

T-SQL





T-SQL History

- Developed in the early 1970 (SEQual)
- ANSI-SQL defined by the American National Standards Institute
- Microsoft implementation is T-SQL, or Transact SQL
- Other implementations include PL/SQL and IBM's SQL Procedural Language.



Categories of T-SQL Statements

- DML – Data Manipulation Language
- DCL – Data Control Language
- DDL – Data Definition Language
- TCL - Transactional Control Language
- DQL - Data Query Language (Select)





Data types

A data type determines the type of data that can be stored in a database column. The most commonly used data types are:

1. Alphanumeric: data types used to store characters, numbers, special characters, or nearly any combination.
2. Numeric
3. Date and Time



Executing Queries

Executing queries occurs when in a query session by:

- Selecting the Execute Icon
- Pressing the F5 key

Note:

Select the Database Before Executing Query or write

Use Keyword + DB Name on top of the Query

✓ Commenting T-SQL Code

- Comments are statements about the meaning of the code
- When used, there is no execution performed on the text

There are two ways to comment code using T-SQL:

- The use of a beginning /* and ending */ creates comments

/*
This is a comment
*/

multi line comment

- The double dash comments to the end of line

--This is a comment

single line comment

Batch

- Recall that a batch is a series of one or more statements submitted and executed at the same time
- Example:

```
delete sales
    where stor_id = "5023"
        and ord_num = "AB-123-DEF-425-1Z3"
delete salesdetail
    where stor_id = "5023"
        and ord_num = "AB-123-DEF-425-1Z3"
select * from sales
    where stor_id = "5023"
select * from salesdetail
    where stor_id = "5023"
go
```

Data Definition Language



Modifying Data in Tables

- Create Table
- Alter Table
- Drop Table

CREATE

+ table + tableNar



Create Table

Syntax:

```
CREATE TABLE [table_name]
("column 1" "data_type", "constraint"
"column 2" "data_type", ... )
```

Example

```
CREATE TABLE Employee
(ID int Primary Key,
FName char(50) Not Null,
LName char(50),
Address char(50),
Birth_Date date);
```



Drop Table

Syntax:

```
Drop Table "Table_Name"
```

Example

```
Drop Table Employee
```

Alter Table

Syntax:

```
ALTER TABLE table_name ADD column_name datatype
```

```
ALTER TABLE table_name DROP column_name
```

Example

```
ALTER TABLE employee ADD City varchar(30)
```

Modifying Data in Tables

Modifying Data in Tables

- Inserting Data into Tables
- Deleting Data from Tables
- Updating Data in Tables



INSERT Fundamentals

- The INSERT statement adds one or more new rows to a table
- INSERT inserts *data_values* as one or more rows into the specified *table_or_view*
- *column_list* is a list of column names used to specify the columns for which data is supplied

INSERT Syntax:

```
INSERT [INTO] table_or_view [(column_list)] data_values
```

INSERT Statement Examples

Using a Simple INSERT Statement

```
INSERT INTO Production.UnitMeasure  
VALUES (N'F2', N'Square Feet', GETDATE());
```

Inserting Multiple Rows of Data (Row Constructor)

```
INSERT INTO Production.UnitMeasure  
VALUES (N'F2', N'Square Feet', GETDATE()), (N'Y2',  
N'Square Yards', GETDATE());
```

INSERT using SELECT

```
INSERT INTO MyTable (PriKey, Description)  
SELECT ForeignKey, Description  
FROM SomeView
```

INSERT using EXECUTE

```
INSERT dbo.SomeTable  
EXECUTE SomeProcedure "predefined procedure"
```

DELETE Fundamentals

- The DELETE statement removes one or more rows in a table or view
- DELETE removes rows from the *table_or_view* parameter that meet the *search condition*
- *table_sources* can be used to specify additional tables or views that can be used by the WHERE clause

DELETE Syntax:

DELETE *table_or_view*

FROM *table_sources*

WHERE *search_condition*

DELETE Statement Definitions

DELETE with no WHERE clause

```
DELETE FROM SomeTable;
```

```
DELETE FROM Sales.SalesPerson;
```

DELETE using a Subquery

```
DELETE FROM SomeTable  
WHERE SomeColumn IN  
(Subquery Definition);
```

```
DELETE FROM  
Sales.SalesPersonQuotaHistory  
WHERE SalesPersonID IN  
(SELECT SalesPersonID  
FROM Sales.SalesPerson  
WHERE SalesYTD > 2500000.00);
```

Defining and Using the TRUNCATE Statement

TRUNCATE TABLE Syntax

```
TRUNCATE TABLE  
[ { database_name.[ schema_name ]. | schema_name . } ]  
table_name  
[ ; ]
```

TRUNCATE TABLE Example

```
TRUNCATE TABLE HumanResources.JobCandidate;
```



You cannot use TRUNCATE TABLE on tables that are referenced by a FOREIGN KEY constraint

TRUNCATE versus DELETE

TRUNCATE TABLE has the following advantages over DELETE:

- Less transaction log space is used
- Fewer locks are typically used

```
DELETE FROM Sales.SalesPerson;
```

```
TRUNCATE TABLE Sales.SalesPerson;
```

UPDATE Fundamentals

- The UPDATE statement changes data values in one, many, or all rows of a table
- An UPDATE statement referencing a table or view can change the data in only one base table at a time
- UPDATE has three major clauses:
 - SET – comma-separated list of columns to be updated
 - FROM – supplies values for the SET clause
 - WHERE – specifies a search condition for the SET clause

UPDATE Syntax:

UPDATE table_or_view

SET column_name = expression

FROM table_sources

WHERE search_condition

UPDATE Statement Definitions

Simple UPDATE Statement

```
UPDATE SomeTable  
SET Column = Value
```

```
UPDATE Sales.SalesPerson  
SET Bonus = 6000;
```

UPDATE with a WHERE clause

```
UPDATE SomeTable  
SET Column = Value  
WHERE SearchExpression
```

```
UPDATE Production.Product  
SET Color = N'Metallic Red'  
WHERE Name LIKE N'Road-250%'  
AND Color = N'Red';
```



Updating with Information from Another Table

UPDATE using a Subquery

```
UPDATE SomeTable  
SET Column = Value  
FROM SomeSubquery
```



```
UPDATE Sales.SalesPerson  
SET SalesYTD = SalesYTD + SubTotal  
FROM Sales.SalesPerson AS sp  
JOIN Sales.SalesOrderHeader AS so  
    ON sp.BusinessEntityID = so.SalesPersonID  
    AND so.OrderDate = (SELECT MAX(OrderDate)  
                        FROM Sales.SalesOrderHeader  
                        WHERE SalesPersonID =  
                              sp.BusinessEntityID);
```

Before

SalesYTD

677558.4653
4557045.0459

After

SalesYTD

721382.488
4593234.5123



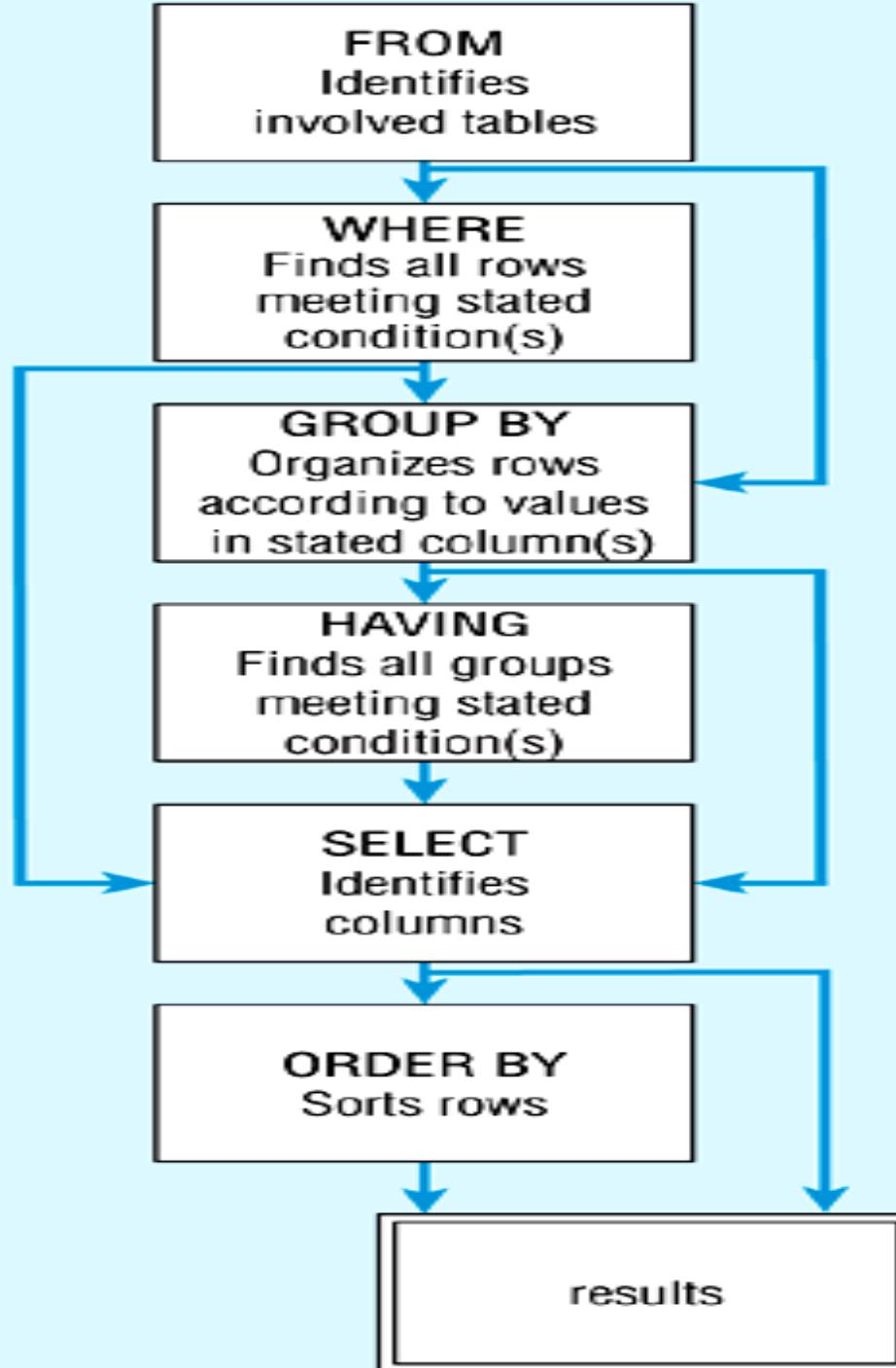
A faint, large watermark-like image of a traditional Chinese building with red lanterns is visible in the background.

Data Query Language

SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
 - **SELECT**
 - List the **columns** (and expressions) that should be returned from the query
 - **FROM**
 - Indicate the **table(s)** or view(s) from which data will be obtained
 - **WHERE**
 - Indicate the **conditions** under which a **row** will be included in the result
 - **GROUP BY**
 - Indicate **columns** to group the results
 - **HAVING**
 - Indicate the **conditions** under which a **group** will be included
 - **ORDER BY**
 - Sorts the result according to specified **columns**

SQL statement processing order



Retrieving Columns in a Table

Displays All Columns in the Employee Table

```
USE AdventureWorks2012;
GO
SELECT *
FROM HumanResources.Employee
```

Displays Only FirstName, LastName and JobTitle Columns

```
USE AdventureWorks2012;
GO
SELECT FirstName, LastName, JobTitle
FROM HumanResources.Employee
```

Filtering Data

- Retrieving Specific Rows in a Table
- Filtering Data by Using Comparison Operators
- Filtering Data by Using String Comparisons
- Filtering Data by Using Logical Operators
- Retrieving a Range of Values
- Retrieving a List of Values

Retrieving Specific Rows in a Table

Simple WHERE clause

```
USE ITI;  
GO  
SELECT *  
FROM Student  
WHERE Age >20
```

WHERE Clause Using a Predicate

```
USE ITI;  
GO  
SELECT *  
FROM Instructor  
WHERE Salary IS NULL;
```

Types of T-SQL Operators

Type	Operators
• Arithmetic operators	• +, -, *, /, % <i>Vacation + SickLeave AS 'Total PTO'</i>
• Assignment operator	• = <i>SET @MyCounter = 1</i>
• Comparison operators	• =, <, >, <>, !, >=, <= <i>IF (@MyProduct <> 0) ...</i>
• Logical operators	• AND, OR, NOT <i>WHERE Department = 'Sales' AND (Shift = 'Evening' OR Shift = 'Night')</i>
• String concatenation operator	• + <i>SELECT LastName + ', ' + FirstName AS Moniker</i>

Using Comparison Operators

- Comparison operators test whether two expressions are the same.
- Comparison operators return a Boolean value of TRUE, FALSE, or UNKNOWN.

Scalar Comparison Operators

```
=, <>, >, >=, <, <=, !=
```

```
USE AdventureWorks2012;
GO
SELECT FirstName, LastName, MiddleName
FROM Person.Person
WHERE ModifiedDate >= '01/01/2004'
```

Using String Comparisons

- String Comparisons are used for data types of text, ntext, char, nchar, varchar, and nvarchar
- Predicates are available for full or partial match comparisons

```
WHERE LastName = 'Johnson'
```

```
WHERE LastName LIKE 'Johns%n'
```

```
WHERE CONTAINS(LastName, 'Johnson')
```

```
WHERE FREETEXT(Description, 'Johnson')
```

Using Logical Operators

- Logical operators are used to combine conditions in a statement

Returns only rows with first name of ‘John’ and last name of ‘Smith’

```
WHERE FirstName = ‘John’ AND LastName = ‘Smith’
```

Returns all rows with first name of ‘John’ and all rows with last name of ‘Smith’

```
WHERE FirstName = ‘John’ OR LastName = ‘Smith’
```

Returns all rows with first name of ‘John’ and last name not equal to ‘Smith’

```
WHERE FirstName = ‘John’ AND NOT LastName = ‘Smith’
```

Operator Precedence

`~` (Bitwise Not)

`*(Multiply), /(Division), %(Modulo)`

`+(Positive), -(Negative), +(Add), (+Concatenate),
-(Subtract), ^(Bitwise Exclusive OR), |(Bitwise OR)`

Comparisons

`=, >, <, >=, <=, <>, !=, !>, !<`

NOT

AND

ALL, ANY, BETWEEN, IN, LIKE, OR, SOME

`=(Assignment)`

Retrieving a Range of Values

- BETWEEN tests for data values within a range of values.

```
SELECT *  
FROM Student  
WHERE age BETWEEN 25 AND 30
```

- BETWEEN uses the same logic as \geq AND \leq

```
SELECT *  
FROM Student  
WHERE age>=25 AND age<=30
```

Retrieving a List of Values

- IN tests a column's values against a list of possible values.

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice  
FROM Sales.SalesOrderDetail  
WHERE ProductID IN (750, 753, 765, 770)
```

- IN uses the same logic as multiple comparisons with the OR predicate between them

```
SELECT SalesOrderID, OrderQty, ProductID, UnitPrice  
FROM Sales.SalesOrderDetail  
WHERE ProductID = 750 OR ProductID = 753  
      OR ProductID = 765 OR ProductID = 770
```

Working with NULL Values

NULL is an UNKNOWN value

NULL is not a zero (0) value or an empty string

NULL values are not equal

Comparing NULL to any other value returns UNKNOWN

A NULL value cannot be included in a calculation.

The special SPARSE keyword can be used to conserve space
in columns that allow NULL values.

Use IS NULL to test for NULL values in an argument.

Work with NULL Values

ISNULL() returns a given value if the column value is NULL

```
SELECT ISNULL(st_fname, ' ')  
FROM Student;
```

NULLIF() returns NULL if both specified expressions are equal

```
select Nullif(st_age,dept_id) from Student  
      where St_Id=7
```

COALESCE() returns the first non NULL expression among its arguments, similar to a CASE statement

```
SELECT CAST(COALESCE(hourly_wage * 40 * 52, salary,  
      commission * num_sales) AS money) AS 'Total Salary'  
FROM wages
```

Formatting Result Sets

- Sorting Data
- Eliminating Duplicate Rows
- Labeling Columns in Result Sets
- Using String Literals
- Using Expressions

Sorting Data

```
SELECT LastName, FirstName, MiddleName  
FROM Person.Person  
ORDER BY LastName, FirstName
```

Eliminating Duplicate Rows

```
SELECT DISTINCT LastName, FirstName, MiddleName  
FROM Person.Person  
ORDER BY LastName, FirstName
```

Labeling Columns in Result Sets

- Aliases are used to create custom column headers in the result set display.
- You can rename actual or derived columns
- The optional **AS** clause can be added to make the statement more readable.
- Both statements below are equivalent

```
SELECT salary*12 as [annual salary]  
From instructor
```

Using String Literals

- String Literals:
- Are constant values.
- Can be inserted into derived columns to format data.
- Can be used as alternate values in functions, such as the ISNULL() function.

```
SELECT (LastName + ', ' + FirstName + ' ' +  
ISNULL(SUBSTRING(MiddleName, 1, 1), ' ')) AS Name  
FROM Person.Person  
ORDER BY LastName, FirstName, MiddleName
```

Using Expressions

- Using mathematical expressions in SELECT and WHERE clauses
- Using functions in expressions

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice *  
    1.1) AS NewPrice  
FROM Production.Product  
WHERE ListPrice > 0 AND (ListPrice/StandardCost) > .8
```

```
SELECT Name, ProductNumber, ListPrice AS OldPrice, (ListPrice *  
    1.1) AS NewPrice  
FROM Production.Product  
WHERE SellEndDate < GETDATE()
```

Joining Data from Multiple Tables

- Querying Multiple Tables by Using Joins
- Applying Joins for Typical Reporting Needs
- Combining and Limiting Result Sets

Querying Multiple Tables by Using Joins

- Fundamentals of Joins
- Categorizing Statements by Types of Joins
- Joining Data Using Inner Joins
- Joining Data Using Outer Joins
- Joining Data Using Cross Joins
- Identifying the Potential Impact of a Cartesian Product

Fundamentals of Joins

Joins:

- Select Specific Columns from Multiple Tables
 - JOIN keyword specifies that tables are joined and how to join them
 - ON keyword specifies join condition
- Query Two or More Tables to Produce a Result Set
 - Use Primary and Foreign Keys as join conditions
 - Use columns common to specified tables to join tables

Simplified JOIN Syntax:

```
FROM first_table join_type second_table [ON  
(join_condition)]
```

Categorizing Statements by Types of Joins

- Inner Join
 - Includes equi-joins and natural joins
 - Use comparison operators to match rows
- Outer Join
 - Includes left, right, or full outer joins
- Cross Join
 - Also called Cartesian products
- Self Join
 - Refers to any join used to join a table to itself

Identifying the Potential Impact of a Cartesian Product

A Cartesian Product:

- Is defined as all possible combinations of rows in all tables
- Results in a rowset containing the number of rows in the first table times the number of rows in the second
- Can result in huge result sets that take several hours to complete!

Joining Tables by Using Non-Equi Joins

- The same Operators and Predicates used for Inner Joins can be used for Not-Equal Joins

```
SELECT DISTINCT p1.ProductSubcategoryID, p1.ListPrice  
FROM Production.Product p1  
INNER JOIN Production.Product p2  
ON p1.ProductSubcategoryID = p2.ProductSubcategoryID  
AND p1.ListPrice <> p2.ListPrice  
WHERE p1.ListPrice < $15 AND p2.ListPrice < $15  
ORDER BY ProductSubcategoryID
```

Result Set:

ProductSubcategoryID	ListPrice
----------------------	-----------

23	8.99
23	9.50

...

(8 row(s) affected)



Equijoins and Non-Equijoins

- Using Equal Operator not satisfy all join conditions.
- Display employees information (name , salary , title) with salary grade for each employee.

```
SELECT e.name, e.salary, j.grade  
FROM employees e, job_grades j  
WHERE e.salary  
BETWEEN j.lowest_sal AND j.highest_sal;
```

Combining and Limiting Result Sets

- Combining Result Sets by Using the UNION Operator
- Limiting Result Sets by Using the EXCEPT and INTERSECT Operators
- Identifying the Order of Precedence of UNION, EXCEPT, and INTERSECT
- Limiting Result Sets by Using the TOP and TABLESAMPLE Operators
- Categorizing Statements that Limit Result Sets

Combining Result Sets by Using the UNION Operator

- UNION combines the results of two or more queries into a single result set that includes all the rows that belong to all queries in the union

```
SELECT * FROM testa  
UNION ALL  
SELECT * FROM testb;
```

Result Set:

columna	columnb
100	test
100	test
...	
(8 row(s) affected)	



The number and order of columns must be the same in all queries and all data types must be compatible



Limiting Result Sets by Using the EXCEPT and INTERSECT Operators

- EXCEPT returns any distinct values from the query to the left of the EXCEPT operand that are not also returned from the right query
- INTERSECT returns any distinct values that are returned by both the query on the left and right sides of the INTERSECT operand

EXCEPT Example:

```
SELECT ProductID  
FROM Production.Product  
EXCEPT  
SELECT ProductID  
FROM Production.WorkOrder
```

INTERSECT Example:

```
SELECT ProductID  
FROM Production.Product  
INTERSECT  
SELECT ProductID  
FROM Production.WorkOrder
```

Order of Precedence of UNION, EXCEPT, and INTERSECT

EXCEPT, INTERSECT, and UNION are evaluated in the context of the following precedence:

- 1 Expressions in parentheses
- 2 The INTERSECT operand
- 3 EXCEPT and UNION evaluated from Left to Right based on their position in the expression

Sub-Queries

- Find the names of employees whose working location = giza

```
Select name  
from Employee  
where Dno in  
( select Dnumber from dept  
      where location='giza')
```

Sub-Queries (Cont'd)

- Display department name with the highest paid employee
 - ✓ 1- Highest Salary
 - ✓ 2- Deptno for this Employee
 - ✓ 3- Department name

```
SELECT dname FROM dept  
WHERE deptno = (SELECT deptno FROM emp  
WHERE sal = (SELECT MAX(sal) FROM EMP));
```

Nested Queries (Cont'd)

- Find the names of employees whose salary is greater than the salary of the employees in department 5

Select Lname , Fname

From employee

Where salary > All (select salary
from employee where Dno=5)

Exists Condition

- Check if the result of correlated subquery is empty
- The EXISTS condition is considered "to be met" if the subquery returns at least one row.

Display suppliers information who have orders.

```
SELECT *
FROM suppliers
WHERE EXISTS
(select *
 from orders
 where suppliers.supplier_id = orders.supplier_id);
```

Exists Condition

- Retrieve the name of employees who have no dependents

Select name

From employee

Where **Not Exists** (select * from dependent where ssn=Essn)

Exists Condition With DML

```
DELETE FROM suppliers  
WHERE Not EXISTS  
(select *  
from orders  
where suppliers.supplier_id = orders.supplier_id);
```

Aggregate Functions

COUNT , SUM , MAX, MIN and AVG

- Find the sum of salaries of all employees , the maximum, the minimum salary, and the average salary

Select Sum(salary) , Max(salary), Min(salary), Avg(salary)
from Employees

Find the total number of employees in the company?

Find the number of employees in the research department?

Note : Group Functions ignore Null values in the columns

Grouping

Apply aggregate functions to a subgroups of tuples

For each department retrieve the department number , the number of employees in the department, and their average salary

Select dno , count(*) , avg(salary)

From employee Group by dno

Grouping (Cont'd)

- For each project on which more than two employees work, retrieve the project number, the project name , and the number of employees who work on the project

Select pnumber, pname ,count(works_on.pno)

From project , Works_on

Where Pnumber = Pno

Group by pnumber, pname

Having count(*) > 2

C. Data Control Language



- Grant Select on table employees to Ahmed;
- Grant All on Table department to Mary, Ahmed;
- Grant Select on table employees to Ahmed with grant Option;

- Revoke update on Table department From Mary;
- Revoke All on Table department From Mary, Ahmed;

Note: Example in the notes

SQLTutorials

- ANSI SQL URL : http://www.w3schools.com/SQL/sql_join.asp
- *MS SQL URL :*

<http://msdn.microsoft.com/en-us/library/bb264565.aspx>