Fpt University

Chapter 8: Database programming on SQL Server

Fpt University

Objectives

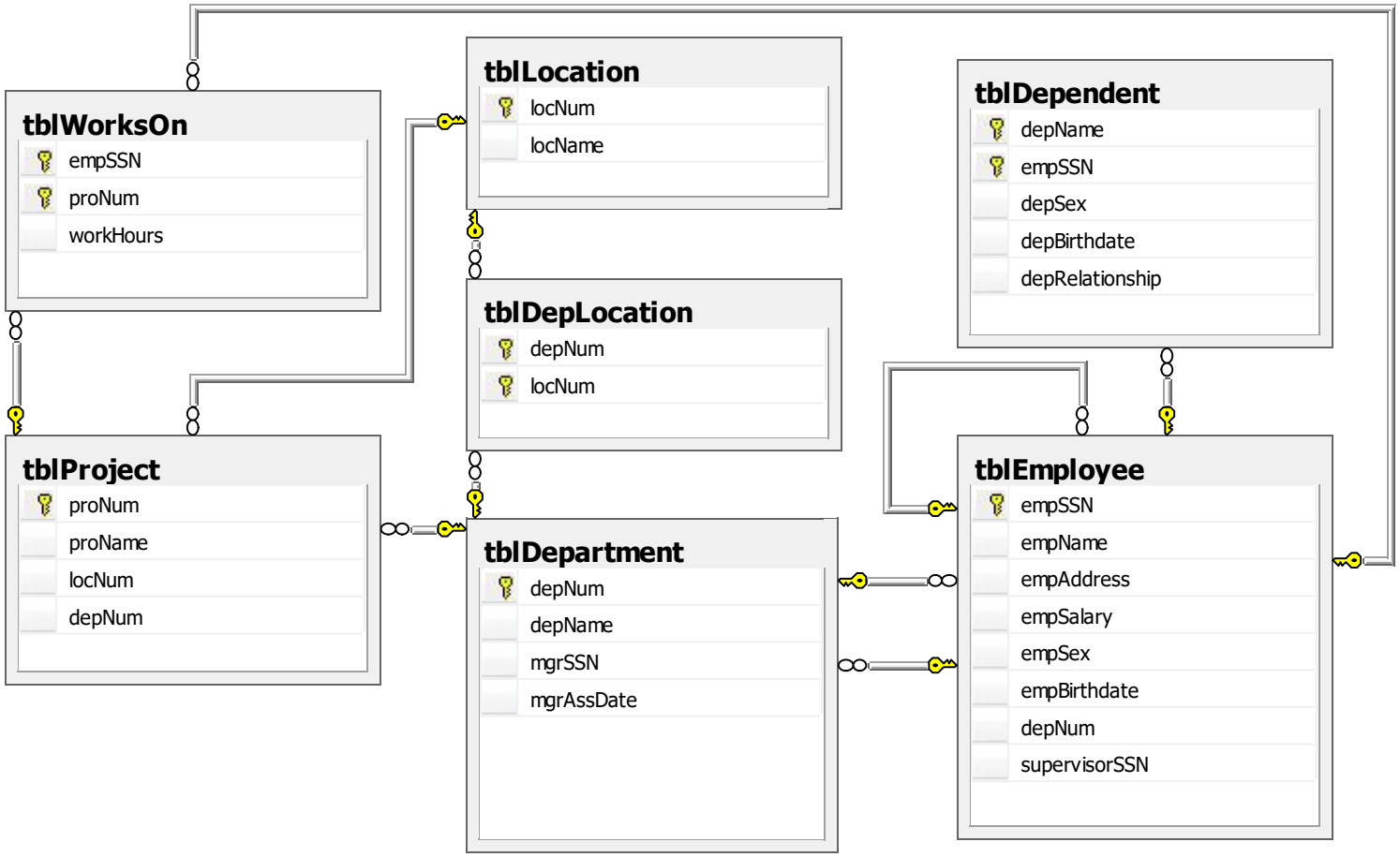
- Understand what stored-procedure are for and how to use
- Understand what functions are for and how to use
- Understand what triggers are for and how to use
- Understand the useful of trigger, function, stored-procedure (compared with SQL statements)
- Understand what cursors are for and how to use
- Understand the difference between T-SQL programming with other programming languages

Fpt University

Contents

- T-SQL Programming
- Stored-procedure
- Functions
- Triggers
- C ursors

Physical Diagram - FUHCompany



Fpt University

Data types in SQL

- **Integer**

bigint	-9,223,372,036,854,775,808	9.223.372.036.854.775,807
int	-2.147.483.648	2.147.483.647
smallint	-32.768	32.767
tinyint	0	255
bit	0	1
decimal	-10 ^ 38 +1	10 ^ 38 -1
numeric	-10 ^ 38 +1	10 ^ 38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214.748.3648	+214.748.3647

Data types in SQL

- **Numeric with floating point (real)**

float	-1.79E + 308	1,79E + 308
real	-3,40E + 38	3,40E + 38

- **Date and Time**

Datetime	'YYYY-MM-DD hh:mm:ss[.mmm]	-'1753-01-01 00:00:00' to '9999-12-31 23:59:59'. - '00:00:00' to '23:59:59:997'
Smalldatetime	YYYY-MM-DD hh:mm:ss	- '1900-01-01' to '2079-06-06' - '00:00:00' to '23:59:59'
Date	YYYY-MM-DD	'0001-01-01' TO '9999-12-31'
Time	'YYYY-MM-DD hh:mm:ss[.nnnnnnn]'	- '0001-01-01' to '9999-12-31' -'00:00:00.0000000' to '23:59:59.9999999'

Data types in SQL

- Strings

CHAR(size)	8000 characters	- Fixed length - non-Unicode characters
VARCHAR(size)	000 characters	- Non-Unicode data - Customizable length
Text	2GB	- Customizable length - Does not contain Unicode characters

- String with Unicode

NCHAR(size)	4000 characters	- Fixed length - Unicode characters
NVARCHAR(size)	4000 characters Or maximum number	- Size is the number of characters to store - Customizable length - If there is a maximum number -> the maximum number of characters is 2GB-Unicode characters
ntext	1.073.741.823	- Customizable length - Unicode characters

Short introduction to T-SQL programming

1. Variables

- Declare a variable

```
DECLARE  @local_variable [AS]  data_type [=initialvalue] , …
```

-*Variable_Name: Declare the name of the variable, the variable name always begins with the character "**@**"*

- Data_type: - SQL Server underlying data types or user-defined data types.

- Data types **text, ntext or image** are **not** allowed in **variable declarations**

- Example

```
DECLARE @empName NVARCHAR(20), @empSSN AS DECIMAL,
        @empSalary DECIMAL=1000
```

Short introduction to T-SQL programming

- **Assign a value into a variable : using SET or SELECT**

```
SET @a=2
SET @empName=’Mai Duy An’
```

- SET: only one variable is allowed to assign a value
- SELEC T: many variables at once

```
SELEC   T @a=1, @b=2, @Lastname='Nguyen’
```

Short introduction to T-SQL programming

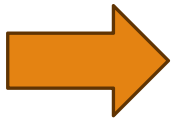
```
DECLARE
    @first_name VARCHAR(MAX),
    @last_name VARCHAR(MAX);
SELECT @last_name= last_name,
       @first_name = first_name
FROM
    employees
WHERE
    salary = 62000;
SELECT
    @first_name AS "Họ",
    @last_name AS "Tên";
```

Short introduction to T-SQL programming

- Assign a value into a variable using SQL command : SELECT or UPDATE

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	90000	2	Trần Phú
3	1003	Everest	Brad	0	71000	NULL	Hoàng Hoa Thám
4	1004	Horvath	Jack	0	90000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	1	Trần Phú
6	1006	Brow	Hilary	1	90000	NULL	An Dương Vương

```
Declare @total int
Select @total=Sum(salary)
From employees
Where salary = 90000
SELECT
    @total AS "Lương"
```

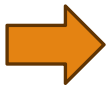


	Lương
1	270000

Short introduction to T-SQL programming

- Assign a value into a variable using SQL command : SELEC T or UPDATE

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	90000	2	Trần Phú
3	1003	Everest	Brad	0	71000	NULL	Hoàng Hoa Thám
4	1004	Horvath	Jack	0	90000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	1	Trần Phú
6	1006	Brow	Hilary	1	90000	NULL	An Dương Vương



	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	90000	2	Trần Phú
3	1003	Everest	Brad	0	71000	NULL	Hoàng Hoa Thám
4	1004	Smith	John	0	90000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	1	Trần Phú
6	1006	Brow	Hilary	1	90000	NULL	An Dương Vương

```
DECLARE
    @first_name VARCHAR(MAX),
    @last_name VARCHAR(MAX);
SELECT@first_name=first_name, @last_name=last_name
FROMemployees
WHEREId_emp=1001
UPDATEemployees
SETfirst_name=@first_name, last_name=@last_name
WHEREfirst_name='jack'
```

Short introduction to T-SQL programming

```
DECLARE @model_salary As INT;
SET @model_salary = 90000;
SELECT
    Id_emp,
    last_name,
    first_name,
salary
FROM
    employees
WHERE
    salary = @model_salary
ORDER BY
    last_name;
```

Short introduction to T-SQL programming

- Display value of a variable : using PRINT or SELECT

```
PRINT   @empName
SELECT  @empSalary
```

```
DECLARE @Ho nvarchar(30), @Ten nvarchar(20)
SET @Ho = N'Hà Thị Bạch'
SET @Ten = N'Lan'
SELECT @Ho = N'Hà Thị Bạch', @Ten = N'Lan'
PRINT N'Họ: ' + @Ho
PRINT N'Tên: ' + @Ten
```

Short introduction to T-SQL programming

```
DECLARE @Ho nvarchar(30), @Ten nvarchar(20)
SET @Ho = N'Hà Thị Bạch'
SET @Ten = N'Lan'
SELECT @Ho = N'Hà Thị Bạch', @Ten = N'Lan'
--PRINT N'Họ: ' + @Ho
--PRINT N'Tên: ' + @Ten
--OR
SELECT @Ho AS 'Họ', @Ten AS 'Tên'
```

Short introduction to T-SQL programming

```
DECLARE
@Diem_toan decimal(3,1) = 9.5,
@Diem_ly decimal(3,1) = 7.5,
@Diem_hoa decimal(3,1) = 8.0
DECLARE @diem_trung_binh decimal(3,1)
SET @Diem_trung_binh = (@Diem_toan + @Diem_ly + @Diem_hoa) / 3
PRINT @Diem_trung_binh
```

Short introduction to T-SQL programming

- C onverts an expression from one data type to a different data type : using C AST or C ONVERT function

C AST(expression AS data_type [(length)])

```
SELECT CAST('25' AS int)+30; → 55

SELECT CONVERT(int, '25') + 30;  → 55
```

Short introduction to T-SQL programming

```
DECLARE @empName NVARCHAR(20), @empSalary DECIMAL
SET @empName=N'Phan Lệ Thuyền'
SET @empSalary=90000
PRINT @empName + ' with salary is ' + CAST(@empSalary AS VARCHAR)
PRINT @empName + ' with salary is ' + CONVERT(VARCHAR, @empSalary)
```

Short introduction to T-SQL programming

2. Flow-control statement

- Statement Blocks: Begin…End
- C onditional Execution:
  - ✓IF ... ELSE Statement
  - ✓C ASE ... WHEN
- Looping: WHILE Statement
- Error handling:
  - ✓@@ERROR
  - ✓TRY … C ATCH

Short introduction to T-SQL programming

Statement Blocks: BEGIN…END

- Groups of statements used with IF, WHILE, and C ASE statements must be grouped together using the BEGIN and END statements. Any BEGIN must have a corresponding END in the same batch.

```
BEGIN
   { sql_statement | statement_block }
END
```

```
BEGIN
   SELEC  T product_id, product_name
   FROM  production.products
   WHERE list_price > 100000;
END
```

Short introduction to T-SQL programming

IF ... ELSE Statement

- Evaluate a Boolean expression and branch execution based on the result

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	90000	2	Trần Phú
3	1003	Everest	Brad	0	71000	NULL	Hoàng Hoa Thám
4	1004	Smith	John	0	90000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	1	Trần Phú
6	1006	Brow	Hilary	1	90000	NULL	An Dương Vương

Short introduction to T-SQL programming

```
DECLARE @nhanvien_salary INT, @ten char(5), @bonus int, @total int
SELECT@nhanvien_salary=salary, @ten = first_name
FROMemployees
WHEREId_emp=1001
IF @nhanvien_salary < 900000
  begin
  set @bonus=10000
  set @total= @bonus + @nhanvien_salary
  PRINT N'Lương bạn '+ @ten + CAST(@total AS VARCHAR)
  end
else
  begin
  set @bonus=20
  set @total= @bonus + @nhanvien_salary
  PRINT N'Lương bạn'+ @ten + convert(varchar,@total)
  end
```

Short introduction to T-SQL programming

C ASE ... WHEN Statement

- Syntax

```
CASE input_expression
     WHEN  when_expression  THEN result_expression
     [WHEN when_expression  THEN result_expression…n]
     [ELSE else_result_expression ]
END
```

We use C ASE in statements such as SELEC T, UPDATE, DELETE and SET, and in clauses such as SELECT list, IN, WHERE, ORDER BY, and HAVING

Short introduction to T-SQL programming

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	NULL	Nguyễn Thái Học
2	1002	Anderson	Jane	0	57500	NULL	Trần Phú
3	1003	Everest	Brad	0	71000	NULL	Hoàng Hoa Thám
4	1004	Horvath	Jack	0	42000	NULL	Trần Phú
5	1005	Horvat	Helen	1	87000	NULL	Trần Phú
6	1006	Brow	Hilary	1	90000	NULL	An Dương Vương

```
DECLARE@emp_sex int, @str NVARCHAR(30)
SELECT @emp_sex=sex
FROM employees
where Id_emp=1005
SET @str=
CASE @emp_sex
WHEN 1 THEN N'giới tính nữ'
WHEN 0 THEN N'Giới tính nam'
ELSE N'Chưa đăng ký giới tính'
END
PRINT @str
```

Short introduction to T-SQL programming

```
DECLARE @womanDayBonus DECIMAL
DECLARE @staff_name NVARCHAR(50)
SELECT @staff_name = empName,@womanDayBonus=
  CASE empSex
 WHEN 'F' THEN 500
 WHEN 'M' THEN 0
  END
FROM tblEmployee
WHERE empSSN=30121050004
PRINT  @staff_name + N' được thưởng ‘ + convert(varchar,@womanDayBonus);
```

Short introduction to T-SQL programming

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	57500	1	Trần Phú
3	1003	Everest	Brad	0	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	0	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương

```
SELECT Id_emp,dept_id,last_name,first_name,
CASE
WHEN dept_id = 1 THEN N'Tập trung chiến dịch truyền thông tháng 12'
WHEN dept_id = 2 THEN N'Du lịch miền tây sông nước'
WHEN dept_id = 3 THEN N'Tập trung chiến dịch truyền thông tháng 12'
ELSE N'Đi hội thảo khoa học tại Mỹ'
END AS Bonus_year
FROM employees
ORDER BY first_name
```

	Id_emp	dept_id	last_name	first_name	Bonus_year
1	1003	2	Everest	Brad	Du lịch miền tây sông nước
2	1005	3	Horvat	Helen	Tập trung chiến dịch truyền thông tháng 12
3	1006	4	Brow	Hilary	Đi hội thảo khoa học tại Mỹ
4	1004	2	Horvath	Jack	Du lịch miền tây sông nước
5	1002	1	Anderson	Jane	Tập trung chiến dịch truyền thông tháng 12
6	1001	1	Smith	John	Tập trung chiến dịch truyền thông tháng 12

Short introduction to T-SQL programming

WHILE Statement : repeats a statement or block of statements as long as a specified condition remains true

- Syntax

```
WHILE boolean_expression
    SQL_statement | block_of_statements
    [BREAK]
    SQL_statement | block_of_statements
    [CONTINUE]
```

Short introduction to T-SQL programming

```
DECLARE @factorial INT, @n INT
SET @n=5
SET @factorial=1
WHILE (@n > 1)
BEGIN
   SET @factorial = @factorial*@n
   SET @n = @n - 1
END
PRINT @factorial
```

Short introduction to T-SQL programming

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	0	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	0	57500	1	Trần Phú
3	1003	Everest	Brad	0	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	0	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương

	Id_emp	last_name	first_name	Lương
1	1002	Anderson	Jane	57500
2	1004	Horvath	Jack	42000
3	1006	Brow	Hilary	90000

	Id_emp	last_name	first_name	Lương
1	1002	Anderson	Jane	115000
2	1004	Horvath	Jack	84000
3	1006	Brow	Hilary	180000

	Id_emp	last_name	first_name	Lương
1	1002	Anderson	Jane	172500
2	1004	Horvath	Jack	126000
3	1006	Brow	Hilary	270000

	Id_emp	last_name	first_name	Lương
1	1002	Anderson	Jane	230000
2	1004	Horvath	Jack	168000
3	1006	Brow	Hilary	360000

```
Declare @max int, @num int;
select @max=count(*)
from employees
set @num=1
while @num<=@max
   Begin
    select Id_emp, last_name, first_name, salary*@num as 'Lương'
    from employees
    where Id_emp % 2 = 0
    set @num=@num+1;
 end;
```

Short introduction to T-SQL programming

Handling error using @@ERROR function

- The @@ERROR system function returns 0 if the last Transact-SQL statement executed successfully; if the statement generated an error, @@ERROR returns the error number.

Short introduction to T-SQL programming

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	1	1	NULL
8	1008	Wilman	Kala	0	1	3	NULL

```
UPDATE employees
    SET salary = 50000
    WHERE salary=1 ;
IF @@ERROR = 547
    BEGIN
    PRINT N'A check constraint violation occurred.';
    END
```

Short introduction to T-SQL programming

Handling error using TRY … C ATCH

- was introduced with SQL Server 2005. Statements to be tested for an error are enclosed in a BEGIN TRY…END TRY block. A CATCH block immediately follows the TRY block, and error-handling logic is stored here

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

Short introduction to T-SQL programming

The CATCH:
Inside the CATCH block, the functions to get detailed information on the error that occurred:

- ERROR_LINE() returns the line number on which the exception occurred.
- ERROR_MESSAGE() returns the complete text of the generated error message.
- ERROR_PROCEDURE() returns the name of the stored procedure or trigger where the error occurred.
- ERROR_NUMBER() returns the number of the error that occurred.
- ERROR_SEVERITY() returns the severity level of the error that occurred.
- ERROR_STATE() returns the state number of the error that occurred.

Short introduction to T-SQL programming

```
BEGIN TRY
INSERT INTO employees(Id_emp,last_name, first_name)
VALUES(1007,'White','Clover Markets'),
      (1008,'Wilman','Kala');
END TRY
BEGIN CATCH
PRINT ERROR_MESSAGE()
PRINT ERROR_LINE()
END CATCH;
```

Short introduction to T-SQL programming

```
BEGIN TRY
DECLARE @num int;
SELECT @num=217/0;
END TRY
BEGIN CATCH
PRINT 'Error occurred, unable to devide by 0'
END CATCH;
```

Short introduction to T-SQL programming

```
BEGIN TRY
SELECT 217/0;
END TRY
BEGIN CATCH
SELECT ERROR_NUMBER() AS ErrorNumer,
       ERROR_SEVERITY() AS ErrorSeverity,
       ERROR_LINE() AS ErrorLine,
       ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

(No column name)

	ErrorNumer	ErrorSeverity	ErrorLine	ErrorMessage
1	8134	16	2	Divide by zero error encountered.

Branching Statements

- If-statement nested within the else-clause

```
DECLARE @Number INT;
SET @Number = 50;
IF @Number > 100
   PRINT 'The number is large.';
ELSE
   BEGIN
      IF @Number < 10
      PRINT 'The number is small.';
   ELSE
      PRINT 'The number is medium.';
   END ;
```

Branching Statements

```
BEGIN
    DECLARE @x INT = 10,
            @y INT = 20;

    IF (@x > 0)
    BEGIN
        IF (@x < @y)
            PRINT 'x > 0 and x < y';
        ELSE
            PRINT 'x > 0 and x >= y';
    END
END
```

Queries in T-SQL programming

Several ways that select-from-where queries are used in PSM(Persistent Stored Modules)

- Subqueries can be used in conditions, or in general, any place a subquery is legal in SQL
- Queries that return a single value can be used as the right sides of assignment statements
- A single-row select statement is a legal statement in PSM
- We can declare and use a cursor for embedded SQL

The three – tier Architecture

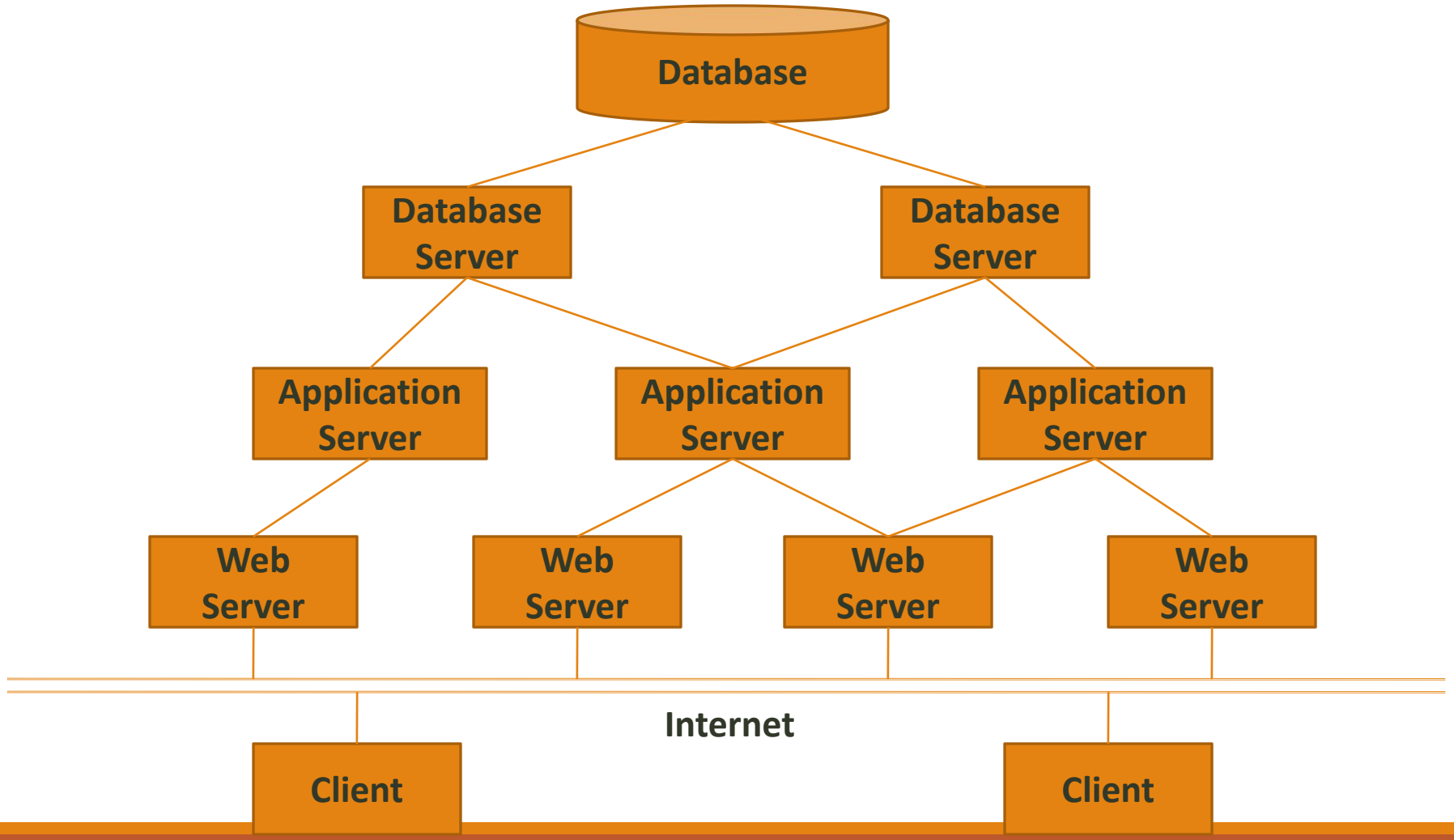
A very common architecture for large database installation

Three different, interacting functions

- Web servers
- Application servers
- Database servers

The processes can run on the same processor or on a large number of processors

The three – tier Architecture



Fpt University

The Webserver Tier

The webserver processes manage the interactions with the user

When a user makes contact, a webserver response to the request, and the user becomes a *client* of this webserver process

Fpt University

The Application Tier

Turning data from the database into a response to the request that it receives from the webserver

One webserver process invoke many application-tier processes, which can be on one or many different machines

The application-tier processes execute the business logic of the organization operating the database

Fpt University

The Database Tier

There can be many processes in the database tier

The processes can be in one or many machines

The database tier executes queries that are requested from the application tier

Advantages of using Stored Procedure

Using stored procedures offer numerous advantages over using SQL statements. These are:

- Reuse of C ode
- Maintainability
- Reduced Client/Server Traffic
- Precompiled Execution
- Improved Security

Stored procedure - Introduction

Persistent, Stored Modules (SQL/PSM)

Help to write procedures in a simple, general-purpose language and to store them in the database

We can use these procedures in SQL queries and other statements to perform computations

Each commercial DBMS offers its own extension of PSM

Creating Stored Procedure under MS SQL Server

C reate stored procedure:

```
C REATE PROC EDURE procedure_name
    [ {@parameter1 data_type} [= default] [OUTPUT] ]
    [ {@parameter2 data_type} [= default] [OUTPUT] ]
    ...
AS
    sql_statement1
    sql_statement2
```

C alling stored procedure

```
EXEC  procedure_name [argument1,
  argument2, …]
```

Creating PSM Functions and Procedures

```
CREATE PROCEDURE SelectAllEmployees
AS
SELECT *
FROM employees
GO;

EXEC SelectAllEmployees
```

Creating PSM Functions and Procedures

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	NULL
8	1008	Wilman	Kala	0	50000	3	NULL

```
CREATE PROCEDURE Selectemployess
@addr nvarchar(30)
AS
SELECT *
FROM employees
WHERE address = @addr
GO;

exec Selectemployess @addr = 'Hoàng Hoa Thám';
```

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám

Creating PSM Functions and Procedures

```
CREATE PROCEDURE Selectemployess
@addr nvarchar(30)
AS
SELECT *
FROM employees
WHERE address = @addr
GO;

exec Selectemployess @addr =N'Trần Phú';
```

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1002	Anderson	Jane	1	57500	1	Trần Phú
2	1004	Horvath	Jack	1	42000	2	Trần Phú
3	1005	Horvat	Helen	1	87000	3	Trần Phú
4	1007	White	Clover Markets	0	50000	1	Trần Phú
5	1008	Wilman	Kala	0	50000	3	Trần Phú

Creating PSM Functions and Procedures

```
CREATE PROCEDURE Selemployee
@addr nvarchar(30),
@code_emp int
AS
SELECT *
FROM employees
WHERE address = @addr AND Id_emp = @code_emp
GO;

exec Selemployee @addr =N'Trần Phú', @code_emp = 1002;
```

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1002	Anderson	Jane	1	57500	1	Trần Phú

Creating PSM Functions and Procedures

```
CREATE PROCEDURE ins_Emp
@Lname char(50),
@Fname char(50),
@EId int
AS
INSERT INTO Employees (Id_emp, last_name,first_name)
  VALUES (@EId,@Lname,@Fname)
  go

EXEC ins_Emp @EId=1009,@Lname='Thuyen',@Fname='Phan'
```

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	Trần Phú
8	1008	Wilman	Kala	0	50000	3	Trần Phú
9	1009	Thuyen	Phan	NULL	NULL	NULL	NULL

Function in SQL Server

- System Defined Function
- User Defined Function
  - *Scalar functions*
  - *Inline table-valued functions*
  - *Multi-statement table-valued functions*

Scalar functions

Scalar function in SQL Server always accepts parameters, either single or multiple and returns a single value.

```
CREATE FUNCTION dbo.function_name (paramet
er_list)
RETURNS data_type AS
BEGIN
    statements
    RETURN value
END
```

Calling a Function in SQL Server

```
SELECT dbo.<FunctionName>(Value)
```

Scalar functions

```
CREATE FUNCTION sum
(@a INT,
 @b INT)
RETURNS INT
AS
BEGIN
    RETURN @a * @b;
END;
SELECT dbo.sum(25, 500) AS sum_result;
```

	sum_result
1	12500

Scalar functions

```
CREATE FUNCTION f_Bonus
(@f_Salary INT)
RETURNS nvarchar(50)
AS
BEGIN
DECLARE @tour NVARCHAR(50);
    IF @f_Salary < 60000
      SET @tour = N'Được tham gia du lịch';
    ELSE
     SET @tour = N'Tham gia chạy deadline dự án của công ty';
RETURN @tour;
END;
SELECT dbo.f_Bonus(60000) AS Tour;
```

	Tour
1	Tham gia chay deadline dự án của công ty

Inline Table-valued Function

```
-- Syntax for creating an Inline table value function
CREATE FUNCTION Function_Name
(
    @Param1 DataType,
    @Param2 DataType,
    @ParamN DataType
)
RETURNS TABLE
AS
RETURN (Select_Statement)

-- Syntax for calling an Inline table value function
SELECT * FROM Function_Name (VALUE)
```

Inline Table-valued Function

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	Trần Phú
8	1008	Wilman	Kala	0	50000	3	Trần Phú
9	1009	Thuyen	Phan	NULL	NULL	NULL	NULL

Id_emp	last_name	first_name
1005	Horvat	Helen
1006	Brow	Hilary

```
CREATE FUNCTION salary_80 (
    @salary_max INT)
RETURNS TABLE
AS
RETURN
    SELECT
        Id_emp,
        last_name,
        first_name
    FROM
        employees
    WHERE
        salary >= @salary_max;
```

```
SELECT *
FROM salary_80 (80000);
```

Inline Table-valued Function

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	Trần Phú
8	1008	Wilman	Kala	0	50000	3	Trần Phú
9	1009	Thuyen	Phan	NULL	NULL	NULL	NULL

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1007	White	Clover Markets	0	50000	1	Trần Phú
2	1008	Wilman	Kala	0	50000	3	Trần Phú

```
CREATE FUNCTION FN_Getsex
(@sex_emp bit)
RETURNS TABLE
AS
RETURN (SELECT * FROM employees WHERE sex = @sex_emp)
go
```

```
SELECT *
FROM FN_Getsex(0)
```

Inline Table-valued Function

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	Trần Phú
8	1008	Wilman	Kala	0	50000	3	Trần Phú
9	1009	Thuyen	Phan	NULL	NULL	NULL	NULL

	Id_emp	last_name	first_name	dept_name
1	1007	White	Clover Markets	phát tri?n
2	1008	Wilman	Kala	n?i dung

```
CREATE FUNCTION FN_Emp
(@sex_age bit)
RETURNS TABLE
AS
RETURN (
       SELECT Id_emp,last_name,first_name, dept_name
    FROM employees Emp
    JOIN department Dept on Emp.dept_id = Dept.dept_id
    WHERE sex = @sex_age)


SELECT *
FROM FN_Emp(0)
```

Multi-Statement Table Valued Function

To define a multi-statement table-valued function, we use a table variable as the return value. Inside the function, we execute one or more queries and insert data into this table variable.

```
-- Syntax For Creating Multi Statement Table valued Function
CREATE FUNCTION FunctionName
(
        @Param1 DataType,
        @Param2 DataType,
        @Paramn DataType
)
RETURNS @TableVariable TABLE (Column_Definitions)
WITH FunctionAttribute
AS
BEGIN
        FunctionBody
        RETURN
END
```

Multi-Statement Table Valued Function

```
CREATE FUNCTION in_emp()
RETURNS @Employee TABLE
(Id_emp int,
        last_name char(50),
        first_name char(50),
        sex bit,
        salary int) AS
BEGIN
    INSERT INTO @Employee
    SELECT E.Id_emp,E.last_name,E.first_name,E.sex,E.salary FROM
employees E;
    UPDATE @Employee SET  last_name = 'Smith' WHERE Id_emp = 1003;
    RETURN
END


SELECT *
FROM in_emp();
```

	Id_emp	last_name	first_name	sex	salary
1	1001	Smith	John	1	62000
2	1002	Anderson	Jane	1	57500
3	1003	Smith	Brad	1	71000
4	1004	Horvath	Jack	1	42000
5	1005	Horvat	Helen	1	87000
6	1006	Brow	Hilary	1	90000
7	1007	White	Clover M...	0	50000
8	1008	Wilman	Kala	0	50000
9	1009	Thuyen	Phan	N...	NULL

Multi-Statement Table Valued Function

```
CREATE FUNCTION count_addr (@address nvarchar(30))
RETURNS @new_table TABLE (DiaChi nvarchar(30), SoLuong int)
AS
BEGIN
DECLARE @count int = 0
SELECT @count = count(*) FROM (
        SELECT Id_emp
        FROM employees
WHERE address = @address
    ) AS Temp
    INSERT INTO @new_table VALUES (@address, @count)
    RETURN
END



SELECT *
FROM count_addr (N'Trần Phú')
```

	Id_emp	last_name	first_name	sex	salary	dept_id	address
1	1001	Smith	John	1	62000	1	Nguyễn Thái Học
2	1002	Anderson	Jane	1	57500	1	Trần Phú
3	1003	Everest	Brad	1	71000	2	Hoàng Hoa Thám
4	1004	Horvath	Jack	1	42000	2	Trần Phú
5	1005	Horvat	Helen	1	87000	3	Trần Phú
6	1006	Brow	Hilary	1	90000	4	An Dương Vương
7	1007	White	Clover Markets	0	50000	1	Trần Phú
8	1008	Wilman	Kala	0	50000	3	Trần Phú
9	1009	Thuyen	Phan	NULL	NULL	NULL	NULL

	DiaChi	SoLuong
1	Trần Phú	5

Triggers

Triggers differ from the other constraints

- Triggers are only awakened when certain events occur (INSERT, UPDATE, DELETE)
- One awakened, the trigger tests a condition.
  - If the condition does not hold, trigger do nothing to response to occurred event
  - If the condition is satisfied, the action associated with trigger is performed by the DBMS

Fpt University

Why uses triggers

Triggers can implement business rules

- E.g. creating a new Order when customer checkout a shopping cart (in online ecommerce websites)

Triggers be used to ensure data integrity

- E.g. Updating derived attributes when underlying data is changed, or maintaining summary data

Triggers in SQL

Some principle features of triggers

- The check of trigger's condition and the action of the trigger may be executed either on the state of database that exists before the triggering event is itself executed or on the state that exists after the triggering event is executed
- The condition and action can refer to both old and/or new values of tuples that were updated in the triggering event
- It is possible to define update events that are limited to a particular attribute or set of attributes
- Trigger executes either
  - Once for each modified tuple
  - Once for all the tuples that are changed in one SQL statement

Implement Trigger with T-SQL

Create Trigger on MS SQL Server syntax

```
CREATE TRIGGER trigger_name ON TableName
    {AFTER {[DELETE] [,] [INSERT] [,] [UPDATE]}
 AS
    sql_statement 1
    sql_statement 2
        …
```

Disable a trigger

```
DISABLE  TRIGGER <trigger_name> ON <table_name>
```

Enable a trigger

```
ENABLE TRIGGER <trigger_name> ON <table_name>
```

Implement Trigger with T-SQL

```
Create TRIGGER deleteStudent1
on Student
For Delete
As
Begin
declare @Id int;
Select @Id=id from deleted;
rollback transaction;
update Student set Deleted=1 where id=@id;
end
Go
```

Using Cursor in MS SQL Server

1. Declare cursor

   DEC LARE cursor_name C URSOR FOR SELEC T Statement

2. Open cursor

   OPEN cursor_name

3. Loop and get values of each tuple in cursor with FETC H statement

   FETC H NEXT | PRIOR | FIRST | LAST
   FROM cursor_name INTO @var1, @var2

4. Using @@FETC_H_STATUS to check fetch status. The 0 value mean FETC H statement was successful.

5. C LOSE cursor_name

6. DEALLOC ATE cursor_name

Example

```
 DECLARE @_lname char(15), @_fname char(15)
  DECLARE Employee_Cursor CURSOR FOR
  SELECT last_name, first_name
  FROM Employees
  OPEN Employee_Cursor
  --Use the FETCH statement to get each row of data from the record
set
  --get data put into variable
  FETCH NEXT FROM Employee_Cursor INTO @_lname, @_fname
  --WHILE by checking the global variable @@FETCH_STATUS (=0 means
success)
  WHILE @@FETCH_STATUS = 0
    BEGIN
      PRINT 'Employee:' + @_fname + ' ' + @_lname
      FETCH NEXT FROM Employee_Cursor INTO @_lname, @_fname
    END
CLOSE Employee_Cursor
DEALLOCATE Employee_Cursor
```