2013	1939105,2554	39573,5766	49	11
12	OVERSEAS - DELUXE	2014	5656454,8057	57135,9071	99	12
13	OVERNIGHT J-FAST	2011	25923,9762	6480,994	4	4
14	OVERNIGHT J-FAST	2012	746999,1169	11318,1684	66	12
15	OVERNIGHT J-FAST	2013	4213890,0375	11450,7881	368	12
16	OVERNIGHT J-FAST	2014	6978378,0565	10785,7466	647	12

Step 3 - The HAVING clause behaves similarly to a SELECT statement. However, it can leverage aggregation. You can use it only with a SELECT statement, and it is typically used with a GROUP BY clause.

```
USE AdventureWorks2019;
SELECT
sm.Name AS ShippingMethod,
YEAR(poh.OrderDate) OrderYear,
SUM(poh.TotalDue) AS [Total Due],
AVG(poh.TotalDue) AS [Average Total Due],
COUNT(poh.EmployeeID) AS [Number Of Employees],
COUNT(DISTINCT poh.EmployeeID) AS [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
ON poh.ShipMethodID = sm.ShipMethodID
GROUP BY sm.Name,YEAR(poh.OrderDate)
HAVING SUM(poh.TotalDue) > 5000000
```

** Execute the query and review the results.

Results | Messages

	ShippingMethod	OrderYear	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	ZY - EXPRESS	2014	9172782,9994	22989,431	399	12
2	OVERSEAS - DELUXE	2014	5656454,8057	57135,9071	99	12
3	OVERNIGHT J-FAST	2014	6978378,0565	10785,7466	647	12
4	CARGO TRANSPORT 5	2013	10301524,0723	21023,5185	490	12
5	CARGO TRANSPORT 5	2014	19776389,2776	21403,0186	924	12

** Now instead of returning every row, the result is limited to only those shipping methods whose annual total due is greater than 5 million.

PROCEDURE 3 - Nested Queries in SQL

Step 1 - Subqueries are queries that are nested inside of another query or statement. They are permitted wherever SQL Server would allow an expression and are indicated by enclosing the subquery in parenthesis. For instance, using the sample AdventureWorks database, we might create a query to find any employees who have more vacation available than the average:

```
USE AdventureWorks2019;
SELECT
  BusinessEntityID,
  LoginID,
  JobTitle,
  VacationHours
FROM
  HumanResources.Employee E1
WHERE
  VacationHours > (SELECT
    AVG(VacationHours)
    FROM HumanResources.Employee E2)
```

Note: In that example, we used the subquery in the WHERE clause and it returned a single value.

** Execute the query and review the results.

Results | Messages

	BusinessEntityID	LoginID	JobTitle	VacationHours
1	1	adventure-works\ken0	Chief Executive Officer	99
2	7	adventure-works\dylan0	Research and Development Manager	61
3	8	adventure-works\diane1	Research and Development Engineer	62
4	9	adventure-works\gigi0	Research and Development Engineer	63
5	25	adventure-works\james1	Vice President of Production	64
6	27	adventure-works\jo0	Production Supervisor - WC60	80
7	40	adventure-works\jolynn0	Production Supervisor - WC60	82

Step 2 - A subquery can itself include one or more subqueries. Any number of subqueries can be nested in a statement. The following query finds the names of employees who are also sales persons.

```
USE AdventureWorks2019;
SELECT LastName, FirstName
FROM Person.Person
WHERE BusinessEntityID IN
    (SELECT BusinessEntityID
     FROM HumanResources.Employee
     WHERE BusinessEntityID IN
        (SELECT BusinessEntityID
         FROM Sales.SalesPerson))
```

** Execute the query and review the results.

Results | Messages

	LastName	FirstName
1	Jiang	Stephen
2	Blythe	Michael

PROCEDURE 4 - Limiting the data returned in your result set

Besides using a WHERE clause in your query, you have several other ways to limit the data returned in your result set. While there is a long list of methods and techniques you can use, SQL Server offers keywords that provide a very simplistic approach to limiting your result set.

Step 1- The TOP keyword limits the number of rows that are returned in a result to either a specific number of rows or a specific percentage of rows. TOP should always be used with the ORDER BY clause. In most cases, you will be looking for the highest or lowest set of values for a given column and sorting the data will provide you with that information.

- For example, if you want to return the top five sales in your Sales table, you add TOP (5) immediately following the keyword SELECT In addition, you include an ORDER BY clause specifying the column that contained the actual sales value for each row as the ordering column.

```
USE AdventureWorks2019;
SELECT TOP(5)
SalesOrderID,
OrderDate,
SalesOrderNumber,
TotalDue
FROM Sales.SalesOrderHeader
ORDER BY
TotalDue DESC
```

** Execute the query and review the results.

Results | Messages

	SalesOrderID	OrderDate	SalesOrderNumber	TotalDue
1	51131	2013-05-30 00:00:00.000	SO51131	187487,825
2	55282	2013-08-30 00:00:00.000	SO55282	182018,6272
3	46616	2012-05-30 00:00:00.000	SO46616	170512,6689
4	46981	2012-06-30 00:00:00.000	SO46981	166537,0808
5	47395	2012-07-31 00:00:00.000	SO47395	165028,7482

Step 2 - DISTINCT returns a unique or distinct list of values of each specified column in a SELECT statement. If there are any duplicate values in the list, all but one duplicate value will be removed. For example, if you execute the following query, the result returns a list of products with some of the product names repeated several times:

```
USE AdventureWorks2019;
SELECT
p.Name AS ProductName
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
```

** Execute the query and review the results.

Results | Messages

	ProductName
1	Sport-100 Helmet, Red
2	Sport-100 Helmet, Red
3	Sport-100 Helmet, Red
4	Sport-100 Helmet, Red
5	Sport-100 Helmet, Red
6	Sport-100 Helmet, Red
7	Sport-100 Helmet, Red
8	Sport-100 Helmet, Red
9	Sport-100 Helmet, Red
10	Sport-100 Helmet, Red
11	Sport-100 Helmet, Red
12	Sport-100 Helmet, Red
13	Sport-100 Helmet, Red

- By placing DISTINCT immediately following the SELECT keyword, you remove any duplicates from the list.

```
USE AdventureWorks2019;
SELECT DISTINCT
p.Name AS ProductName
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
```

** Execute the query and review the results.

Results | Messages

	ProductName
1	Sport-100 Helmet, Red
2	Sport-100 Helmet, Black
3	Mountain Bike Socks, M
4	Mountain Bike Socks, L
5	Sport-100 Helmet, Blue
6	AWC Logo Cap
7	Long-Sleeve Logo Jersey, S
8	Long-Sleeve Logo Jersey, M
9	Long-Sleeve Logo Jersey, L
10	Long-Sleeve Logo Jersey, XL

Step 3 - If you want to limit the result to only products that have not shipped, you add the WHERE clause. In this example, the WHERE clause needs to identify all the SalesOrderProduct rows that contain a NULL CarrierTrackingNumber. Use this query to return a distinct list of products that have not been shipped:

```
USE AdventureWorks2019;
SELECT DISTINCT
p.Name AS ProductName
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail sd
ON p.ProductID = sd.ProductID
WHERE
sd.CarrierTrackingNumber IS NULL
order by productname
```

** Execute the query and review the results.

Results | Messages

	ProductName
1	All-Purpose Bike Stand
2	AWC Logo Cap
3	Bike Wash - Dissolver
4	Classic Vest, L
5	Classic Vest, M
6	Classic Vest, S
7	Fender Set - Mountain
8	Half-Finger Gloves, L
9	Half-Finger Gloves, M
10	Half-Finger Gloves, S

Step 4 - Often you will have two SELECT statements that may need to be combined into one result for consumption by an application or end user. Using the UNION keyword, you can accomplish just that. UNION has two variations:

- Just UNION, which removes any duplicate rows in your result set.
- UNION ALL, which includes duplicates. If duplicates are possible, you should use UNION ALL; it is much faster because it does not have to include DISTINCT.
- Use this query to return a list of products that are black and silver:

```
USE AdventureWorks2019;
SELECT
Name AS ProductName
FROM Production.Product
WHERE
Color = 'Black'
UNION
SELECT
Name AS ProductName
FROM Production.Product
WHERE
Color = 'Silver'
```

** Execute the query and review the results.

Results | Messages

	ProductName
1	Chain
2	Chainring
3	Chainring Bolts
4	Chainring Nut
5	Freewheel
6	Front Brakes
7	Front Derailleur
8	Front Derailleur Cage
9	Front Derailleur Linkage
10	Full-Finger Gloves, L
11	Full-Finger Gloves, M

SUMMARY

The potential of the SELECT statement is limited only by your knowledge of the table and data relationships and how you can manipulate them with the SELECT syntax. In this LAB, you learned about the commonly used SELECT methods, which should provide a solid foundation for your T-SQL experimentation. You explored several methods that will assist you in joining data from multiple tables and then limiting the result. You learned to write to nested queries in SQL.

PROCEDURE 5 - ASSIGNMENT (Upload the solution to the CATs by your ID)

You can access your assignment at the time your section starts. Then, you can download the assignment from the Assignments section in CATS.