About main source

Course Ref. 2071

T-SQL Query

Contents

T-SQL Query

- **Data Definition Language (CREATE, ALTER, DROP)**
- **Data Control Language (GRANT, DENY, REVOKE)**
- **Data Manipulation Language(SELECT, INSERT, UPDATE, DELETE)**
- **Batch Directives (GO, EXEC)**
- **Variables**

```
USE northwind

DECLARE  @EmpID  varchar(11),@vlName char(20)

SET @vlname = 'Dodsworth'

SELECT @EmpID = employeeid  FROM  employees

 WHERE LastName = @vlname

SELECT @EmpID AS EmployeeID

GO
```

- **System Functions (Aggregate Functions, Scalar Functions(DB_NAME()),Rowset Functions)**

```
SELECT *
 FROM OPENQUERY
  (OracleSvr, 'SELECT name, id FROM owner.titles')
```

- **Using Comparison Operators**

```
USE northwind

SELECT lastname, city

FROM employees

WHERE country = 'USA'

GO
```

Using String Comparisons

```
USE northwind

SELECT companyname

FROM customers

WHERE companyname LIKE '%Restaurant%'

GO
```

Using Logical Operators

```
USE northwind

SELECT productid, productname, supplierid, unitprice
 FROM  products
 WHERE (productname LIKE 'T%' OR productid = 46)
  AND  (unitprice > 16.00)

GO
```

Retrieving a Range of Values

```
USE northwind

SELECT productname, unitprice

FROM products

WHERE unitprice BETWEEN 10 AND 20

GO
```

Using a List of Values as Search Criteria

```
USE northwind

SELECT companyname, country

FROM suppliers

WHERE country IN ('Japan', 'Italy')

GO
```

- **Retrieving Unknown Values**

```
USE northwind

SELECT companyname, fax

FROM suppliers

WHERE fax IS NULL

GO
```

- **Sorting Data**

```
USE northwind

SELECT productid, productname, categoryid, unitprice

 FROM products

 ORDER BY categoryid, unitprice DESC

GO
```

- **Eliminating Duplicate Rows**

```
USE northwind

SELECT DISTINCT country

 FROM suppliers

 ORDER BY country

GO
```

- **Changing Column Names**

```
USE northwind

SELECT  firstname AS First, lastname AS Last

              ,employeeid AS 'Employee ID:'

FROM employees
GO
```

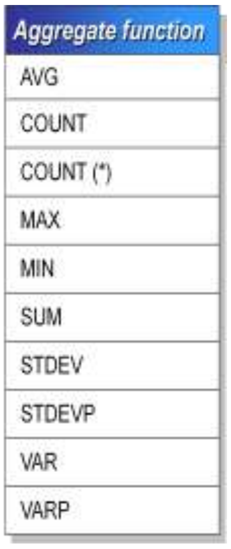
- **Using Literals**

```
USE northwind

SELECT  firstname, lastname

              ,'Identification number:', employeeid

FROM employees
GO
```

- **Listing the TOP *n* Values**

```
USE northwind

SELECT TOP 5 orderid, productid, quantity

 FROM [order details]

 ORDER BY quantity DESC
GO

USE northwind

SELECT TOP 5 WITH TIES orderid, productid, quantity

 FROM [order details]

 ORDER BY quantity DESC
GO
```

- **Using Aggregate Functions**



Aggregate function
AVG
COUNT
COUNT (*)
MAX
MIN
SUM
STDEV
STDEVP
VAR
VARP

- **Using Aggregate Functions with Null Values**

```
USE northwind

SELECT COUNT (*)

 FROM employees
GO

USE northwind

SELECT COUNT(reportsto)

 FROM employees

GO
```

- **Using the GROUP BY Clause**

```
USE northwind

SELECT productid

   ,SUM(quantity) AS total_quantity

 FROM orderhist

 WHERE productid = 2

 GROUP BY productid
GO
```

- **Using the GROUP BY Clause with the HAVING Clause**

```
USE northwind

SELECT productid, SUM(quantity)

   AS total_quantity

 FROM orderhist

 GROUP BY productid

 HAVING SUM(quantity)>=30
GO
```

- **Using the GROUP BY Clause with the ROLLUP Operator**

```
USE northwind

SELECT productid, orderid, SUM(quantity) AS total_quantity

 FROM orderhist

 GROUP BY productid, orderid

 WITH ROLLUP

 ORDER BY productid, orderid
GO
```

- **Using the GROUP BY Clause with the CUBE Operator**

```
USE northwind

SELECT productid, orderid, SUM(quantity) AS total_quantity

 FROM orderhist

 GROUP BY productid, orderid

 WITH CUBE

 ORDER BY productid, orderid
GO
```

- **Using the GROUPING Function**

```
SELECT productid, GROUPING (productid)

     ,orderid, GROUPING (orderid)

     ,SUM(quantity) AS total_quantity

 FROM orderhist

 GROUP BY productid, orderid

 WITH CUBE

 ORDER BY productid, orderid
GO
```

Using the COMPUTE and COMPUTE BY Clauses

```
USE northwind

SELECT productid, orderid

     ,quantity

FROM orderhist

ORDER BY productid, orderid

COMPUTE SUM(quantity)
GO

USE northwind

SELECT productid, orderid, quantity

 FROM orderhist

 ORDER BY productid, orderid

 COMPUTE SUM(quantity) BY productid

 COMPUTE SUM(quantity)
GO
```

- **Using Aliases for Table Names**

```
USE joindb

SELECT buyer_name, s.buyer_id, qty

 FROM buyers AS b  INNER JOIN sales AS s

  ON b.buyer_id = s.buyer_id

GO
```

- **Using Inner Joins**

```
USE joindb

SELECT buyer_name, sales.buyer_id, qty

FROM buyers  INNER JOIN sales

ON buyers.buyer_id = sales.buyer_id
GO
```

- **Using Outer Joins**

```
USE joindb

SELECT buyer_name, sales.buyer_id, qty

 FROM buyers  LEFT OUTER JOIN sales

  ON buyers.buyer_id = sales.buyer_id
GO
```

- **Using Cross Joins**

```
USE joindb

SELECT buyer_name, qty

 FROM buyers

 CROSS JOIN sales
GO
```

- **Joining More Than Two Tables**

```
SELECT buyer_name, prod_name, qty

 FROM buyers

 INNER JOIN sales

  ON buyers.buyer_id = sales.buyer_id

 INNER JOIN produce

  ON sales.prod_id = produce.prod_id
GO
```

- **Joining a Table to Itself**

```
USE joindb

SELECT a.buyer_id AS buyer1, a.prod_id

     ,b.buyer_id AS buyer2

 FROM sales AS a

 JOIN sales AS b

  ON a.prod_id = b.prod_id

WHERE  a.buyer_id > b.buyer_id

GO
```

- **Combining Multiple Result Sets**

```
USE northwind

SELECT  (firstname + ' ' + lastname) AS name
      ,city, postalcode

 FROM employees

UNION

SELECT companyname, city, postalcode

 FROM customers

GO
```

- **Using a Subquery as a Derived Table**

```
USE northwind

SELECT T.orderid, T.customerid

 FROM ( SELECT orderid, customerid

        FROM orders ) AS T

GO
```

- **Using a Subquery as an Expression**

```
USE pubs

SELECT title, price

     ,( SELECT AVG(price) FROM titles) AS average

     ,price-(SELECT AVG(price) FROM titles) AS difference

 FROM titles

 WHERE type='popular_comp'

GO
```

- **Mimicking a JOIN Clause**

```
USE pubs

SELECT DISTINCT t1.type

 FROM titles AS t1

 WHERE t1.type IN

   (SELECT t2.type

     FROM titles AS t2

     WHERE t1.pub_id <> t2.pub_id)

GO
```

- **Mimicking a HAVING Clause**
  - **Subquery with the Same Result As a HAVING Clause**

```
USE pubs

SELECT t1.type, t1.title, t1.price

 FROM titles AS t1

 WHERE t1.price > ( SELECT AVG(t2.price)  FROM titles AS t2

                    WHERE t1.type = t2.type )

GO
```

  - **Using a HAVING Clause Without a Subquery**

```
USE pubs

SELECT t1.type, t1.title, t1.price

 FROM titles AS t1

 INNER JOIN titles AS t2  ON t1.type = t2.type

 GROUP BY t1.type, t1.title, t1.price

  HAVING t1.price > AVG(t2.price)

GO
```

- **Using the EXISTS and NOT EXISTS Clauses**

```
USE northwind

SELECT lastname, employeeid

 FROM employees AS e

 WHERE EXISTS (SELECT * FROM orders AS o

               WHERE e.employeeid = o.employeeid

                AND  o.orderdate = '9/5/97')

GO
```

- **Inserting a Row of Data by Values**

```
USE northwind

INSERT customers

      (customerid, companyname, contactname, contacttitle
     ,address, city, region, postalcode, country, phone
     ,fax)

VALUES ('PECOF', 'Pecos Coffee Company', 'Michael Dunn'
         ,'Owner', '1900 Oak Street', 'Vancouver', 'BC'
         ,'V3F 2K1', 'Canada', '(604) 555-3392'
         ,'(604) 555-7293')

GO
```

- **Using the INSERT…SELECT Statement**

```
USE northwind

INSERT customers

 SELECT substring(firstname, 1, 3)

        + substring (lastname, 1, 2)

      ,lastname, firstname, title, address, city

      ,region, postalcode, country, homephone, NULL

  FROM employees

GO
```

- **Creating a Table Using the SELECT INTO Statement**

```
USE northwind

SELECT productname AS products

     ,unitprice AS price

     ,(unitprice * 1.1) AS tax

 INTO #pricetable

 FROM products

GO
```

- **Inserting Partial Data**

```
USE northwind

INSERT shippers (companyname)

VALUES ('Fitch & Mather')

GO

USE northwind

SELECT *

FROM shippers

WHERE companyname = 'Fitch & Mather'

GO
```

- **Inserting Data by Using Column Defaults**

```
USE northwind

INSERT shippers (companyname, phone)

 VALUES ('Kenya Coffee Co.', DEFAULT)

GO
```

- **Using the DELETE Statement**

```
USE northwind
DELETE orders
 WHERE DATEDIFF(MONTH, shippeddate, GETDATE()) >= 6

GO
```

- **Using the TRUNCATE TABLE Statement**

```
USE northwind

TRUNCATE TABLE orders

GO
```

- **Updating Rows Based on Data in the Table**

```
USE northwind

UPDATE products

 SET unitprice = (unitprice * 1.1)

GO
```

- **Getting Information About Full-Text Indexes**
  - **sp_help_fulltext_catalogs**
  - **sp_help_fulltext_tables**
  - **sp_help_fulltext_columns**
  - **Using Transact-SQL Functions**

```
USE northwind

SELECT

 DATABASEPROPERTY('Northwind','IsFullTextEnabled')

GO
```

- **CONTAINS Predicate**

```
SELECT plant_id, common_name, price

 FROM plants

 WHERE CONTAINS( *, ' "English Thyme" ' )

GO
```

- **FREETEXT Predicate**

```
SELECT *

 FROM news_table

 WHERE FREETEXT( description,

 '"The Fulton County Grand Jury said Friday an

 investigation of Atlanta‘s recent primary

 election produced no evidence that any

 irregularities took place."')

GO
```

- **CONTAINS and FREETEXT Predicates**

```
USE northwind

SELECT Description

 FROM Categories

 WHERE CategoryName <> 'Seafood‘

  AND  CONTAINS(Description, ' sauces AND seasonings ')

GO
```

- **CONTAINS Within a Subquery**

```
USE pubs

SELECT T.title, P.pub_name

 FROM publishers AS P

 INNER JOIN titles AS T  ON P.pub_id = I.pub_id

 WHERE P.pub_id = (SELECT pub_id FROM pub_info WHERE CONTAINS

       (pr_info, ' moonbeam AND ontario AND "flying saucer" '))

GO
```

- **Displaying the Text of a Programming Object**

```
USE library

EXEC sp_helptext 'dbo.OverdueView'

GO
```

- **Defining Views**

```
USE library

GO

CREATE VIEW dbo.UnpaidFinesView (Member, TotalUnpaidFines)

AS

SELECT member_no, (sum(fine_assessed-fine_paid))

 FROM loanhist

 GROUP BY member_no

 HAVING SUM(fine_assessed-fine_paid) > 0
GO

SELECT *

 FROM UnpaidFinesView

GO
```

- **Example: Viewing Information from Multiple Tables**

```
USE library

GO

CREATE VIEW dbo.birthdayview

  (lastname, firstname, birthday)

AS

SELECT lastname, firstname

      ,CONVERT(char(8), birth_date, 2)

 FROM member

 INNER JOIN juvenile

  ON member.member_no = juvenile.member_no

GO
```

- **Creating a User-defined Function**

```
USE northwind

GO

CREATE FUNCTION fn_NewRegion ( @myinput nvarchar(30) )

  RETURNS nvarchar(30)

BEGIN

  IF @myinput IS NULL

   SET @myinput = 'Not Applicable'

  RETURN @myinput

END

GO
```

Recommended Practices

1. Use SQL Query Analyzer to Work Graphically and Interactively
2. Use the Object Browser to Locate and Script Objects
3. Use Templates as Starting Points to Create Objects
4. Use the osql Command-line Utility for Batch Files and Scheduling
5. Save Commonly Used Transact-SQL Scripts to Files
6. Use the DISTINCT Clause to Eliminate Duplicate Rows in the Result Set
7. Improve the Readability of a Result Set by ChangingColumn Names or by Using Literals
8. In Multi-Line Column Lists, Place Commas Before theColumn Names, Excluding the First Column
9. Index Frequently Aggregated Columns
10. Avoid Using Aggregate Functions with Null Values
11. Use the ORDER BY Clause to Guarantee a Sort Order
12. Use the ROLLUP Operator Instead of the CUBE Operator
13. Avoid Using the COMPUTE or COMPUTE BY Clause
14. Join Tables on Primary and Foreign Keys
15. Reference All Columns of Composite Primary Key in the ON Clause When Composite Key Relates Tables
16. Limit the Number of Tables in a Join
17. Use Subqueries to Break Down a Complex Query
18. Use Table Name Aliases for Correlated Subqueries
19. Use the INSERT…SELECT Statement to Add Rows from Other Sources to an Existing Table
20. Use the EXISTS Operator Instead of the IN Operator
21. Always Write a SELECT Statement That Does Not Modify Data Before You Actually Modify Data
22. Improve the Readability of a Result Set by Changing Column Names or by Using Literals
23. Always Include a WHERE Clause with the DELETE and UPDATE Statements
24. Use Full-Text Indexes on CHAR, NCHAR, VARCHAR, NVARCHAR, TEXT, NTEXT and IMAGE Data Types
25. Use the Full-Text Index and Catalog Properties for Troubleshooting
26. Use the *top_n_by_rank* Argument to Restrict Result Set Size
27. Verify Object Definition Text with EXEC sp_helptext
28. Use Views to Capture and Reuse Queries
29. Use Stored Procedures to Encapsulate Complex Procedures
30. Use User-defined Functions to Encapsulate Expressions

Performance Considerations

1. Not Search Conditions May Slow Data Retrieval
2. LIKE Search Conditions Slow Data Retrieval
3. Exact Matches or Ranges May Speed Data Retrieval
4. ORDER BY Clause May Slow Data Retrieval
5. All Data Modifications Occur Within a Transaction
6. Data Page Allocation May Occur
7. Modifying Indexed Data Incurs Additional Overhead
8. Indexes Can Assist Search Criteria