Querying with Transact-SQL

Mohammad Sbeeh

Software Architect & Sr. Technical Consultant Lead | Trainer (MCT) | PSM

https://www.linkedin.com/feed/

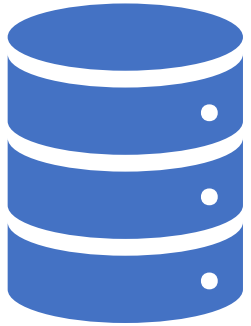
www.sbeeh.com

Outlines

- Module 1: Getting Started with Transact-SQL
- Module 2: Sorting and Filtering Query Results
- Module 3: Using Joins
- Module 4: Using Subqueries
- Module 5: Using Built-in Functions
- Module 6: Modifying Data

Lab Environment Setup

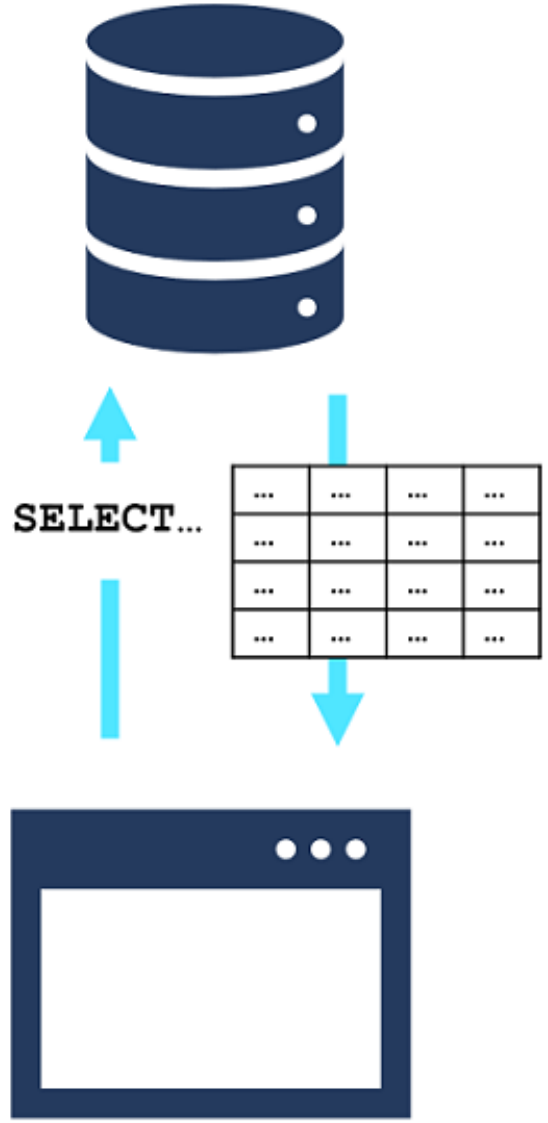
Microsoft SQL Server Express 2019

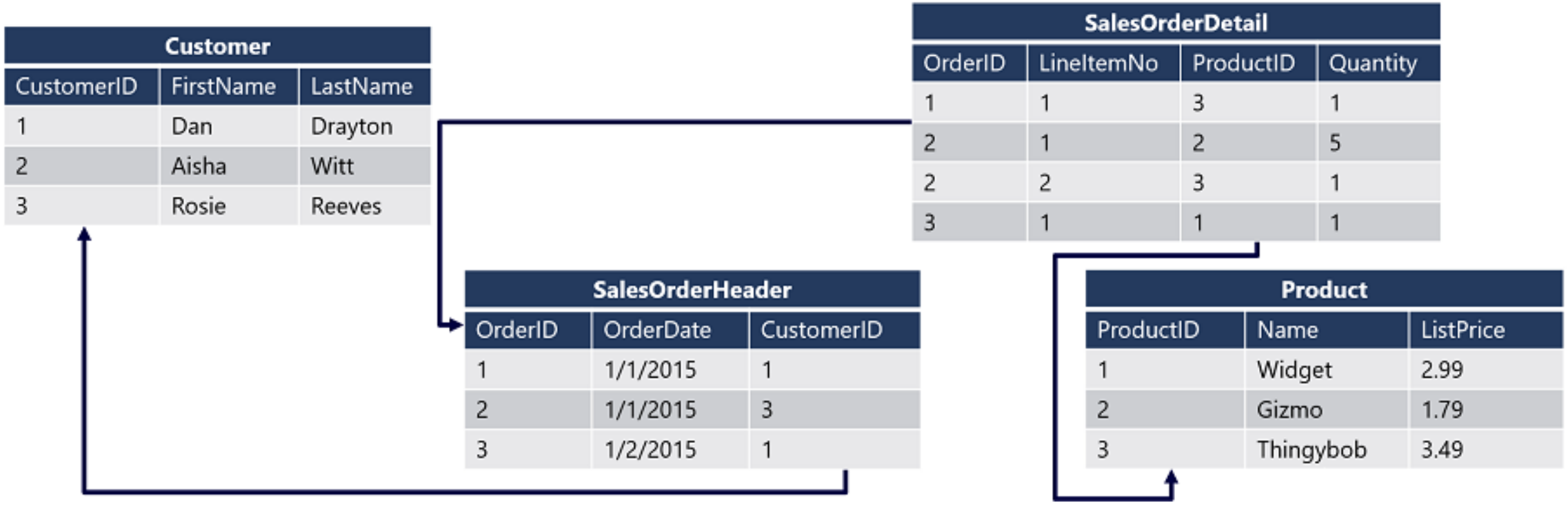


AdventureWorks LT Database

Module 1: Introduction to Transact-SQL

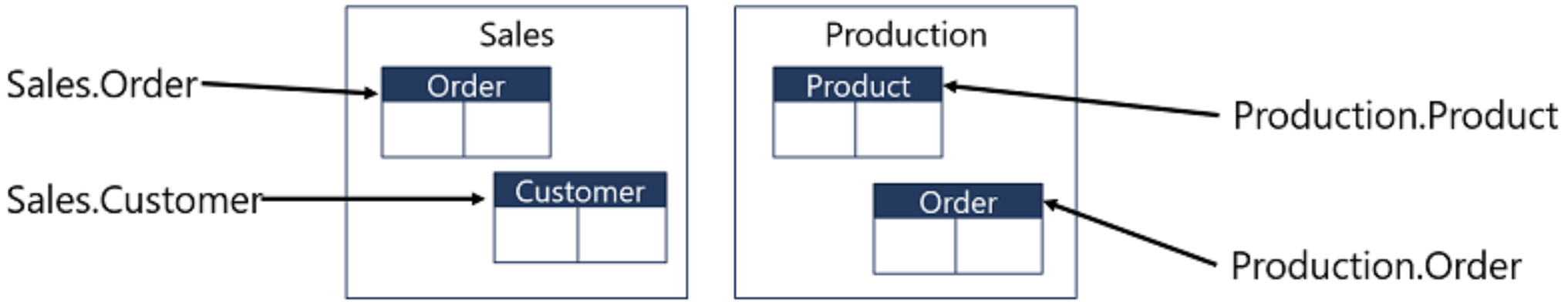
- SQL is used to communicate with relational databases.
- SQL statements are used to perform tasks such as updating data in a database or retrieving data from a database.





Relational data

- SQL is most often (though not always) used to query data in relational databases. A relational database is one in which the data has been organized in multiple tables (technically referred to as relations), each representing a particular type of entity (such as a customer, product, or sales order).



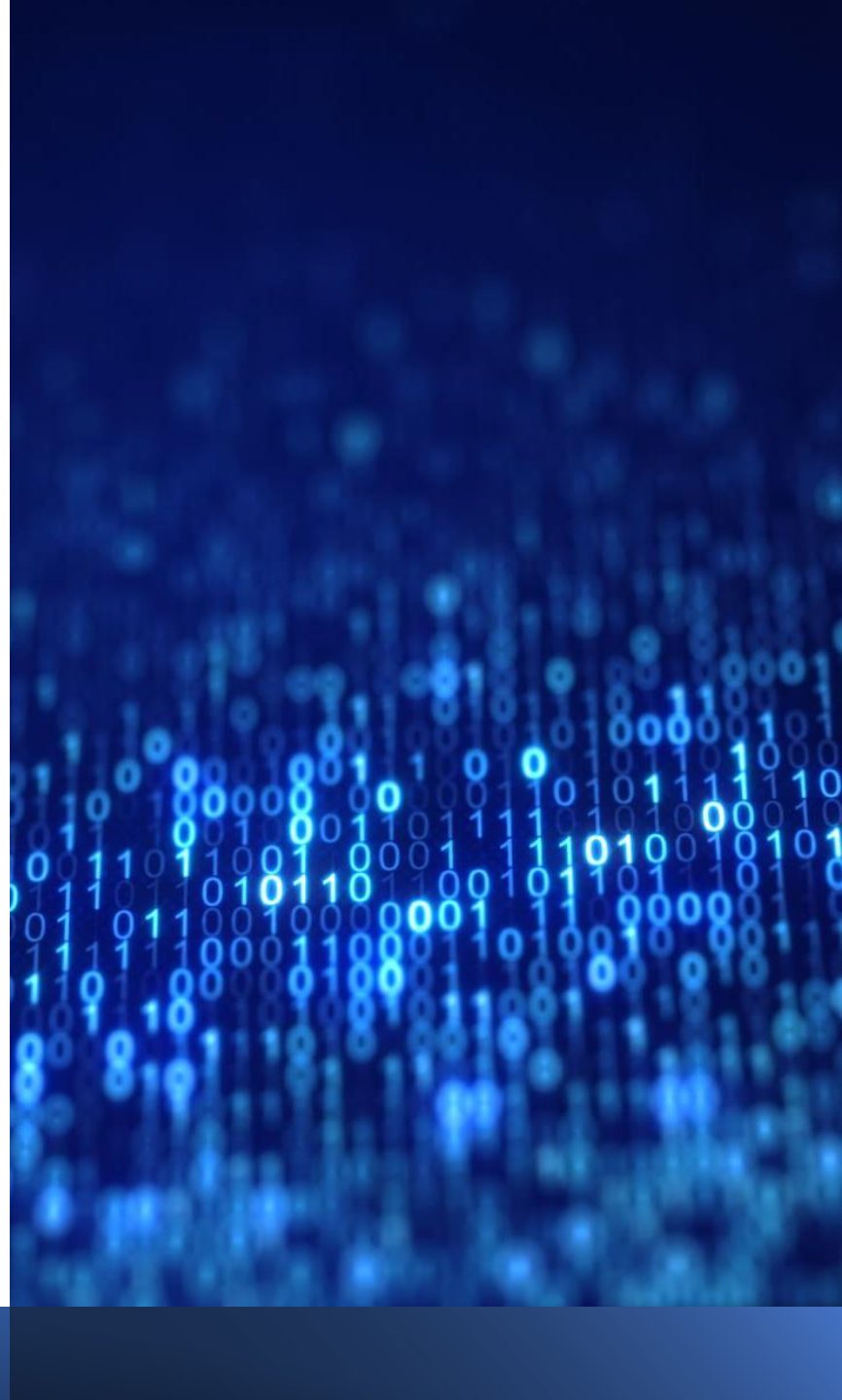
Work with schemas

- SQL Server database systems, tables are defined within schemas to create logical namespaces in the database
- Database systems such as SQL Server use a hierarchical naming system. This multi-level naming helps to disambiguate tables with the same name in different schemas. The fully qualified name of an object includes the name of a database server instance in which the database is stored, the name of the database, the schema name, and the table name. For example: **Server1.StoreDB.Sales.Order**.

Explore the structure of SQL statements

The SQL statements are grouped together into several different types of statements. These different types are:

- **Data Manipulation Language (DML)** is the set of SQL statements that focuses on querying and modifying data. DML statements include SELECT, the primary focus of this training, and modification statements such as INSERT, UPDATE, and DELETE.
- **Data Definition Language (DDL)** is the set of SQL statements that handles the definition and life cycle of database objects, such as tables, views, and procedures. DDL includes statements such as CREATE, ALTER, and DROP.
- **Data Control Language (DCL)** is the set of SQL statements used to manage security permissions for users and objects. DCL includes statements such as GRANT and DENY.
- We'll focus on DML statements. "CRUD" operations to create, read, update, or delete application data.



Examine the SELECT statement

- Select Statement
- Selecting all columns
- Selecting specific columns
- Selecting expressions
- Specifying column aliases

A null value in a relational database is used when the value in a column is unknown or missing. A null is neither an empty string (for character or datetime data types) nor a zero value (for numeric data types)

Work with data types

- [Data types (Transact-SQL) - SQL Server | Microsoft Learn]

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

Work with data types

- CAST and TRY_CAST
- CONVERT and TRY_CONVERT
- PARSE and TRY_PARSE
- STR

Handle NULLs

- A NULL value means no value or unknown.
- It does not mean zero or blank, or even an empty string.
  - ISNULL
  - NULLIF

Module 2: Sort and filter results in T-SQL

- The Transact-SQL SELECT statement supports sorting query results by applying the ORDER BY clause, and filtering results using the WHERE clause

Sort your results

- In the logical order of query processing, ORDER BY is the last phase of a SELECT statement to be executed.

- SELECT<select_list>
- FROM <table_source>
- ORDER BY <order_by_list> [ASC|DESC];

Limit the sorted results

- The TOP clause is a Microsoft-proprietary extension of the SELECT clause. TOP will let you specify how many rows to return, either as a positive integer or as a percentage of all qualifying rows.

- SELECT TOP (N) <column_list> FROM <table_source> WHERE <search_condition> ORDER BY <order list> [ASC|DESC];

- Using WITH TIES
- Using PERCENT

Page results

- An extension to the ORDER BY clause called OFFSET-FETCH enables you to return only a range of the rows selected by your query. It adds the ability to supply a starting point (an offset) and a value to specify how many rows you would like to return (a fetch value). This extension provides a convenient technique for paging through results.

Using OFFSET-FETCH

- 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

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

Remove duplicates

- DISTINCT
  - SELECT City, CountryRegion FROM Production.Supplier ORDER BY CountryRegion, City;
  - By default, the SELECT clause includes an implicit ALL keyword that results in this behavior:
  - SELECT ALL City, CountryRegion FROM Production.Supplier ORDER BY CountryRegion, City;
  - SELECT DISTINCT City, CountryRegion FROM Production.Supplier ORDER BY CountryRegion, City;

Filter data with predicates Completed

- The simplest SELECT statements with only SELECT and FROM clauses will evaluate every row in a table. By using a WHERE clause, you define conditions that determine which rows will be processed and potentially reduce result set.
- usually using the basic comparison operators:
  - = (equals)
  - <> (not equals)
  - > (greater than)
  - >= (greater than or equal to)
  - < (less than)
  - <= (less than or equal to)

Filter data with predicates Completed

- SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ProductCategoryID = 2;

Filter data with predicates Completed

- IS NULL / IS NOT NULL
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ProductName IS NOT NULL;
- Multiple conditions
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ProductCategoryID = 2 AND ListPrice < 10.00;
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE (ProductCategoryID = 2 OR ProductCategoryID = 3) AND (ListPrice < 10.00);

Comparison operators

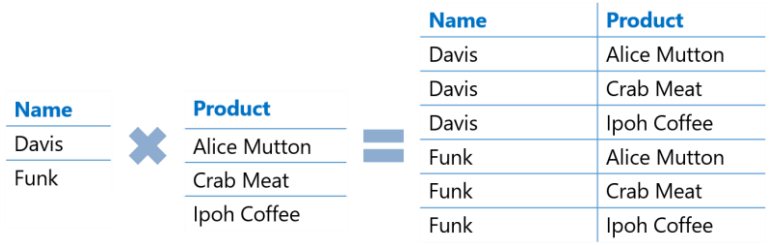
- IN
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ProductCategoryID = 2 OR ProductCategoryID = 3 OR ProductCategoryID = 4;
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ProductCategoryID IN (2, 3, 4);
- BETWEEN
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ListPrice >= 1.00 AND ListPrice <= 10.00;
  - SELECT ProductCategoryID AS Category, ProductName FROM Production.Product WHERE ListPrice BETWEEN 1.00 AND 10.00;
  - SELECT ProductName, ModifiedDate FROM Production.Product WHERE ModifiedDate BETWEEN '2012-01-01' AND '2012-12-31';
  - SELECT ProductName, ListPrice, ModifiedDate FROM Production.Product WHERE ModifiedDate BETWEEN '2012-01-01 00:00:00.000' AND '2012-12-31 23:59:59.999';

Comparison operators

- LIKE
  - SELECT Name, ListPrice FROM SalesLT.Product WHERE Name LIKE '%mountain%’;
  - The **%** wildcard represents any string of 0 or more characters, so the results include products with the word "mountain" anywhere in their name
  - SELECT ProductName, ListPrice FROM SalesLT.Product WHERE ProductName LIKE 'Mountain Bike Socks, _’;
  - The following results only include products that begin with "Mountain Bike Socks, " and a single character after:

Module 3: Combine multiple tables with JOINs in T-SQL

- Relational databases usually contain multiple tables that are linked by common key fields. This normalized design minimizes duplication of data but means that you'll often need to write queries to retrieve related data from two or more tables.



Name
Davis
Funk

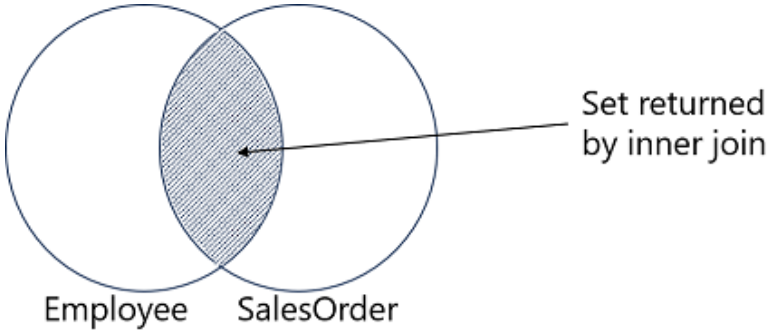
×

Product
Alice Mutton
Crab Meat
Ipoh Coffee

=

Name	Product
Davis	Alice Mutton
Davis	Crab Meat
Davis	Ipoh Coffee
Funk	Alice Mutton
Funk	Crab Meat
Funk	Ipoh Coffee

Use inner joins



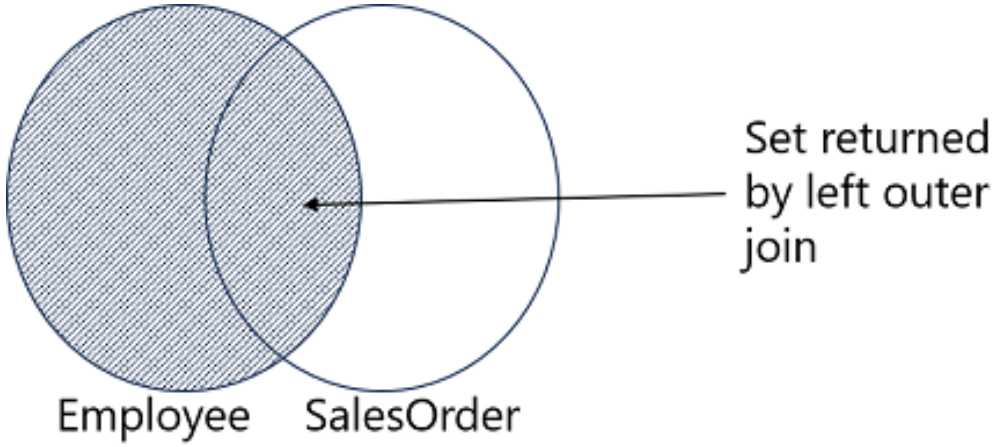
- The most frequent type of JOIN in T-SQL queries is INNER JOIN. Inner joins are used to solve many common business problems, especially in highly normalized database environments.
- To retrieve data that has been stored across multiple tables, you will often need to combine it via INNER JOIN queries.
- SELECT emp.FirstName, ord.Amount FROM HR.Employee AS emp JOIN Sales.SalesOrder AS ord ON emp.EmployeeID = ord.EmployeeID;
- The result of the completed query is a list of employees and their order amounts. Employees that do not have any associated orders have been filtered out by the ON clause, as have any orders that happen to have a EmployeeID that doesn't correspond to an entry in the HR.Employee table.

Use inner joins

- An INNER JOIN is the default type of JOIN, and the optional INNER keyword is implicit in the JOIN clause. When mixing and matching join types, it can be useful to specify the join type explicitly, as shown in this hypothetical example:
- SELECT emp.FirstName, ord.Amount FROM HR.Employee AS emp INNER JOIN Sales.SalesOrder AS ord ON emp.EmployeeID = ord.EmployeeID;

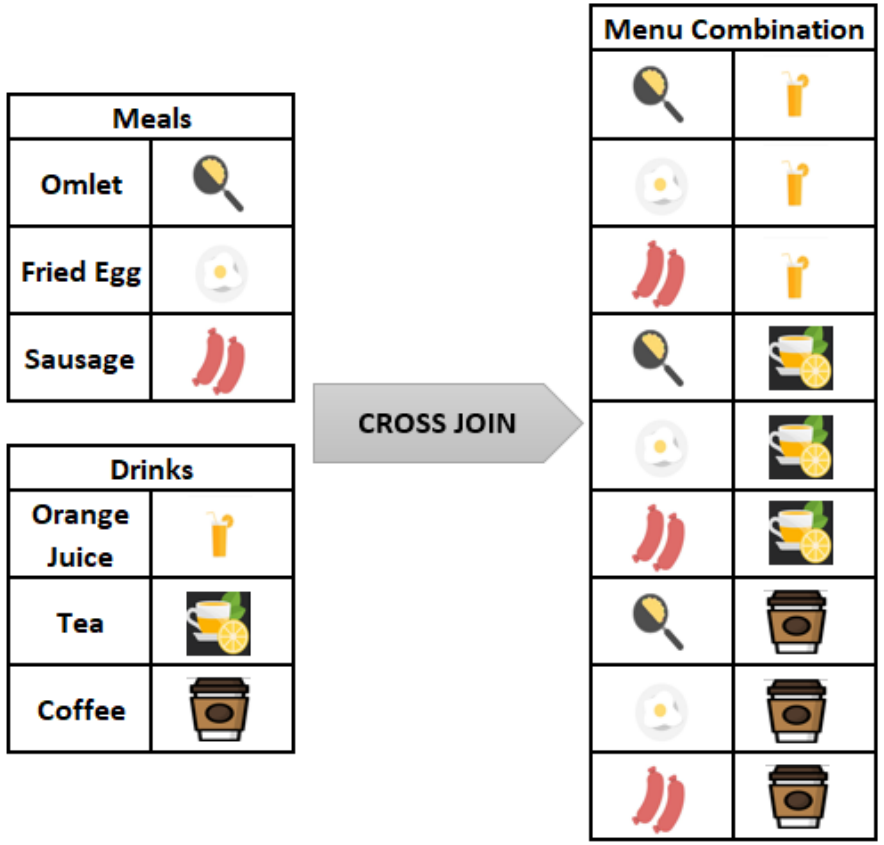
Use outer joins

- SELECT emp.FirstName, ord.Amount FROM HR.Employee AS emp LEFT OUTER JOIN Sales.SalesOrder AS ord ON emp.EmployeeID = ord.EmployeeID;
- This example uses a LEFT OUTER JOIN operator, which directs the query processor to preserve all rows from the table on the left (HR.Employee) and displays the Amount values for matching rows in Sales.SalesOrder. However, all employees are returned, whether or not they have taken a sales order. In place of the Amount value, the query will return NULL for employees with no matching sales orders.



Use cross joins

- SELECT <select_list> FROM table1 AS t1
  CROSS JOIN table2 AS t2;



Use self joins

- There may be scenarios in which you need to retrieve and compare rows from a table with other rows from the same table.

Module 4: Write Subqueries in T-SQL

- Sometimes, when using Transact-SQL to retrieve data from a database, it can be easier to simplify complex queries by breaking them down into multiple simpler queries that can be combined to achieve the desired results. Transact-SQL supports the creation of subqueries, in which an inner query returns its result to an outer query.
- Scalar subqueries (return a single value)
- Multi-valued subqueries (return results using the IN operator)

Write Subqueries in T-SQL

Labs

Module 5: Use built-in functions and GROUP BY in Transact-SQL

- When retrieving data from tables in a database, it's often useful to be able to manipulate data values by using functions; to format, convert, aggregate, or otherwise affect the output from the query

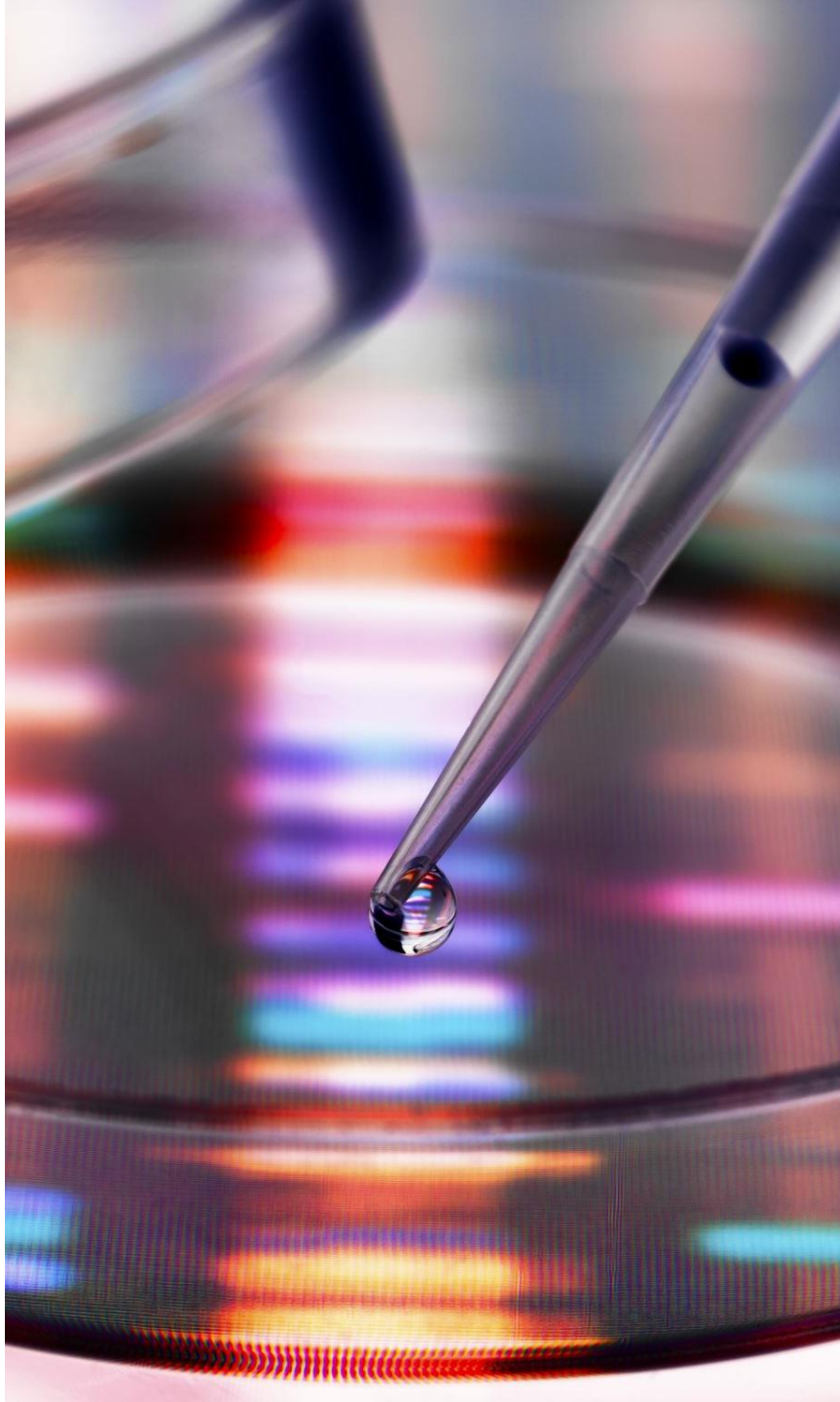
Categorize built-in functions

Functions in T-SQL can be categorized as follows:

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 in a FROM clause in a T-SQL statement.
Aggregate	Take one or more input values, return a single summarizing value.

Use scalar functions

- Lab



Use ranking and rowset functions

- Demo

Use aggregate functions

- Demo

Summarize data with GROUP BY

- Demo

Filter groups with HAVING

- Demo

Module 6: Modify data with T-SQL

- Data analysts and business users simply need to retrieve data from a database for reporting or analysis. However, when developing an application, or even during some complex analysis, you may need to insert, modify, or delete data.

Insert data

- INSERT [INTO] <Table> [(column_list)] VALUES ([ColumnName or an expression or DEFAULT or NULL],…n)

Generate automatic values

- The IDENTITY property

Update data

- UPDATE <TableName> SET <ColumnName> = { expression | DEFAULT | NULL } {,…n} WHERE <search_conditions>;

Delete data

- DELETE [FROM] <TableName> WHERE <search_conditions>;

Merge data based on multiple tables

- MERGE INTO schema_name.table_name AS TargetTbl USING (SELECT <select_list>) AS SourceTbl ON (TargetTbl.col1 = SourceTbl.col1) WHEN MATCHED THEN UPDATE SET TargetTbl.col2 = SourceTbl.col2 WHEN NOT MATCHED [BY TARGET] THEN INSERT (<column_list>) VALUES (<value_list>) WHEN NOT MATCHED BY SOURCE THEN DELETE;

END

Questions and Answers