

Day1:

Kiểu dữ liệu trong SQL:

- BIT trả về true hoặc false
- Approximate_Data: khai báo biến kiểu float và gán giá trị theo biến, tiếp theo thực hiện tính toán để lấy ra giá trị của biến

Ví dụ float

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database '1FWADIEUNI1-LT' is selected. In the Results pane, a query is run:

```
-- Approximate numeric
-- float(n) n<=24 4bytes, (53) n>24 8bytes. real=float(24)

DECLARE @Float1 float, @Float2 float, @Float3 float, @Float4 float;
SET @Float1 = 54;
SET @Float2 = 3.1;
SET @Float3 = 0 + @Float1 + @Float2;
SELECT @Float3 - @Float1 - @Float2 AS "Should be 0";
```

The results show a single row with the value '1 133226762955019E-15'. A status bar at the bottom indicates 'Query executed successfully.'

- DateTime: Hàm getdate để trả về ngày tháng năm h, phút,s

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database '1FWADIEUNI1-LT' is selected. In the Results pane, a query is run:

```
-- Date and Time
SELECT getdate()
DECLARE @d date
SET @d = getdate()
SELECT @d

GO

DECLARE @t time
SET @t = getdate()
SELECT @t

GO
-- DateTime and DateTime2
DECLARE @d1 datetime = '2/25/2013 11:59:59.997'
DECLARE @d2 datetime2 = '2/25/2013 11:59:59.999'
SELECT @d1 as 'DateTime', @d2 'DateTime2'

GO
-- DateTimeOFFSET
```

The results show a single row with the value '1 2014-08-20 11:00:39.293'. A status bar at the bottom indicates 'Query executed successfully.'

- Datetimeoffset kiểu dữ liệu real time, sử dụng kèm theo hàm CAST:

```
SELECT CAST(GETDATE() AS DATETIMEOFFSET)
```
- Kiểu dữ liệu binary: thường thực hiện theo giá trị hexa và dạng số nguyên(int)
 - Hàm cast để chuyển đổi

```

DECLARE @BinaryVariable2 binary(2);

SET @BinaryVariable2 = 123456;
SET @BinaryVariable2 = @BinaryVariable2 + 1;

SELECT @BinaryVariable2;

SELECT CAST( @BinaryVariable2 AS int);
GO

```

Results

| (No column name) |
|------------------|
| 1 0xE241 |
| (No column name) |
| 1 57921 |

Query executed successfully.

- Dữ liệu rowversion: có độ rộng là 8bytes
- Kiểu dữ liệu SQL_variant: là kiểu dữ liệu xác định theo kiểu dữ liệu khác

```

INSERT INTO MyTest (myKey, myValue) VALUES (3, 0);
INSERT INTO MyTest (myKey, myValue) VALUES (4, 0);
GO

SELECT * FROM MyTest

-- SQL VARIANT
DECLARE @t SQL VARIANT
SET @t = '2012-12-12'
SELECT @t
SET @t = 1
SELECT @t

```

Results

| (No column name) |
|------------------|
| 1 2012-12-12 |
| (No column name) |
| 1 1 |

Query executed successfully.

• Basic SQL - Lecture 03: SQL Data Types & Operations - Demo Operations

- Toán tử Exists: kiểm tra sự tồn tại của một đối tượng
- Toán tử so sánh và toán tử logic(SQL Comparison operators) : lọc ra các bản ghi thỏa mãn điều kiện

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'Fsoft_Training' is selected. In the center pane, a query window titled 'SQL Operations.sql ... Training (sa (54))' contains the following T-SQL code:

```
GO
-- usage of SQL Comparison Operators
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnAge >= 30 AND trn.TrnSalary >= 2500

GO
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnAge >= 30 OR trn.TrnSalary >= 2500

GO
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnAge IS NOT NULL

GO
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnName LIKE 'Ko%'

GO
SELECT * FROM dbo.Trainer trn
```

The 'Results' tab shows the output of the last SELECT statement, displaying 7 rows of data:

| | Id | TmName | TmAge | TmAddress | TmSalary |
|---|----|---------|-------|-----------|----------|
| 1 | 1 | Ramesh | 32 | Ahmedabad | 2000 |
| 2 | 2 | Khilan | 28 | Delhi | 1500 |
| 3 | 3 | Kaushik | 26 | Kota | 1600 |
| 4 | 4 | Chatali | 35 | Mumbai | 2800 |
| 5 | 5 | Hardik | 27 | Bhopal | 3000 |

The status bar at the bottom indicates: 'Query executed successfully.' and '1FWADIEUNI1-LT (10.0 RTM) | sa (54) | Fsoft_Training | 00:00:00 | 7 rows'.

Ví dụ trên dùng toán tử logic AND, OR và LIKE

- Toán tử IS NOT NULL là lọc ra bản ghi có giá trị và ko null
- Toán tử LIKE để lọc ra phần tử có chuỗi cần tìm

The screenshot shows two examples of T-SQL queries in the SSMS interface:

```
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnName LIKE 'Ko%'
```

Below this, another query is shown:

```
SELECT * FROM dbo.Trainer trn
WHERE trn.TrnAge IN ( 25, 27 )
```

Both queries have a 'GO' command preceding them. The status bar at the bottom indicates: 'Ready' and '4:49 PM 8/21/2014'.

- Toán tử IN để lọc ra bản ghi trong cột có giá trị cần tìm
- Toán tử BETWEEN lọc ra bản ghi trong bảng có giá trị trong khoảng cần tìm

```
□ SELECT * FROM dbo.Trainer trn  
└ WHERE trn.TrnAge BETWEEN 25 AND 27
```

- Toán tử EXISTS lọc ra bản ghi có sẵn và kết hợp với điều kiện WHERE + điều kiện và in ra màn hình

```
□ IF EXISTS (SELECT TrnAge FROM dbo.Trainer WHERE TrnSalary >= 5000)  
    PRINT 'YES'  
ELSE  
    PRINT 'NO'  
GO
```

- Toán tử ALL đảm bảo rằng bản ghi đã được lọc trong phần tử dựa trên điều kiện

```
□ SELECT * FROM dbo.Trainer  
WHERE TrnAge > ALL  
(  
    SELECT TrnAge FROM dbo.Trainer  
    WHERE TrnSalary BETWEEN 1000 AND 2000  
)  
GO
```

- Toán tử ANY

Toán tử **ANY** là một toán tử logic so sánh một giá trị với một tập các giá trị cột đơn trả về bởi một **subquery**.

Sau đây là cú pháp của toán tử **ANY**:

```
1 | scalar_expression comparison_operator ANY (subquery)
```

Basic SQL - Lecture 04: DDL Statements

- DDL - Data Definition Language là ngôn ngữ cho phép chúng ta định nghĩa cấu trúc dữ liệu trong SQL Server bao gồm: tạo, thay đổi, xoá các bảng và thiết lập các ràng buộc...
- Các câu lệnh DDL gồm: CREATE , ALTER , DROP , TRUNCATEMột cơ sở SQL service gồm nhiều đối tượng như:

A SQL Server database has lot of objects like:

- Database
- Schema
- Tables
- Views
- Stored Procedures
- Functions
- Rules
- Defaults
- Triggers

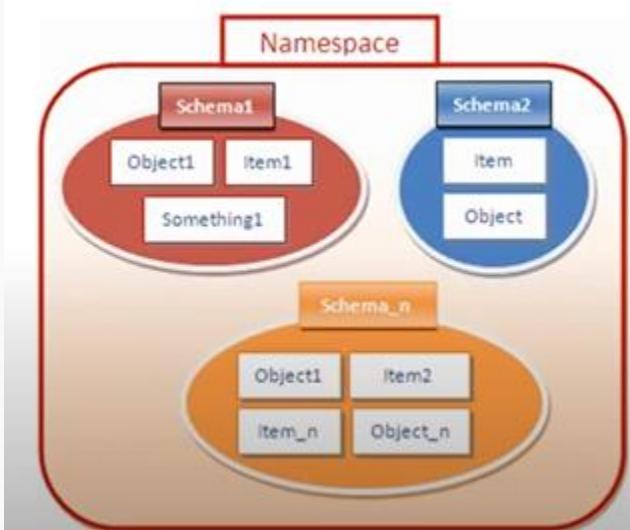
- Một SQL là tập hợp các dữ liệu liên quan đến nhau

Create

Rename

Drop

- Namespace: chứa một tập hợp bên trong nó bao gồm table stored procedure và view
- Schema object: các đối tượng có mục đích liên quan đến nhau được gọi là schema



Ví dụ mặc định của schema là dbo, schema là danh giới đặt tên và danh giới bảo mật

- ràng buộc trên bảng(Constraints)
- khi tạo ràng buộc cần quan tâm đến: kiểu dữ liệu(Data type), chỉ mục(index)
ràng buộc SQL duy trì tính nhất quán dữ liệu, giúp ngăn chặn dữ liệu không hợp lệ, duy trì tính hợp lệ của dữ liệu

□ We will **focus** on the following constraints:

- NOT NULL
- CHECK
- UNIQUE
- PRIMARY KEY
- DEFAULT
- FOREIGN KEY

Default: ràng buộc mặc định nhằm chỉ định giá trị ban đầu khởi tạo cho cột khi insert dữ liệu giá trị mà bạn không chỉ định giá trị cho cột thì giá trị default sẽ được dùng

- Struncate statement: dùng để xoá hoàn toàn dữ liệu trong bảng, dùng truncate sẽ tăng performance nhanh hơn delete(nhưng điểm truncate không thể dùng where hay from như delete)

Basic SQL - Lecture 05: Table Indexes, Sequences, View

1. Index trong SQL: tương tự chỉ mục để tìm kiếm dữ liệu trên các cột trong bảng
 - Đầu tiên SQL sẽ tìm dữ liệu trong index, sau đó index sẽ xác định dòng dữ liệu cần tìm
 - Index có thể tạo trong hàng cột của bảng ngoại trừ các cột có kiểu dữ liệu lớn như IMG, varchar.v.v



1.2 Table Indexes

```
CREATE TABLE dbo.PhoneBook
```

```
(
```

```
    LastName      varchar(50) NOT NULL  
    , FirstName    varchar(50) NOT NULL  
    , PhoneNumber  varchar(50) NOT NULL  
);
```

```
SELECT PhoneNumber  
FROM dbo.PhoneBook  
WHERE LastName = 'Logan' AND FirstName = 'Todd';
```



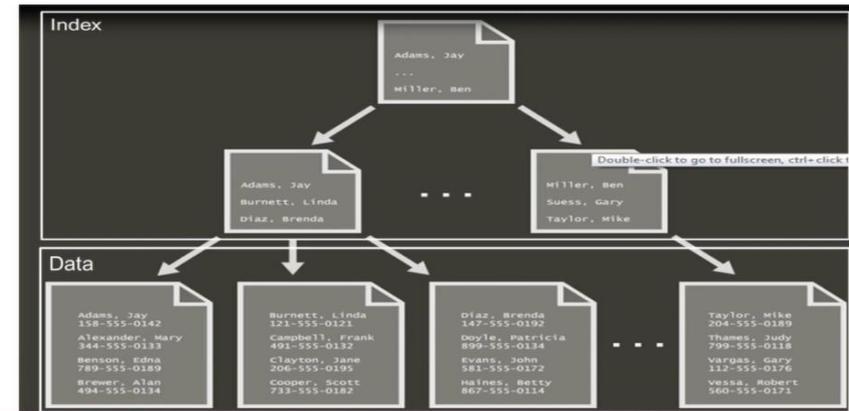
Non-clustered index



1.3 Clustered Index

Syntax

```
CREATE CLUSTERED INDEX IX_PhoneBook_CI  
ON dbo.PhoneBook (LastName, FirstName)
```



@Copyright 2014 FSOFT. All rights reserved.

7/19

Creating an Index:

1.5 Creating an Index

Syntax

```
CREATE INDEX index_name  
ON table_name (column1_name, column2_name, ...)
```

Deleting an index

Syntax

```
DROP INDEX table_name.index_name
```

@Copyright 2014 FSOFT. All rights reserved.

10/19

2. Sequences

- giống với Identity là tạo dãy tăng tự động cho các cột
- khác: giá trị đc khởi tạo trong bộ nhớ thay vì đc khởi tạo từ ổ cứng như identity do đó hiệu xuất cao hơn
- Creating sequences

2.2 Creating Sequence

Syntax

```
CREATE SEQUENCE [schema_name . ] sequence_name  
[AS[built_in_integer_type|user-defined_integer_type]]  
[ START WITH <constant> ]  
[ INCREMENT BY <constant> ]  
[ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]  
[ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]  
[ CYCLE | { NO CYCLE } ]  
[ { CACHE [ <constant> ] } | { NO CACHE } ] [ ; ]
```

@Copyright 2014 FSOFT. All rights reserved.

12/19

Ví dụ:

2.3 Example

- Creating a sequence that increases by 1

```
CREATE SEQUENCE Test.CountBy1  
    START WITH  
        INCREMENT BY 1 ;  
    GO
```

- Creating a sequence that decreases by 1

```
CREATE SEQUENCE Test.CountByNeg1  
    START WITH 0  
        INCREMENT BY -1 ;  
    GO
```

- Creating a sequence that increases by 5

```
CREATE SEQUENCE Test.CountBy5  
    START WITH 5  
        INCREMENT BY 5 ;  
    GO
```

- Creating a sequence that starts with a designated number

```
CREATE SEQUENCE Test.ID_Seq  
    START WITH 24329  
        INCREMENT BY 1 ;  
    GO
```

13/19

3. Bảng ảo trong SQL(View)

- View là bảng ảo mà dữ liệu trong đó được mô tả bằng câu truy vấn và view không chứa dữ liệu.
- Lý do cho phép sử dụng view là: cho phép giới hạn người truy cập dữ liệu, view giúp giảm độ phức tạp cho người dùng cuối

3.2 Creating a View

Syntax

```
CREATE VIEW View_Name [list of column names]  
AS  
SELECT...
```

Example:

```
CREATE VIEW view_EmployeeByDpt  
AS  
SELECT ID, NAME, AGE, DEPT_NAME  
FROM   EMP,DEPARTMENT  
WHERE  EMP.DEP_ID = DEPARTMENT.DEPT_ID
```

Table: EMP

| ID | NAME | AGE | DEP_ID |
|----|------|-----|--------|
| 1 | John | 25 | 3 |
| 2 | Mike | 30 | 2 |
| 3 | Parm | 25 | 1 |
| 4 | Todd | 23 | 4 |
| 5 | Sara | 35 | 1 |
| 6 | Ben | 40 | 3 |

Table: DEPARTMENT

| DEPT_ID | DEPT_NAME |
|---------|-----------|
| 1 | IT |
| 2 | Payroll |
| 3 | HR |
| 4 | Admin |

```
SELECT * FROM  
view_EmployeeByDpt
```

view_EmployeeByDpt

| ID | NAME | AGE | DEPT_NAME |
|----|------|-----|-----------|
| 1 | John | 25 | HR |
| 2 | Mike | 30 | Payroll |
| 3 | Parm | 25 | IT |
| 4 | Todd | 23 | Admin |
| 5 | Sara | 35 | IT |
| 6 | Ben | 40 | HR |

15/19

Basic SQL - Lecture 06: DML Statements

DML - Data Manipulation Language là ngôn ngữ thao tác dữ liệu.

Các câu lệnh DML gồm: SELECT, INSERT, UPDATE, DELETE...

Insert statement: được sử dụng để thêm 1 hoặc nhiều bản ghi vào các cột trong bảng

Sql Insert into Statement

```
INSERT INTO agents VALUES ("A001","Jodi","London",.12,"075-1248798");
```

| agent_code | agent_name | working_area | commission | phone_no |
|------------|------------|--------------|------------|-------------|
| A001 | Jodi | London | .12 | 075-1248798 |

Table : agents

The **INSERT INTO** statement is used to adds one or more rows to a table or a view

- Ví dụ về insert:

Syntax

(1) Inserting data to all columns

```
INSERT INTO table_name  
VALUES (value1,value2,value3,...);
```

Ex: USE Fsoft_Training

```
INSERT INTO dbo.Persons  
VALUES ( 1,'Tom', 'B. Erichsen','Skagen 21','Stavanger')
```

Syntax

(2) Inserting data to selected columns

```
INSERT INTO table_name(column1,column2,column3,...)  
VALUES (value1,value2,value3,...);
```

Ex: USE Fsoft_Training

```
INSERT INTO dbo.Customer (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

Update statement: được sử dụng để thay đổi dữ liệu đã tồn tại của một bảng hoặc bảng ảo

Syntax

```
UPDATE table_name  
SET column1=value1,column2=value2,...  
WHERE some_column=some_value;
```



NOTICE : The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Ex: USE Fsoft_Training

```
UPDATE dbo.Customer  
SET PostalCode = '4006'  
WHERE Country = 'Norway'  
SELECT @@ROWCOUNT AS ROW_COUNT
```

| Results | | Messages |
|---------|---|-----------|
| | | ROW_COUNT |
| 1 | 2 | |

Delete Statement: được sử dụng để xoá dữ liệu ở một bảng hoặc bảng ảo, khi delete bạn nên dùng truncate

- Remove one or more rows from a table or view

| CustomerId | CustomerName | ContactName |
|------------|---------------------|--------------------|
| 1 | Alfreds Futterkiste | Maria Anders |
| 2 | Around the Horn | Thomas Hardy |
| 3 | Berglunds snabbköp | Christina Berglund |
| 4 | Antonio Moreno | Antonio Moreno |
| 5 | Ana Trujillo | Ana Trujillo |



Tips:

- To delete all the rows in a table, use TRUNCATE TABLE. TRUNCATE TABLE is faster than DELETE and uses fewer system and transaction log resources
- TRUNCATE TABLE has restrictions, for example, the table cannot participate in replication



Ví dụ về delete:

Syntax

```
DELETE FROM table_name  
WHERE some_column=some_value;
```



Notice:

- The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!
- The DELETE FROM command cannot delete any rows of data that would violate FOREIGN KEY or other constraints

Ex: USE Fsoft_Training
DELETE dbo.Customer
WHERE Country = 'Germany'
SELECT @@ROWCOUNT AS ROW_COUNT

| Results | | Messages | |
|-----------|---|----------|--|
| ROW_COUNT | | | |
| 1 | 1 | | |

Select statement: Giúp lấy về các dòng SQL và cho phép một hay nhiều dòng hoặc cột trong bảng

Ví dụ Select:

4. SELECT statement (2/4)

Syntax

```
SELECT [ALL/DISTINCT/TOP] <Column name1>, <Column name2>,...  
      FROM <Table name>  
      [WHERE <Search condition>]  
      [GROUP BY grouping columns]  
      [HAVING search condition]  
      [ORDER BY sort specification]
```

Ex 1: USE Adventure Works

GO

SELECT ProductID, Name

FROM Production.Product

ORDER BY Name ASC;

(504 rows)

Ex 2: SELECT DISTINCT E.Title

FROM HumanResources.Employee

ORDER BY E.Title;

(67 rows)

| Results | | Messages | |
|-----------|-----|----------|--|
| ProductID | | Name | |
| 1 | 1 | | |
| 2 | 879 | | |
| 3 | 712 | | |
| 4 | 3 | | |
| 5 | 2 | | |
| 6 | 877 | | |
| 7 | 316 | | |
| 8 | 843 | | |
| 9 | 952 | | |
| 10 | 324 | | |
| 11 | 322 | | |
| 12 | 320 | | |
| 13 | 321 | | |
| 14 | 866 | | |
| 15 | 865 | | |
| 16 | 864 | | |
| 17 | 505 | | |

Query ex

Query executed successfully.

@Copyright 2014 FSOFT. All rights reserved.

11/16

- Select into: giúp lấy dữ liệu từ 1 bảng và insert chúng vào 1 bảng khác

Ví dụ Select into:

4. SELECT statement (3/4)

- The **SELECT INTO** statement selects data from one table and inserts it into a different table

Syntax

```
SELECT *
    INTO new_table_name
    FROM old_table_name
```



Tip:
The **SELECT INTO** statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT *
    INTO newtable
    FROM table1
    WHERE 1=0;
```

@Copyright 2014 FSOFT. All rights reserved.

12/16

- SQL Alias: SQL bí danh được sử dụng để cung cấp cho 1 bảng dữ liệu hoặc 1 cột trong 1 bảng tên tạm thời, về cơ bản bí danh(Alias) được tạo ra để tên bảng dễ đọc hơn
Ví dụ về Alias:

4. SELECT statement (4/4)

Syntax

SQL Alias Syntax

- For table

```
SELECT column_name(s)
    FROM table_name AS alias_name
```

- For Column(s)

```
SELECT column_name AS alias_name
    FROM table_name
```

Ex: USE AdventureWorks

```
GO
SELECT c.CustomerID, s.Name
FROM Sales.Customer AS c
JOIN Sales.Store AS s
ON c.CustomerID = s.SalesPersonID
```

@Copyright 2014 FSOFT. All rights reserved.

13/16

Từ khoá DISTINCT: loại bỏ các bản ghi trùng lặp

Basic SQL - Lecture 07: SELECT Options

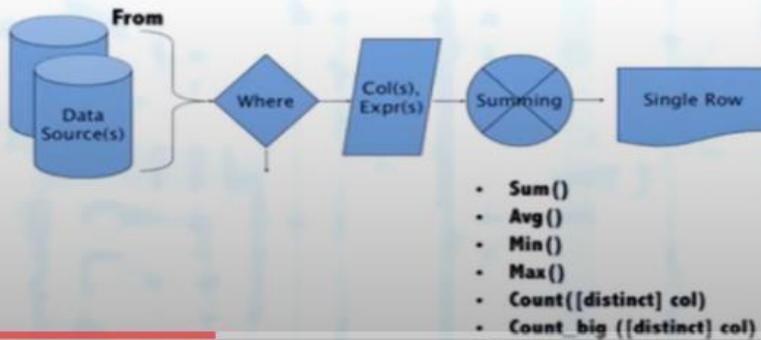
- SQL function: Hàm là một đối tượng trong CSDL bao gồm một tập hợp nhiều câu lệnh SQL đc nhóm lại với nhau thành 1 nhóm

- Điểm khác biệt giữa hàm và thủ tục là:
 - Hàm trả về một giá trị thông qua tên hàm, điều này cho phép ta sử dụng hàm như một biểu thức. Ví dụ câu lệnh truy vấn, update
 - 1.2 Aggregate function: nhóm hàm tập hợp có tác dụng với nhiều giá trị nhưng chỉ trả về 1 giá trị
 - Hàm tập hợp có thể sử dụng trong câu lệnh select và trong mệnh đề having

following:

- ✓ The select list of a SELECT statement
- ✓ A HAVING clause

Aggregation



- Một số hàm trong nhóm hàm tập hợp

1.3 Aggregate Functions

Each function eliminates NULL values and operates on Non-NULL values

| Function | Description |
|----------|---|
| AVG () | Return the average value in a column |
| COUNT() | Return the total number of values in a given column |
| COUNT(*) | Return the number of rows |
| MAX () | Return the largest value in a column |
| MIN () | Return the smallest value in a column |

1.4 Scalar Functions

| Function | Description |
|----------|---|
| LEN() | Return the length of a text field |
| ROUND() | Round a numeric field to the number of decimals specified |
| NOW() | Return the current system date and time |
| FORMAT() | Format how a field is to be displayed |

@Copyright 2014 FSOFT. All rights reserved.

6/16

2. SQL Clauses

2.1 GROUP BY

Trong câu lệnh SELECT có chức năng tổng hợp dữ liệu

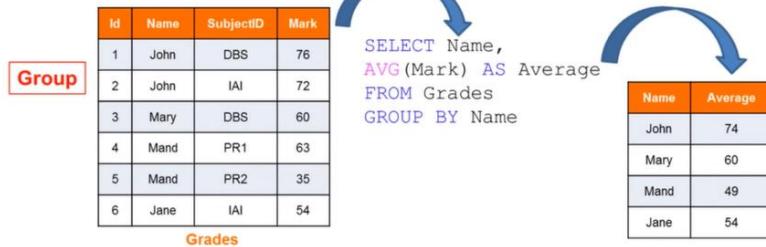
2.1 GROUP BY

Sometimes we want to apply aggregate functions to groups of rows

Syntax

```
SELECT column_name, aggregate_function(column_name)
  FROM table_name
 WHERE column_name operator value
 GROUP BY column_name;
```

Example: Find the average mark of each student



@Copyright 2014 FSOFT. All rights reserved.

7/16

2.2 Mệnh đề HAVING giúp kiểm tra điều kiện với các hàm tập hợp

2.2 HAVING

- **HAVING** is like a **WHERE** clause, except that it applies to the results of a **GROUP BY** query
- It can be used to select groups which satisfy a given condition

Example

:

| Id | Name | SubjectID | Mark |
|----|------|-----------|------|
| 1 | John | DBS | 76 |
| 2 | John | IAI | 72 |
| 3 | Mary | DBS | 60 |
| 4 | Mand | PR1 | 63 |
| 5 | Mand | PR2 | 35 |
| 6 | Jane | IAI | 54 |

```
SELECT Name, AVG(Mark) AS Average
FROM Grades
GROUP BY Name
HAVING AVG(Mark) >= 50
```

| Name | Average |
|------|---------|
| John | 74 |
| Mary | 60 |
| Jane | 54 |

8/16

- Mệnh đề WHERE tham chiếu tới các bản ghi trong bảng và không sử dụng được với các hàm tập hợp
- Mệnh đề HAVING tham chiếu tới các bản ghi trong bảng và có thể sử dụng với các hàm tập hợp(HAVING luôn đi kèm GROUP BY)

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
WHERE AVG(Mark) >= 50
GROUP BY Name
```

```
SELECT Name,
AVG(Mark) AS Average
FROM Grades
GROUP BY Name
HAVING AVG(Mark) >= 50
```

2.3 ORDER BY

Mệnh đề sắp xếp(ORDER BY) theo thứ tự tăng dần hoặc giảm dần các bản ghi

2.4 ORDER BY

The SQL **ORDER BY clause** is used to sort (ascending or descending) the records in the result set for a SELECT statement

Syntax

```
SELECT column_name, column_name  
FROM table_name  
[WHERE conditions]  
ORDER BY column_name, column_name [ASC|DESC]
```

Example

Group

| ID | Name | SubjectID | Mark |
|----|------|-----------|------|
| 1 | John | DBS | 76 |
| 2 | John | IAI | 72 |
| 3 | Mary | DBS | 60 |
| 4 | Mand | PR1 | 63 |
| 5 | Mand | PR2 | 35 |
| 6 | Jane | IAI | 54 |

Grades

```
SELECT Name,  
AVG(Mark) AS Average  
FROM Grades  
GROUP BY Name  
ORDER BY Average DESC
```

| Name | Average |
|------|---------|
| John | 74 |
| Mary | 60 |
| Jane | 54 |
| Mand | 49 |

10/16

3.1 UNION Operator:

Được dùng để kết hợp các dòng nhiều bảng khác nhau, Union sẽ kết hợp kết quả của 2 hay nhiều câu lệnh select lại thành 1 kết quả

- Lưu ý: mỗi câu lệnh select phải có cấu trúc

Và để sử dụng đc toán tử Union bạn phải chú ý đến 2 điều kiện: dữ liệu các cột phải tương ứng (kiểu dữ liệu trong câu select bảng này phải giống kiểu kiểu dữ liệu trong câu select bảng kia)

- Số lượng cột trong mỗi câu truy vấn phải bằng nhau tuy nhiên tên cột có thể khác nhau
- Về cú pháp toán tử UNION sẽ lọc các bản ghi có giá trị trùng nhau, nếu sử dụng từ khoá ALL

3.1 UNION Operator

The SQL UNION operator combines the result of two or more SELECT statements

Syntax

```
SELECT column_name(s) FROM table1  
UNION  
SELECT column_name(s) FROM table2;
```

 Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION

```
SELECT Column1, Column2 FROM Table1  
UNION  
SELECT Column1, Column2 FROM  
Table2;
```

| Table 1 | | UNION | | Table 2 | |
|----------|----------|-------|--|----------|----------|
| Column 1 | Column 2 | | | Column 1 | Column 2 |
| a | a | | | b | a |
| a | b | | | a | b |
| a | c | | | b | c |

The UNION operator selects only distinct values by default.

Result

| Column 1 | Column 2 |
|----------|----------|
| a | a |
| a | b |
| b | a |
| b | c |

Duplicate rows are displayed only once.

```
SELECT Column1, Column2 FROM Table1  
UNION ALL  
SELECT Column1, Column2 FROM  
Table2;
```

| Table 1 | | UNION ALL | | Table 2 | |
|----------|----------|-----------|--|----------|----------|
| Column 1 | Column 2 | | | Column 1 | Column 2 |
| a | a | | | b | a |
| a | b | | | a | b |
| a | c | | | b | c |

Duplicate rows are repeated in the result set.

@Copyright 2014 FSOFT. All rights reserved.

11/16

3.2 Select Into Statement

- With SQL, you can copy information from one table into another
- The SELECT INTO statement selects data from one table and inserts it into a **new table**

Syntax

(1) Copy all columns into the new table:

```
SELECT *
  INTO newtable [IN externaldb]
  FROM table1;
```

(2) Copy only the columns we want into the new table:

```
SELECT column_name(s)
  INTO newtable [IN externaldb]
  FROM table1;
```

@Copyright 2014 FSOFT. All rights reserved.

12/16

3.3 INSERT INTO SELECT Statement

- The INSERT INTO SELECT statement selects data from one table and inserts it into an **existing table**
- Any existing rows in the target table are unaffected

Syntax

(1) Copy all columns from one table to another, existing table:

```
INSERT INTO table2
SELECT * FROM table1;
```

(2) Copy only the columns we want to into another, existing table:

```
INSERT INTO
table2(column_name(s))
SELECT column_name(s)
FROM table1;
```

@Copyright 2014 FSOFT. All rights reserved.

13/16

ROUND để làm tròn số thập phân

Basic SQL - Lecture 08: Built-in Functions

Các built-in function được sử dụng trong các biểu thức SELECT để tính toán các giá trị và thao tác dữ liệu.

Built-in function gồm các loại chính sau:

Các hàm chuyển đổi (Conversion Functions)

Các hàm ngày giờ (Date and Time Functions)

Các hàm xử lý chuỗi (String Functions)

1.1. CAST function: chuyển đổi một biểu thức từ kiểu dữ liệu này sang kiểu dữ liệu khác có thể sử dụng hàm CAST

- Hàm CAST dùng để chuyển đổi dữ liệu kiểu variable
- Hàm CAST cung cấp kiểu dữ liệu đến tham số động hoặc một giá trị null
- 1.2 CONVERT Function
- Hàm Convert dùng để chuyển đổi dữ liệu

Syntax

For CONVERT:

CONVERT (data_type [(length)] , expression [, style])

- Style (0 hoặc 100): mon dd yyyy hh:miAM (or PM)

1.2 CONVERT Function (2/3)

| Without century (yy) | With century (yyyy) | Standard | Input/Output |
|----------------------|---------------------|------------------------|------------------------------------|
| - | 0 or 100 | Default | mon dd yyyy hh:miAM (or PM) |
| 1 | 101 | U.S. | mm/dd/yyyy |
| 2 | 102 | ANSI | yy.mm.dd |
| 3 | 103 | British/French | dd/mm/yyyy |
| 4 | 104 | German | dd.mm.yy |
| 5 | 105 | Italian | dd-mm-yy |
| 6 | 106 | - | dd mon yy |
| 7 | 107 | - | Mon dd, yy |
| 8 | 108 | - | hh:mi:ss |
| - | 9 or 109 | Default + milliseconds | mon dd yyyy hh:mi:ss:mmmAM (or PM) |
| 10 | 110 | USA | mm-dd-yy |
| 11 | 111 | JAPAN | yy/mm/dd |

©Copyright 2014 FSOFT. All rights reserved.

5/17

2.1 GETDATE()&DATEPART() Function

- GETDATE(): hàm trả về ngày giờ hiện tại của hệ thống
- DATEPART(): được sử dụng lấy về ngày giờ có thể là ngày tháng năm, giờ phút giây(giá trị trả về phụ thuộc vào đối số đầu tiên của hàm này)

Syntax

GETDATE()

DATEPART(datepart, date)

Example

SELECT GETDATE()

SELECT DATEPART(YYYY, GETDATE())

Result

| Results | |
|-------------|-------------------------|
| CurrentDate | |
| 1 | 2014-08-28 09:47:08.030 |
| Date_Part | |
| 1 | 2014 |

2.1 GETDATE() & DATEPART () Function (2/2)

| datepart | Abbreviation |
|-------------|--------------|
| year | yy, YYYY |
| quarter | qq, q |
| month | mm, m |
| dayofyear | dy, y |
| day | dd, d |
| week | wk, ww |
| weekday | dw, w |
| hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |
| microsecond | mcs |
| nanosecond | ns |

2.2 DAY, MONTH, YEAR Function

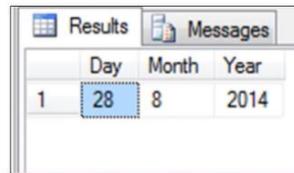
- Returns an integer representing the day/month/year (day of the month) c specified date

Syntax

```
DAY(date)  
MONTH(date)  
YEAR(date);
```

Example `SELECT DAY(GETDATE()) AS [Day],
MONTH(GETDATE()) AS [Month],
YEAR(GETDATE()) AS [Year]`

Result



| | Day | Month | Year |
|---|-----|-------|------|
| 1 | 28 | 8 | 2014 |

@Copyright 2014 FSOFT. All rights reserved.

Lấy về ngày tháng, năm của tháng ngày hiện tại

2.4 DATEADD Function

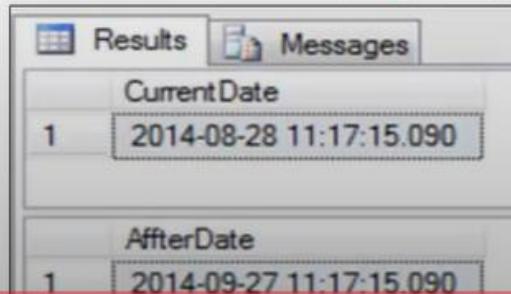
- Giúp cộng thêm hoặc trừ đi giá trị ngày tháng phụ thuộc vào đối số thứ 2(number)

Syntax

DATEADD(datepart,number,date)

Example : `DECLARE @dt datetime
SET @dt = GETDATE()
SELECT @dt AS CurrentDate
SELECT DATEADD(day, 30, @dt) AS AfterDate`

Result



| | CurrentDate |
|---|-------------------------|
| 1 | 2014-08-28 11:17:15.090 |

| | AfterDate |
|---|-------------------------|
| 1 | 2014-09-27 11:17:15.090 |

2.4 DATEDIFF Function

- The DATEDIFF() function returns the time between two dates

Syntax

```
DATEDIFF (datepart,startdate,enddate)
```

Example

```
DECLARE @date1 DATETIME  
DECLARE @date2 DATETIME  
SET @date1= '2012-04-07 20:12:22.013'  
SET @date2= '2014-02-27 22:14:10.013'  
SELECT DATEDIFF(month, @date1, @date2) AS 'Month'
```

Result

| Month | |
|-------|----|
| 1 | 22 |

@Copyright 2014 FSOFT, All rights reserved.

11/17

- 3. String FUNCTION
- 3.1 RTRIM, LTRIM FUNCTION
- - giúp cắt bỏ kí tự trắng trong một chuỗi kí tự
- - LTRIM giúp cắt bỏ kí tự trắng bên trái chuỗi
- - RTRIM giúp cắt bỏ kí tự trắng bên phải chuỗi

Syntax

```
LTRIM (str)  
RTRIM (str)
```

Example

```
SELECT LTRIM(' Sample ');
```

```
SELECT RTRIM(' Sample ');
```

Result

| Results | |
|----------|--------|
| LtrimStr | Sample |
| RtrimStr | Sample |

SUBSTRING Function: chả về chuỗi con trong chuỗi ban đầu

Syntax

SUBSTRING(str, position, length)

Example

```
SELECT SUBSTRING('Bill Gates', 0 ,5) As Result
```

Result

| Results | |
|---------|--------|
| | Result |
| 1 | |
| | Bill |

Advanced SQL - Lecture 01: JOINs in SQL Server

JOIN: cho phép kết nối dữ liệu từ nhiều bảng lại với nhau

Join được dùng để lấy dữ liệu từ hai hay nhiều bảng và trả về dữ liệu trong cùng một tập kết quả.

Join có các loại sau:

Inner Join

Outer Join

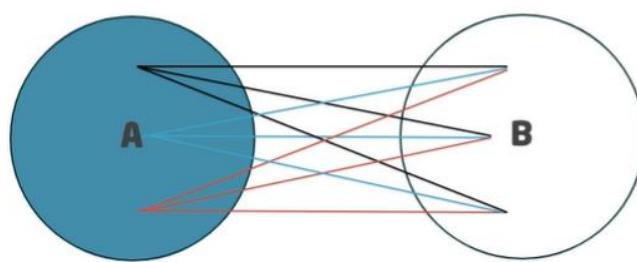
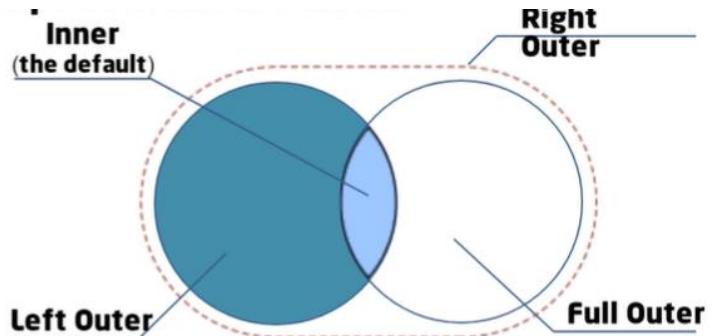
Cross Join

Self Join

Excluding Join

◆ Types of Join in SQL:

- ◊ **Inner Join**
- ◊ **Outer Join**
- ◊ **Cross Join**
- ◊ **Self Join**



| Column Name | Data Type | Nullable | Default | Primary Key |
|-------------|--------------|----------|---------|-------------|
| EMP_ID | VARCHAR (5) | No | - | 1 |
| EMP_NAME | VARCHAR (20) | Yes | - | - |
| DT_OF_JOIN | DATE | Yes | - | - |
| EMP_SUPV | VARCHAR (5) | Yes | - | - |

1 - 4

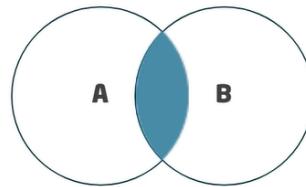
| Constraint | Type | Table |
|-------------|------|----------|
| SYS_C004074 | C | EMPLOYEE |
| EMP_ID | P | EMPLOYEE |
| EMP_SUPV | R | EMPLOYEE |

Foreign key
Referencing EMP_ID of this table

- Khi bạn cần truy vấn các cột dữ liệu từ nhiều bảng khác nhau để trả về cùng 1 kết quả thì bạn dùng join

2.1 INNER JOIN: lấy phần trung và trả về tất cả bản ghi trong cả 2 bảng

2. INNER JOIN (1/2)



◆ **The INNER JOIN selects all rows from both tables as long as there is a match between the columns in both tables**

◊ **Eliminate the rows that do not match with a row from the other table**

◆ Syntax

```
SELECT col_names
FROM Table_A A
INNER JOIN Table_B B
ON A.Col1 = B.Col1
```

3. Outer join: lấy về các bản ghi có mặt trong cả 2 bảng và các bản ghi chỉ xuất hiện ở 1 trong 2 bảng được chỉ ra trong mệnh đề from miễn là các dòng đó thoả mãn điều kiện WHERE hoặc HAVING
- Có 3 loại outer join:

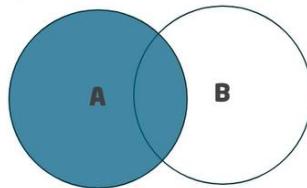
◇ **LEFT OUTER JOIN (or LEFT JOIN)**

◇ **RIGHT OUTER JOIN (or RIGHT JOIN)**

◇ **FULL OUTER JOIN (or FULL JOIN)**

- LEFT JOIN: trả về tất cả bản ghi ở bên trái

LEFT OUTER JOIN (1/2)



- ◆ Return all of the records in the left table (table A) regardless if any of those records has a match in the right table (table B)
 - ◇ In the results where there is no matching condition, the row contains NULL values for the right table's columns
- ◆ Syntax

```
SELECT col_names  
      FROM Table_A A  
      LEFT JOIN Table_B B  
        ON A.Col1 = B.Col1
```

LEFT OUTER JOIN (2/2)

◆ Example:

Customer

| CustID | CustName | BirthDate | Country |
|--------|-----------------|-----------|---------|
| 1 | Davolio Nancy | 12/8/1968 | Germany |
| 2 | Fuller Andrew | 2/19/1952 | Mexico |
| 3 | Leverling Janet | 8/30/1963 | Mexico |

Order

| OrderID | CustID | OrderDate | ShipperID |
|---------|--------|------------|-----------|
| 10308 | 2 | 2013-09-18 | 3 |
| 10309 | 3 | 2013-09-19 | 1 |
| 10310 | 77 | 2013-09-20 | 2 |

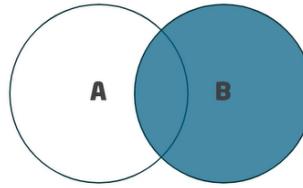
```
SELECT c.CustName, o.OrderID
FROM Customer c
LEFT JOIN [Order] o
ON c.CustID = o.CustID
ORDER BY c.CustName;
```

◆ Result:

| Results | |
|-------------------|---------|
| CustName | OrderID |
| 1 Davolio Nancy | NULL |
| 2 Fuller Andrew | 10308 |
| 3 Leverling Janet | 10309 |

- Right JOIN: trả về tất cả các bản ghi ở bản bên phải và ko quan tâm nó có đối sánh với bản bên trái hay không

RIGHT OUTER JOIN (1/2)



- Return all of the records in the right table (table B) regardless if any of those records have a match in the left table (table A)
 - In the results where there is no matching condition, the row contains NULL values for the left table's columns
- Syntax

```
SELECT col_names
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Col1 = B.Col1
```



RIGHT OUTER JOIN (2/2)

◆ Example:

Customer

| CustID | CustName | BirthDate | Country |
|--------|-----------------|-----------|---------|
| 1 | Davolio Nancy | 12/8/1968 | Germany |
| 2 | Fuller Andrew | 2/19/1952 | Mexico |
| 3 | Leverling Janet | 8/30/1963 | Mexico |

Order

| OrderID | CustID | OrderDate | ShipperID |
|---------|--------|------------|-----------|
| 10308 | 2 | 2013-09-18 | 3 |
| 10309 | 3 | 2013-09-19 | 1 |
| 10310 | 77 | 2013-09-20 | 2 |

```
SELECT c.CustName, o.OrderID  
FROM Customer c  
    RIGHT JOIN [Order] o  
        ON c.CustID = o.CustID  
ORDER BY c.CustName;
```

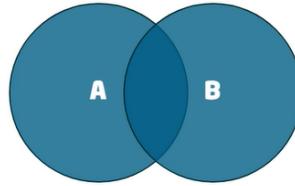
◆ Result:

| Results | |
|-------------------|---------|
| CustName | OrderID |
| 1 NULL | 10310 |
| 2 Fuller Andrew | 10308 |
| 3 Leverling Janet | 10309 |

- FULL JOIN: Trả về tất cả bản ghi ở cả 2 bản và là sự kết hợp của left join và right join



FULL OUTER JOIN (1/2)



◆ Return all of the records from both tables, joining records from the left table (table A) that match records from the right table (table B)

◆ Syntax

```
SELECT col_names  
FROM Table_A A  
    FULL JOIN Table_B B  
        ON A.Col1 = B.Col1
```



FULL OUTER JOIN (2/2)

◆ Example:

Customer

| CustID | CustName | BirthDate | Country |
|--------|-----------------|-----------|---------|
| 1 | Davolio Nancy | 12/8/1968 | Germany |
| 2 | Fuller Andrew | 2/19/1952 | Mexico |
| 3 | Leverling Janet | 8/30/1963 | Mexico |

Order

| OrderID | CustID | OrderDate | ShipperID |
|---------|--------|------------|-----------|
| 10308 | 2 | 2013-09-18 | 3 |
| 10309 | 3 | 2013-09-19 | 1 |
| 10310 | 77 | 2013-09-20 | 2 |

```
SELECT c.CustName, o.OrderID  
FROM Customer c  
    FULL JOIN [Order] o  
        ON c.CustID = o.CustID  
ORDER BY c.CustName;
```

◆ Result:

| CustName | OrderID |
|-----------------|---------|
| NULL | 10310 |
| Davolio Nancy | NULL |
| Fuller Andrew | 10308 |
| Leverling Janet | 10309 |

Advanced SQL - Lecture 02: Subqueries

1. What is a SUBQUERY?

Truy vấn con(Subqueries) là một truy vấn được lồng vào bên trong 1 truy vấn lớn hơn

- 1 subqueries có thể lồng bên trong câu lệnh select,insert,update,delete hay bên trong 1 truy vấn con khác
- Bạn có thể sử dụng các toán tử so sánh như >,< or =. Toán tử so sánh cũng có thể là 1 toán tử nhiều dòng IN,ANY hoặc ALL
- 1 truy vấn con được đặt như 1 phần của truy vấn ngoài còn gọi là outer query

1.WHAT IS A SUBQUERY?

◆ Syntax (example: subquery within the Where) :

```
SELECT      select_list  
FROM        table  
WHERE       expr operator  
           (SELECT      select_list  
            FROM        table);
```

◆ Ex:

| Table <i>Store_Information</i> | | | Table <i>Geography</i> | | Sale_Sum |
|--------------------------------|-------|-------------|------------------------|-------------|----------|
| Store_Name | Sales | Txn_Date | Region_Name | Store_Name | 2050 |
| Los Angeles | 1500 | Jan-05-1999 | East | Boston | |
| San Diego | 250 | Jan-07-1999 | East | New York | |
| Los Angeles | 300 | Jan-08-1999 | West | Los Angeles | |
| Boston | 700 | Jan-08-1999 | West | San Diego | |

```
SELECT SUM (Sales) AS Sale_Sum FROM Store_Information  
WHERE Store_Name IN  
(SELECT Store_Name FROM Geography WHERE Region_Name = 'West');
```

© Copyright 2015 FSOFT. All rights reserved

4/26

- Subquery hoạt động như thế nào?
 - Inner query độc lập với outer query
 - Inner query thực thi trước và lưu lại kết quả sau
 - Outer query được chạy sau đó và sử dụng kết quả đã lưu trước đó của inner query
- 2. Các loại truy vấn con (subquery type)
 - Single row subquery: truy vấn con 1 dòng đơn sẽ trả về kết quả tối đa 1 bản ghi cho truy vấn bên ngoài. Có thể đặt truy vấn con loại này trong mệnh đề WHERE, HAVING hoặc FROM của truy vấn ngoài

2.1 SINGLE ROW SUBQUERY

◆ A single row subquery returns zero or one row to the outer SQL statement.
You can place a subquery in a WHERE clause, a HAVING clause, or a FROM clause
of a SELECT statement

◆ Exam: Single Row subqueries in WHERE clause

AGENTS

| agent_code | agent_name | working_area | commission | phone_no |
|------------|------------|--------------|------------|--------------|
| A007 | Ramasundar | Bangalore | 0.15 | 077-25814763 |
| A003 | Alex | London | 0.13 | 075-12458969 |
| A008 | Alford | New York | 0.12 | 044-25874365 |
| A011 | Ravi Kumar | Bangalore | 0.15 | 077-45625874 |
| A010 | Santakumar | Chennai | 0.14 | 007-22388644 |
| A012 | Lucida | San Jose | 0.12 | 044-52981425 |
| A005 | Anderson | Brisban | 0.13 | 045-21447739 |
| A001 | Subbarao | Bangalore | 0.14 | 077-12346674 |
| A002 | Mukesh | Mumbai | 0.11 | 029-12358964 |
| A006 | McDen | London | 0.15 | 078-22255588 |
| A004 | Ivan | Torento | 0.15 | 008-22544166 |
| A009 | Benjamin | Hampshire | 0.11 | 008-22536178 |

| AGENT_NAME | AGENT_CODE | PHONE_NO |
|------------|------------|--------------|
| Alex | A003 | 075-12458969 |

```
SELECT agent_name, agent_code, phone_no  
FROM agents  
WHERE agent_code =  
(SELECT agent_code FROM agents WHERE agent_name = 'Alex')
```

© Copyright 2015 FSOFT. All rights reserved

7/26

- Multiple row subquery: loại này trả về ít nhất 1 bản ghi trong tập truy vấn con, có thể sử dụng các toán tử IN, ANY, ALL trong truy vấn ngoài để điều khiển kết quả trả về của toàn bộ truy vấn

2.2 MULTIPLE ROW SUBQUERY

- ◆ **Multiple row subquery returns one or more rows to the outer SQL statement. You may use the IN, ANY, or ALL operator in outer query to handle a subquery that returns multiple rows**
- ◆ **Ex: Multiple row Subquery in a WHERE clause**

ORDERS

| ord_num | ord_amount | advance_amount | ord_date | cust_code | agent_code | ship_city |
|---------|------------|----------------|------------|-----------|------------|-----------|
| 200105 | 2500.00 | 500.00 | 2008-07-18 | C00025 | A011 | Bangalore |
| 200112 | 2000.00 | 400.00 | 2008-05-30 | C00016 | A007 | London |
| 200113 | 4000.00 | 600.00 | 2008-06-10 | C00022 | A002 | Mumbai |
| 200117 | 800.00 | 200.00 | 2008-10-20 | C00014 | A001 | New York |
| 200130 | 2500.00 | 400.00 | 2008-07-30 | C00025 | A011 | Bangalore |

```
SELECT ord_num, ord_amount, ord_date, cust_code, agent_code
FROM orders
WHERE agent_code IN
(
    SELECT agent_code FROM agents
    WHERE working_area='Bangalore'
)
```

| ORD_NUM | ORD_AMOUNT | ORD_DATE | CUST_CODE | AGENT_CODE |
|---------|------------|-----------|-----------|------------|
| 200130 | 2500 | 30-JUL-08 | C00025 | A011 |
| 200105 | 2500 | 18-JUL-08 | C00025 | A011 |
| 200117 | 800 | 20-OCT-08 | C00014 | A001 |

© Copyright 2015 FSOFT. All rights reserved 8/26

- Multiple column subquery: truy vấn con trả về tập kết quả chứa nhiều cột

2.3 MULTIPLE COLUMN SUBQUERY

- ◆ **You can write subqueries that return multiple columns**
- ◆ **Ex: Multiple column Subquery in a FROM clause**

```
SELECT ord_num, agent_code, ord_date, ord_amount
FROM orders
WHERE (agent_code, ord_amount) IN
(
    SELECT agent_code, MIN(ord_amount)
    FROM orders
    GROUP BY agent_code
)
```

| ORD_NUM | AGENT_CODE | ORD_DATE | ORD_AMOUNT |
|---------|------------|-----------|------------|
| 200124 | A007 | 20-JUN-08 | 500 |
| 200126 | A002 | 24-JUN-08 | 500 |
| 200120 | A002 | 20-JUL-08 | 500 |
| 200123 | A002 | 16-SEP-08 | 500 |
| 200104 | A004 | 13-MAR-08 | 1500 |
| 200121 | A004 | 23-SEP-08 | 1500 |
| 200116 | A009 | 13-JUL-08 | 500 |
| 200105 | A011 | 18-JUL-08 | 2500 |
| 200130 | A011 | 30-JUL-08 | 2500 |
| 200131 | A012 | 26-AUG-08 | 900 |
| 200135 | A010 | 16-SEP-08 | 2000 |
| 200115 | A013 | 08-FEB-08 | 2000 |
| 200117 | A001 | 20-OCT-08 | 800 |
| 200111 | A008 | 10-JUL-08 | 1000 |
| 200118 | A006 | 20-JUL-08 | 500 |
| 200103 | A005 | 15-MAY-08 | 1500 |
| 200127 | A003 | 20-JUL-08 | 2500 |

© Copyright 2015 FSOFT. All rights reserved 9/26

- Correlated subquery:là loại truy vấn đặc biệt trong đó truy vấn con có tham chiếu tới một hoặc vài cột của truy vấn ngoài, nói cách khác truy vấn con có quan hệ với truy vấn ngoài

2.4 CORRELATED SUBQUERY

- ◆ Reference one or more columns in the outer SQL statement.
The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement
- ◆ Ex: Correlated Subquery in a FROM clause

```
SELECT * FROM orders o
WHERE agent_code IN
(
    SELECT agent_code FROM agents
    a
    WHERE o.ship_city =
        a.working_area
)
```

- ◆ Result:

| ord_num | ord_amount | advance_amount | ord_date | cust_code | agent_code | ship_city |
|---------|------------|----------------|------------|-----------|------------|-----------|
| 200105 | 2500.00 | 500.00 | 2008-07-18 | C00025 | A011 | Bangalore |
| 200113 | 4000.00 | 600.00 | 2008-06-10 | C00022 | A002 | Mumbai |
| 200130 | 2500.00 | 400.00 | 2008-07-30 | C00025 | A011 | Bangalore |

© Copyright 2015 FSOFT. All rights reserved

10/26

- Nested subquery: truy vấn lồng là truy vấn con lồng bên trong các truy vấn con khác

2.5 NESTED SUBQUERY

- ◆ A subquery can be nested inside other subqueries.
- ◆ Ex: Nested Subquery in a WHERE clause

```
SELECT *
FROM orders
WHERE ship_city IN
(
    SELECT DISTINCT working_area
    FROM agents
    WHERE agent_code IN
    (
        SELECT agent_code
        FROM agents
        WHERE commission >= 0.14
    )
)
```

- ◆ Result:

| ord_num | ord_amount | advance_amount | ord_date | cust_code | agent_code | ship_city |
|---------|------------|----------------|------------|-----------|------------|-----------|
| 200105 | 2500.00 | 500.00 | 2008-07-18 | C00025 | A011 | Bangalore |
| 200112 | 2000.00 | 400.00 | 2008-05-30 | C00016 | A007 | London |
| 200130 | 2500.00 | 400.00 | 2008-07-30 | C00025 | A011 | Bangalore |

© Copyright 2015 FSOFT. All rights reserved

11/26

3. COMMON case use subquery

Các trường hợp điển hình dùng subquery

- Subquery với bí danh(ALIAS): nhiều câu lệnh trong đó subquery và outer query tham chiếu tới cùng 1 bảng
- Subquery với IN/NOT IN: kết quả của subquery khi dùng với IN hoặc NOT IN là một danh sách 0 hoặc nhiều giá trị sau khi subquery trả về kết quả, outer query sẽ sử dụng chúng để kiểm tra phần tử so sánh có nằm trong hay không nằm trong tập kết quả trả về subquery

- Subquery với EXISTS/NOT EXISTS: trường hợp này toán tử kết hợp với subquery tạo thành một chức năng kiểm tra sự tồn tại.
- Subquery in update, delete, insert, select
- 4. Các quy tắc sử dụng subquery
 - Truy vấn con phải được đặt trong ngoặc
 - Các truy vấn con trả về một hay nhiều dòng thì chỉ sử dụng được với toán tử đa trị chẳng hạn như toán tử IN
 - Truy vấn con có thể bao gồm mệnh đề WHERE, GROUP BY, HAVING
 - Truy vấn con không bao gồm mệnh đề COMPUTE hoặc FOR BROWSE
 - Bạn chỉ được phép chứa 1 mệnh đề ORDER BY khi có mệnh đề TOP
 - Bạn có thể lồng truy vấn con lên đến 32 cấp

Advanced SQL - Lecture 03: CTE & Ranking Functions, SQL Code Practice

Biểu thức bảng và các hàm ranking trong SQL

- Biểu thức bảng(Common table expression): là cách tạo ra bộ dữ liệu trung gian từ các phép truy vấn và được sử dụng lại trong phép truy vấn ngay sau đó
- Biểu thức bảng(CTE) tương tự như 1 bảng dẫn xuất ở chỗ nó không được lưu trữ một số đối tượng và chỉ kéo dài trong suốt thời gian của câu truy vấn, nhưng lại khác bảng dẫn xuất ở chỗ biểu thức bảng(CTE) có thể tự tham chiếu tới bản thân nó và có thể tham chiếu nhiều lần trong 1 câu truy vấn
- Mục đích của biểu thức bảng(CTE): nó giúp tạo truy vấn đệ quy, thay thế view cho một số trường hợp, cho phép nhóm 1 cột từ truy vấn con, tham chiếu tới bảng kết quả nhiều lần trong cùng 1 lệnh

1.COMMON TABLE EXPRESSIONS (1/3)

◆ **A CTE can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE. It can be used:**

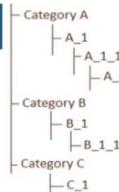
◇ **This is used to store result of a complex sub query for further use.(As a temporary table)**

◇ **Create a recursive query**

```
Syntax ;WITH CTE_Name [ col_names ]
AS
(
  CTE_query_definition
)
```

1.COMMON TABLE EXPRESSIONS (2/3)

Ex



| Column Name | Data Type | Allow Nulls |
|--------------------|----------------|-------------------------------------|
| id | bigint | <input type="checkbox"/> |
| name | nvarchar(512) | <input type="checkbox"/> |
| description | nvarchar(1024) | <input checked="" type="checkbox"/> |
| parent_id | bigint | <input checked="" type="checkbox"/> |

| id | name | description | parent_id |
|-----------|-------------|--------------------|------------------|
| 13 | Category A | NULL | NULL |
| 14 | Category B | NULL | NULL |
| 15 | Category C | NULL | NULL |
| 16 | A_1 | NULL | 13 |
| 17 | A_1_1 | NULL | 16 |
| 18 | A_1_2 | NULL | 17 |
| 19 | B_1 | NULL | 14 |
| 20 | B_1_1 | NULL | 19 |
| 21 | C_1 | NULL | 15 |
| NULL | NULL | NULL | NULL |

| Results | | | |
|-----------|------------------------|------------------------|-----------------|
| id | Name | Descriptions | ParentId |
| 1 | Laptop | Laptop | 0 |
| 2 | Ultrabook | Ultrabook | 0 |
| 3 | Netbook | Netbook | 0 |
| 4 | Desktop | Desktop | 0 |
| 5 | Linh kiện máy tính | Linh kiện máy tính | 0 |
| 6 | Thiết bị văn phòng | Thiết bị văn phòng | 0 |
| 7 | Main board | Main board | 5 |
| 8 | CPU | CPU | 5 |
| 9 | RAM | RAM | 5 |
| 10 | HDD | HDD | 5 |
| 11 | Nguồn máy tính | Nguồn máy tính | 5 |
| 12 | CD/DVD Reader | CD/DVD Reader | 5 |
| 13 | Case | Case | 5 |
| 14 | Card đồ họa | Card đồ họa | 5 |
| 15 | Card mạng | Card mạng | 5 |
| 16 | Card sound | Card sound | 5 |
| 17 | Máy in phun | Máy in phun | 6 |
| 18 | Máy in phun 4 màu CMYK | Máy in phun 4 màu CMYK | 17 |
| 19 | Máy in phun 6 màu CMYK | Máy in phun 6 màu CMYK | 17 |
| 20 | Máy in phun 8 màu CMYK | Máy in phun 8 màu CMYK | 17 |

| Category |
|--------------------|
| Id |
| Name |
| Description |
| ParentID |

© Copyright 2015 FSOFT. All rights reserved

4/29

1.COMMON TABLE EXPRESSIONS (3/3)

♦ Solution for this example:

```

WITH temp(id, name, alevel)
as
(
    Select id, name, 0 as alevel
    From Category Where parent_id is null
    Union All
    Select b.id, b.name, a.level + 1 From temp as a, Category as b
    Where a.id = b.parent_id
)
Select * From temp
  
```

| id | name | alevel |
|-----------|---------------|---------------|
| 1 | 13 Category A | 0 |
| 2 | 14 Category B | 0 |
| 3 | 15 Category C | 0 |
| 4 | 21 C_1 | 1 |
| 5 | 19 B_1 | 1 |
| 6 | 20 B_1_1 | 2 |
| 7 | 16 A_1 | 1 |
| 8 | 17 A_1_1 | 2 |
| 9 | 18 A_1_2 | 3 |

© Copyright 2015 FSOFT. All rights reserved

5/29

- Trong biểu thức bảng có thể tạo truy vấn đệ quy()

1.COMMON TABLE EXPRESSIONS RECURSIVE

◆ Recursive Queries Using Common Table Expressions

Syntax

```
WITH cte_name ( col_names )
AS
(
    -- Anchor member is defined.
    CTE_query_definition
    UNION ALL
    -- Recursive member is defined referencing cte_name.
    CTE_query_definition
)
-- Statement using the CTE
SELECT *
FROM cte_name
```

© Copyright 2015 FSOFT. All rights reserved

6/29

2. Ranking function

Các hàm xếp hạng ranking cho phép bạn đánh số liên tục hay xếp loại cho tập hợp kết quả, các hàm này có thể được sử dụng để cung cấp số thứ tự trong hệ thống đánh số tuần tự khác nhau

- Có 4 hàm ranking:

2.RANKING FUNCTIONS (2/6)

◆ Let's take following sample table and data to know about RANK, RANK_DENSE, NTILE and ROW_NUMBER

Ex

```
CREATE TABLE ExamResult(FullName varchar(50), Subject varchar(20),
Marks int)
```

```
INSERT INTO ExamResult VALUES('Adam','Maths',70)
INSERT INTO ExamResult VALUES ('Adam','Science',80)
INSERT INTO ExamResult VALUES ('Adam','Social',60)
```

```
INSERT INTO ExamResult VALUES('Rak','Maths',60)
INSERT INTO ExamResult VALUES ('Rak','Science',50)
INSERT INTO ExamResult VALUES ('Rak','Social',70)
```

```
INSERT INTO ExamResult VALUES('Sam','Maths',90)
INSERT INTO ExamResult VALUES ('Sam','Science',90)
INSERT INTO ExamResult VALUES ('Sam','Social',80)
```

© Copyright 2015 FSOFT. All rights reserved

9/29

- Rank: trả về thứ tự bên trong dòng của tập kết quả, thứ hạng này phụ thuộc vào tiêu chí đưa ra để xếp hạng. Hàm Rank cho phép chọn vùng xếp hạng, có nghĩa là thứ hạng chỉ có nghĩa trong vùng đó mà thôi.

2.RANKING FUNCTIONS (4/6)

- ◆ **Rank**: Returns the rank of each row within the partition of a result set

Ex

```
SELECT      FullName, Subject, Marks, RANK() OVER(PARTITION BY  
          FullName ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY FullName, Subject
```

| | FullName | Subject | Marks | Rank |
|---|----------|---------|-------|------|
| 1 | Adam | Maths | 70 | 2 |
| 2 | Adam | Science | 80 | 1 |
| 3 | Adam | Social | 60 | 3 |
| 4 | Rak | Maths | 60 | 2 |
| 5 | Rak | Science | 50 | 3 |
| 6 | Rak | Social | 70 | 1 |
| 7 | Sam | Maths | 90 | 1 |
| 8 | Sam | Science | 90 | 1 |
| 9 | Sam | Social | 80 | 3 |

© Copyright 2015 FSOFT. All rights reserved

11/29

- Rank_DENSE:Tương tự như rank nhưng hàm này không cung cấp khoảng cách giữa các số xếp loại, nghĩa là nếu có 2 số cùng thứ hạng 1 thì thứ hạng tiếp theo vẫn là 2

2.RANKING FUNCTIONS (5/6)

- ◆ **Dense Rank**: Returns the rank of rows within the partition of a result set, without any gaps in the ranking

Ex

```
SELECT      FullName, Subject, Marks, DENSE_RANK() OVER  
          (PARTITION BY FullName ORDER BY Marks DESC) Rank  
FROM ExamResult  
ORDER BY FullName
```

Dense Rank

| | FullName | Subject | Marks | Rank |
|---|----------|---------|-------|------|
| 1 | Adam | Science | 80 | 1 |
| 2 | Adam | Maths | 70 | 2 |
| 3 | Adam | Social | 60 | 3 |
| 4 | Rak | Social | 70 | 1 |
| 5 | Rak | Maths | 60 | 2 |
| 6 | Rak | Science | 50 | 3 |
| 7 | Sam | Maths | 90 | 1 |
| 8 | Sam | Science | 90 | 1 |
| 9 | Sam | Social | 80 | 2 |

Rank

| | FullName | Subject | Marks | Rank |
|---|----------|---------|-------|------|
| 1 | Adam | Maths | 70 | 2 |
| 2 | Adam | Science | 80 | 1 |
| 3 | Adam | Social | 60 | 3 |
| 4 | Rak | Maths | 60 | 2 |
| 5 | Rak | Science | 50 | 3 |
| 6 | Rak | Social | 70 | 1 |
| 7 | Sam | Maths | 90 | 1 |
| 8 | Sam | Science | 90 | 1 |
| 9 | Sam | Social | 80 | 3 |

© Copyright 2015 FSOFT. All rights reserved

12/29

- NTILE: được sử dụng để phân tán các bản ghi hoặc các dòng thành các nhóm theo số nhóm được chỉ định. Số nhóm tùy thuộc vào đối số N(Nếu N=3 thì chia bản ghi thành 3 nhóm và hệ quản trị sẽ tự xác định bản ghi rồi chia ra cho 3 để ra được số nhóm)



2.RANKING FUNCTIONS (6/6)

- ◆ **Ntile:** Distributes the rows in an ordered partition into a specified number of groups

Ex

```
SELECT    FullName, Subject, Marks, NTILE(2) OVER  
          (ORDER BY Marks DESC) Quartile  
FROM ExamResult
```

| | FullName | Subject | Marks | Quartile |
|---|----------|---------|-------|----------|
| 1 | Rak | Science | 50 | 1 |
| 2 | Adam | Social | 60 | 1 |
| 3 | Rak | Maths | 60 | 1 |
| 4 | Adam | Maths | 70 | 1 |
| 5 | Rak | Social | 70 | 1 |
| 6 | Adam | Science | 80 | 2 |
| 7 | Sam | Social | 80 | 2 |
| 8 | Sam | Maths | 90 | 2 |
| 9 | Sam | Science | 90 | 2 |

© Copyright 2015 FSOFT. All rights reserved

13/29

- ROW_NUMBER: Trả về số thứ tự của mỗi dòng trong một phân vùng của tập kết quả, phân vùng tùy thuộc vào lựa chọn của người viết



2.RANKING FUNCTIONS (3/6)

- ◆ **Row Number:** Returns the sequential number of a row within a partition of a result set

Ex

```
SELECT    FullName, Subject, Marks,  
          ROW_NUMBER() OVER(ORDER BY FullName) RowNumber  
FROM ExamResult  
ORDER BY FullName, Subject
```

| | FullName | Subject | Marks | RowNumber |
|---|----------|---------|-------|-----------|
| 1 | Adam | Maths | 70 | 1 |
| 2 | Adam | Science | 80 | 2 |
| 3 | Adam | Social | 60 | 3 |
| 4 | Rak | Maths | 60 | 4 |
| 5 | Rak | Science | 50 | 5 |
| 6 | Rak | Social | 70 | 6 |
| 7 | Sam | Maths | 90 | 7 |
| 8 | Sam | Science | 90 | 8 |
| 9 | Sam | Social | 80 | 9 |

© Copyright 2015 FSOFT. All rights reserved

10/29

3. SQL code practice

Lưu ý trong code SQL

- Khai báo tường minh danh sách cột trả về trong lệnh select
- Thay vì dùng select * thì điều này sẽ cải thiện hiệu năng truy vấn
- Ngăn ngừa những lỗi tiềm ẩn liên quan đến việc thay đổi CSDL trong tương lai

◆ **Explicitly Name Columns in SELECT Statements**

◇ **Improve performance**

◇ **Prevent potential failures related to some database schema change in the future**

Ex

```
SELECT ContactID, FirstName, LastName  
FROM Person.Contact
```

◆ **Instead of:**

```
SELECT * FROM Person.Contact
```

- Đối với lệnh insert thì tương tự như lệnh select: thì việc khai báo tương minh danh sách cột giúp code rõ ràng và sáng hơn tránh những nhầm lẫn đáng tiếc về thứ tự cột. Việc khai báo này sẽ giúp chúng ta không phải quan tâm đến thứ tự cột trong DB mà chỉ cần quan tâm có những cột nào để truyền dữ liệu tương ứng

◆ **Explicitly Name Columns in INSERT Statements**

◇ **Prevent potential failures related to some database schema change in the future**

◇ **Prevent error with identity column**

Ex

```
INSERT dbo.Employee (FirstName, LastName, NationalIDNumber,  
ManagerID, Title, BirthDate, MaritalStatus,  
Gender)  
VALUES ('Bill', 'Gates', '123456', NULL, 'CEO', '1959-01-01', 'M',  
'M')  
◆ Instead of .
```

```
INSERT dbo.Employee  
VALUES ('Bill', 'Gates', '123456', NULL, 'CEO', '1959-01-01', 'M',  
'M')
```

- Luôn chỉ định schema cho các truy vấn , việc này giúp ngăn ngừa lỗi tiềm ẩn liên quan đến thay đổi schema hoặc thay đổi quyền trong các schema trong tương lai



◆ Always specific schema for tables in query

- ◇ Prevent potential failures related to some database schema change or permission change on schema in the future

```
SELECT A.Key1 , B.Col1 , C.Col2  
FROM dbo.TableAA  
INNER JOIN dbo.TableB B  
ON A.Key1 = B.Key1  
INNER JOIN dbo.TableC C  
ON A.Key1 = C.Key1  
WHERE A.Col1 = '123'  
AND B.Col2 like 'A%'
```

```
SELECT A.Key1 , B.Col1 , C.Col2  
FROM Tabl  
INNER JOIN Ta  
ON A.Key1 =  
INNER JOIN TableC C  
ON A.Key1 = C.Key1  
WHERE A.Col1 = '123'  
AND B.Col2 like 'A%'
```

- Nên cung cấp bí danh cho các bảng, điều này giúp cho truy vấn trở nên gọn gàng và dễ đọc hơn.



◆ Always provides alias for tables in query

- ◇ Make query more clearer and easier to read

```
SELECT A.Key1 , B.Col1 , C.Col2  
FROM dbo.TableAA  
INNER JOIN dbo.TableB B  
ON A.Key1 = B.Key1  
INNER JOIN dbo.TableC C  
ON A.Key1 = C.Key1  
WHERE A.Col1 = '123'  
AND B.Col2 like 'A%'
```

```
SELECT TableA.Key1 , TableB.Col1 ,  
TableC.Col2  
FROM dbo.  
INNER JOIN TableB  
ON TableA.Key1 = TableB.Key1  
INNER JOIN dbo.TableC  
ON TableA.Key1 = TableC.Key1  
WHERE TableA.Col1 = '123'  
AND TableB.Col2 like 'A%'
```

- Hạn chế sử dụng hàm trong mệnh đề WHERE, vì sử dụng hàm sẽ làm giảm hiệu xuất truy vấn

◆ **Avoid SQL Server functions in the WHERE clause**

◇ **Improve performance**

```
SELECT EmailAddress  
FROM person.contact  
WHERE EmailAddress like 'As%'
```

```
SELECT EmailAddress  
FROM person.contact  
WHERE len(EmailAddress) = 2 = 'As'
```

- Chỉ sử dụng từ khoá DISTINCT và UNION ALL khi thực sự cần thiết hay khi không biết dữ liệu có trùng lặp hay không vì với các từ khoá này SQL ENGINE sẽ phải xử lý thêm các thao tác sắp xếp và loại bỏ bản ghi trùng lặp dẫn đến hiệu năng giả đáng kể

Advanced SQL - Lecture 04: SQL Language Elements

Phần tử trong SQL

- Comment : --ghi chú trong sql
- 1. Định danh(Identifiers) : tên các đối tượng, đối tượng ở đây là CSDL, bảng, bảng ảo, chỉ mục, hàm thủ tục.v.v.
- Một định danh được tạo khi đối tượng được tạo ra hoặc được định nghĩa ra
- Định danh được dùng để tham chiếu đến đối tượng
- Có 2 loại định danh:
 - Regular identifiers: tên các bảng
 - Delimited Identifiers: loại này sử dụng dấu "" hoặc dấu[] để bao quanh định danh, loại này dùng cho các định danh bị trùng trong sql hoặc các định danh có chứa khoảng trắng giữa các từ



2.IDENTIFIERS

- ◆ The database object name is referred to as its identifier
 - ◇ An object identifier is created when the object is defined
 - ◇ The identifier is used to reference the object

- ◆ There are 2 types of Identifiers:

- ◇ Regular Identifiers:

- Example: Orders, Customers, Employee...

- ◇ Delimited Identifiers: Are enclosed in double quotation marks ("") or brackets ([])

- [My Table]
 - [1Person]

Ex

```
✓ Select * From [My Table]  
Where [Order]=40
```

© Copyright 2015 FSOFT. All rights reserved

4/17

2. Variables

- Biến trong SQL: biến cần phải khai báo trước câu lệnh tham chiếu đến nó, khi khai báo biến trong SQL ta sử dụng @. Còn đối với các biến toàn cục globle variable thì sẽ sử dụng 2 biến @@, khi khai báo biến chúng ta dùng DECLARE
- Để thực hiện gán giá trị ban đầu cho biến chúng ta dùng từ khoá set @tênbien = giá trị



3.VARIABLES

- ◆ Declare a variable

- ◇ Must be DECLARE and start with @ symbol

```
DECLARE @limit money  
DECLARE @min_range int, @hi_range int
```

- ◆ Assign a value into a variable using SET

```
SET @min_range = 0, @hi_range = 100  
SET @limit = $10
```

- ◆ Assign a value into a variable using

SELECT

```
SELECT @price = price FROM titles  
WHERE title_id = 'PC2091'
```

© Copyright 2015 FSOFT. All rights reserved

5/17

3. Control-of-flow

Khi chúng ta thực hiện 1 chương trình theo chế độ mặc định thì các câu lệnh được thực hiện tuần tự, chúng ta có thể thực hiện điều khiển luồng của chương trình bằng cách dùng các câu lệnh điều khiển như sau:



The T-SQL control-of-flow language keywords are:

CASE ... WHEN

BEGIN ... END

WHILE

BREAK /
CONTINUE

IF...ELSE

GOTO

RETURN

TRY...CATCH

- IF...ELSE: chúng ta có thể thực hiện một tập hợp các câu lệnh khác của SQL dựa trên điều khiển được chỉ ra câu lệnh SQL sau từ khoá IF chỉ được thực hiện nếu điều kiện đúng, trong trường hợp mệnh đề IF sai thì tập hợp các câu lệnh sau từ khoá else sẽ được thực hiện.



IF...ELSE

Define conditional and, optionally, alternate execution when a condition is false

Syntax

```
IF Boolean_expression  
    SQL_statement | block_of_statements  
[ ELSE  
    SQL_statement | block_of_statements]
```

- GOTO: lệnh goto sẽ chuyển luồng xử lý tới vị trí khác được chỉ định bởi nhãn

GOTO

◇ Alter the flow of execution to a label. The Transact-SQL statement or statements that follow GOTO are skipped and processing continues at the label

Syntax

```
--Define the label
```

label :

```
--Alter the execution:
```

```
GOTO label
```

- CASE...WHEN: hàm CASE kiểm định giá trị trên danh sách các điều kiện đưa ra sau đó trả về 1 hoặc nhiều kết quả

CASE ... WHEN

◇ Evaluate a list of conditions and returns one of multiple possible result expressions

Syntax

```
CASE input_expression
```

```
    WHEN when_expression1 THEN result_expression1
```

```
    WHEN when_expression2 THEN result_expression2
```

```
ELSE else_result_expression
```

```
END
```

- TRY...CATCH: thay vì luôn phải check biến error của SQL lúc thực hiện transaction thì bây giờ ta thực hiện các chuỗi lệnh đó trong try catch nếu lỗi thì rollback lại

TRY... CATCH

- ◊ **Provide error handling for T-SQL that is similar to the exception handling in the C# / Java**

Syntax

```
BEGIN TRY  
    { sql_statement | statement_block }  
END TRY  
BEGIN CATCH  
    [ { sql_statement | statement_block } ]  
END CATCH
```

- While : với cấu trúc lặp thì người lập trình có thể chỉ định 1 hoặc nhiều câu lệnh sẽ được lặp lại nhiều lần trong khi giá trị của biểu thức điều kiện có thể đúng
 - Thực tế cấu trúc lặp WHILE bị giới hạn trong nhiều trường hợp, bởi vì bản thân các lệnh truy vấn cập nhật dữ liệu như select,update,delete trong transaction SQL đã tự động thực hiện việc lặp từ dòng dữ liệu đầu tiên đến dòng dữ liệu cuối cùng trong bảng, vì vậy cấu trúc lặp while thông thường được dùng với các biến kiểu dữ liệu con trả

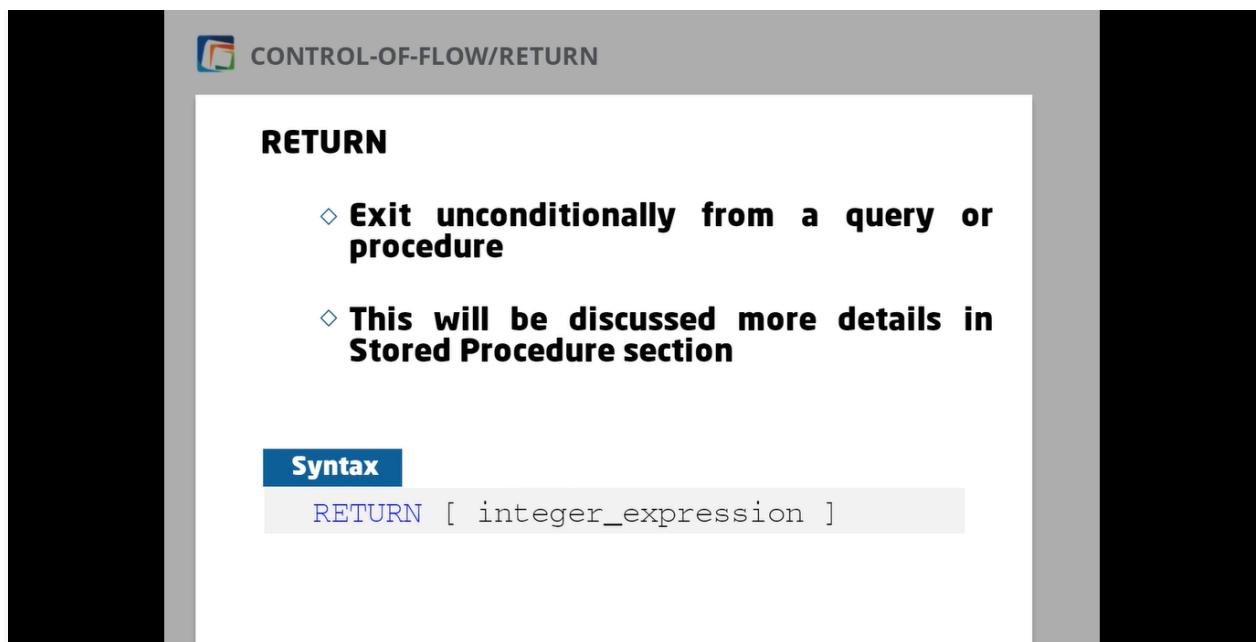
WHILE

- ◊ **Set a condition for the repeated execution of an statement block**
- ◊ **The statements are executed repeatedly as long as the specified condition is true**
- ◊ **The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords**

Syntax

```
WHILE Boolean_expression  
    { sql_statement | statement_block |  
    BREAK | CONTINUE }
```

- Return : câu lệnh này giúp trả về giá trị của một hàm, thủ tục lưu trữ hoặc lệnh này sẽ giúp thoát khỏi thủ tục lưu trữ hay các truy vấn

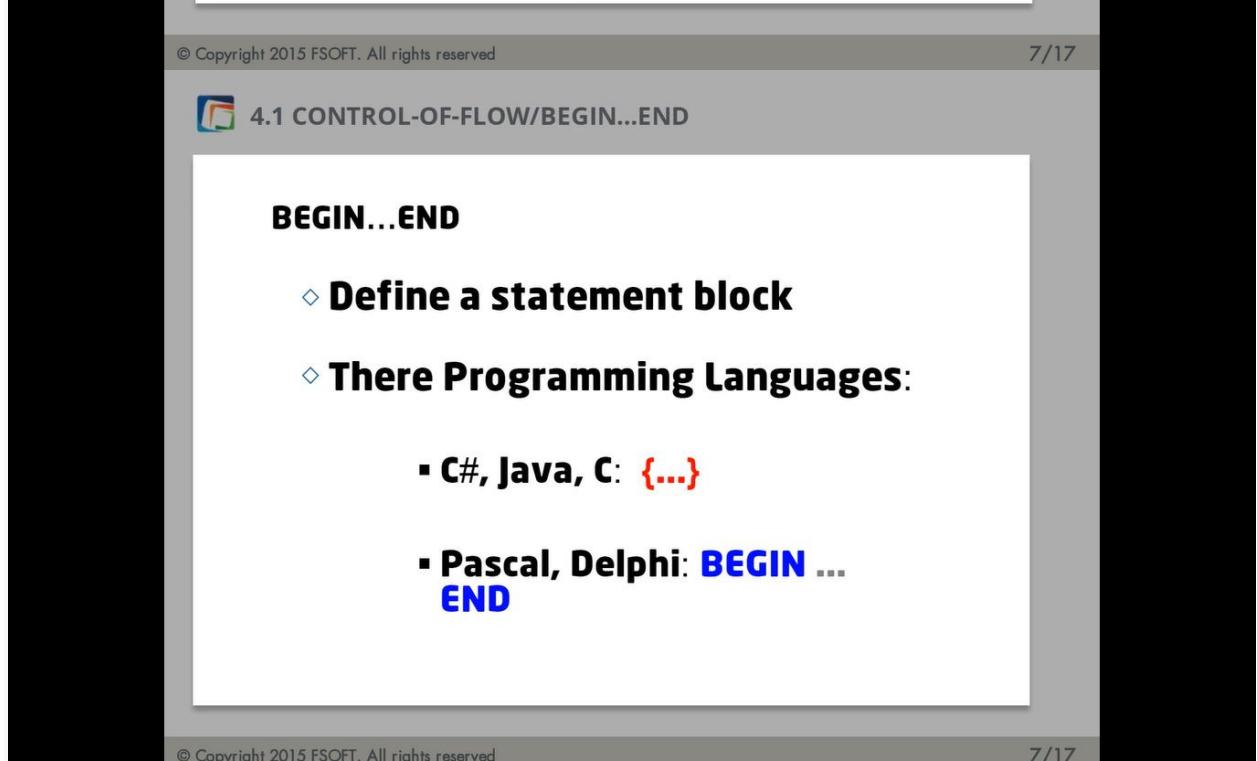


RETURN

- ◊ **Exit unconditionally from a query or procedure**
- ◊ **This will be discussed more details in Stored Procedure section**

Syntax

```
RETURN [ integer_expression ]
```



BEGIN...END

- ◊ **Define a statement block**
- ◊ **These Programming Languages:**
 - C#, Java, C: {...}
 - Pascal, Delphi: **BEGIN ... END**

© Copyright 2015 FSOFT. All rights reserved 7/17

4.1 CONTROL-OF-FLOW-BEGIN...END

© Copyright 2015 FSOFT. All rights reserved 7/17

Advanced SQL - Lecture 05: Stored Procedure

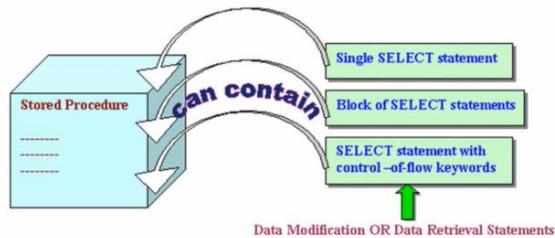
Thủ tục lưu trữ là tập hợp các câu lệnh SQL được mô tả như 1 khối các câu lệnh đơn lẻ dùng để thực thi 1 nhiệm vụ cụ thể nó giúp ích khi 1 nhiệm vụ được thực hiện nhiều lần. Khi đó thay vì viết lại các đoạn code thì ta chỉ việc gọi thủ tục

1. Stored procedure: thủ tục lưu trữ là tập hợp các câu lệnh SQL được biên dịch trước, gọi là pre compiles

- Thủ tục lưu trữ được đặt tên và được xử lý như một khối lệnh thống nhất chứ không phải thực hiện rời rạc. SQL server cũng cấp sẵn các thủ tục được lưu trong hệ thống giúp thực hiện một số công việc thường xuyên. Nó được gọi là thủ tục hệ thống(System stored procedure)
- Thủ tục lưu trữ trong SQL cũng tương tự như khái niệm thủ tục trong các ngôn ngữ lập trình khác bởi vì nó chấp nhận biến đầu vào và trả lại kết quả khi thực hiện. chứa những câu lệnh dùng trong lập trình, có thể thao tác với CSDL và có thể gọi đến các thủ tục khác và trả lại giá trị trạng thái khi thủ tục được gọi

STORED PROCEDURE OVERVIEW (1/5)

- ◆ A stored procedure (SP) is a collection of SQL statements that SQL Server compiles into a single execution plan.
- ◆ It can accept input parameters, return output values as parameters, or return success or failure status messages



© Copyright 2015 FSOFT. All rights reserved

3/25

- Thủ tục lưu trữ có thể trả về dữ liệu 1 trong 4 cách
 - Trả về theo đối số đầu ra(output parameter): trả về dữ liệu kiểu số nguyên hoặc kiểu kí tự hay biến con trả

STORED PROCEDURE OVERVIEW (2/5)

- ◆ **Stored procedures return data in four ways:**

- ◇ ***Output parameters*, which can return either data (such as an integer or character value) or a cursor variable (cursors are result sets that can be retrieved one row at a time).**

Ex

```
USE AdventureWorks
GO
CREATE PROCEDURE GetImmediateManager
    @employeeID INT, @managerID INT OUTPUT
AS
BEGIN
    SELECT @managerID = ManagerID
    FROM HumanResources.Employee
    WHERE EmployeeID = @employeeID
END
```

© Copyright 2015 FSOFT. All rights reserved

4/25

STORED PROCEDURE OVERVIEW (3/5)

- ◆ **Stored procedures return data in four ways:**

- ◇ ***Example (code java):***

Ex

```
public static void executeStoredProcedure(Connection con) {
    try {
        CallableStatement cstmt = con
            .prepareCall("{call dbo.GetImmediateManager(?, ?)}");
        cstmt.setInt(1, 5);
        cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
        cstmt.execute();
        System.out.println("MANAGER ID: " + cstmt.getInt(2));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



© Copyright 2015 FSOFT. All rights reserved

5/25

- Trả về kết quả sử dụng từ khóa return

STORED PROCEDURE OVERVIEW (4/5)

◆ **Stored procedures return data in four ways:**

- ◇ **Return codes**, which are always an integer value.

Ex

```
CREATE PROC dbo.TestReturn (@InValue int)
AS
    Return @InValue + 10
GO
-- Call SP
DECLARE @ReturnValue
INT EXEC @ReturnValue = TestReturn 3
SELECT ReturnValue=@ReturnValue
```

© Copyright 2015 FSOFT. All rights reserved

6/25

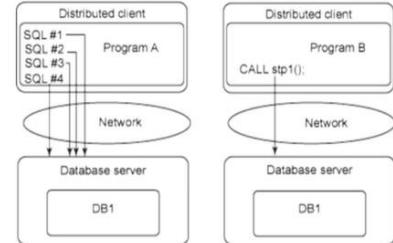
- Trả về một tập kết quả với mỗi lệnh select chứa trong thủ tục lưu trữ hoặc trong một thủ tục lưu trữ khác được gọi bởi thủ tục lưu trữ hiện hành
- Trả về kết quả thông qua một con trả toàn cục mà có thể tham chiếu tới bên ngoài con trả lưu trữ
- Lợi ích của thủ tục lưu trữ:
 - Giảm dung lượng chuyển đổi qua lại giữa client và server
 - Bảo mật hơn

BENEFIT OF USING SP (1/2)

◆ **Benefit of Using SP**

- ◇ **Reduced server/client network traffic:**

Only the call to execute the procedure is sent across the network



- ◇ **Stronger security**

When calling a procedure over the network, only the call to execute the procedure is visible. Therefore, malicious users cannot see table and database object names, embed Transact-SQL statements of their own, or search for critical data

© Copyright 2015 FSOFT. All rights reserved

8/25

- Có thể tái sử dụng code, có thể viết 1 lần và gọi nhiều lần
- Cải thiện hiệu năng

STORED PROCEDURE VS. SQL STATEMENT

SQL Statement

First Time

- Check syntax
- Compile
- Execute
- Return data

Second Time

- Check syntax
- Compile
- Execute
- Return data

Stored Procedure

Creating

- Check syntax
- Compile

First Time

- Execute
- Return data

Second Time

- Execute
- Return data

Cú pháp cơ bản của SP:

CREATE A SP- SYNTAX

◆ **Create / Modify a SP**

Syntax:

CREATE PROC[EDURE] procedure_name

[@parameter_name data_type][= default] OUTPUT[,....,

AS

SQL_statement_block



EXEC, UPDATE, DELETE A SP

- ◆ **Execute a Procedure:**

EXEC[UTE] procedure_name

- ◆ **Update a Procedure**

ALTER PROC[EDURE] procedure_name

[@parameter_name data_type]

[= default] [OUTPUT]

[,...,n]

AS

SQL_statement(s)

- ◆ **Delete a Procedure**

DROP PROC[EDURE] procedure_name

© Copyright 2015 FSOFT. All rights reserved

12/25

Hạn chế khi dùng SP:



STORED PROCEDURE DISADVANTAGES

- ◆ **Make the database server high load in both memory and processors**

- ◆ **Difficult to write a procedure with complexity of business logic**
- ◆ **Difficult to debug**

- ◆ **Not easy to write and maintain**

© Copyright 2015 FSOFT. All rights reserved

14/25

Advanced SQL - Lecture 06: Trigger

Tương tự như thủ tục lưu trữ đó là trigger, trigger bao gồm tập các lệnh SQL kết hợp với nhau nhưng trigger phải được gắn liền với một bảng nào đó và được tự động thực thi xảy ra một trong các thao tác như insert, update, delete làm thay đổi dữ liệu của bảng

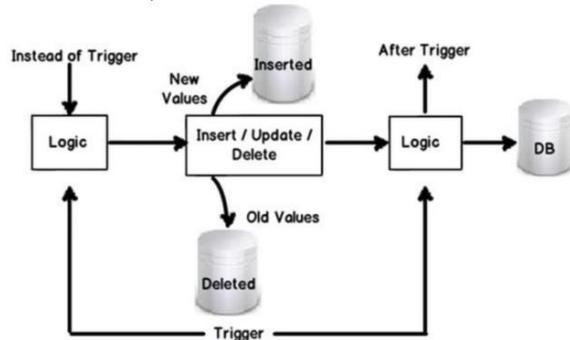
- Một số điểm khác biệt của trigger và SP được nói đến như:
 - Trigger được viết bên trong 1 bảng, được chỉ định và thực thi khi có sự thay đổi dữ liệu

- Không thể thực thi trong các thủ tục khác
 - Trigger không thể phân quyền đến từng user trong SQL, nó được phân quyền với quyền của bảng. Với SP thì bạn có thể phân quyền đến từng user
1. What is trigger?

Trigger là một thủ tục lưu trữ đặc biệt và nó sẽ được tự động thực thi khi có sự kiện tương ứng tác động. Trigger còn được gọi là bẫy lỗi, một bẫy lỗi được gắn liền với 1 bảng và được thực thi khi xuất hiện một sự kiện đặc biệt trong bảng khi insert, update, delete

WHAT IS A TRIGGER? (1/2)

- ◆ **A trigger is a special type of stored procedure that is executed automatically as part of a data modification.**
- ◆ **A trigger is created on a table and associated with one or more actions linked with a data modification (INSERT, UPDATE, or DELETE).**



© Copyright 2015 FSOFT. All rights reserved

3/27

- Các bẫy lỗi sẽ được kích hoạt tự động khi một hành động xảy ra
- Các bẫy lỗi được xử dụng phổ biến để ép các thao tác tuân theo một quy tắc nhất định, chúng giám sát sự thay đổi để chắc chắn rằng sự thay đổi đó là phù hợp với các quy tắc nghiệp vụ, vì vậy các bẫy lỗi thường được sử dụng để đảm bảo tính toàn vẹn dữ liệu
- Một số mục đích thông thường của bẫy lỗi như sau:
 - Duy trì bản sao và dẫn xuất của dữ liệu
 - Sử dụng trong trường hợp cột phải có ràng buộc phức tạp mà không biểu diễn được bằng constraints
 - Làm thay đổi theo tầng hoặc xoá dữ liệu liên quan đến bảng khác trong 1 CSDL
 - Thiết lập giá trị ban đầu cho cột: giá trị này thường phức tạp và không thể sử dụng default
 - Huỷ bỏ những thay đổi không đúng để đảm bảo toàn vẹn dữ liệu
- Có 3 loại trigger:
 - DML trigger: là các trigger cho phép biến đổi thao tác dữ liệu, nó sẽ được kích hoạt khi có thay đổi dữ liệu trên bảng hoặc bảng ảo. Loại này cũng thường được dùng để đảm bảo quy tắc nghiệp vụ hoặc toàn vẹn dữ liệu
 - DLL trigger: loại này được kích hoạt khi có sự thay đổi cấu trúc bảng, bảng ảo hay thủ tục lưu trữ qua các lệnh create, alter, drop với ý nghĩa đảm bảo an toàn CSDL mỗi khi có sự thay đổi về định nghĩa dữ liệu thì người thực hiện có nhu cầu xử lý tự động để ngăn chặn hoặc ghi nhận thông tin như ai thực hiện, thời gian, nội dung.v.v. thì người thực hiện có thể làm được điều này bằng DDL trigger
- DML trigger chia làm 2 loại:

- After trigger: trigger loại này sẽ được tự động gọi ngay sau khi các câu lệnh DML vừa kết thúc. Khi trigger được tạo nếu không có thiết lập gì thì mặc định là after và không sử dụng cho bảng ảo
- Instead of trigger: trigger loại này hoạt động khác một chút: B1:các lệnh DML vẫn được gọi để thực thi. B2: dữ liệu chưa bị thay đổi. B3: các lệnh instead of trigger sẽ được thực thi. Như vậy dựa trên cách thức hoạt động của nó thì ta có thể thấy thực chất các lệnh DML tác động trên bảng có gắn trigger instead of thì nó chỉ mang tính chất phát sinh sự kiện kích hoạt bên trong trigger thực thi.

- Cú pháp trigger

DML TRIGGER SYNTAX (1/2)

- ◆ **Create a trigger:**
`CREATE TRIGGER Trigger_name
ON table | view
[WITH ENCRYPTION]
{FOR | AFTER | INSTEAD OF}
{[DELETE] [,][INSERT] [,][UPDATE]}
AS Sql_statement`
- ◆ **Update a trigger**
`ALTER TRIGGER trigger_name
ON (table | view)
[WITH ENCRYPTION]{{(FOR | AFTER | INSTEAD OF)
{ [DELETE][,][INSERT][,][UPDATE] }
[NOT FOR REPLICATION] AS sql_statement [...n]}`
- ◆ **Delete a trigger**
`DROP TRIGGER { trigger_name }`

© Copyright 2015 FSOFT. All rights reserved

7/27

Ví dụ:

DML TRIGGER SYNTAX (2/2)

Ex

```
USE AdventureWorks
GO
CREATE TRIGGER [Person].[CHECK_CONT]
ON [Person].[Contact]
FOR INSERT
AS BEGIN
    DECLARE @ContactID NVARCHAR(50)
    SET @ContactID = (SELECT ContactID FROM
INSERTED)
    IF (@ContactID < 500)
        BEGIN
            PRINT 'ContactID < 500'
            ROLLBACK TRAN
        END
END
```

© Copyright 2015 FSOFT. All rights reserved

8/27

Bật tắt trigger:

DISABLE/ENABLE SYNTAX

**Disable trigger <trigger_name> ON
<table_name>**

Disable syntax

Enable syntax

**Enable trigger <trigger_name> ON
<table_name>**

© Copyright 2015 FSOFT. All rights reserved

9/27

Delete and inserted table:

- Đoạn mã chứa trong 1 trigger có thể truy cập đến 2 bảng logic các bảng này là bộ phận của bẫy lỗi và được gọi là inserted và deleted
- Bảng inserted và deleted chứa ảnh của dữ liệu trước và sau quá trình cập nhật dữ liệu trong bảng không bị tác động bởi thao tác cập nhật thì sẽ không nằm trong bảng inserted và deleted. Chúng được lưu trữ trong bộ nhớ mà không phải lưu trữ trên đĩa
- Bảng inserted chứa dữ liệu được thêm mới trong hành động insert hoặc update. Bảng này có ở cả 2 loại trigger

- Bảng delete chứa dữ liệu bị xoá trong hành động delete hoặc update, bảng này có ở cả 2 loại trigger

 DELETED AND INSERTED TABLES

- ◆ When you create a trigger, you have access to two temporary tables (the deleted and inserted tables).
- ◆ They are referred to as tables, but they are different from true database tables. They are stored in memory—not on disk.
- ◆ When the insert, update or delete statement is executed. All data will be copied into these tables with the same structure.



- ◆ The values in the inserted and deleted tables are accessible only within the trigger. Once the trigger is completed, these tables are no longer accessible.

© Copyright 2015 FSOFT. All rights reserved

10/27

Advanced SQL - Lecture 07: UDF & SQL Code Practice

User define function: Hàm do người dùng định nghĩa

- Hàm là đối tượng CSDL tương tự như thủ tục lưu trữ, điểm khác biệt giữa hàm và thủ tục là: Hàm trả về giá trị thông qua tên hàm, còn thủ tục thì không. Điều này cho phép ta sử dụng hàm như một biểu thức chẳng hạn bạn có thể gọi hàm trong danh sách chọn của lệnh select. Ngoài những hàm do CSDL cung cấp sẵn người sử dụng có thể định nghĩa thêm các hàm có thể phục vụ mục đích riêng của mình
- Hàm do người dùng định nghĩa thông thường sẽ nhận một tham số đầu vào
- Thực hiện một công việc và trả về kết quả, giá trị trả về có thể là giá trị đơn hoặc tập kết quả
- Phân loại hàm người dùng(UDF type) :Về phân loại tùy thuộc kết quả trả về của hàm mà chia loại như sau
 - Hàm vô hướng: tức là trả về giá trị đơn, giá trị trả về thuộc một trong các giá trị kiểu (in,char,varchar.v.v.v). Nhưng không hỗ trợ trả về kiểu Text, ntext, image, timestamp
 - Hàm trả về một bảng: loại này chia thành:
 - Hàm trả về bảng từ truy vấn đơn
 - Hàm trả về bảng từ truy vấn nhiều câu lệnh
- Cú pháp UDF scalar function

UDF FUNCTIONS SYNTAX (1/3)

◆ UDF Scalar Function Syntax

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name data_type [ = default ] [ READONLY ] } [ ,...n ] ] )
RETURNS return_data_type
[ AS ]
BEGIN
    function_body
    RETURN scalar_expression
END
```

© Copyright 2015 FSOFT. All rights reserved

5/16

Cú pháp trả về inline table-valued function

UDF FUNCTIONS SYNTAX (2/3)

◆ UDF Inline Table-valued Functions Syntax

```
CREATE FUNCTION [schema_name.]function_name
( [ { @parameter_name data_type [ = default ] } [ ,...n ] ] )
RETURNS TABLE
[ WITH < function_option > [ ,...n ] ]
[ AS ]
RETURN [ ( ) select_statement [ ) ]
```

© Copyright 2015 FSOFT. All rights reserved

6/16

Hàm multi-statement table-valued function

**◆ UDF Multi-statement Table-Valued Functions:**

```
CREATE FUNCTION [ schema_name. ] function_name
(( [ { @parameter_name data_type [ = default ] [ READONLY ] } [ ,...n ] ] ))
RETURNS @return_variable TABLE <table_type_definition>
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
    RETURN
END
```

• Code practice

- Lời khuyên liên quan đến việc transaction: nên dùng các transaction cho thao tác thay đổi dữ liệu delete, insert, update. Điều đó giúp ta đảm bảo đặc tính “ACID” quan trọng của CSDL
 - Atomicity, Consistency, Isolation, Durability: tính nguyên tố, nhất quán, tách biệt và bền vững
 - Transaction được dùng để đảm bảo tính toàn vẹn dữ liệu khi xảy ra cập nhật, cập nhật sẽ được hiểu theo nghĩa rộng là các hành động sửa đổi dữ liệu như insert, update, delete. Khi một transaction bao gồm nhiều lệnh cập nhật nó đảm bảo các lệnh cập nhật đều thành công hoặc trong trường hợp 1 lệnh gặp sự cố thì toàn bộ transaction bị huỷ bỏ, khi đó dữ liệu trở về trạng thái ban đầu. Nói cách khác transaction ngăn chặn tình huống dữ liệu cập nhật nửa chừng (trong đó 1 phần đc cập nhật, một phần bị bỏ qua)
 - Transaction properties(ACID): Atomicity là tính nguyên tố, thuộc tính này đảm bảo mỗi transaction là một khối duy nhất được thực hiện trọn vẹn hoặc hoàn toàn không được thực hiện trọn vẹn hoặc hoàn toàn không được thực hiện, nếu có một lỗi nào đó xảy ra trong transaction nó sẽ đc rollback về trạng thái ban đầu
 - Consistency: tính nhất quán SQL server ở mọi thời điểm dữ liệu đều phải nhất quán, tuân theo các ràng buộc được định nghĩa. Ví dụ: trường kiểu ngày phải có dữ liệu kiểu ngày giờ, bản ghi bán hàng phải có mã sản phẩm hợp lệ. Khi transaction được thực hiện dữ liệu sau khi được cập nhật phải ở trạng thái nhất quán, nếu transaction gây ra những vi phạm về ràng buộc dữ liệu thì hệ thống sẽ không cho phép thực hiện tiếp và huỷ bỏ toàn bộ transaction
 - Isolation: tính tách biệt cũng như các server khác thì SQL có thể đáp ứng nhiều yêu cầu xảy ra cùng lúc nhưng mỗi transaction được đảm bảo theo một ngữ cảnh riêng biệt của nó và không bị ảnh hưởng bởi các transaction khác, khi 2 transaction cùng cập nhật 1 dữ liệu thi SQL đảm bảo chúng thực hiện tuần tự không đâm lên chan của nhau

- Durability: tính bền vững khi transaction thực hiện xong đã commit những cập nhật đã trở nên cố định và dữ liệu sẽ luôn cố định, khi hệ thống gặp sự cố thì trong quá trình khôi phục nó sẽ đảm bảo dữ liệu cho những transaction đã commit
- Lời khuyên liên quan đến câu lệnh dùng Stored procedure
- Lời khuyên khi dùng truy vấn con(subquery)
- Lời khuyên dùng cho biểu thức câu lệnh truy vấn

SQL CODE PRACTICE

