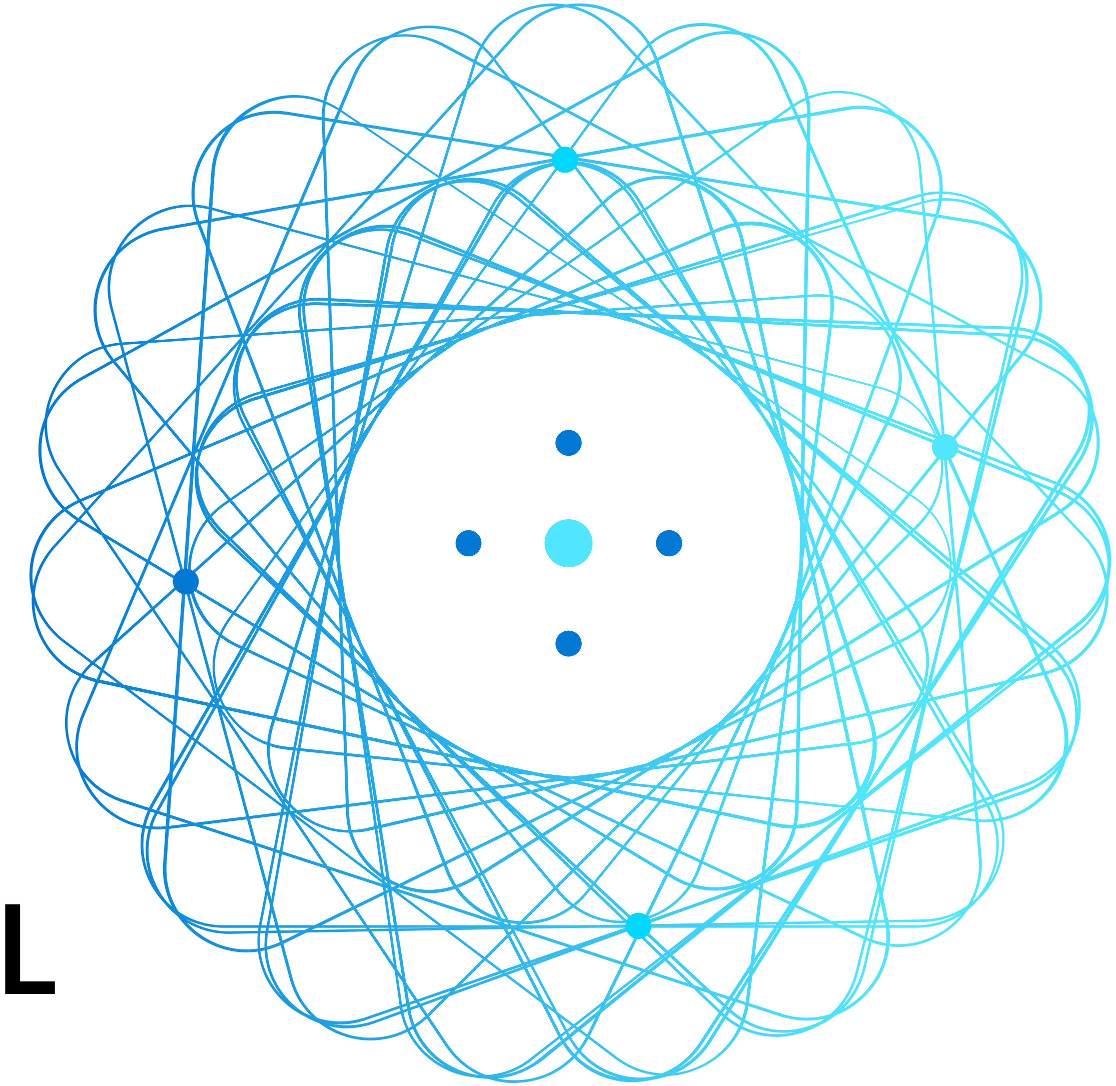
Microsoft Azure



Course DP-080: Querying Data with Microsoft Transact-SQL

About This Course

Learn how to write queries using SQL Server and Azure SQL Database

- This course focuses on learning core Transact-SQL syntax used to work with data for reporting and application development
  - Using SELECT to retrieve columns from a table
  - Sorting and filtering query results
  - Using joins and subqueries to retrieve data from multiple tables
  - Using built-in functions, aggregations, and groupings
  - Inserting, updating, and deleting data
- Additional learning materials are available on Microsoft Learn

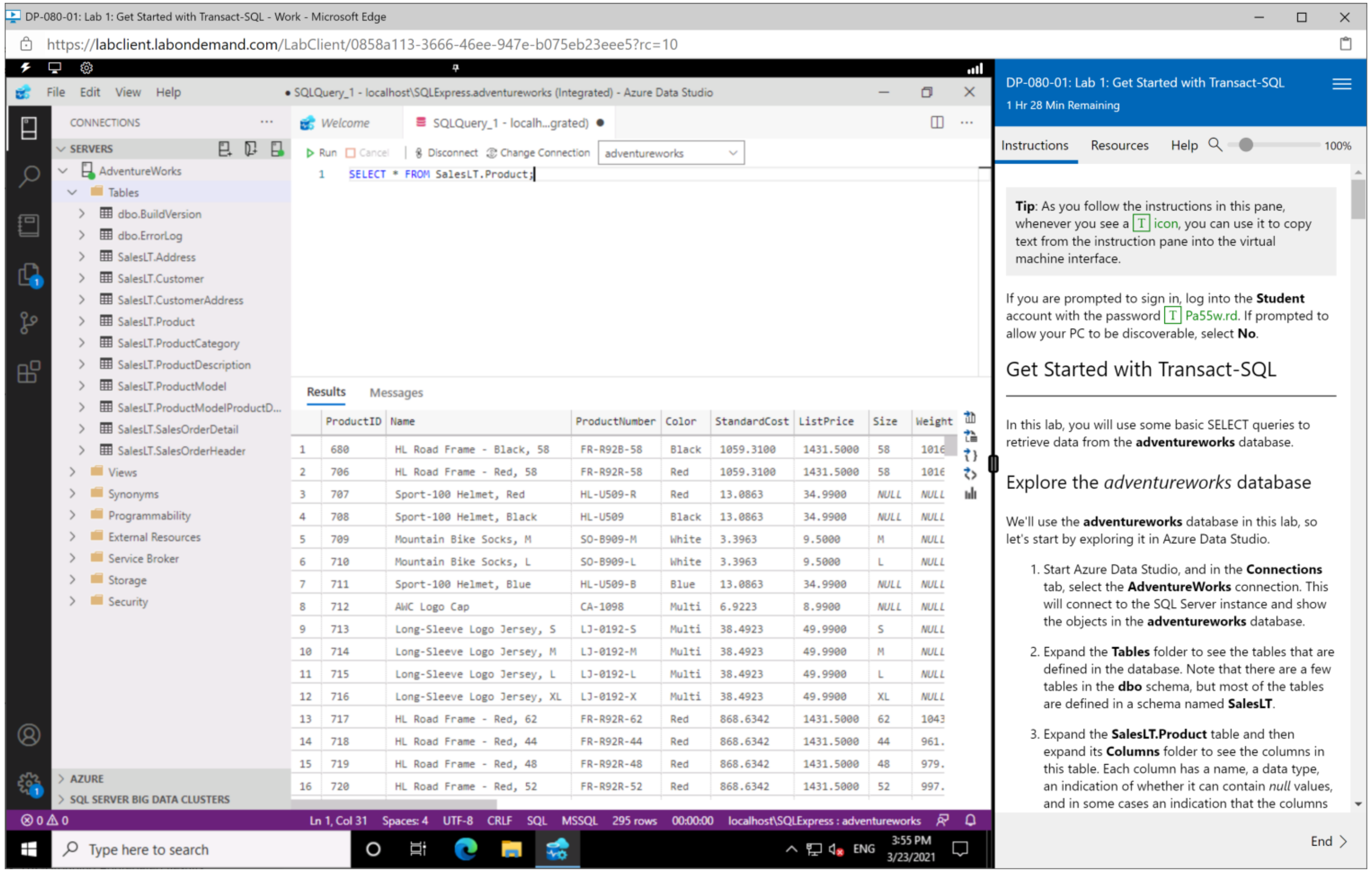
Course Agenda

Module 1: Getting Started with Transact-SQL
Module 2: Sorting and Filtering Query Results
Module 3: Using Joins and Subqueries
Module 4: Using Built-in Functions
Module 5: Modifying Data

Lab Environment

Hosted Virtual Machine

- Windows 10
- SQL Server Express
- Azure Data Studio



```
SELECT * FROM SalesLT.Product;
```

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000	58	1016
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	1059.3100	1431.5000	58	1016
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	13.0863	34.9900	NULL	NULL
4	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900	NULL	NULL
5	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	NULL
6	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	NULL
7	711	Sport-100 Helmet, Blue	HL-U509-B	Blue	13.0863	34.9900	NULL	NULL
8	712	AWC Logo Cap	CA-1098	Multi	6.9223	8.9900	NULL	NULL
9	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	38.4923	49.9900	S	NULL
10	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	NULL
11	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	NULL
12	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	NULL
13	717	HL Road Frame - Red, 62	FR-R92R-62	Red	868.6342	1431.5000	62	1043
14	718	HL Road Frame - Red, 44	FR-R92R-44	Red	868.6342	1431.5000	44	961.
15	719	HL Road Frame - Red, 48	FR-R92R-48	Red	868.6342	1431.5000	48	979.
16	720	HL Road Frame - Red, 52	FR-R92R-52	Red	868.6342	1431.5000	52	997.

Tip: As you follow the instructions in this pane, whenever you see a T icon, you can use it to copy text from the instruction pane into the virtual machine interface.

If you are prompted to sign in, log into the **Student** account with the password T Pa55w.rd. If prompted to allow your PC to be discoverable, select **No**.

Get Started with Transact-SQL

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

Explore the *adventureworks* database

We'll use the **adventureworks** database in this lab, so let's start by exploring it in Azure Data Studio.

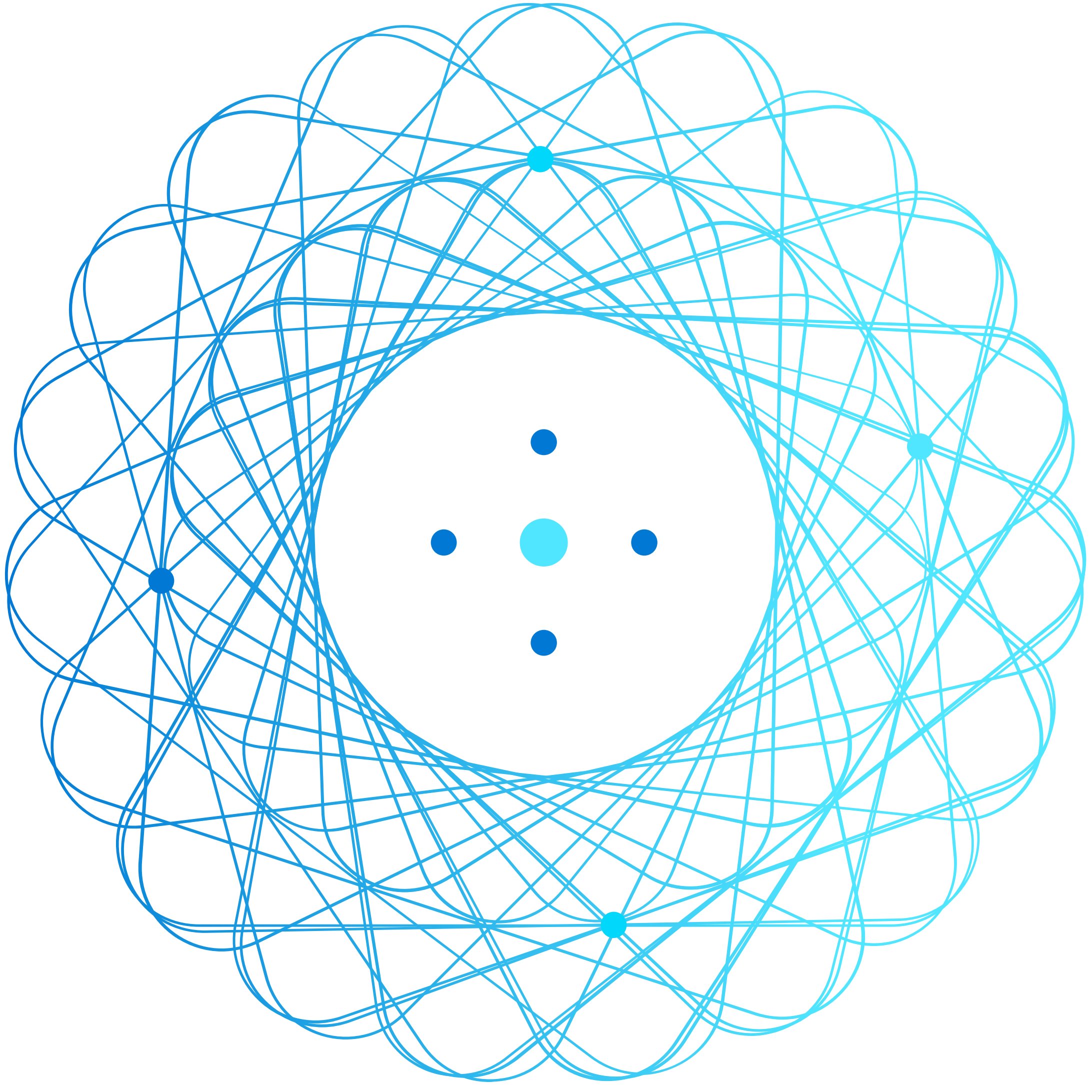
1. Start Azure Data Studio, and in the **Connections** tab, select the **AdventureWorks** connection. This will connect to the SQL Server instance and show the objects in the **adventureworks** database.
2. Expand the **Tables** folder to see the tables that are defined in the database. Note that there are a few tables in the **dbo** schema, but most of the tables are defined in a schema named **SalesLT**.
3. Expand the **SalesLT.Product** table and then expand its **Columns** folder to see the columns in this table. Each column has a name, a data type, an indication of whether it can contain *null* values, and in some cases an indication that the columns

Bring your own environment:
https://microsoftlearning.github.io/dp-080-Transact-SQL/

Microsoft Azure

Microsoft Azure

Module 1: Getting Started with Transact-SQL



Module Agenda

- Introduction to Transact-SQL
- Using the SELECT Statement

Lesson 1: Introduction to Transact-SQL

What is Transact-SQL?

Structured Query Language (SQL)

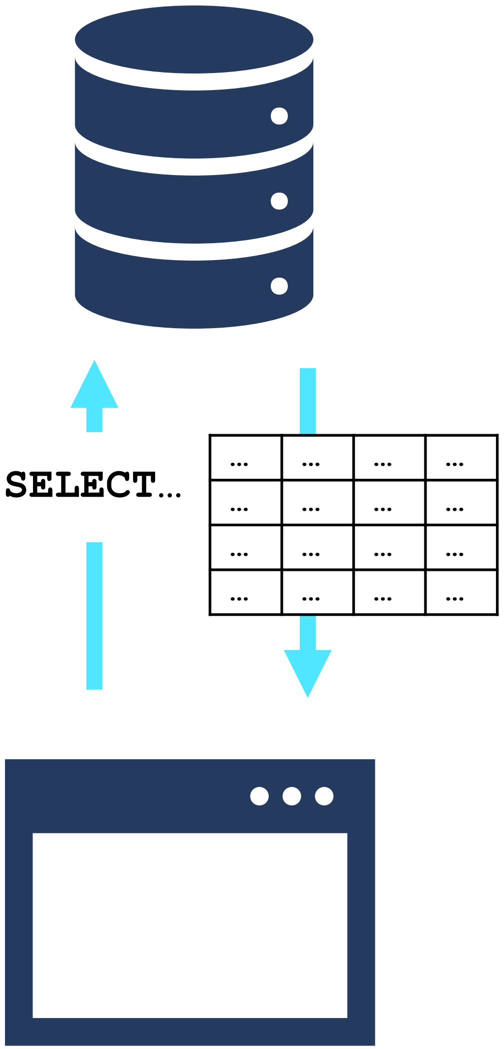
- Developed in the 1970s as a language for querying databases
- Adopted as a standard by ANSI and ISO standards bodies
- Widely used across multiple database systems

Microsoft's implementation is Transact-SQL

- Often referred to as T-SQL
- Query language for SQL Server, Azure SQL Database, and other Microsoft relational database services

SQL is *declarative*, not *procedural*

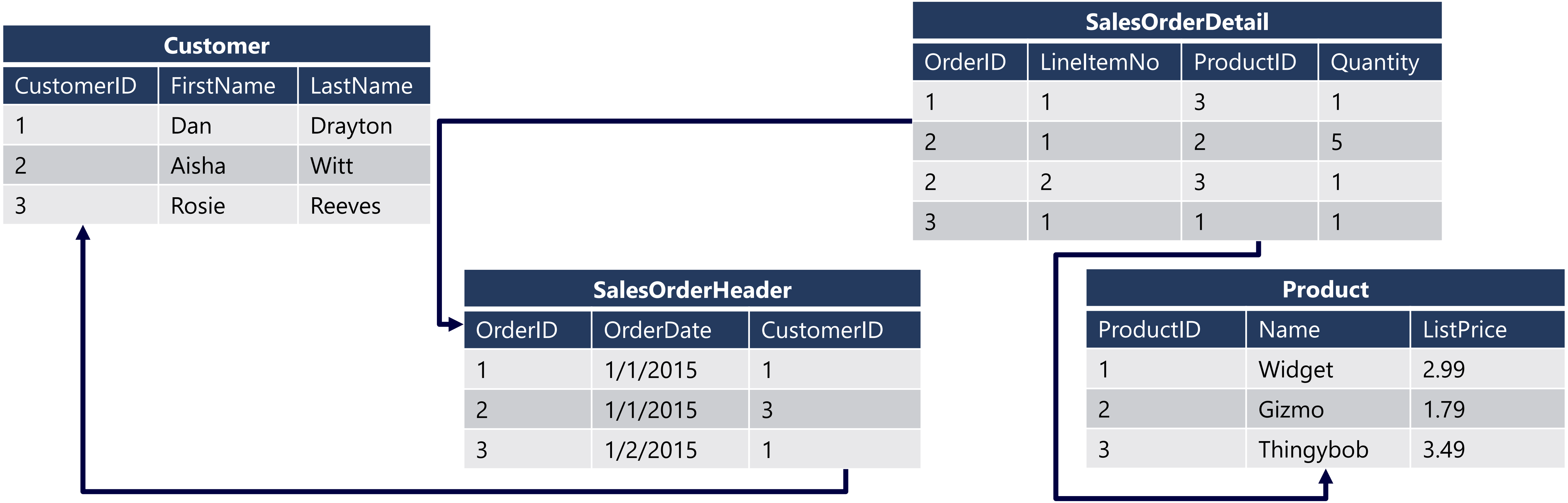
- Describe what you want, don't specify steps



…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

Relational Databases

- Entities are represented as *relations* (tables), in which their attributes are represented as *domains* (columns)
- Most relational databases are *normalized*, with relationships defined between tables through *primary* and *foreign* keys



Customer

CustomerID	FirstName	LastName
1	Dan	Drayton
2	Aisha	Witt
3	Rosie	Reeves

SalesOrderDetail

OrderID	LineItemNo	ProductID	Quantity
1	1	3	1
2	1	2	5
2	2	3	1
3	1	1	1

SalesOrderHeader

OrderID	OrderDate	CustomerID
1	1/1/2015	1
2	1/1/2015	3
3	1/2/2015	1

Product

ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

Schemas and Object Names

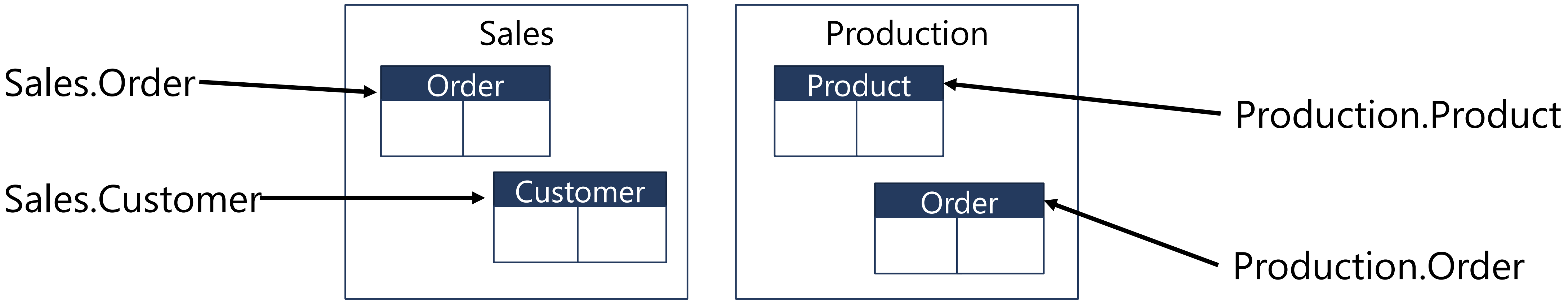
Schemas are namespaces for database objects

- **Fully-qualified names:**

  [*server_name*.][*database_name*.][*schema_name*.]*object_name*

- **Within database context, best practice is to include schema name:**

  schema_name.object_name



SQL Statement Types

Data Manipulation Language (DML)	Data Definition Language (DDL)	Data Control Language (DCL)
Statements for querying and modifying data: • SELECT • INSERT • UPDATE • DELETE	Statements for defining database objects: • CREATE • ALTER • DROP	Statements for assigning security permissions: • GRANT • REVOKE • DENY

Focus of this course

Lesson 2: Using the SELECT Statement

The SELECT Statement

	Element	Expression	Role
5	SELECT	<select list>	Defines which columns to return
1	FROM	<table source>	Defines table(s) to query
2	WHERE	<search condition>	Filters rows using a predicate
3	GROUP BY	<group by list>	Arranges rows by groups
4	HAVING	<search condition>	Filters groups using a predicate
6	ORDER BY	<order by list>	Sorts the output

```
SELECT OrderDate, COUNT(OrderID) AS Orders
FROM Sales.SalesOrder
WHERE Status = 'Shipped'
GROUP BY OrderDate
HAVING COUNT(OrderID) > 1
ORDER BY OrderDate DESC;
```

Basic SELECT Query Examples

All columns

```
SELECT * FROM Production.Product;
```

Specific columns

```
SELECT Name, ListPrice
FROM Production.Product;
```

Expressions and Aliases

```
SELECT Name AS Product, ListPrice * 0.9 AS SalePrice
FROM Production.Product;
```

Data Types

Exact Numeric	Approximate Numeric	Character	Date/Time	Binary	Other
tinyint	float	char	date	binary	cursor
smallint	real	varchar	time	varbinary	hierarchyid
int		text	datetime	image	sql_variant
bigint		nchar	datetime2		table
bit		nvarchar	smalldatetime		timestamp
decimal/numeric		ntext	datetimeoffset		uniqueidentifier
numeric					xml
money					geography
smallmoney					geometry

- Compatible data types can be implicitly converted
- Explicit conversion requires an explicit conversion function:

```
CAST / TRY_CAST
CONVERT / TRY_CONVERT
PARSE / TRY_PARSE
STR
```

NULL Values

NULL represents a *missing* or *unknown* value

ANSI behaviour for NULL values:

- The result of any expression containing a NULL value is NULL

```
2 + NULL = NULL

'MyString: ' + NULL = NULL
```

- Equality comparisons (=) always return false for NULL values, use IS NULL

`NULL = NULL` returns false

`NULL IS NULL` returns true

Useful functions:

`ISNULL(`*`column/variable, value`*`)`: Returns *value* if the column or variable is NULL

`NULLIF(`*`column/variable, value`*`)`: Returns NULL if the column or variable is *value*

`COALESCE(`*`column/variable1, column/variable2, ...`*`)`: Returns the value of the first non-NULL column or variable in the list

Lab: Get Started with Transact-SQL

Explore the *AdventureWorks* database

Use SELECT queries to retrieve data

Handle NULL values

Work with data types

Module Review

You must return the *Name* and *Price* columns from a table named *Product* in the *Production* schema. In the resulting rowset, you want the *Name* column to be named *ProductName*. Which of the following Transact-SQL statements should you use?

- □ `SELECT * FROM Product AS Production.Product;`
- ☑ `SELECT Name AS ProductName, Price FROM Production.Product;`
- □ `SELECT ProductName, Price FROM Production.Product;`

You must retrieve data from a column that is defined as char(1). If the value in the column is a digit between 0 and 9, the query should return it as an integer value. Otherwise, the query should return NULL. Which function should you use?

- □ CAST
- □ NULLIF
- ☑ TRY_CONVERT

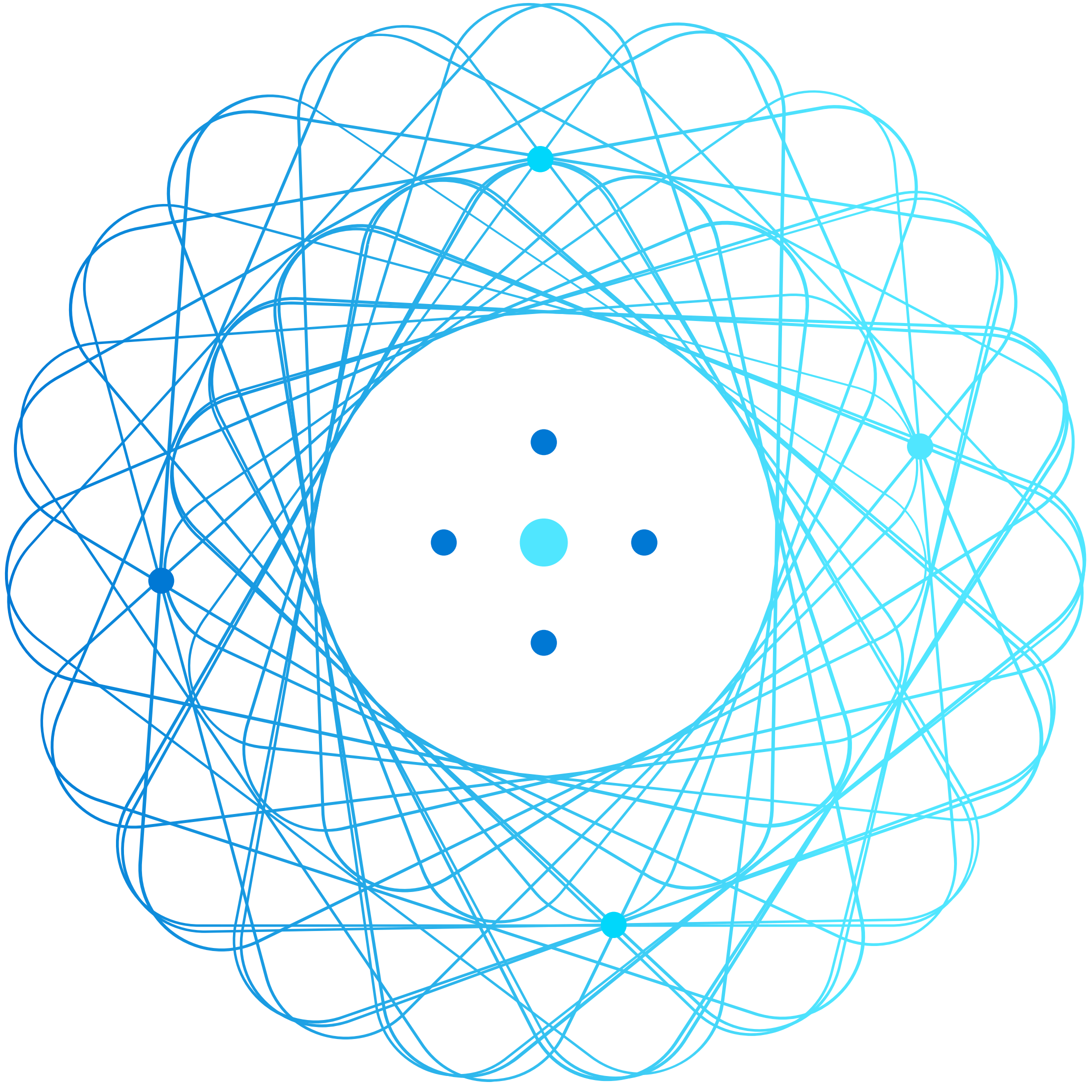
You must return the *Cellphone* column from the *Sales.Customer* table. *Cellphone* is a varchar column that permits NULL values. For rows where the *Cellphone* value is NULL, your query should return the text 'None'. What query should you use?

- ☑ `SELECT ISNULL(Cellphone, 'None') AS Cellphone FROM Sales.Customer;`
- □ `SELECT NULLIF(Cellphone, 'None') AS Cellphone FROM Sales.Customer;`
- □ `SELECT CONVERT(varchar, Cellphone) AS None FROM Sales.Customer;`

Microsoft Azure

Microsoft Azure

Module 2: Sorting and Filtering Query Results



Module Agenda

- Sorting Query Results
- Filtering Query Results

Lesson 1: Sorting Query Results

Sorting Results

Use ORDER BY to sort results by one or more columns

- Aliases created in SELECT clause are visible to ORDER BY
- You can order by columns in the source that are not included in the SELECT clause
- You can specify ASC or DESC (ASC is the default)

```
SELECT ProductCategoryID AS Category, ProductName
FROM Production.Product
ORDER BY Category ASC, Price DESC;
```

Limiting Sorted Results

Use TOP to limit the number or percentage of rows returned by a query

- Works with ORDER BY clause to limit rows by sort order
- Added to SELECT clause:

```
SELECT TOP N [Percent] [WITH TIES]
```

```
SELECT TOP 10 Name, ListPrice
FROM Production.Product
ORDER BY ListPrice DESC;
```

Paging Through Results

OFFSET-FETCH is an extension to the ORDER BY clause:

- Allows returning a requested range of rows
- Provides a mechanism for paging through results
- Specify number of rows to skip, number of rows to retrieve

```
SELECT ProductID, ProductName, ListPrice
FROM Production.Product
ORDER BY ListPrice DESC
   OFFSET 0 ROWS -- Skip zero rows
   FETCH NEXT 10 ROWS ONLY; -- Get the next 10
```

Lesson 2: Filtering Query Results

Removing Duplicates

SELECT ALL

Default behavior includes duplicates

```
SELECT City, CountryRegion
FROM Production.Supplier
ORDER BY CountryRegion, City;
```

City	CountryRegion
Aurora	Canada
Barrie	Canada
Brampton	Canada
Brossard	Canada
Brossard	Canada
Burnaby	Canada
Burnaby	Canada
Burnaby	Canada
Calgary	Canada
Calgary	Canada

SELECT DISTINCT

Removes duplicates

```
SELECT DISTINCT City, CountryRegion
FROM Production.Supplier
ORDER BY CountryRegion, City;
```

City	CountryRegion
Aurora	Canada
Barrie	Canada
Brampton	Canada
Brossard	Canada
Burnaby	Canada
Calgary	Canada

Filtering and Using Predicates

```
SELECT ProductCategoryID AS Category, ProductName
FROM Production.Product
WHERE ProductCategoryID = 2
   AND ListPrice < 10.00
ORDER BY Category, Price DESC;
```

Predicates and Operators	Description
= < >	Compares values for equality / non-equality.
IN	Determines whether a specified value matches any value in a subquery or a list.
BETWEEN	Specifies an inclusive range to test.
LIKE	Determines whether a specific character string matches a specified pattern, which can include wildcards.
AND	Combines two Boolean expressions and returns TRUE only when both are TRUE.
OR	Combines two Boolean expressions and returns TRUE if either is TRUE.
NOT	Reverses the result of a search condition.

Lab: Sort and Filter Query Results

Sort results using the
ORDER BY clause

Restrict results using TOP

Retrieve pages of results with OFFSET and FETCH

Use the ALL and DISTINCT options

Filter results with the WHERE clause

Module Review

You write a Transact-SQL query to list the available sizes for products. Each individual size should be listed only once. Which query should you use?

□ `SELECT Size FROM Production.Product;`

☑ `SELECT DISTINCT Size FROM Production.Product;`

□ `SELECT ALL Size FROM Production.Product;`

You must return the InvoiceNo and TotalDue columns from the Sales.Invoice table in decreasing order of TotalDue value. Which query should you use?

□ `SELECT * FROM Sales.Invoice ORDER BY TotalDue, InvoiceNo;`

☑ `SELECT InvoiceNo, TotalDue FROM Sales.Invoice ORDER BY TotalDue DESC;`

□ `SELECT TotalDue AS DESC, InvoiceNo FROM Sales.Invoice;`

Complete this query to return only products that have a Category value of 2 or 4:

```
SELECT Name, Price FROM Production.Product
```

□ `ORDER BY Category;`

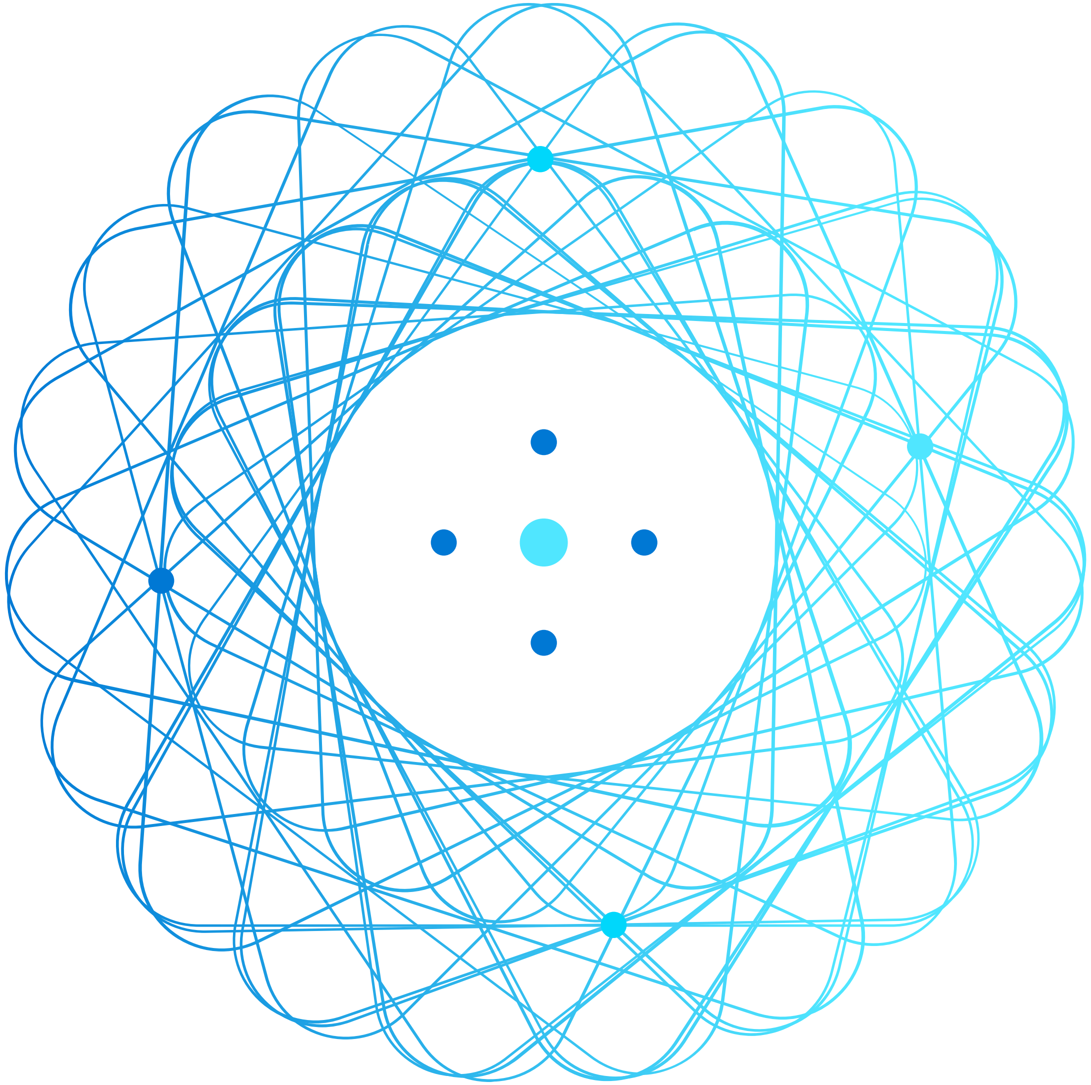
□ `WHERE Category BETWEEN 2 AND 4;`

☑ `WHERE Category IN (2, 4);`

Microsoft Azure

Microsoft Azure

Module 3: Using Joins and Subqueries



Module Agenda

- Using Joins
- Using Subqueries

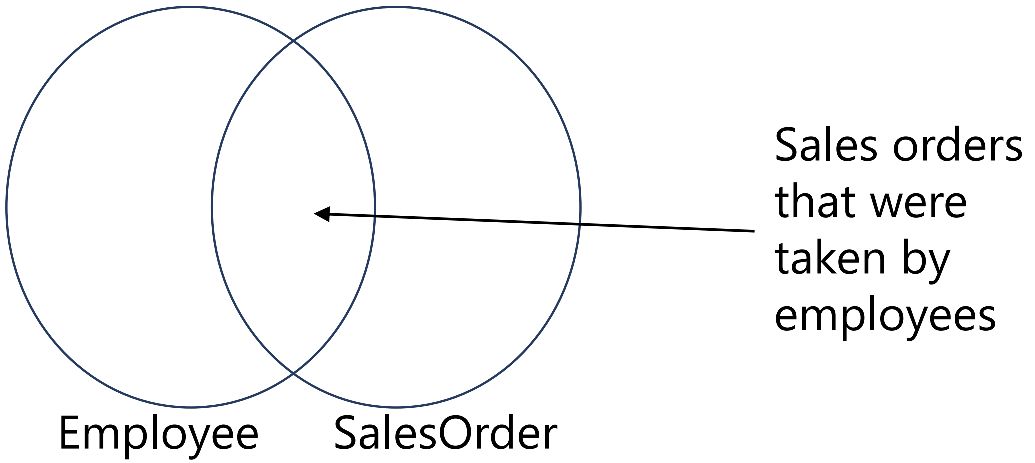
Lesson 1: Using Joins

Join Concepts

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

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



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!

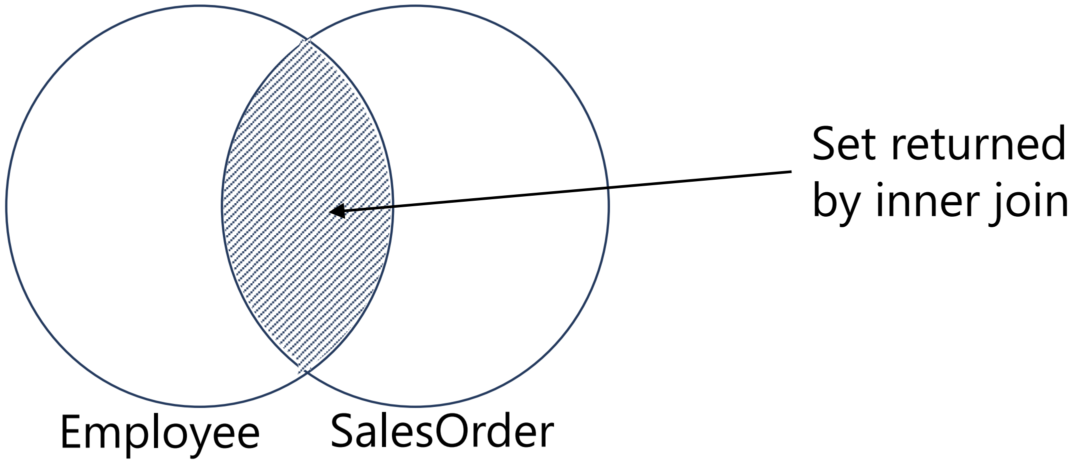
```
SELECT ...
FROM   Table1, Table2
WHERE  <predicate>;
```

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
```

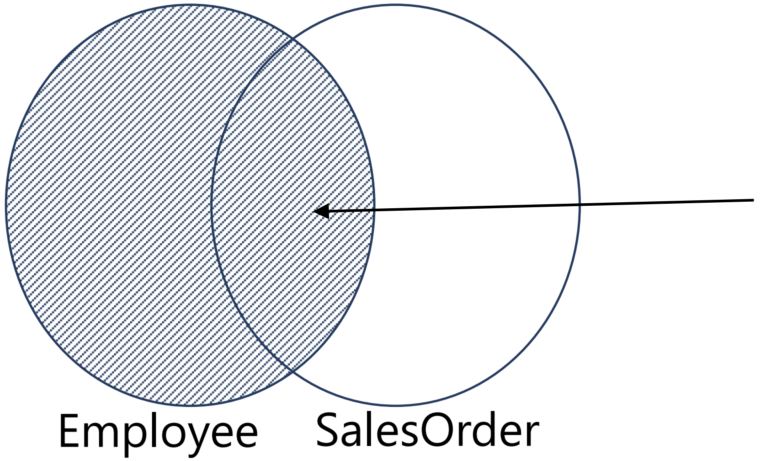


Outer Joins

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;
```



Set returned by left outer join

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

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

Lab: Query Multiple Tables with Joins

Use inner joins

Use outer joins

Use a cross join

Use a self join

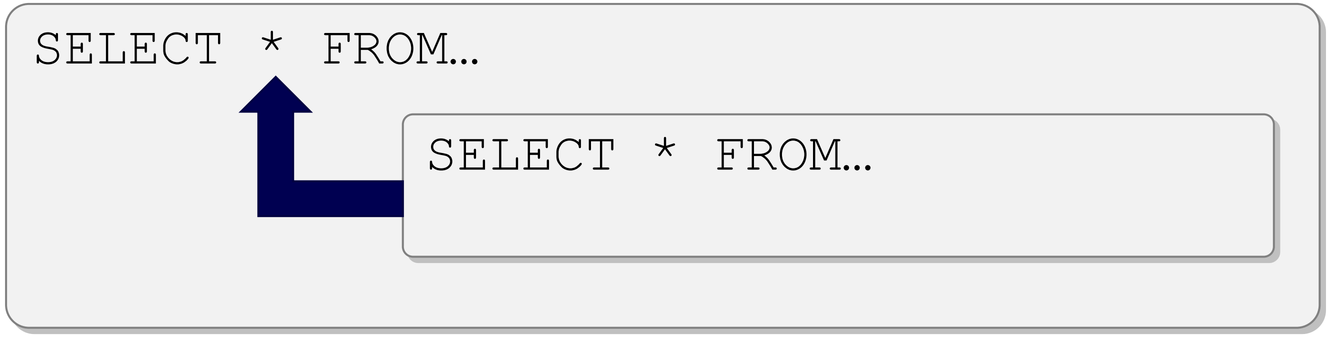
Lesson 2: Using Subqueries

Introduction to Subqueries

Subqueries are nested queries: queries within queries

Results of inner query passed to outer query

- Inner query acts like an expression from perspective of the outer query



SELECT * FROM…

SELECT * FROM…

Scalar or Multi-Valued Subqueries?

Scalar subquery returns single value to outer query

- Can be used anywhere single-valued expression is used: SELECT, WHERE, and so on

```sql
SELECT SalesOrderID, ProductID, OrderQty
FROM Sales.SalesOrderDetail
WHERE SalesOrderID =
    (SELECT MAX(SalesOrderID)
     FROM Sales.SalesOrderHeader);
```

Multi-valued subquery returns multiple values as a single column set to the outer query

- Used with IN predicate

```sql
SELECT CustomerID, SalesOrderID
FROM Sales.SalesOrderHeader
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Sales.Customer
    WHERE CountryRegion = 'Canada');
```

Self-Contained or Correlated Subqueries?

Most subqueries are self-contained and have no connection with the outer query other than passing results to it

Correlated subqueries refer to elements of tables used in outer query

- Dependent on outer query, cannot be executed separately
- Behaves as if inner query is executed once per outer row
- May return scalar value or multiple values

```
SELECT SalesOrderID, CustomerID, OrderDate
FROM SalesLT.SalesOrderHeader AS o1
WHERE SalesOrderID =
    (SELECT MAX(SalesOrderID)
     FROM SalesLT.SalesOrderHeader AS o2
     WHERE o2.CustomerID = o1.CustomerID)
ORDER BY CustomerID, OrderDate;
```

Lab: Use Subqueries

Use simple subqueries

Use correlated subqueries

Module Review

You must return a list of all sales employees that have taken sales orders. Employees who have not taken sales orders should not be included in the results. Which type of join is required?

- ☑ INNER
- □ LEFT OUTER
- □ FULL OUTER

What dows the following query return?

```
SELECT p.Name, c.Name FROM Store.Product AS p CROSS JOIN Store.Category AS c;
```

- □ Only data rows where the product name is the same as the category name.
- □ Only rows where the product name is not the same as the category name.
- ☑ Every combination of product and category name.

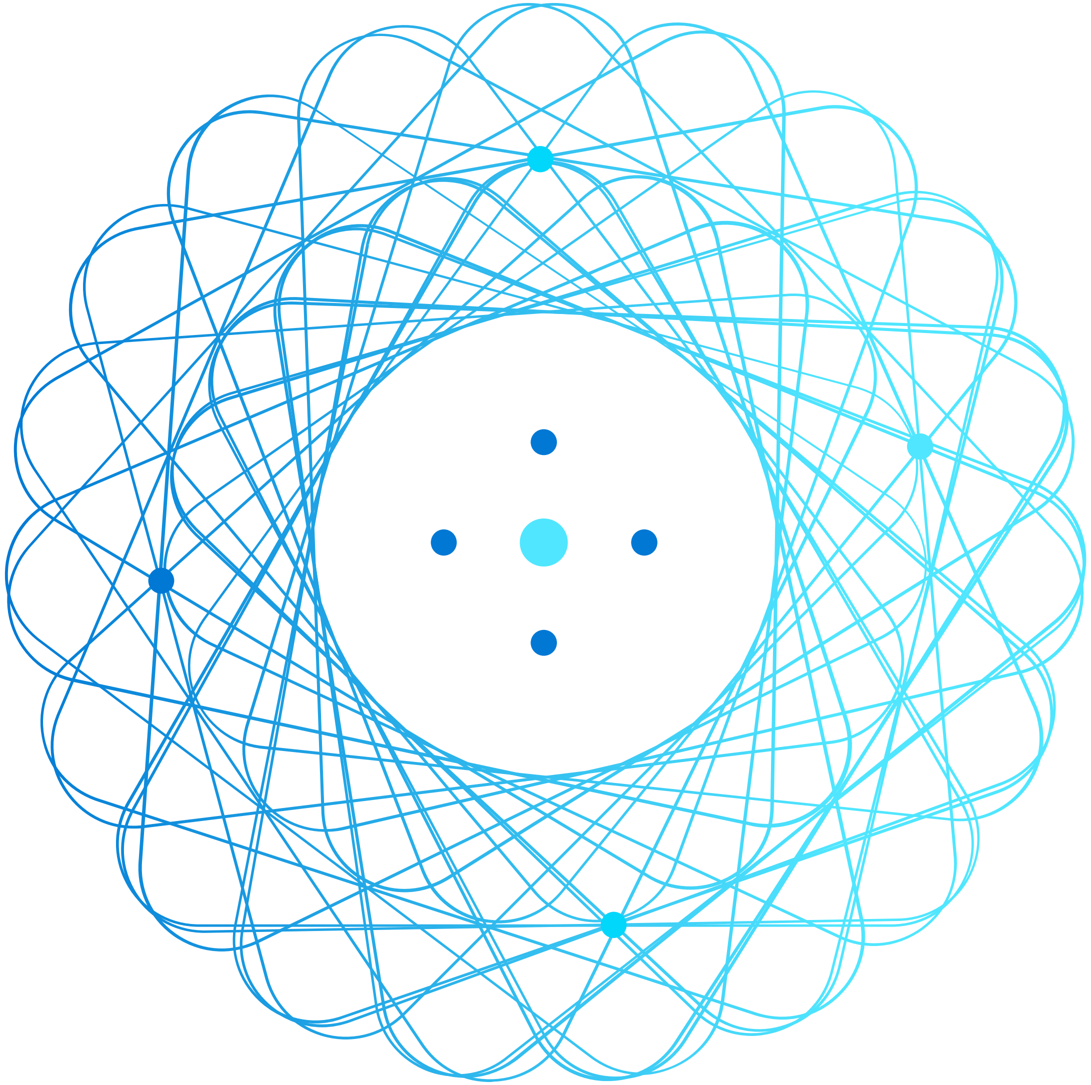
A correlated subquery...

- □ Returns a single scalar value
- □ Returns multiple columns and rows
- ☑ References a value in the outer query

Microsoft Azure

Microsoft Azure

Module 4: Using Built-in Functions



Module Agenda

- Getting Started with Scalar Functions
- Grouping Aggregated Results

Lesson 1: Getting Started with Scalar Functions

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

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

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;
```

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

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;
```

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

Lesson 2: Grouping Aggregated Results

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;
```

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;
```

Lab: 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

Module Review

Which OrderState value does this query return for rows with a Status value of 2:

`SELECT OrderNo, CHOOSE(Status, 'Ordered', 'Shipped', 'Delivered') AS OrderState FROM Sales.Order;`

- ☑ Shipped
- □ Delivered
- □ NULL

Which query returns the number of customers in each city?

- □ `SELECT City, COUNT(*) AS CustomerCount FROM Sales.Customer;`
- ☑ `SELECT City, COUNT(*) AS CustomerCount FROM Sales.Customer GROUP BY City;`
- □ `SELECT City, COUNT(*) AS CustomerCount FROM Sales.Customer ORDER BY City;`

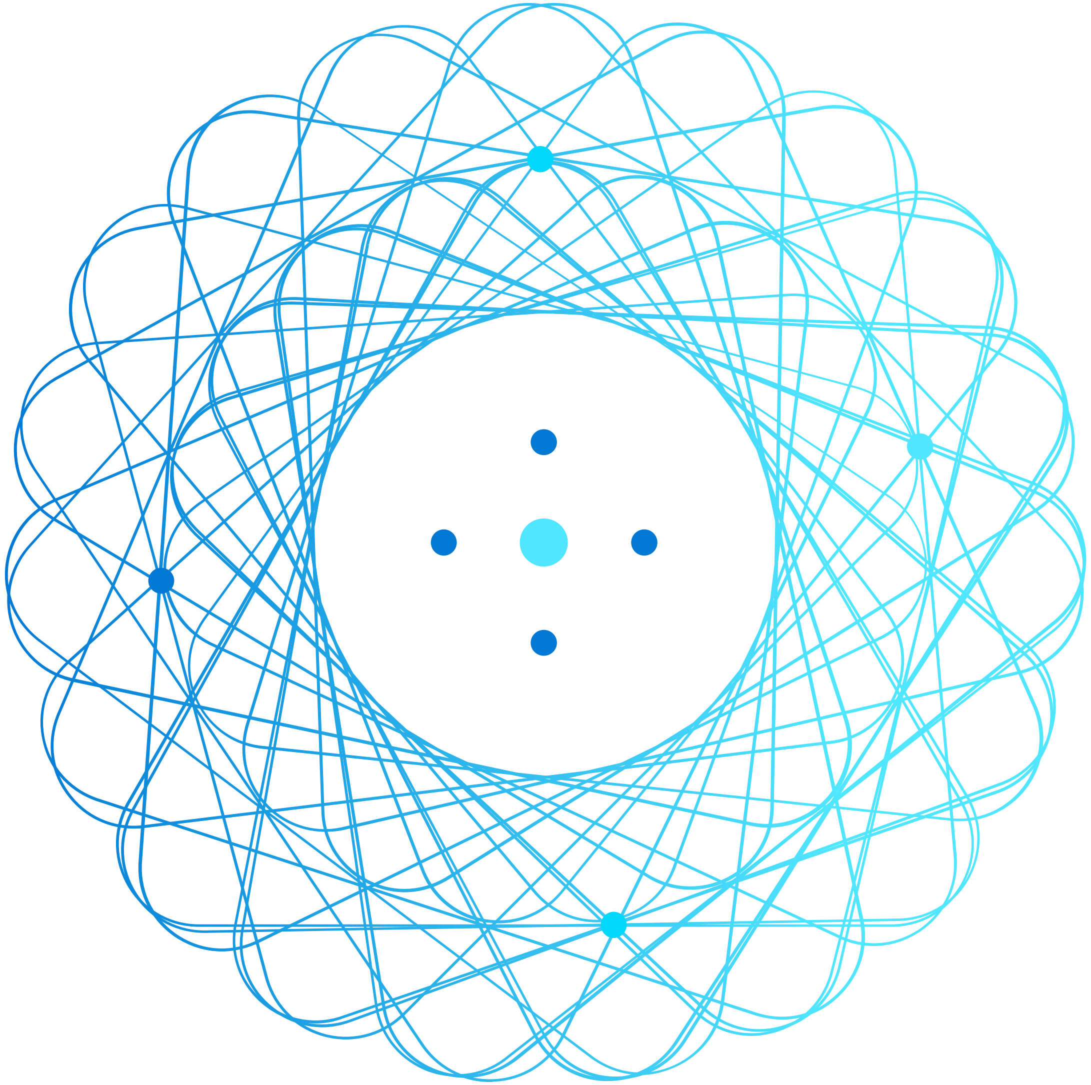
Which query returns a row for each category with an average price over 10.00?

- □ `SELECT Category, AVG(Price) FROM Store.Product WHERE AVG(Price) > 10.00;`
- □ `SELECT Category, AVG(Price) FROM Store.Product GROUP BY Category WHERE AVG(Price) > 10.00;`
- ☑ `SELECT Category, AVG(Price) FROM Store.Product GROUP BY Category HAVING AVG(Price) > 10.00;`

Microsoft Azure

Microsoft Azure

Module 5: Modifying Data



Module Agenda

- Inserting Data into Tables
- Modifying and Deleting Data

Lesson 1: Inserting Data into Tables

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

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

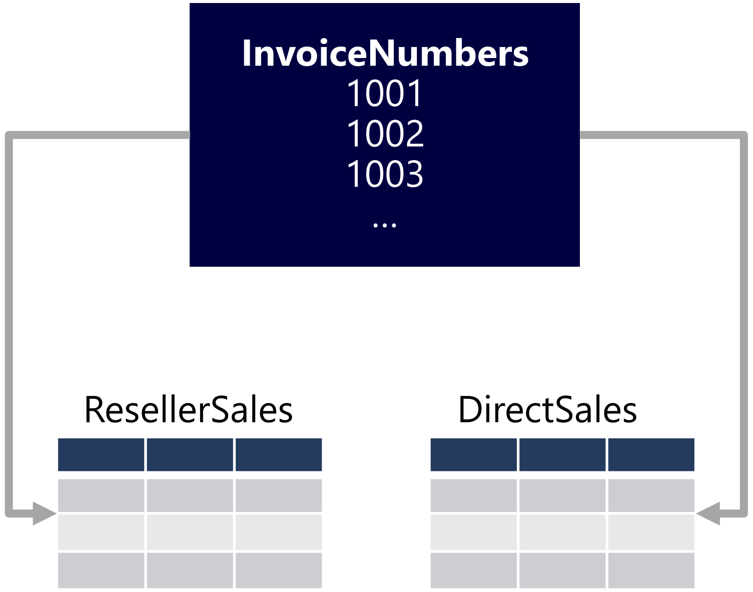
```
INSERT INTO Sales.Promotion (PromotionName,StartDate,ProductModelID,Discount,Notes)
VALUES
('Clearance Sale', '01/01/2021', 23, 0.10, '10% discount')
…
SELECT SCOPE_IDENTITY() AS PromotionID;
```

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



InvoiceNumbers
1001
1002
1003
…

ResellerSales

DirectSales

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

Lesson 2: Modifying and Deleting Data

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;
```

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;
```

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);
```

Lab: Modifying Data

Insert data

Update data

Delete data

Module Review

You want to insert data from the Store.Product table into an existing table named Sales.Offer. Which statement should you use?

☑ INSERT INTO Sales.Offer SELECT ProductID, Name, Price*0.9 FROM Store.Product;

☐ SELECT ProductID, Name, Price*0.9 FROM Store.Product INTO Sales.Offer;

☐ INSERT INTO Sales.Offer (ProductID, Name, Price*0.9) VALUES (Store.Product);

You need to determine the most recently inserted IDENTITY column in the Sales.Invoice table. Which statement should you use?

☐ SELECT SCOPE_IDENTITY() FROM Sales.Invoice;

☑ SELECT IDENT_CURRENT('Sales.Invoice');

☐ SELECT NEXT VALUE FOR Sales.Invoice;

You must increase the Price of all products in category 2 by 10%.

Which statement should you use?

☐ UPDATE Store.Product SET Price = Price*1.1, Category = 2;

☑ UPDATE Store.Product SET Price = Price*1.1 WHERE Category = 2;

☐ SELECT Price*1.1 FROM Store.Product WHERE Category = 2 INTO Store.Product;

Microsoft Azure