October 9, 2017

Programming SQL Server

Trainer: Do Thi Thanh Thanh

DXC.technology

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
  - Have the basic knowledge about MS SQL Server
  - Be able to create database objects in MS SQL Server
  - Be able to retrieve and execute queries in MS SQL Server

Agenda

- Introduction
- SQL Database Objects
- SQL Elements
- Q & A

Course Audience and Prerequisite

- The course is for technical associates
- The following are prerequisites to this course:
  - Basic SQL
  - Basic knowledge about relation database

Assessment Disciplines

- Class Participation: 100%
- Assignment: 50%
- Final Exam: 50%
- Passing Scores: 7

Duration and Course Timetable

- Course Duration: 6 hours
- Course Timetable:
  - Break 15 minutes

Further References

- http://msdn.microsoft.com/en-us/library/bb418439 (v=SQL.10).aspx
- https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation

Set Up Environment

- To complete the course, your PC must install:
  - MS SQL Server Express

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

SQL Database Objects

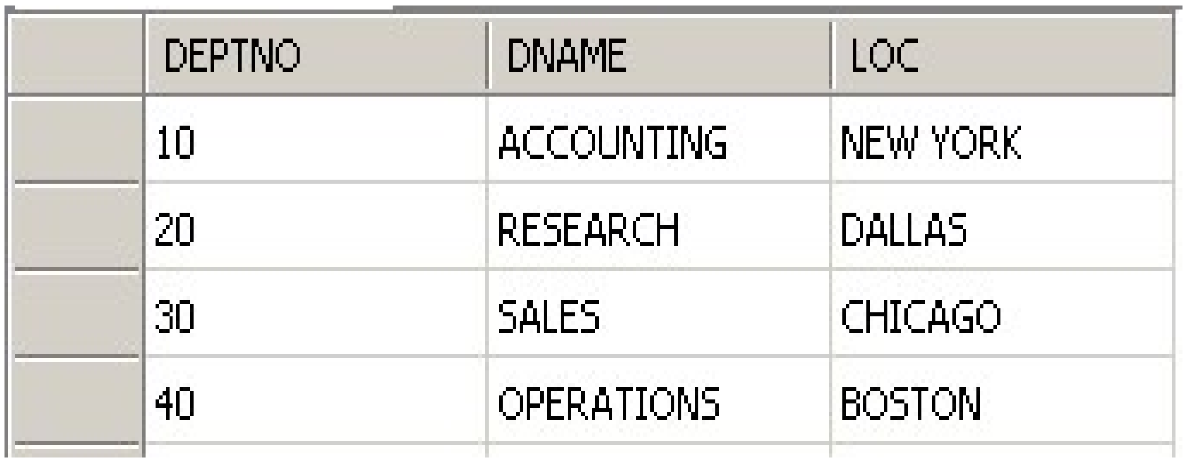
SQL Database Objects

- Tables
- Views
- Indexes
- Triggers
- Functions
- Stored Procedures

SQL Database Objects - Tables

- Contain the data stored in a database
- Are composed of rows and columns

<u>Example</u>



	DEPTNO	DNAME	LOC
	10	ACCOUNTING	NEW YORK
	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

SQL Database Objects - Tables

- 2 types
  - System tables: specify the configuration of the server
  - User-defined tables
    - *Normal tables*
    - *Partitioned tables*: are used in large table to manage subsets data easily
    - *Temporary tables*: are used to keep data at the execute-time, including local temporary tables and global temporary tables

SQL Database Objects - Tables

- Constraints: define the rules relating to the values permitted in columns of table
  - NOT NULL
  - CHECK
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY

SQL Database Objects - Tables

- System data types: almost is used to define data type of columns in table

Exact numerics	Int, bigint, smallint, tinyint, bit, decimal, numeric, money, smallmoney
Approximate numerics	Float, real
Date and time	Datetime, smalldatetime
Character strings	Char, varchar, text
Unicode character strings	nchar, nvarchar, ntext
Binary strings	Binary, varbinary, image
Other data type	**Cursor**, timestamp, sql variant, uniqueidentifier, **table**, xml

SQL Database Objects - Tables

- Use **CREATE TABLE** statement to create a table

  CREATE TABLE *table_name*

  ***(** { <column_name> <data_type> <constraint>} **)** [;]*

Example:

```
CREATE TABLE Dept_BK(
DeptNo   INT not null,
DName    VARCHAR(14),
Loc      VARCHAR(13));
```

SQL Database Objects - Tables

- Use **ALTER TABLE** statement to modify the structure of table: add/drop columns, enable/disable triggers and constraints

Example:

```
ALTER TABLE Dept_BK ADD Description VARCHAR(1000);
ALTER TABLE Dept_BK DROP COLUMN Description;
ALTER TABLE Emp DISABLE TRIGGER tg_EmpDel;
ALTER TABLE Emp ENABLE TRIGGER tg_EmpDel;
```

SQL Database Objects - Tables

- Use **DROP TABLE** statement to drop a table in a database

  DROP TABLE *table_name*

Example:

```
DROP TABLE Dept_BK;
```

SQL Database Objects - Views

- Are virtual tables which contain data is defined on a query
- Can be used to combine data from multiple tables and make it accessible from a single result set

SQL Database Objects - Views

- Use **CREATE VIEW** statement to create a view in a current database

  CREATE VIEW *view_name* [**(***column* [**,**...*n*] **)**] **AS** *select_statement*

Example:

```
CREATE VIEW vw_GetEmpInfo (id, name, job) AS
SELECT Empno, Ename, Job
FROM    Emp;
```

- Use **ALTER VIEW** statement to change the view

  ALTER VIEW *view_name* [**(***column* [**,**...*n*] **)**] **AS** *select_statement*

SQL Database Objects - Views

Example:

```
ALTER VIEW w_GetEmpInfo (id, name, job, dept) AS
SELECT e.EmpNo, e.Ename, e.Job,  m.Dname
FROM Emp e INNER JOIN Dept  m ON    DeptNo=m.DeptNo
```

- Use **DROP VIEW** statement to delete the view

  DROP VIEW *view_name*

  Example:

  ```
  DROP VIEW vw_GetEmpInfo;
  ```

SQL Database Objects – Indexes

- Are data structure associated with a table or view
- Are used to speed up the retrieval of rows from its associated table or view
- Advantage
  - Improve performance of query
  - Reduce accessing of I/O disk
- Disadvantage
  - Take disk space
  - Performance of query can be slowed down if using much more indexes in a table or placing wrong indexes for columns

SQL Database Objects – Indexes

- Usage

CREATE {CLUSTERED | NONCLUSTERED} INDEX *Index_name* **ON** *table_name*

AS *(<column_name> [ASC | DESC])*

Example:

```
CREATE INDEX idxEmpolyee_Dept ON Emp
(DeptNo ASC,
 EmpNo ASC)
```

SQL Database Objects – Indexes

- When designing an index, specify items below
  - Understand the characteristics of database
  - Understand the most frequently used queries
  - Be familiar with the columns used in queries
  - Determine with index options enhance performance
  - Decide the optimal storage location for the index

SQL Database Objects – Triggers

- Purpose: perform other actions after / before / at the time a standard action of event happens in database
- 2 types
  - Data Manipulation Language (DML) triggers: response to DML statements including INSERT, UPDATE, DELETE statements
  - Data Definition Language (DDL) triggers: response to DDL statement such as CREATE, ALTER, DROP statements

SQL Database Objects – Triggers

- Usage
  - <u>DDL triggers</u>

CREATE TRIGGER *trigger_name*

ON *{ ALL SERVER | DATABASE }*

{ FOR | AFTER } { event_type | event_group} [,…n] { INSERT | UPDATE | DELETE }

***AS** { <sql_statement>}*

  - <u>DML triggers</u>

CREATE TRIGGER *trigger_name*

ON *{table_name | view_name}*

{ FOR | AFTER | INSTEAD OF } { INSERT | UPDATE | DELETE }

***AS**{ <sql_statements> }*

SQL Database Objects – Triggers

Example:

```
CREATE TRIGGER tg_EmpDel ON  Emp
   INSTEAD OF DELETE
AS
BEGIN
   SET NOCOUNT ON

   RAISERROR (‘You should not delete it, just mark it as deleted', 10, 1);

   IF @@TRANCOUNT > 0
             ROLLBACK TRANSACTION;
END
```

SQL Database Objects – Triggers

- When designing a trigger, specify items below
  - A table can have only one type of INSTEAD OF trigger. It is used to override standard actions of the event that they would trigger on
  - A table can have multiple ALTER triggers as long as they have different name
  - Can not create DML trigger on system or temporary table

SQL Database Objects – Functions

- Purpose: perform a certain set of operations and return value appropriate
- 2 types
  - *System functions*

Aggregate functions	MAX(), MIN(), COUNT(), …
Mathematical functions	TAN(), SIN(), COS(), POWER(), …
Security functions	SUSER_ID(), CURRENT_USER(), IS_ MEMBER(), SESSION_USER(), …
String functions	ASCII(), CHAR(), STR(), RTRIM(), …
Date/time functions	DATEADD(), DATEDIFF(), GETDATE,…
Conversion functions	CAST(), CONVERT()
…	

SQL Database Objects – Functions

Example:

- To get the total number of employees in company

```
SELECT  COUNT(EmpNo)
FROM Emp
WHERE DelFlag <>
```

- To convert the number to char

```
SELECT  CAST 123 AS VARCHAR
```

Or

```
SELECT  CONVERT(VARCHAR(5), 123)
```

SQL Database Objects – Functions

- *User-defined functions*: can return scalar-value or table-valued
  - Scalar-valued functions

```
CREATE FUNCTION function_name
( <@parameter1><data_type> )
RETURNS <data_type>
[AS]
BEGIN
  -- Declare the return variable here
  DECLARE <@return_value> <data_type>
  -- Add the T-SQL statements to compute the return value here
  SELECT <@return_value> = <@parameter1>
  -- Return the result of the function
  RETURN <@return_value>
END[;]
```

SQL Database Objects – Functions

Example

```
CREATE FUNCTION
(@sString1              VARCHAR(10),
@sString2               VARCHAR(10))
RETURNS INT
AS
BEGIN
        DECLARE @iResult INT
        IF LEN(@sString1) >= LEN(@sString2)  SET @iResult = 1
        ELSE    SET @iResult = 0
        RETURN @iResult
END
```

SQL Database Objects – Functions

- <u>Inline table-valued functions</u>

CREATE FUNCTION *function_name*

(*<@parameter1><data_type>,*

  <@parameter2> <data_type>)

RETURNS TABLE

[AS]

RETURN

(

  -- Add the SELECT statement with parameters reference here

  SELECT 0

)[;]

SQL Database Objects – Functions

Example

```
CREATE FUNCTION GetEmpInfor
(@EmpNo int)
RETURNS TABLE
AS
RETURN
(
   SELECT e.EmpNo AS ID, e.EName AS Name, e.Job, m.EName AS Manager
   FROM    Emp e LEFT OUTER JOIN Emp m ON e.Mgr = m.EmpNo
   WHERE  e.EmpNo = @EmpNo
)
```

SQL Database Objects – Functions

- Multi-statement table-valued functions

CREATE FUNCTION *function_name*

(*<@parameter1>   <data_type>,*

  <@parameter2>   <data_type>)

RETURNS *@return_variable* **TABLE** < table_type_definition >

[AS]

BEGIN

  -- *Function body*

  RETURN

END*[;]*

SQL Database Objects – Functions

Example

```
CREATE FUNCTION GetEmpInfor(@EmpNo int)
RETURNS @EmpInfor TABLE (ID int,
                         Name varchar(10),
                         Job varchar(9),
                         Manager varchar(10))
BEGIN
    INSERT INTO @EmpInfor
    SELECT e.EmpNo, e.EName, e.Job, m.EName
    FROM    Emp e LEFT OUTER JOIN Emp m ON e.Mgr = m.EmpNo
    WHERE  e.EmpNo = @EmpNo
    RETURN
END
```

SQL Database Objects – Functions

- To drop a SP in database

  DROP { PROC | FUNCTION } *function_name;*

  Example:

  ```
  DROP FUNCTION GetEmpInfor;
  ```

SQL Database Objects – Functions

- To execute/test function, use **SELECT** statement
  - Scalar-valued functions

    SELECT *function_name (<parameters>)[;]*
  - Table-valued functions

    SELECT * **FROM** *function_name (<parameters>)[;]*

Example:

```
SELECT CompareString('Johnson', 'Peter')

SELECT * FROM  GetEmpInfor (7369)
```

SQL Database Objects – Stored Procedures

- Are methods for storing and executing T-SQL programs
- Can be called and executed by application
- What do they do?
  - Accept input parameter and return multiple values as output parameter
  - Contain programming statements that perform operations such as calling other SPs
  - Return a status value to a calling SP, indicating success or failure

SQL Database Objects – Stored Procedures

- Why do we use it?
  - Improve security
  - Allow modular programming
  - Allow for delayed binding
  - Reduce network traffic

SQL Database Objects – Stored Procedures

- 2 types
  - <u>System procedures</u>: are used to perform administrator task in SQL Server, have a prefix **sp_**

    <u>Example:</u> sp_changedbowner, …
  - <u>User-defined procedures</u>: are created by users to perform what they want to retrieve data or something like that

SQL Database Objects – Stored Procedures

- Usage
  - To create or alter a stored procedure (SP) in database

CREATE | ALTER {PROC | PROCEDURE} *procedure_name*

[{<@parameters> <data_type>} [= default_value] [OUT[PUT]]]

AS *{ <sql_statements> }* [;]

Example

```
CREATE PROCEDURE GetEmpInfor
        @EmpID    int
AS
BEGIN
        SELECT EmpNo, Ename, Job
        FROM   Emp
        WHERE  EmpNo = @EmpID
END
```

SQL Database Objects – Stored Procedures

- To drop a SP in database

DROP { PROC | PROCEDURE } *procedure_name;*

<u>Example:</u>

```
DROP PROCEDURE GetEmpInfor;
```

- To execute/test a SP in database

EXECUTE | EXEC *procedure_name <parameters>;*

<u>Example:</u>

```
EXECUTE GetEmpInfor 1;
```

SQL Database Objects – Stored Procedures

- When designing SPs, specify items below
  - Must have a permission to create or alter SPs
  - Just create SPs in the current DB
  - Name must be unique and should not have a prefix **sp_**
  - Maximum number of parameter in SP is 2100

SQL Database Objects – Summary

- In this section, you have learned:
  - The purpose of the basic database objects including tables, views, indexes, triggers, functions, stored procedures
  - How to implement them

SQL Elements

SQL Elements

- Tools
- T-SQL
- SELECT statement
- JOIN clause
- Conditions
- INSERT statement
- UPDATE statement
- DELETE statement
- MERGE statement

SQL Elements - Tools

- There are 3 tools access or change data in MS SQL Server 2005
  - SQL Server Management Studio: is a main tool for creating and editing T-SQL statement
  - Sqlcmd utility: is a command-line utility that executes T-SQL statements and scripts
  - Bcp utility: is used to copy a large amount of data from instance SQL Server to data files or insert a large number of rows into an existing table

SQL Elements - TSQL

- T-SQL is an extension of SQL language including elements
  - Identifiers
  - Expressions
  - Operator in expression
  - Comments
  - Reserved keyword

SQL Elements – SELECT Statement

- Purpose: SELECT statement is used to display one or more columns from one or more tables. It can meet the conditions
- Usage

SELECT *select_list*
[INTO new_table_name]
FROM *table_list*
[WHERE search_conditions]
[GROUP BY group_by_list]
[HAVING search_conditions]
[ORDER BY order_list [ASC | DESC]]

SQL Elements – JOIN Clause

- Purpose
  - Represent how the tables relate to each other for navigation purposes to obtain the correct data
  - Retrieve from two or more tables, based on the matching value in columns common to both tables
  - Used in not only *FROM* clause but also *WHERE* clause

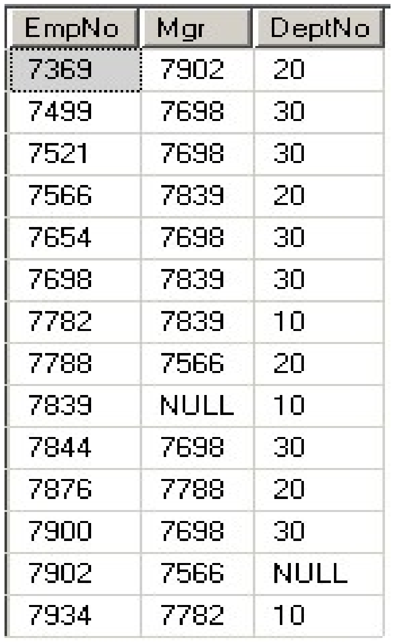
SQL Elements – JOIN Clause

- Types
  - Inner Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
  - Self join

SQL Elements – JOIN Clause

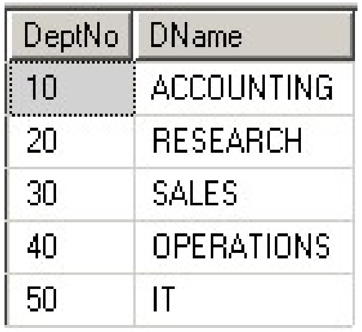
Example

Table **Emp**



EmpNo	Mgr	DeptNo
7369	7902	20
7499	7698	30
7521	7698	30
7566	7839	20
7654	7698	30
7698	7839	30
7782	7839	10
7788	7566	20
7839	NULL	10
7844	7698	30
7876	7788	20
7900	7698	30
7902	7566	NULL
7934	7782	10

Table **Dept**

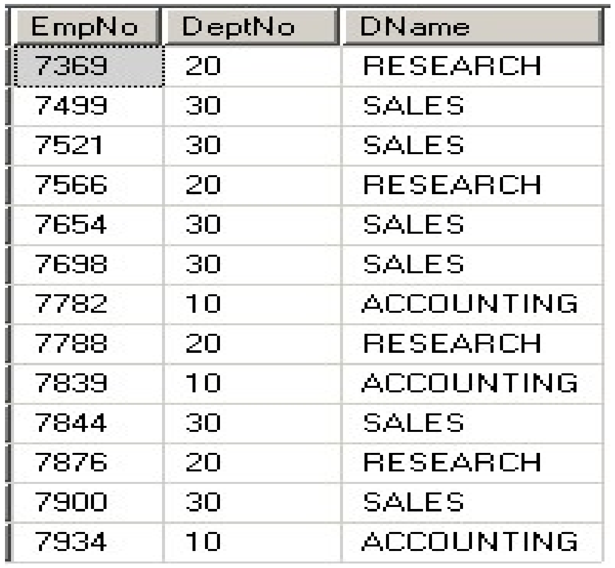


DeptNo	DName
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS
50	IT

SQL Elements – JOIN Clause

- INNER JOIN

```
SELECT e.EmpNo, e.DeptNo, d.DName
FROM Emp e INNER JOIN Dept d ON e.DeptNo = d.DeptNo
```

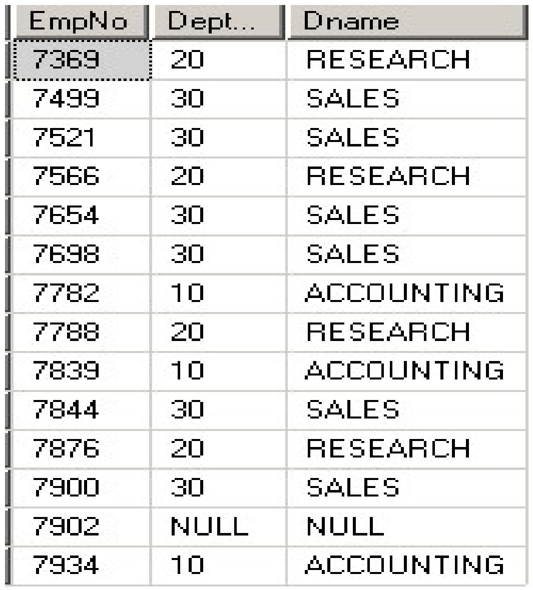


EmpNo	DeptNo	DName
7369	20	RESEARCH
7499	30	SALES
7521	30	SALES
7566	20	RESEARCH
7654	30	SALES
7698	30	SALES
7782	10	ACCOUNTING
7788	20	RESEARCH
7839	10	ACCOUNTING
7844	30	SALES
7876	20	RESEARCH
7900	30	SALES
7934	10	ACCOUNTING

SQL Elements – JOIN Clause

- LEFT OUTER JOIN

```
SELECT e.EmpNo, e.DeptNo, d.DName
FROM Emp e LEFT OUTER JOIN Dept d ON e.DeptNo = d.DeptNo
```

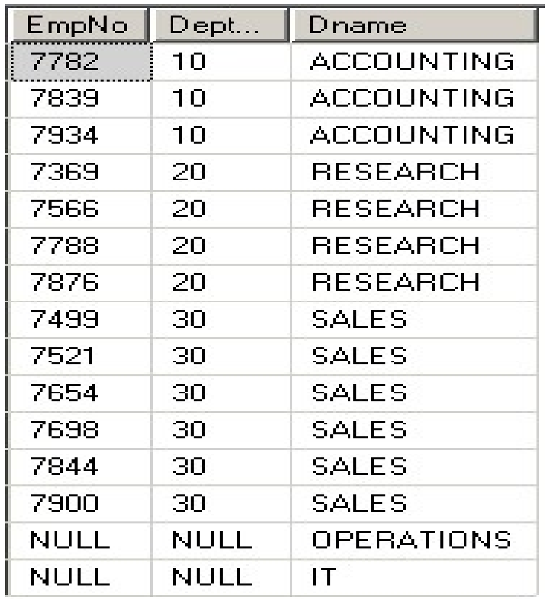


EmpNo	Dept...	Dname
7369	20	RESEARCH
7499	30	SALES
7521	30	SALES
7566	20	RESEARCH
7654	30	SALES
7698	30	SALES
7782	10	ACCOUNTING
7788	20	RESEARCH
7839	10	ACCOUNTING
7844	30	SALES
7876	20	RESEARCH
7900	30	SALES
7902	NULL	NULL
7934	10	ACCOUNTING

SQL Elements – JOIN Clause

- RIGHT OUTER JOIN

```
SELECT e.EmpNo, e.DeptNo, d.DName
FROM Emp e RIGHT OUTER JOIN Dept d ON e.DeptNo = d.DeptNo
```

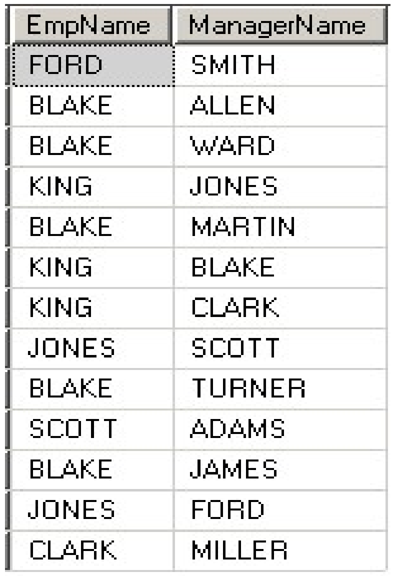


EmpNo	Dept...	Dname
7782	10	ACCOUNTING
7839	10	ACCOUNTING
7934	10	ACCOUNTING
7369	20	RESEARCH
7566	20	RESEARCH
7788	20	RESEARCH
7876	20	RESEARCH
7499	30	SALES
7521	30	SALES
7654	30	SALES
7698	30	SALES
7844	30	SALES
7900	30	SALES
NULL	NULL	OPERATIONS
NULL	NULL	IT

SQL Elements – JOIN Clause

- SELFJOIN

```
SELECT e.EmpNo, e.DeptNo, d.DName
FROM Emp e INNER JOIN Dept d ON e.DeptNo = d.DeptNo
```



EmpName	ManagerName
FORD	SMITH
BLAKE	ALLEN
BLAKE	WARD
KING	JONES
BLAKE	MARTIN
KING	BLAKE
KING	CLARK
JONES	SCOTT
BLAKE	TURNER
SCOTT	ADAMS
BLAKE	JAMES
JONES	FORD
CLARK	MILLER

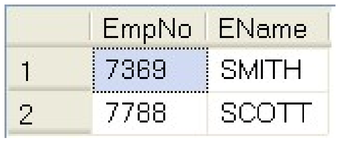
SQL Elements – Conditions

- Arithmetic operators: +, -, * , /
- Comparison conditions: >, >=, =, <=, <, <>
- Logical conditions: AND, OR, NOT
- Range conditions: BETWEEN … AND
- NULL conditions: IS [NOT] NULL
- Others: [NOT] IN, [NOT] EXISTS, [NOT] LIKE
- Set operators: combine or compare the result set of two or more SELECT statements
  - UNION
  - UNION ALL
  - EXCEPT
  - INTERSECT

SQL Elements – Conditions

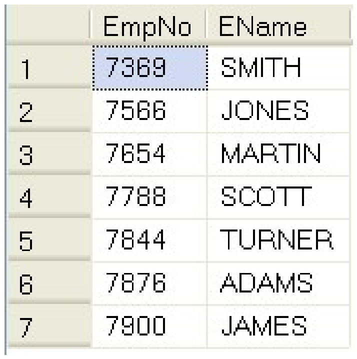
Example

```
SELECT e1.EmpNo, e1.EName
FROM    Emp e1
WHERE EName LIKE '%S%'
INTERSECT
SELECT e2.EmpNo, e2.EName
FROM    Emp e2
WHERE EName LIKE '%T%'
```



	EmpNo	EName
1	7369	SMITH
2	7788	SCOTT

```
SELECT e1.EmpNo, e1.EName
FROM    Emp e1
WHERE  EName LIKE '%S%'
UNION
SELECT e2.EmpNo, e2.EName
FROM    Emp e2
WHERE EName LIKE '%T%'
```



	EmpNo	EName
1	7369	SMITH
2	7566	JONES
3	7654	MARTIN
4	7788	SCOTT
5	7844	TURNER
6	7876	ADAMS
7	7900	JAMES

SQL Elements – INSERT Statements

- Purpose: insert a row into an existing table or rows into new table
- Allow to insert multiple rows in a single statement (MSSQL 2008)
- Usage

INSERT *[INTO] table_or_view*
[{column_list}] data_values

OR

SELECT *column_list*
INTO *new_table*
FROM *table_source*
[WHERE search_conditions]

SQL Elements – INSERT Statements

Example

```
INSERT INTO Emp (EmpNo, EName, Mgr, Job, DeptNo, HireDate, Sal)
VALUES (7935, 'JOHN', 7698, 'SALESMAN', 20, '10/12/2000', 900)

SELECT *
INTO    Dept_BK
FROM  Dept

INSERT INTO Dept (DeptName)
VALUES (‘IT’), (‘HR’), (‘Finance’)
```

SQL Elements – UPDATE Statements

- Purpose: change data values in one or more rows in a specific table or view
- Usage

UPDATE *table_or_view*
SET *column = expression*
[WHERE search_conditions]

<u>Example</u>

```
UPDATE Emp
SET     DeptNo = 30
WHERE EmpNo = 7935
```

SQL Elements – DELETE Statements

- Purpose: delete data one or more rows in a specific table or view
- Usage

DELETE *table_or_view*

[FROM table_source]

[WHERE search_conditions]

Example

```
DELETE Dept
WHERE DeptNo = 50
```

SQL Elements – MERGE Statements

- Perform INSERT/UPDATE/DELETE statements on a target table based on the results of a join with a source table
- Perform INSERT/UPDATE/DELETE statements on checking whether a row exists and then executing inserts or updates.
- Using

```
MERGE <target table/date set>
USING  <source table/data set>
       ON <join conditions between the source and target data
WHEN MATCHED THEN <SQL statement>
WHEN  TARGET] NOT MATCHED THEN <SQL statement>
WHEN SOURCE NOT MATCHED THEN <SQL statement>
```

SQL Elements – Summary

- In this section, you should have learned to
  - Retrieve data: SELECT statement
  - Manipulate data: INSERT/DELETE/UPDATE/MERGE statements

Questions & Answer

Thank You!

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
2017-10-05	1.0	Change the course by using DXC template	Thanh Do	Quang Tran