N.I.L.C.

Querying SQL Server Intermediate

Derek Pike

1

Introductions

- Name
- Title/function
- Job responsibility
- Database querying experience
- Your expectations for the course

2

Other Information

- Time table (approx):

  Start 9:30 am

  Break 10:45am – 11:00 am

  Lunch 12:00pm – 1:00pm

  Break 2:45pm – 3:00pm

  Finish about 4:30pm

3

Course Outline

- Working with multiple tables – understanding and using SQL Joins
- Using SQL Aliases
- Connecting multiple tables with Union
- Additional Filters, finding specific information.
- Working with SQL Functions to handle tasks
- Grouping information
- Copying data to new tables and temporary tables
- Introduction to Deleting and Updating data.

4

Basic SELECT Syntax

```
SELECT select_list
[ INTO new_table ]
[ FROM table_source ]
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

5

Go Deploy Labs

Labs can be access for 6 months

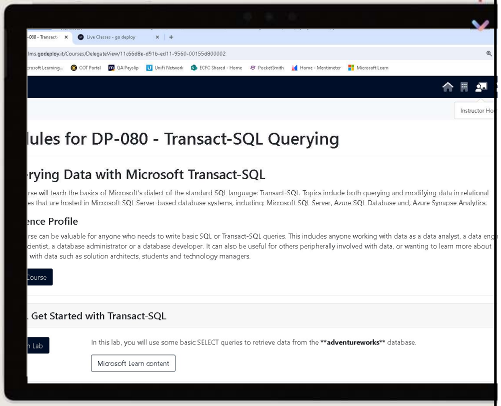
https://lms.godeploy.it

Signin, or register if you do not have an account

Redeem the lab key: **4RXSS9**

www.aka.ms/MyMicrosoftLearnProfile

https://lms.godeploy.it



lules for DP-080 - Transact-SQL Querying

rying Data with Microsoft Transact-SQL

rse will teach the basics of Microsoft's dialect of the standard SQL language: Transact-SQL. Topics include both querying and modifying data in relational es that are hosted in Microsoft SQL Server-based database systems, including: Microsoft SQL Server, Azure SQL Database and, Azure Synapse Analytics.

ence Profile

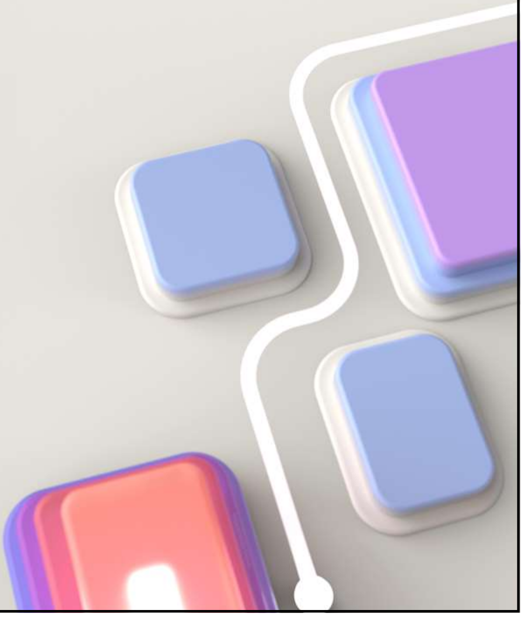
rse can be valuable for anyone who needs to write basic SQL or Transact-SQL queries. This includes anyone working with data as a data analyst, a data eng cientist, a database administrator or a database developer. It can also be useful for others peripherally involved with data, or wanting to learn more about with data such as solution architects, students and technology managers.

. Get Started with Transact-SQL

In this lab, you will use some basic SELECT queries to retrieve data from the **adventureworks** database.

6

https://learn.microsoft.com/en-us/training/modules/query-multiple-tables-with-joins/

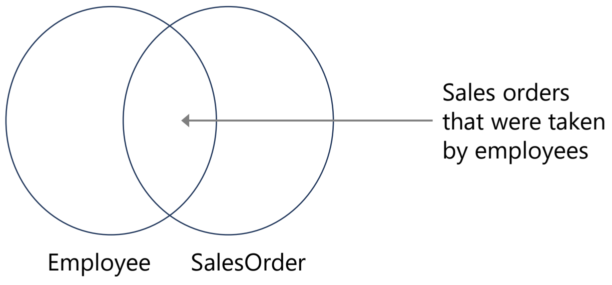


1: Using joins

7

Join concepts

It can help to think of the tables as sets in a Venn diagram



Combine rows from multiple tables by specifying matching criteria

Usually based on primary key – Foreign key relationships

For example, return rows that combine data from the **Employee** and **SalesOrder** tables by matching the **Employee.EmployeeID** primary key to the **SalesOrder.EmployeeID** foreign key

8

Join syntax

ANSI SQL-92

- Tables joined by JOIN operator in FROM clause
  - Preferred syntax

```
SELECT ...
FROM Table1 JOIN Table2
       ON <predicate>;
```

ANSI SQL-89

- Tables listed in FROM clause with join predicate in WHERE clause
  - Not recommended: can lead to accidental Cartesian products!

```
SELECT ...
FROM   Table1, Table2
WHERE  <predicate>;
```

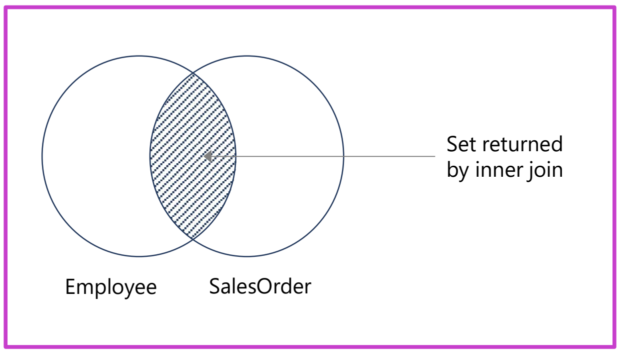
9

Inner joins

Return only rows where a match is found in both input tables

- Match rows based on criteria supplied in the join predicate
- If join predicate operator is =, also known as *equi-join*

```
SELECT emp.FirstName, ord.Amount
FROM HR.Employee AS emp
[INNER] JOIN Sales.SalesOrder AS ord
  ON emp.EmployeeID = ord.EmployeeID
```



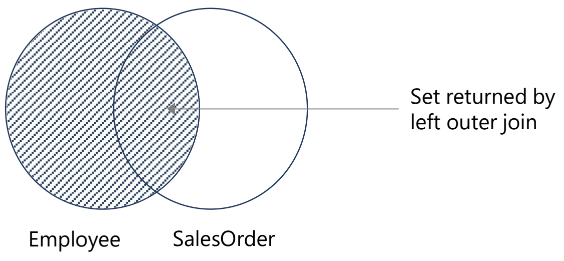
10

Outer joins

Return all rows from one table and any matching rows from second table

- Outer table's rows are "preserved"
  - Designated with LEFT, RIGHT, FULL keyword
  - All rows from preserved table output to result set
- Matches from inner table retrieved
- NULLs added in places where attributes do not match

```
SELECT emp.FirstName, ord.Amount
FROM HR.Employee AS emp
LEFT [OUTER] JOIN Sales.SalesOrder AS ord
  ON emp.EmployeeID = ord.EmployeeID;
```



11

Cross joins

Combine all rows from both tables

- All possible combinations output
- Logical foundation for inner and outer joins
  - Inner join starts with Cartesian product, adds filter
  - Outer join takes Cartesian output, filtered, adds back non-matching rows (with NULL placeholders)

Cartesian product output is typically undesired

- Some useful exceptions:
  - Table of numbers
  - Generating data for testing

Employee

EmployeeID	FirstName
1	Dan
2	Aisha

Product

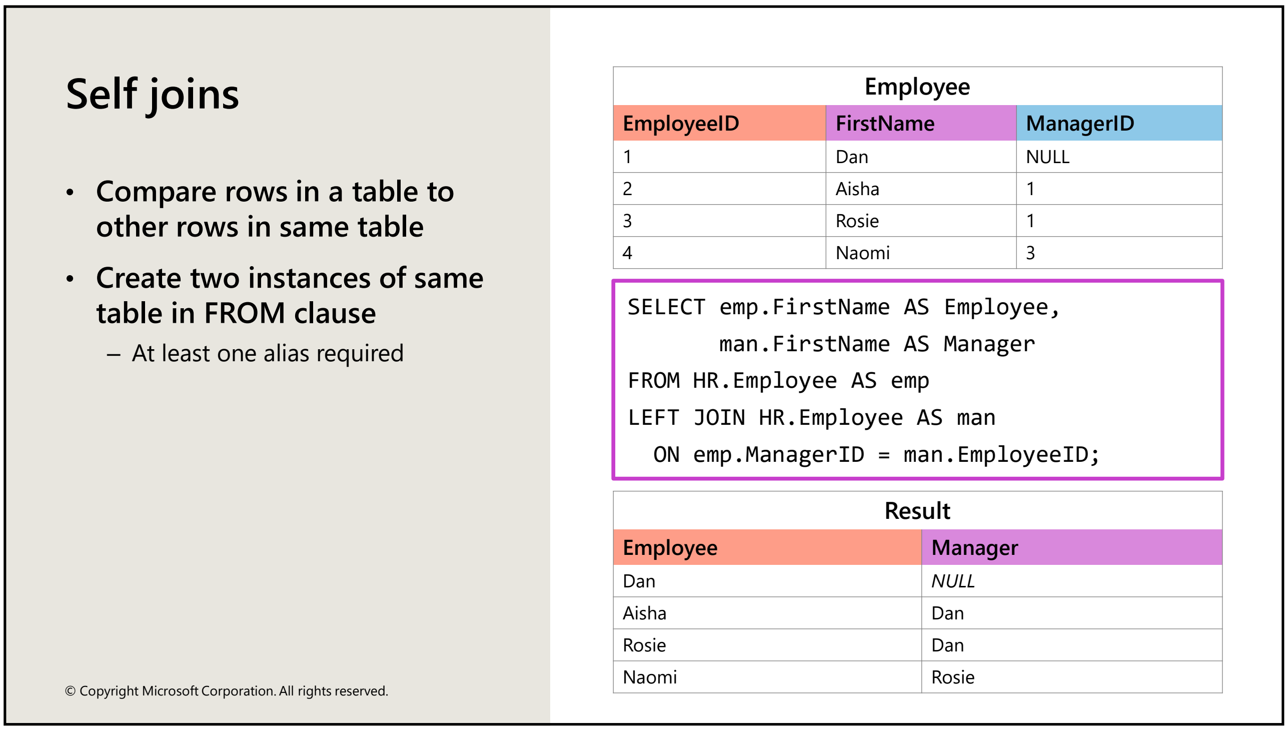
ProductID	Name
1	Widget
2	Gizmo

```
SELECT emp.FirstName, prd.Name
FROM HR.Employee AS emp
CROSS JOIN Production.Product AS prd;
```

Result

FirstName	Name
Dan	Widget
Dan	Gizmo
Aisha	Widget
Aisha	Gizmo

12



Self joins

- Compare rows in a table to other rows in same table
- Create two instances of same table in FROM clause
  - At least one alias required

Employee

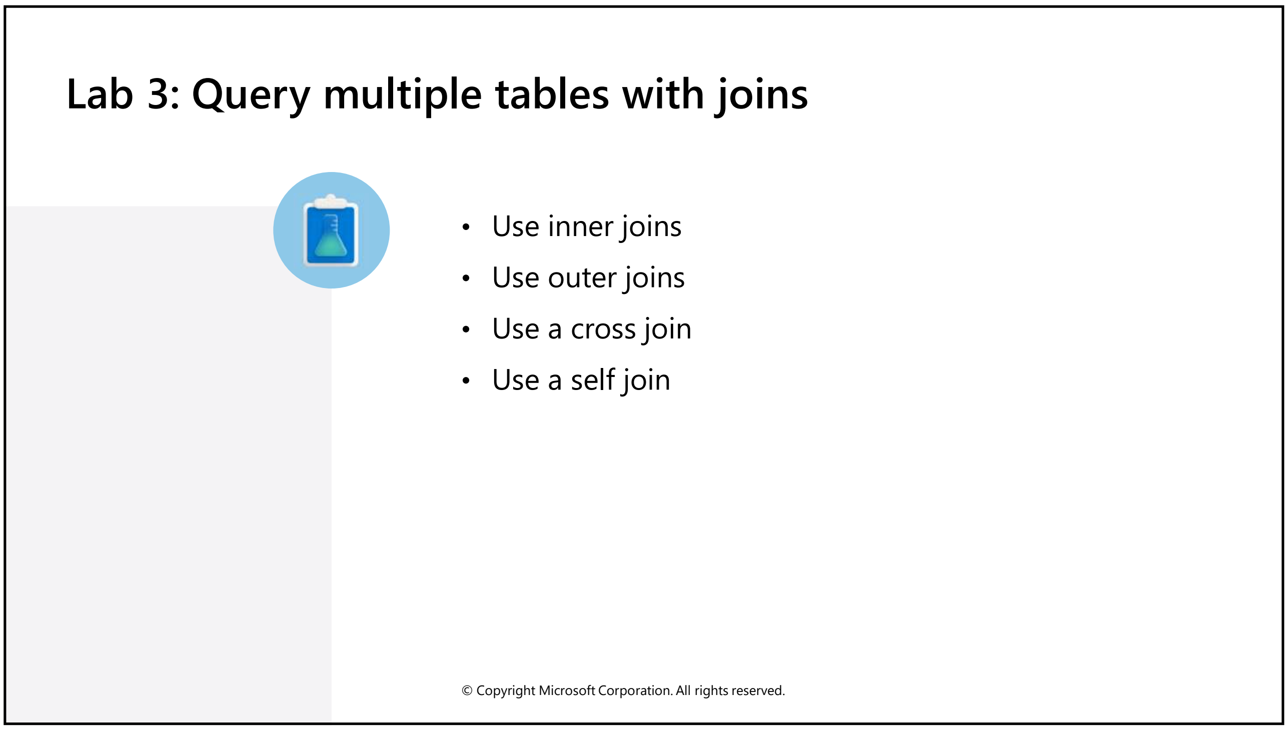
EmployeeID	FirstName	ManagerID
1	Dan	NULL
2	Aisha	1
3	Rosie	1
4	Naomi	3

```
SELECT emp.FirstName AS Employee,
       man.FirstName AS Manager
FROM HR.Employee AS emp
LEFT JOIN HR.Employee AS man
  ON emp.ManagerID = man.EmployeeID;
```

Result

Employee	Manager
Dan	*NULL*
Aisha	Dan
Rosie	Dan
Naomi	Rosie

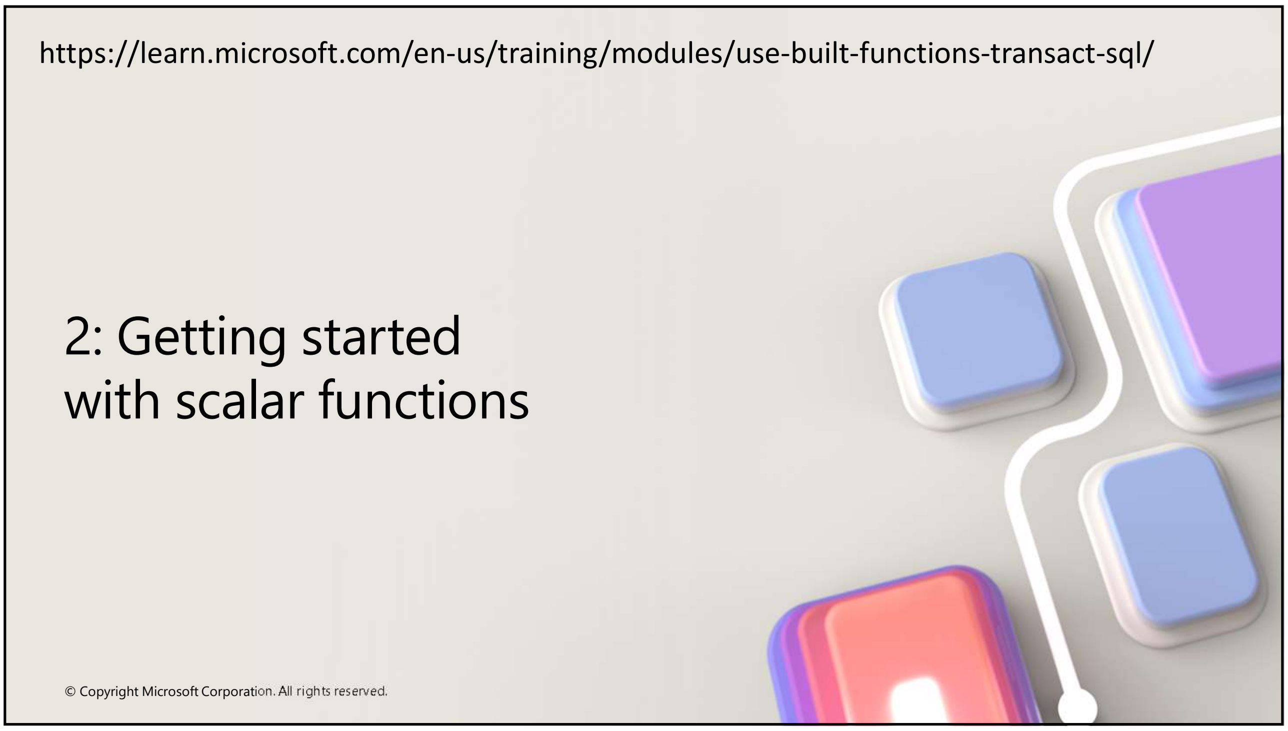
13



Lab 3: Query multiple tables with joins

- Use inner joins
- Use outer joins
- Use a cross join
- Use a self join

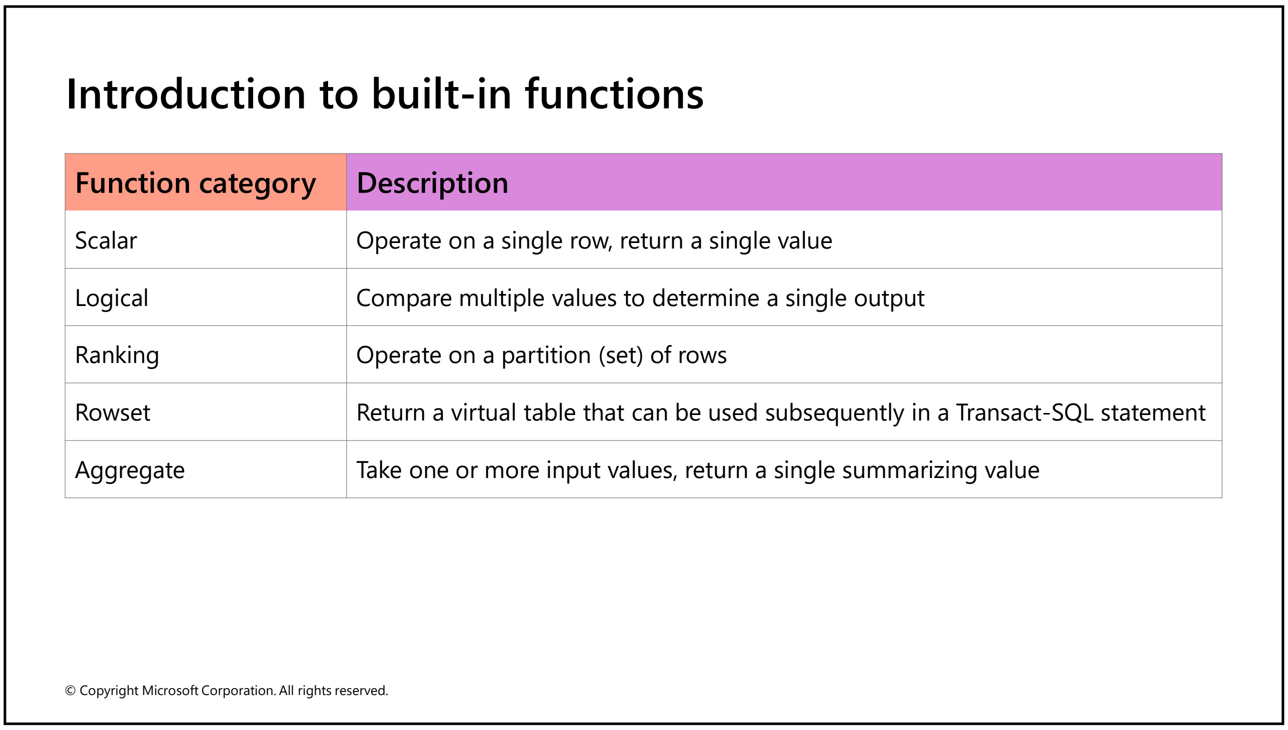
14



https://learn.microsoft.com/en-us/training/modules/use-built-functions-transact-sql/

2: Getting started with scalar functions

15



Introduction to built-in functions

Function category	Description
Scalar	Operate on a single row, return a single value
Logical	Compare multiple values to determine a single output
Ranking	Operate on a partition (set) of rows
Rowset	Return a virtual table that can be used subsequently in a Transact-SQL statement
Aggregate	Take one or more input values, return a single summarizing value

16

Scalar functions

Operate on elements from a single row as inputs, return a single value as output

- Return a single (scalar) value
- Can be used like an expression in queries
- May be deterministic or non-deterministic

```
SELECT UPPER(ProductName) AS Product,
       ROUND(ListPrice, 0) AS ApproxPrice,
       YEAR(SaleStartDate) AS SoldSince
FROM Production.Product;
```

Scalar function categories

- Configuration
- Conversion
- Cursor
- Date and Time
- Mathematical
- Metadata
- Security
- String
- System
- System Statistical
- Text and Image

17

Logical functions

Output is determined by comparative logic

IIF

- Evaluate logical expression, return first value if true and second value if false

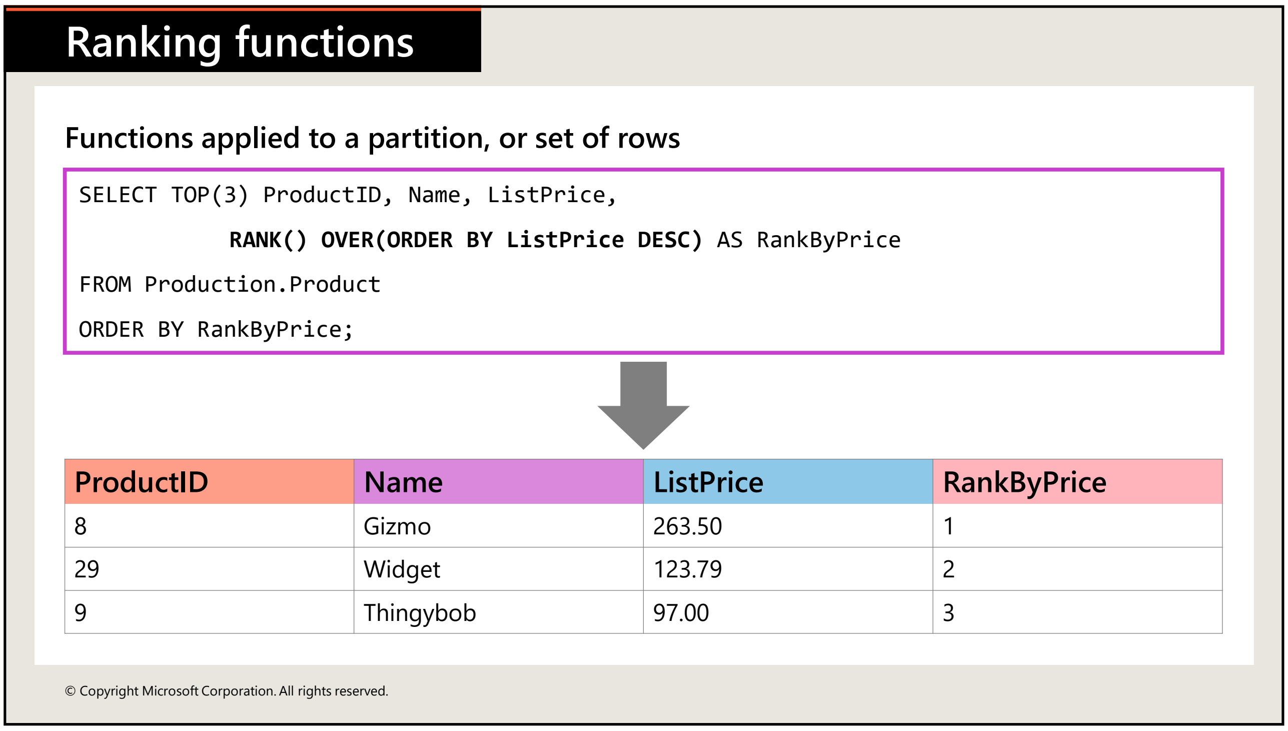
```
SELECT AddressType,
       IIF(AddressType = 'Main Office', 'Billing', 'Mailing') AS UseFor
FROM Sales.CustomerAddress;
```

CHOOSE

- Return value based ordinal position of expression in 1-based list

```
SELECT SalesOrderID, Status,
       CHOOSE(Status, 'Ordered', 'Shipped', 'Delivered')   AS OrderStatus
FROM Sales.SalesOrderHeader;
```

18



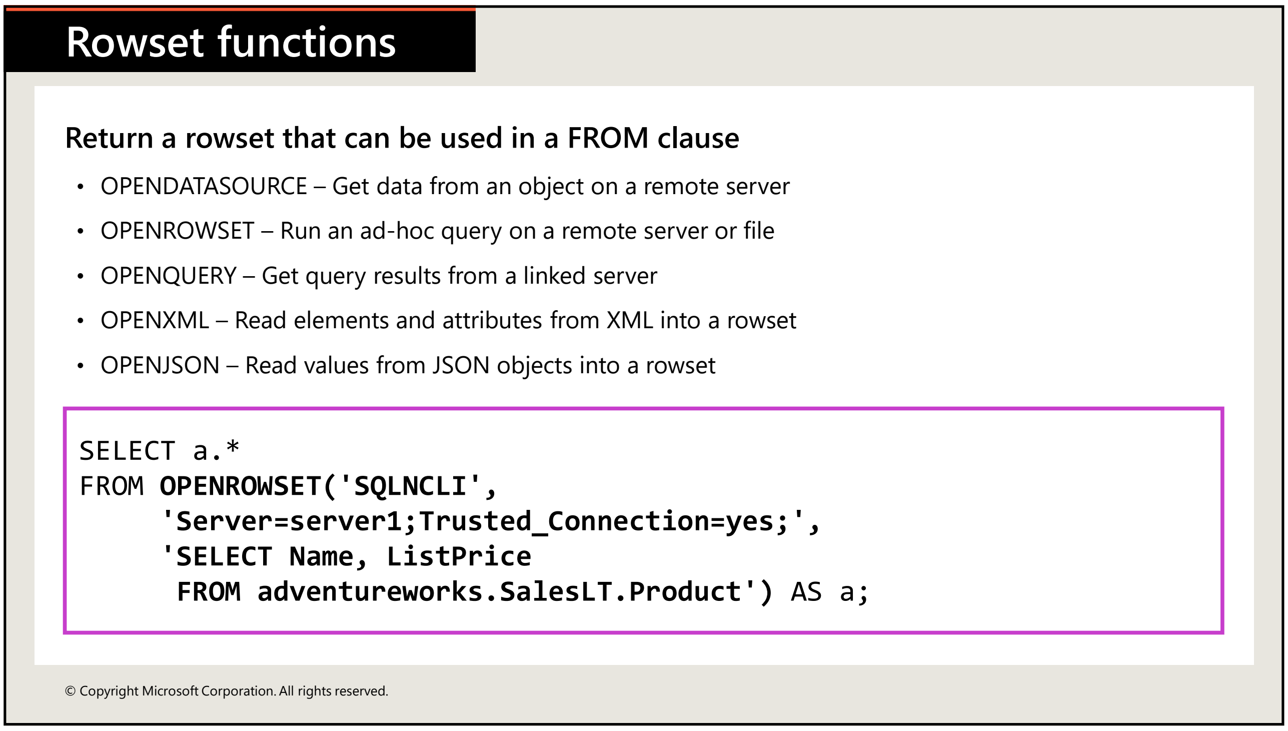
Ranking functions

Functions applied to a partition, or set of rows

```
SELECT TOP(3) ProductID, Name, ListPrice,
           RANK() OVER(ORDER BY ListPrice DESC) AS RankByPrice
FROM Production.Product
ORDER BY RankByPrice;
```

ProductID	Name	ListPrice	RankByPrice
8	Gizmo	263.50	1
29	Widget	123.79	2
9	Thingybob	97.00	3

19



Rowset functions

Return a rowset that can be used in a FROM clause

- OPENDATASOURCE – Get data from an object on a remote server
- OPENROWSET – Run an ad-hoc query on a remote server or file
- OPENQUERY – Get query results from a linked server
- OPENXML – Read elements and attributes from XML into a rowset
- OPENJSON – Read values from JSON objects into a rowset

```
SELECT a.*
FROM OPENROWSET('SQLNCLI',
     'Server=server1;Trusted_Connection=yes;',
     'SELECT Name, ListPrice
      FROM adventureworks.SalesLT.Product') AS a;
```

20

Aggregate functions

Functions that operate on sets, or rows of data

- Summarize input rows
- Without GROUP BY clause, all rows are arranged as one group

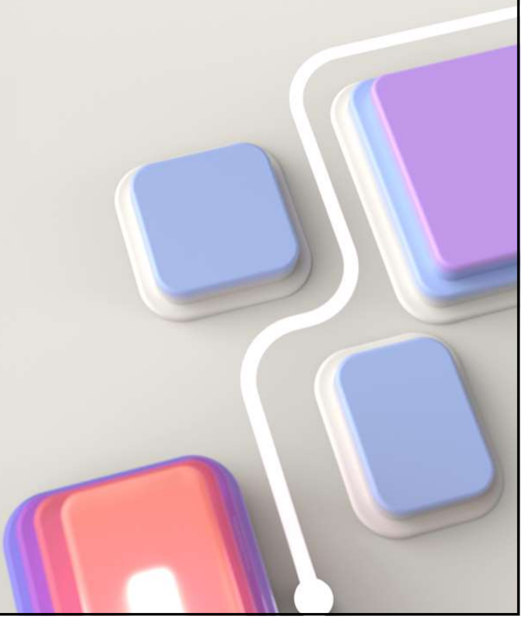
```
SELECT COUNT(*) AS OrderLines,
       SUM(OrderQty*UnitPrice) AS TotalSales
FROM  Sales.OrderDetail;
```

OrderLines	TotalSales
542	714002.9136

21

https://learn.microsoft.com/en-us/training/modules/use-built-functions-transact-sql/

3: Grouping aggregated results



22

Grouping with GROUP BY

- GROUP BY creates groups for output rows, according to unique combination of values specified in the GROUP BY clause
- GROUP BY calculates a summary value for aggregate functions in subsequent phases
- Detail rows are not available after GROUP BY clause is processed

```
SELECT CustomerID, COUNT(*) AS OrderCount
FROM Sales.SalesOrderHeader
GROUP BY CustomerID;
```

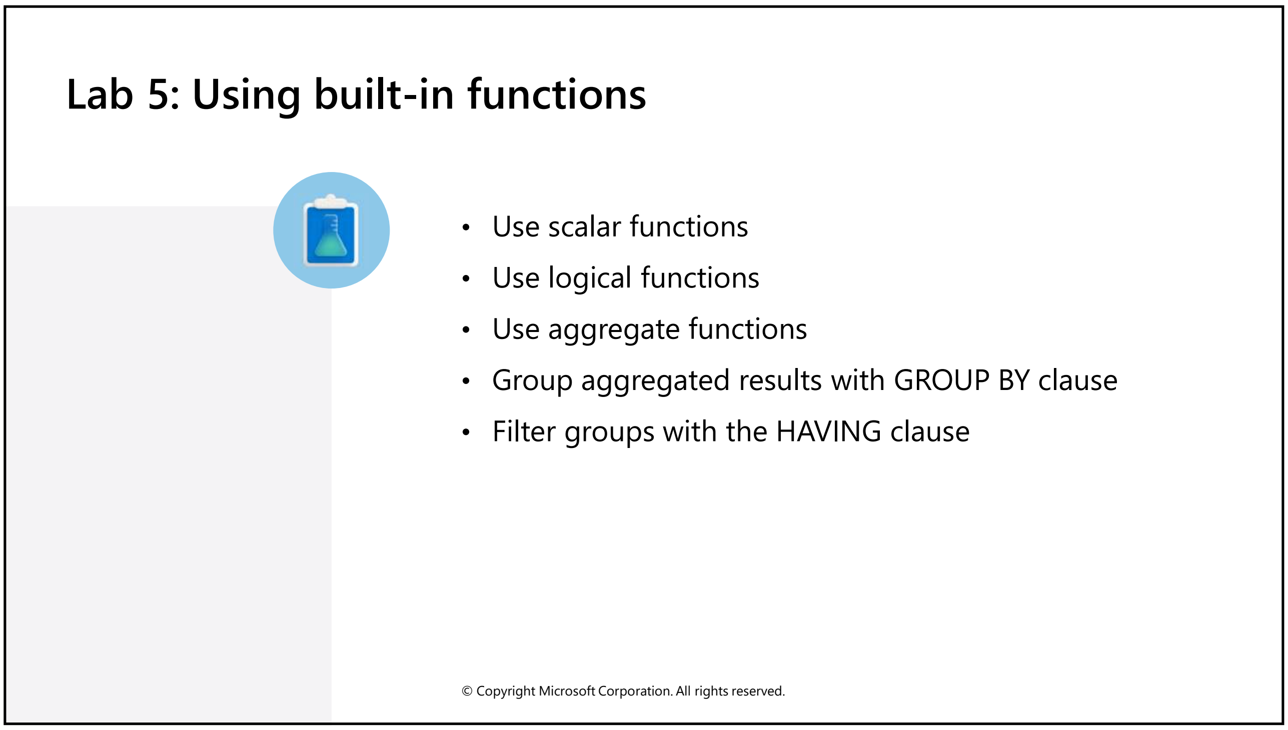
23

Filtering groups with HAVING

- HAVING clause provides a search condition that each group must satisfy
- WHERE clause is processed before GROUP BY, HAVING clause is processed after GROUP BY

```
SELECT CustomerID, COUNT(*) AS Orders
FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING COUNT(*) > 10;
```

24



Lab 5: Using built-in functions

- Use scalar functions
- Use logical functions
- Use aggregate functions
- Group aggregated results with GROUP BY clause
- Filter groups with the HAVING clause

25

https://learn.microsoft.com/en-us/training/modules/modify-data-with-transact-sql/

4: Inserting data into tables

26

Options for inserting data into tables

INSERT...VALUES

- Inserts explicit values
- You can omit identity columns, columns that allow NULL, and columns with default constraints
- You can also explicitly specify NULL and DEFAULT

INSERT...SELECT

Inserts the results returned by a query into an existing table

SELECT...INTO

Creates a new table from the results of a query

27

Identity columns

IDENTITY property of a column generates sequential numbers automatically for insertion into a table

- Optional seed and increment values can be specified when creating the table
- Use system variables and functions to return last inserted identity:

  @@IDENTITY: The last identity generated in the session

  SCOPE_IDENTITY(): The last identity generated in the current scope

  IDENT_CURRENT('<*table_name*>'): The last identity inserted into a table

```
INSERT INTO Sales.Promotion (PromotionName,StartDate,ProductModelID,Discount,Notes)
VALUES
('Clearance Sale', '01/01/2021', 23, 0.10, '10% discount')
…
SELECT SCOPE_IDENTITY() AS PromotionID;
```

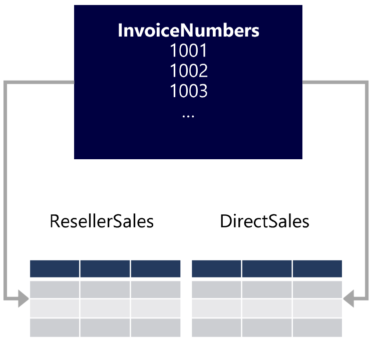
28

Sequences

Sequences are objects that generate sequential numbers

- Exist independently of tables, so offer greater flexibility than Identity
- Use SELECT NEXT VALUE FOR to retrieve the next sequential number

Can be set as the default value for a column



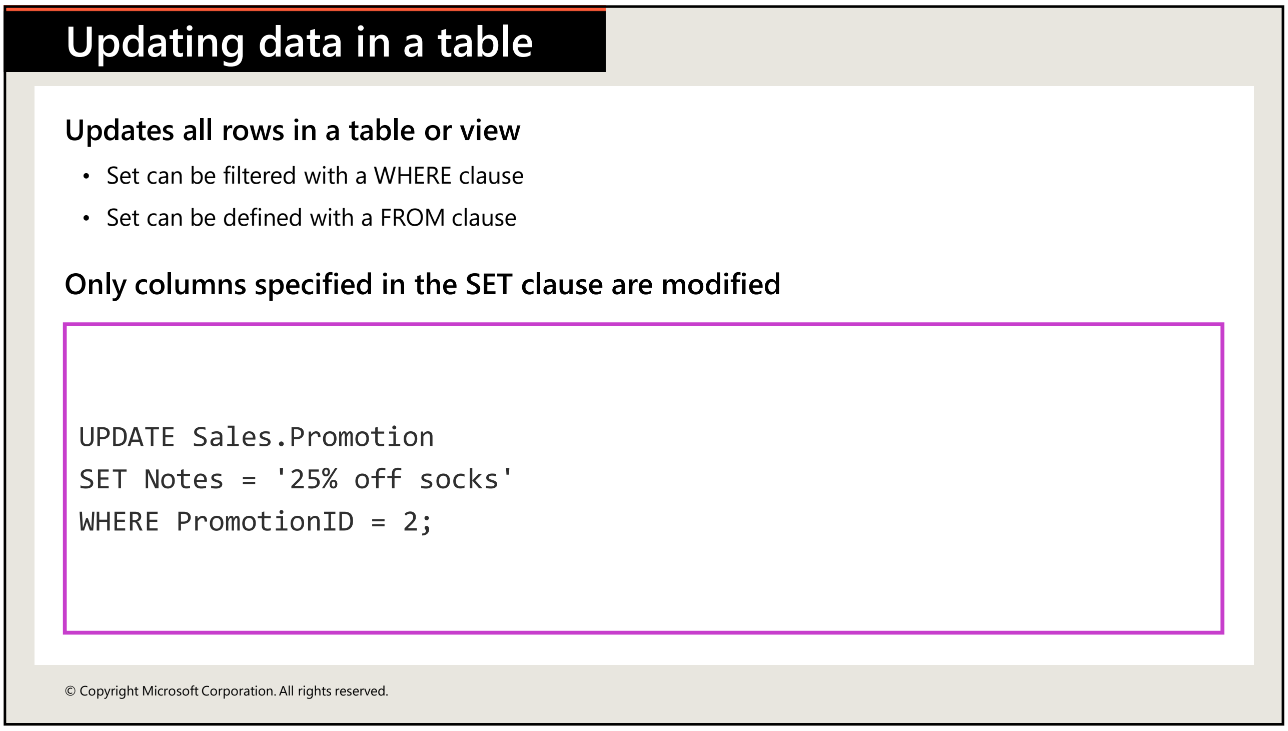
```
CREATE SEQUENCE Sales.InvoiceNumber AS INT
START WITH 1000 INCREMENT BY 1;
…
SELECT NEXT VALUE FOR Sales.InvoiceNumber;
```

29

https://learn.microsoft.com/en-us/training/modules/modify-data-with-transact-sql/

5: Modifying and deleting data

30



Updating data in a table

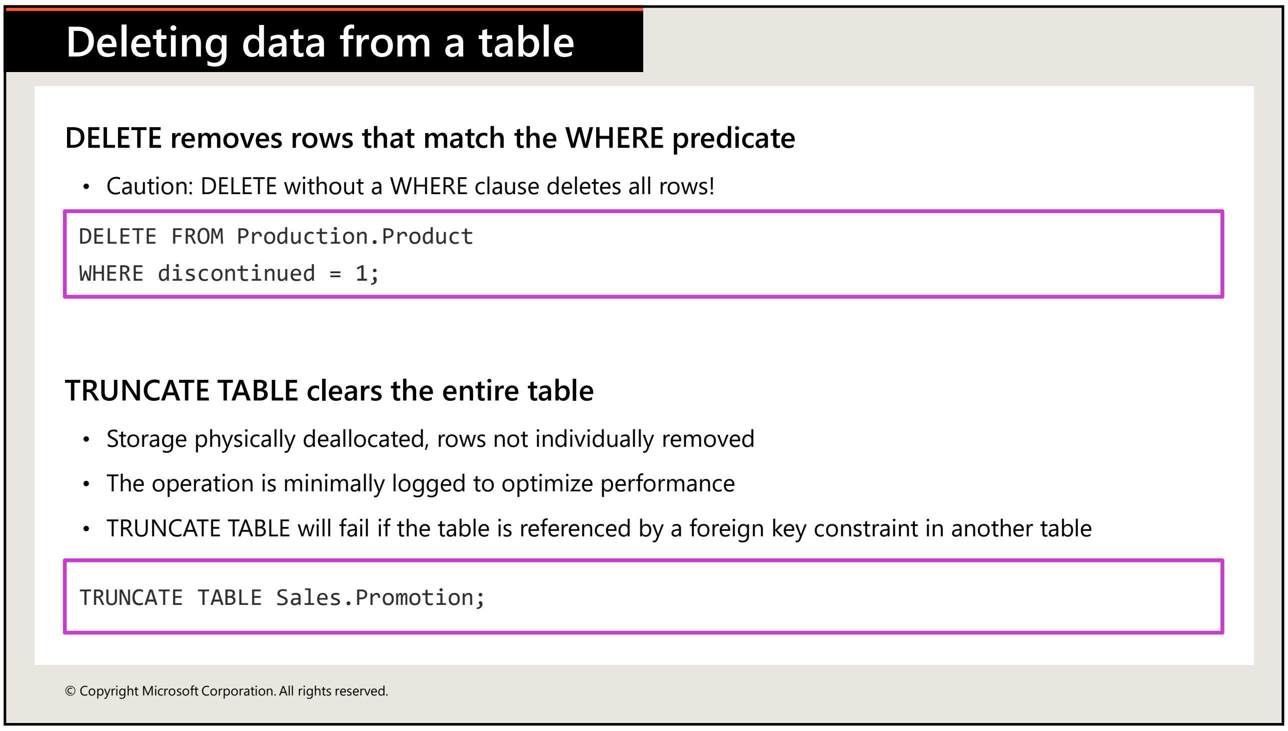
Updates all rows in a table or view

- Set can be filtered with a WHERE clause
- Set can be defined with a FROM clause

Only columns specified in the SET clause are modified

```
UPDATE Sales.Promotion
SET Notes = '25% off socks'
WHERE PromotionID = 2;
```

31



Deleting data from a table

DELETE removes rows that match the WHERE predicate

- Caution: DELETE without a WHERE clause deletes all rows!

```
DELETE FROM Production.Product
WHERE discontinued = 1;
```

TRUNCATE TABLE clears the entire table

- Storage physically deallocated, rows not individually removed
- The operation is minimally logged to optimize performance
- TRUNCATE TABLE will fail if the table is referenced by a foreign key constraint in another table

```
TRUNCATE TABLE Sales.Promotion;
```

32

Merging data in a table

MERGE modifies data based on a condition

- When the source matches the target
- When the source has no match in the target
- When the target has no match in the source

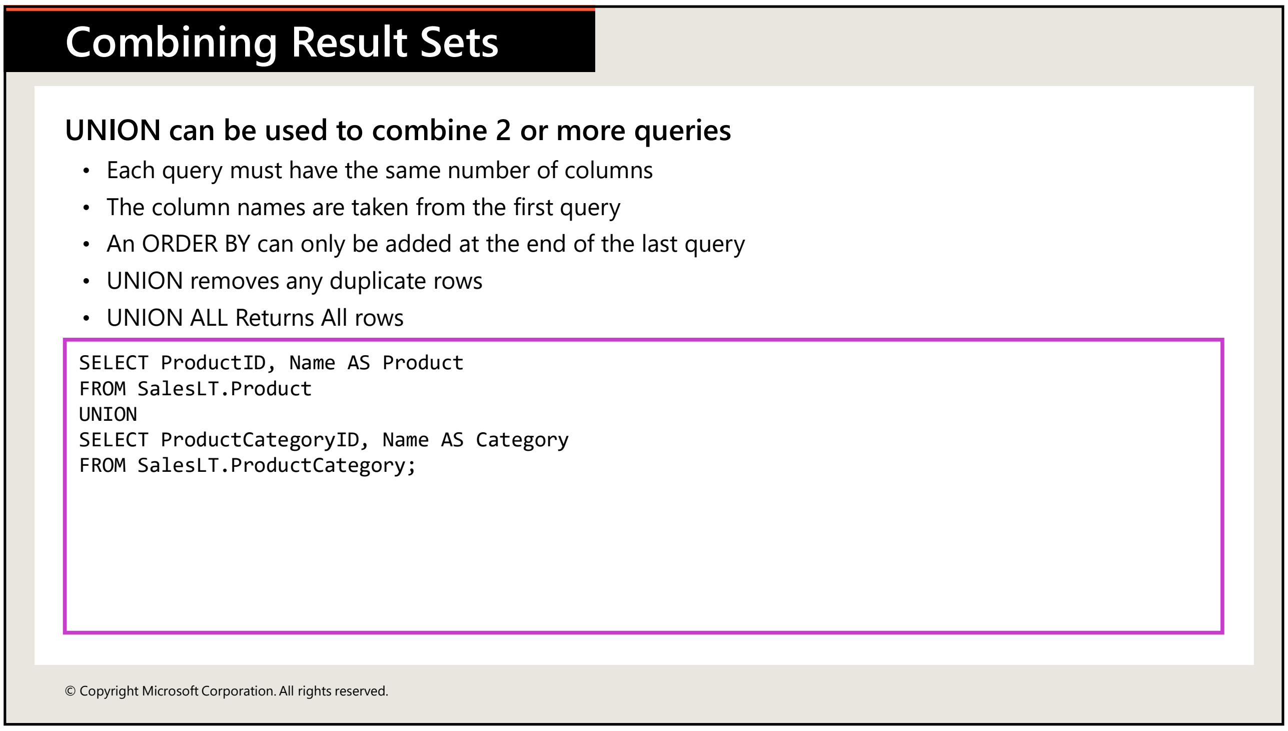
```
MERGE INTO Sales.Invoice as i
USING Sales.InvoiceStaging as s
ON i.SalesOrderID = s.SalesOrderID
WHEN MATCHED THEN
    UPDATE SET i.CustomerID = s.CustomerID,
                i.OrderDate = GETDATE(),
                i.PONumber = s.PONumber,
                i.TotalDue = s.TotalDue
WHEN NOT MATCHED THEN
    INSERT (SalesOrderID, CustomerID, OrderDate, PONumber, TotalDue)
    VALUES (s.SalesOrderID, s.CustomerID, s.OrderDate, s.PONumber, s.TotalDue);
```

33

Lab 6: Modifying data

- Insert data
- Update data
- Delete data

34



Combining Result Sets

UNION can be used to combine 2 or more queries

- Each query must have the same number of columns
- The column names are taken from the first query
- An ORDER BY can only be added at the end of the last query
- UNION removes any duplicate rows
- UNION ALL Returns All rows

```
SELECT ProductID, Name AS Product
FROM SalesLT.Product
UNION
SELECT ProductCategoryID, Name AS Category
FROM SalesLT.ProductCategory;
```

35