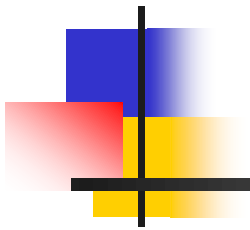


Session 4

Transact-SQL Language

Session Objectives

- Liệt kê các tính năng chính của Transact-SQL
- Hiểu được biến, kiểu dữ liệu, chú thích
- Tóm tắt các hàm, biểu thức trong Transact-SQL
- Giải thích ngôn ngữ định nghĩa dữ liệu và các lệnh của ngôn ngữ định nghĩa dữ liệu
- Giải thích ngôn ngữ thao tác dữ liệu và các lệnh của ngôn ngữ thao tác dữ liệu
- Giải thích ngôn ngữ điều khiển dữ liệu và các lệnh của ngôn ngữ điều khiển dữ liệu
- Giải thích cách thực thi câu lệnh Transact-SQL:
  - Một câu lệnh đơn lẽ
  - Tập lệnh
  - Scripts
- Liệt kê và giải thích những tính năng nâng cao của Transact-SQL

Transact-SQL Language -1

- Structured Query Language (SQL) là một ngôn ngữ rất phổ dụng trong lĩnh vực CSDL.
- Microsoft xây dựng Transact-SQL dựa trên ngôn ngữ vấn tin có cấu trúc chuẩn.
- Nó cung cấp một ngôn ngữ bao hàm toàn diện để định nghĩa bảng, chèn, xóa, thay đổi và truy cập dữ liệu trong bảng.
- Transact-SQL là một ngôn ngữ mạnh, nó hỗ trợ các tính năng khác như: kiểu dữ liệu, đối tượng tạm thời, thủ tục lưu trữ và thủ tục hệ thống.
- Nó còn cho phép chúng ta định nghĩa đối tượng con trỏ, khai báo biến, cấu trúc rẽ nhánh, vòng lặp, bẫy lỗi và Transaction.

Transact-SQL Language -2

- Example for Transact-SQL statement:

```
SELECT * FROM Employee
```

- The statement retrieves all records from Employee table.

Variables in Transact-SQL - 1

- Biến là vùng nhớ trong bộ nhớ được đặt tên để chứa giá trị dữ liệu. Dữ liệu có thể truyền cho câu lệnh SQL bằng biến cục bộ. Tên biến cục bộ phải khai báo bắt đầu bằng ký hiệu @.
- Biến có thể phân thành 2 loại:
  - Local Variables (Biến cục bộ)
  - Global Variables (Biến toàn cục)

Variables in Transact-SQL - 2

- Local Variables
  - Trong Transact-SQL, biến cục bộ được khai báo và sử dụng tạm thời khi thực thi câu lệnh SQL.

Cú pháp:

```
DECLARE
{
@local_variable [AS] data_type
}
```

where,

`@local_variable` specifies the name of a variable.

Variable names must begin with the '@'.

`data_type` is a system or user-defined data type.

Variables in Transact-SQL - 3

- Câu lệnh SET hoặc SELECT dùng để gán giá trị cho biến.

  Cú pháp:

```
SET @local_variable = value
    Hoặc
SELECT @local_variable = value
```

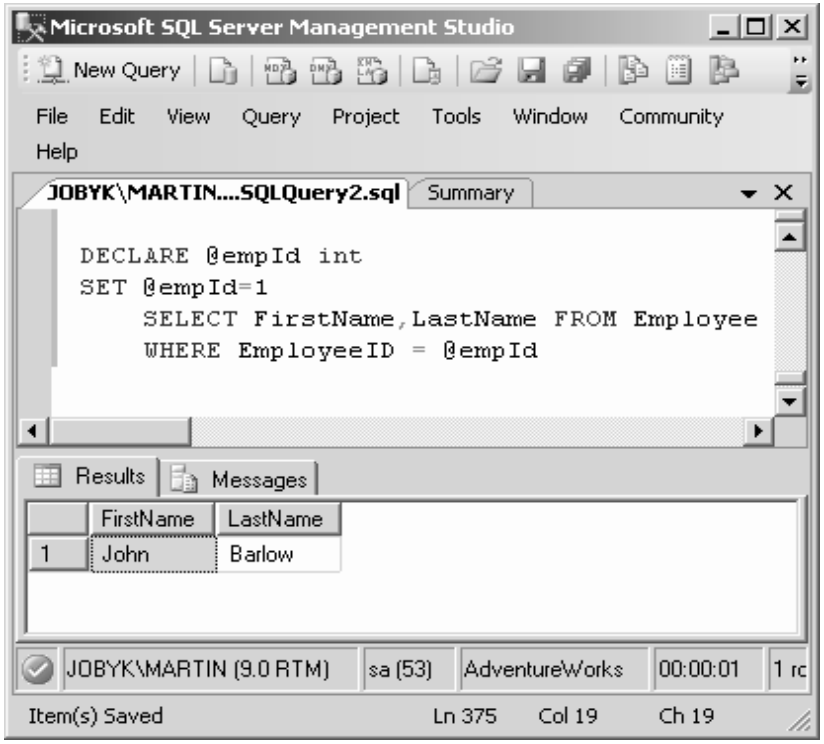
Variables in Transact-SQL - 4

Ví dụ: DECLARE @empID int

SET @empID = 1

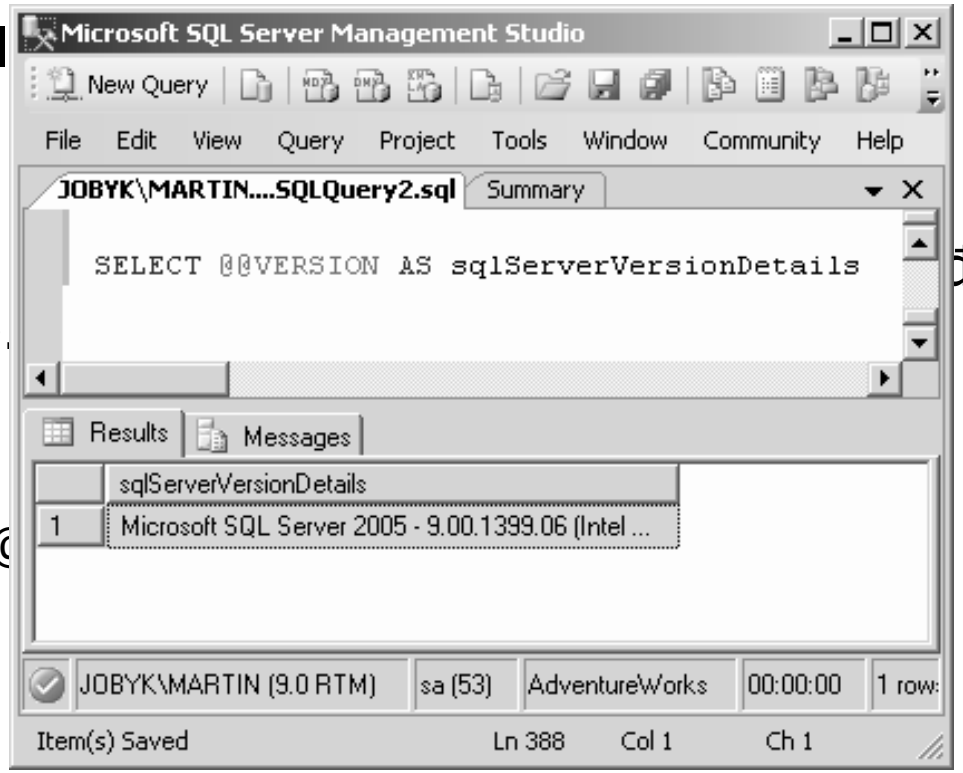
SELECT FirstName, LastName FROM Employee

WHERE EmployeeID = @empID



Variables in Transact-SQL - 5

- Global Variabl
  - Biến toàn
  - Biến toàn đầu bởi hai ký hiệu @
- Ví dụ:
  - SELECT @@

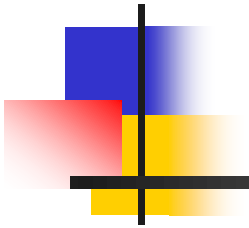


Data Types in Transact-SQL - 1

- Kiểu dữ liệu là thuộc tính định nghĩa loại dữ liệu mà đối tượng có thể chứa.
- Transact-SQL bao gồm nhiều kiểu dữ liệu chẳng hạn như: varchar, text, int,....
- Các đối tượng dưới đây sử dụng kiểu dữ liệu:
  - **Cột trong bảng**
  - **Biến**
  - Tham số trong thủ tục lưu trữ
  - Hàm trả về giá trị
  - Thủ tục lưu trữ trả về giá trị

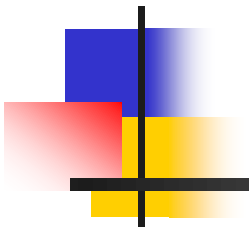
Data Types in Transact-SQL - 2

Data Types in SQL Server 2005			
tinyint	real	datetime	varbinary
smallint	char	smalldatetime	uniqueidentifier
int	nchar	image	numeric
bigint	varchar	money	timestamp
bit	nvarchar	smallmoney	sql_variant
decimal	text	xml	table
float	ntext	cursor	binary



Comments in Transact-SQL - 1

- Microsoft SQL Server hỗ trợ hai loại chú thích sau:
  - -- chú thích một dòng
  - /* . . . */ chú thích nhiều dòng



Data Definition Language

- Data Definition Language (DDL), được dùng để định nghĩa (xây dựng), thay đổi hoặc xóa cấu trúc của các đối tượng CSDL, chẳng hạn: bảng, view, trigger, thủ tục lưu trữ, …
- Câu lệnh DDL:
  - CREATE object_name
  - ALTER object_name
  - DROP object_name

  Với object_name: TABLE, VIEW, …

CREATE DATABASE – Example 1

```
CREATE DATABASE NhanSu
ON PRIMARY
 (NAME = NhanSu_DATA,
  FILENAME = 'C:\SQL\Tdata\NhanSu.mdf'
 )
LOG ON
 (NAME = NhanSu_LOG,
  FILENAME = 'c:\sql\Tdata\NhanSu.ldf'
 )
```

CREATE DATABASE - Example 2

```
create database BookStores
on primary (
name=‘BookStores_data',
filename='C:\inetpub\wwwroot\project\database\ BookStores.mdf',
size=2MB,
maxsize=50MB,
filegrowth=10%
)
log on (
    name=' BookStores _log',
    filename='C:\inetpub\wwwroot\project\database\
    BookStores.ldf',
    size=2MB,
    maxsize=50MB,
    filegrowth=10%
)
```

CREATE TABLE

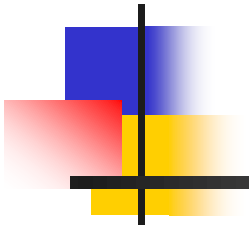
- CREATE TABLE statement
  - The CREATE TABLE statement creates a table in an existing database.

Syntax:

```
CREATE TABLE <Table_Name>
(Column_Name1 Data_Type, Column_Name2 Data_Type,...,
Column_NameN Data_Type )
```

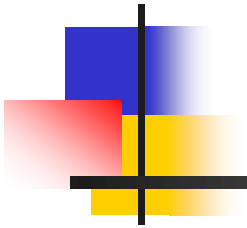
Example:

```
CREATE TABLE Contacts
(MailID varchar(20),
 Name text,
 TelephoneNumber int)
```



Data Types in SQL

- Characters:
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length
- Numbers:
  - BIGINT, INT, SMALLINT, TINYINT
  - REAL, FLOAT -- differ in precision
  - MONEY
- Times and dates:
  - DATETIME
  - TIMESTAMP
- Binary objects (such as pictures, sound, etc.)
  - BLOB -- stands for "Binary Large OBject"
- Others... All are simple



Column attributes in SQL

- Keys:
  - PRIMARY KEY -- primary key of the table
  - KEY -- foreign key (will be indexed)
  - INDEX -- field to be indexed for fast search
- Null values:
  - NOT NULL -- field must be filled in
- Default value:
  - DEFAULT '*value*' -- value to be used if user gives none

Column Nullability

- The nullability feature of a column determines whether rows in the table can contain a null value for that column.
- Nullability of a column can be defined either when creating a table or modifying a table.

- NULL keyword indicates that null values are allowed in the column.
- NOT NULL keywords indicate that null values are not allowed in columns.

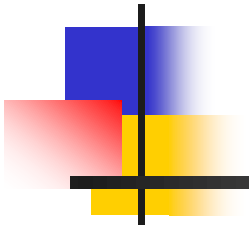
Example:

```
CREATE TABLE StoreDetails
(StoreID int NOT NULL, Name varchar(40) NULL)
```

DEFAULT Definition -1

- A DEFAULT definition can be given for a column to assign it a default value if no value is given at the time of creation.

- A DEFAULT definition for a column can be created at the time of table creation or added at a later stage to an existing table.



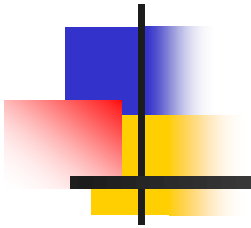
DEFAULT Definition - 2

Example:

```
CREATE TABLE StoreProduct( ProductID int NOT NULL, Name
varchar(40) NOT NULL, Price money NOT NULL DEFAULT (100))
```

Example:

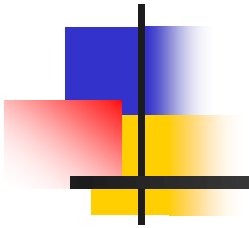
```
INSERT INTO StoreProduct (ProductID, Name) VALUES (111,
'Rivets')
```



DEFAULT Definition -3

DEFAULT definitions cannot be created on columns defined with:

- A timestamp data type
- An IDENTITY or ROWGUIDCOL property
- An existing default definition or default object



IDENTITY Property -1

Syntax:

```
CREATE TABLE <table_name> (column_name data_type [ IDENTITY
[(seed_value, increment_value)]] NOT NULL )
```

where,

seed_value is a value from which identity values will start.

increment_value is an increment value by which to increase the identity value each time.

Example:

```
CREATE TABLE ContactPhone ( Person_ID int
IDENTITY(500,1) NOT NULL, MobileNumber bigint NOT NULL )
```

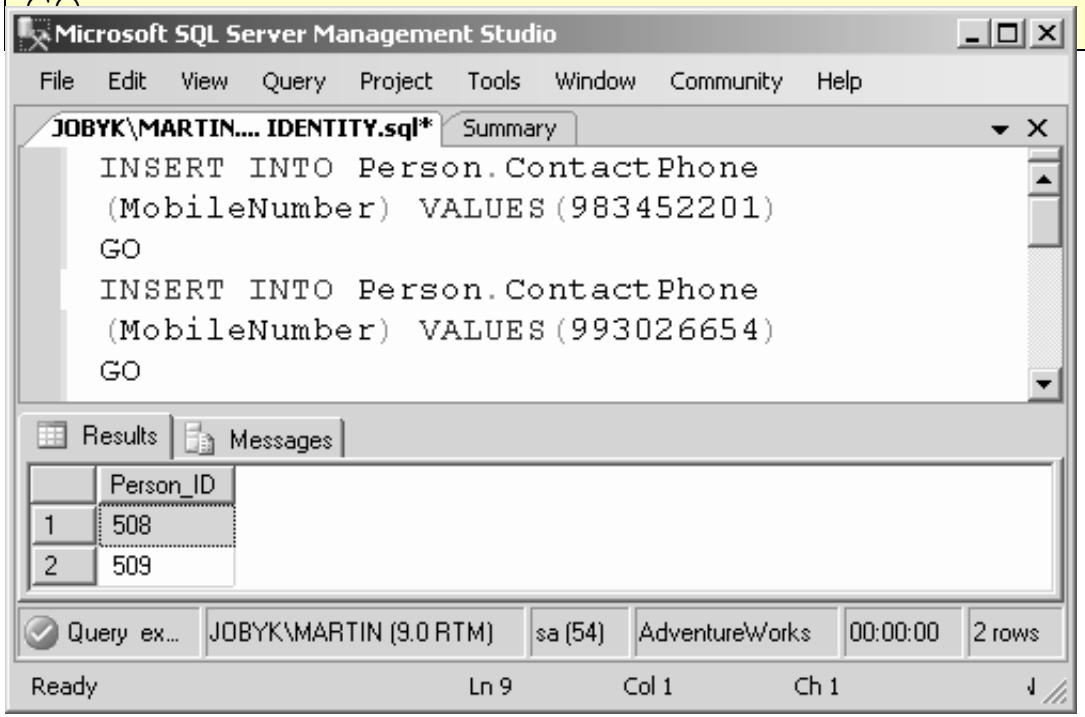
Here, in this example, 500 is the seed identity value and 1 is the increment.

IDENTITY Property -2

Later, if a SELECT statement is written to display only the Person ID, the output can be seen as shown.

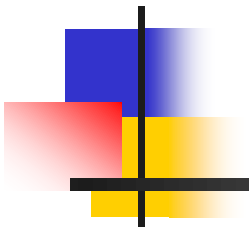
Example:

```
INSERT INTO ContactPhone VALUES (983452201)
GO
INSERT INTO ContactPhone VALUES (993026654)
GO
```



CREATE TABLE

- create table monHoc
  (maMH char(10) primary key,
   tenMH nvarchar(50))
- create table sinhVien
  (maSV char(10) primary key,
   tenSV nvarchar(30),
   ngaySinh smallDateTime,
   diachi nvarchar(50))



- create table ketQua
  (maSV char(10),
   maMH char(10),
   diem real,
   primary key (maSV,maMH),
   foreign key (MaSV) references sinhVien,
  foreign key (maMH) references monHoc
)

ALTER TABLE

- ALTER TABLE statement
  - The ALTER TABLE statement is used to modify a table definition by altering, adding, or dropping columns and constraints or by disabling or enabling constraints.

Syntax:

```
ALTER TABLE <Table_Name> ALTER COLUMN [<Column_name>
<New_data_type>]| ADD [<Column_name> <Data_Type>]
| DROP COLUMN [<Column _Name>]
```

where,

`<Table_Name>` is the name of the table to be modified.

ALTER COLUMN specifies the column to be changed or altered.

`<Column_Name>` names the column to be altered, added or dropped.

`<New_data_type>` is the new data type for the altered column.

`ADD` specifies a column to be added to the table.

`DROP COLUMN` specifies the column to be removed from the table.

ALTER TABLE example

1. Create

```
CREATE TABLE SinhVien(
        MaSV char(6) PRIMARY KEY,
        TenSV nvarchar(30) NOT NULL,
        Tuoi int
)
```

2. Add a column named DiaChi with data type nvarchar(50) into SinhVien table

```
ALTER TABLE SinhVien
ADD DiaChi nvarchar(50)
```

3. change from a column named TenSV to nvarchar(40)

```
ALTER TABLE SinhVien
ALTER COLUMN TenSV nvarchar(40)
```

4. delete a column named DiaChi

```
ALTER TABLE SinhVien
DROP COLUMN DiaChi
```

DROP DATABASE & TABLE

- DROP DATABASE/TABLE statement
  - The DROP DATABASE statement removes all tables
  - The DROP TABLE statement removes a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

Syntax:

```
DROP DATABASE/TABLE <Database_Name/Table_Name>
```

where,

`<Database_Name/Table_Name>` is the name of the Database or the table to be modified.

ATTN: All data will be destroyed along with the table. The system will not ask twice!

CONVERT

- Conversion Functions

Syntax:

```
CONVERT(datatype[(length)],expression [,style]
```

Where,

- *datatype: datatype is needed conversion*
- *Length : length of data*
- *Expression:* specifies name of column,const, function, variable or subquery
- *Style:* specifies *styles of date (dd/mm/yyyy hay mm/dd/yyyy)*

Example:

```
SELECT 'Employee ID: ' + convert(char(4), EmployeeID)
FROM Employee
```

Convert & Cast

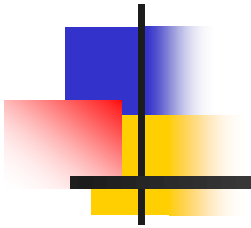
--dd/mm/yyyy

- select convert(char(12),getdate(),103)

  --dd/mm/yy
- select convert(char(12),getdate(),3)
- print cast(1234 as char(20))

--Jan 10 2009 10:58PM

- print cast(getdate() as char(20))
- print CONVERT(int, '5678')
- print CONVERT(decimal(5,2), '123.456789')



Data Manipulation Language

- Data manipulation language is used to select, insert, update, and delete data in the objects defined with DDL.
- There are three types of Data Manipulation Languages:
  - SELECT statement
  - INSERT statement
  - UPDATE statement
  - DELETE statement

INSERT (1)

The INSERT statement adds a new row to a table.

Syntax:

```
INSERT INTO <Table_name>
VALUES <Values>
```

where,

<Table_Name> is the name of the table in which row is to be inserted.
[INTO] is an optional keyword used between INSERT and the target table.
<Values> specifies the values for columns of the table.

Example:

```
USE AdventureWorks
INSERT INTO Contacts
VALUES('john@abc.com','John',5432677,5432678)
```

This statement adds a new row to the table Contacts.

INSERT (2)

- insert into **monHoc** values ('CF','Computer fundamentals')
- insert into **monHoc** values ('C','Elementary Programming with C')
- insert into **sinhVien** values ('01','Tran Van Cong', '1986-12-30','HN')
- insert into **sinhVien (maSV,tenSV)** values ('05','Tran Tho Ngo')

SELECT (1)

- SELECT statement
  - The SELECT statement retrieves rows from the database and enables the selection of one or many rows or columns from a table.

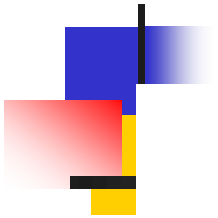
    Syntax:

```
SELECT <Column_name(s)> FROM <Table_name>
```

where,

`<Table_name>` is the name of the table from which rows are to be retrieved.

`<Column_name(s)>` is the name of the column or the names of the columns to be retrieved.



Selection Operations

What goes in the WHERE clause:

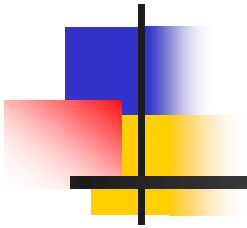
- x = y, x < y, x <= y, etc
  - For numbers, they have the usual meanings
  - For CHAR and VARCHAR: lexicographic ordering
  - For dates and times: chronological ordering
- Negation: NOT
- Pattern matching on strings: *s* LIKE *p* (also: NOT LIKE)
- Strings are enclosed in single quotes: '*string*'
- Multiple criteria can be joined by AND or OR
- Probe for empty fields with IS NULL
- Special operators: BETWEEN *x* AND *y*, IN(), …

Operators

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!	Not
BETWEEN	Between a range
LIKE	Search for an ordered pattern
IN	Within a range

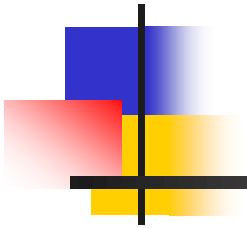
Wild Card characters

Wildcard	Description	Example
_	It will display a single character.	SELECT * FROM Person.Contact WHERE Suffix LIKE 'Jr_'
%	It will display a string of any length.	SELECT * FROM Person.Contact WHERE LastName LIKE 'B%'
[ ]	It will display a single character Within the range enclosed in the Brackets.	SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'C[AN][DY]'
[^]	It will display any single character not within the range enclosed in the Brackets.	SELECT * FROM Sales.CurrencyRate WHERE ToCurrencyCode LIKE 'A[^R][^S]'



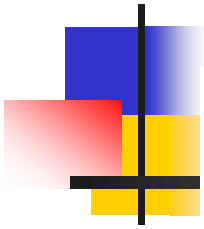
Select (2)

- select * from monHoc
- select * from monHoc where maMH like 'C%'
- select * from monHoc where maMH like '%F%'
- select maSV, tenSV, convert(char(12),ngaysinh,103) as ngaysinh from sinhVien where diachi in ('HP','HN')
- select maSV, tenSV,cast(ngaysinh as char(12)) from sinhVien where ngaySinh between '1986-1-1' and '1986-12-31'



Select (3)

- select a.TenSV, b.MaMH, b.diem from sinhVien a, ketQua b where a.maSV = b.MaSV
- select top 3 * from ketQua
- select top 25 percent * from ketQua
- select a.TenSV, b.MaMH, b.diem from sinhVien a inner join ketQua b on a.maSV = b.MaSV
- select a.TenSV, b.TenMH, c.diem from sinhVien a, monHoc b, ketQua c where a.maSV = c.MaSV and b.maMH = c.maMH



Simple SQL Query

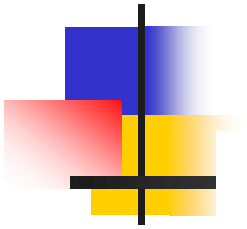
Product

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  PName, Price, Supplier
FROM    Product
WHERE   Price > 100
```

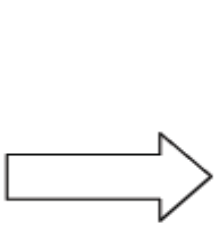
"selection" and "projection"

PName	Price	Supplier
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi



Eliminating Duplicates

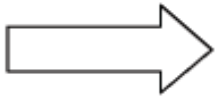
```
SELECT  category
FROM    Product
```



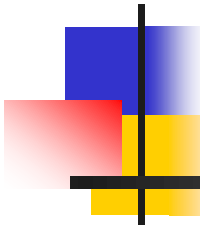
Category
Gadgets
Gadgets
Photography
Household

Compare to:

```
SELECT  DISTINCT category
FROM    Product
```



Category
Gadgets
Photography
Household



Ordering the Results

```
SELECT   pname, price, supplier
FROM     Product
WHERE    category='gadgets' AND price > 50
ORDER BY  price ASC
```

- ASC order in ascending order
- DESC in descending order
- List of several attributes can be given for nested ordering (e.g. order by price first and within all products of same price, order by supplier:

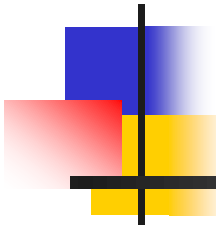
  ORDER BY price, supplier ASC)

Ordering the Results

```
SELECT   category
FROM     Product
ORDER BY Pname  ASC
```

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Category
Gadgets
Household
Gadgets
Photography



Joins in SQL

- Connect two or more tables:

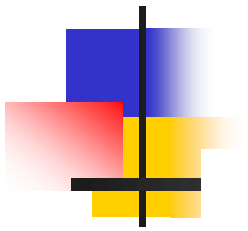
Product

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is the Connection between them ?



Joins

Product(pname,price,category,supplier)
Company(cname,stockPrice,country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Supplier=CName AND Country='Japan'
         AND Price <= 200
```

Join between Product and Company

Joins in SQL

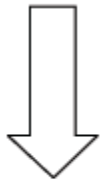
Product

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

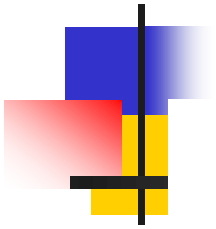
Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  PName, Price
FROM    Product, Company
WHERE   Supplier=CName AND Country='Japan'
        AND Price <= 200
```



PName	Price
SingleTouch	$149.99



Joins

Product(pname,price,category,supplier)
Company(cname,stockPrice,country)

Find all countries that manufacture some product in the 'Gadgets' category.

```
SELECT   Country
FROM     Product, Company
WHERE    Supplier=CName AND Category='Gadgets'
```

Joins in SQL

Product

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  Country
FROM    Product, Company
WHERE   Supplier=CName AND Category='Gadgets'
```

Country
USA
USA

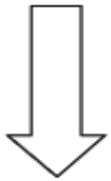
Why does USA appear twice ?

Internal joint table

RDBMS internally builds a joint table with ALL matches:

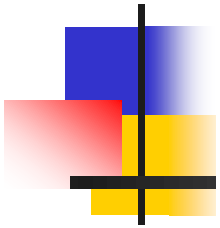
PName	Price	Category	Supplier	Cname	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

```
SELECT  Country
FROM    Product, Company
WHERE   Supplier=CName AND Category='Gadgets'
```



Country
USA
USA

First the tables are joint and entries duplicated
Then the selection is made!



Disambiguating Attributes

- Sometimes two relations have the same attr:

  Person(pname, address, worksfor)
  Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor=cname
```

Which address ?

```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor=Company.cname
```

Joining together multiple tables

- SELECT name,phone,zip FROM people, phonenumbers, address WHERE people.addressid=address.addressid AND people.id=phonenumbers.id;

People

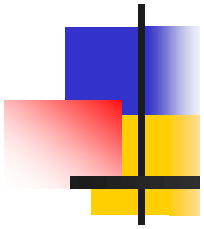
Id	Name	Addressid
1	Joe	1
2	Jane	2
3	Chris	3

PhoneNumbers

PhoneID	Id	Phone
1	1	5532
2	1	2234
3	1	3211
4	2	3421
5	3	2341
6	3	3211

Address

AddressID	Company	Address	Zip
1	ABC	123	12345
2	XYZ	456	14454
3	PDQ	789	14423



Renaming Columns

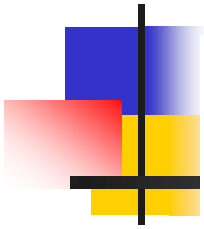
Product

PName	Price	Category	Supplier
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  Pname AS prodName, Price AS askPrice
FROM    Product
WHERE   Price > 100
```

Query with renaming

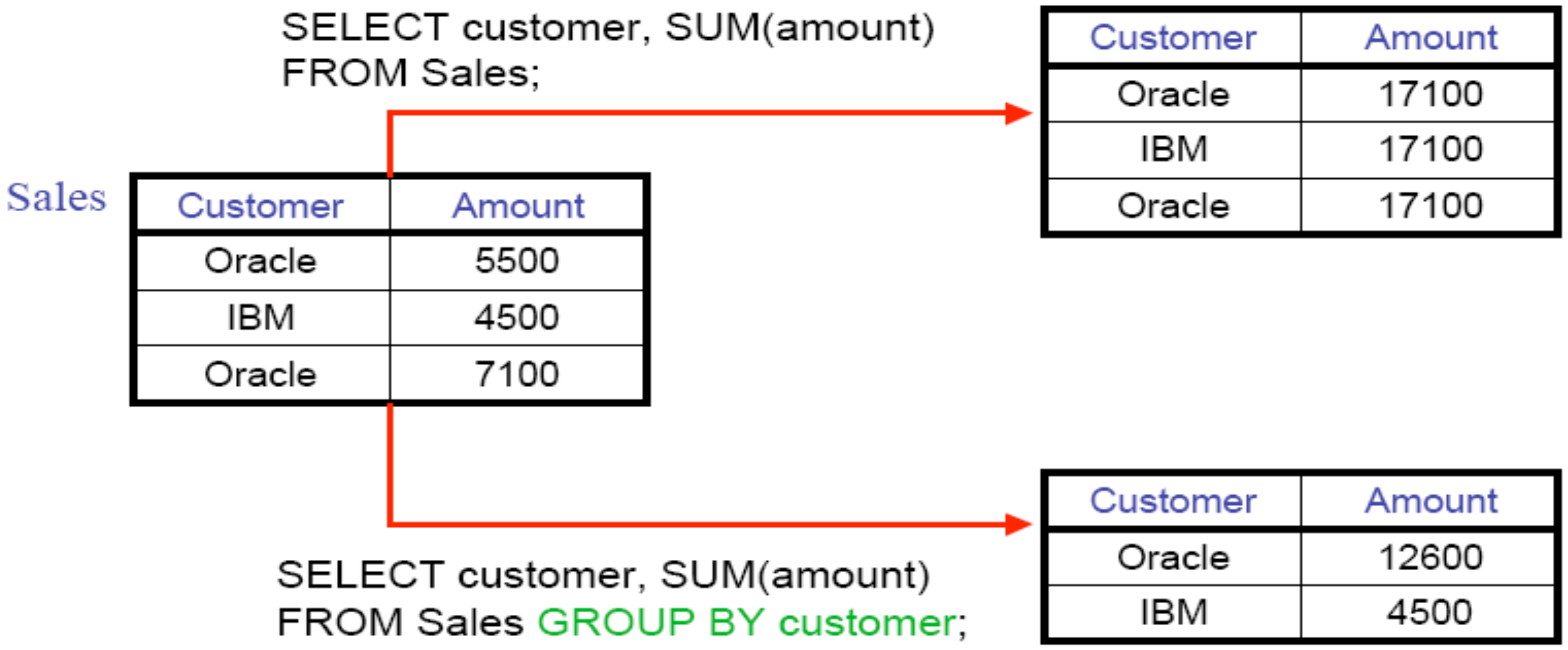
prodName	askPrice
SingleTouch	$149.99
MultiTouch	$203.99



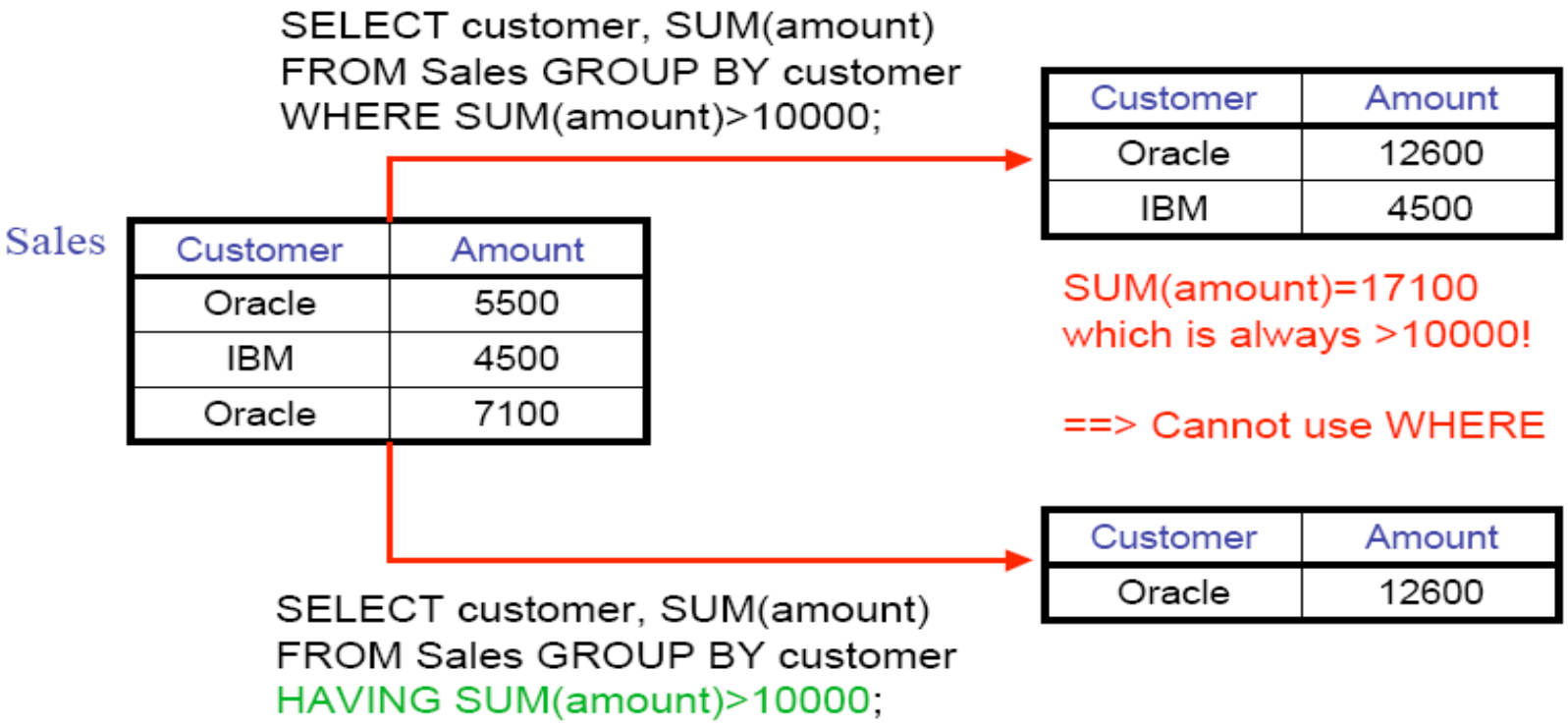
GROUP BY/HAVING

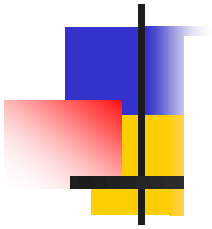
- The “GROUP BY” clause allows you to group results together before applying aggregation functions like:
  - AVG(), COUNT(), MAX(), MIN(), SUM()
  - COUNT DISTINCT
- HAVING allows you to search the grouped results

GROUP BY/HAVING



GROUP BY/HAVING





GROUP BY Examples

```
SELECT company,count(company)
   FROM contacts
   GROUP BY company;
```

- How many times does each company appear in the contacts table?

```
SELECT company,count(company)
   FROM contacts
   GROUP BY company
   HAVING count(company) > 5;
```

- Only show counts for companies that appear more than 5 times.

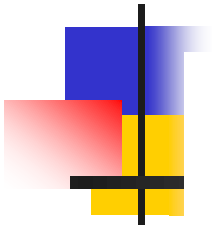
Group by

	maSV	maMH	diem	type
1	01	CF	12.5	W
2	01	C	17	W
3	01	C	13	P
4	01	HDJ	19	W
5	01	HDJ	14	P
6	02	CF	16	W
7	02	C	14	W
8	02	C	18	P
9	02	HDJ	22	W
10	02	HDJ	15	P

	maSV	maMH	diemTB
1	01	C	15
2	02	C	16
3	01	CF	12.5
4	02	CF	16
5	01	HDJ	16.5
6	02	HDJ	18.5

	maSV	maMH	diemTB
1	01	C	15
2	02	C	16
3	02	CF	16

- select maSV, maMH, avg(diem) as diemTB from ketQua group by maSV, maMH
- select maSV, maMH, avg(diem) as diemTB from ketQua where maMH like 'C%' group by maSV, maMH having avg(diem)>14 order by maSV, maMH



Nested SELECTs

- The WHERE subclause of a SELECT statement can contain another SELECT statement:

```
SELECT * FROM contacts
      WHERE zip NOT IN (
            SELECT zip FROM address
                  WHERE state='NY'
            );
```

(Select all contacts that have a ZIP code outside NY state.)

Update (1)

- UPDATE statement
  - The UPDATE statement modifies the data in the table.

Syntax:

```
UPDATE <Table_name>
SET <Column_Name = Value>
[WHERE <Search condition>]
```

where,

<Table_Name> is the name of the table in which row is to be inserted.

<Column_Name> is the name of the column in the table in which record is to be updated.

<Value> specifies the new value for the modified column.

<Search condition> specifies the condition to be met for the rows to be deleted.

Update (2)

- create table monHoc (maMH char(10) primary key, tenMH nvarchar(50), wtest bit,ptest bit)
- insert into monHoc (MaMH, tenMH) values ('CF','Computer fundamentals')
- insert into monHoc (MaMH, tenMH) values ('C','Elementary Programming with C')
- insert into monHoc (MaMH, tenMH) values ('HDJ','HTML, DHTML and JavaScript')
- update monHoc set wtest=1
- update monHoc set ptest=1 where MaMH in ('C','HDJ')
- update monHoc set ptest=0 where MaMH ='CF'

Delete (1)

- DELETE statement
  - The DELETE statement removes rows from a table.

Syntax:

```
DELETE FROM <Table_name>
[WHERE <Search condition>]
```

where,

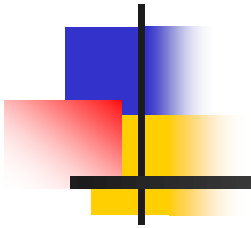
`<Table_Name>` is the name of the table in which row is to be inserted.

`WHERE` clause is to specify the condition. If where clause is not included in the DELETE statement, all the records in the table will be deleted.

<Search condition> specifies the condition to be met for the rows to be deleted.

Delete (2)

- create table monHoc
  (maMH char(10) primary key,
  tenMH nvarchar(50),
  wtest bit,
  ptest bit)
- insert into monHoc (MaMH, tenMH) values ('CF','Computer fundamentals')
- insert into monHoc (MaMH, tenMH) values ('C','Elementary Programming with C')
- insert into monHoc (MaMH, tenMH) values ('HDJ','HTML, DHTML and JavaScript')
- delete from monHoc where maMH ='C'
- delete from monHoc



CASE

- Evaluates a list of conditions and returns one of multiple possible result expressions.
- CASE can be used in any statement or clause that allows a valid expression. For example, you can use CASE in statements such as SELECT, UPDATE, DELETE and SET, and in clauses such as select_list, IN, WHERE, ORDER BY, and HAVING.

- Systax:

Simple CASE expression:

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

Searched CASE expression:

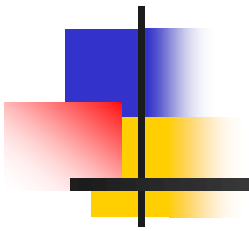
```
CASE
 WHEN Boolean_expression THEN result_expression [...n]
   [ ELSE else_result_expression ]
END
```

Case – Example 1

```
UPDATE CLASS
 set TimeSlot =
  CASE
    when right(ClassCode,1)='G' then '7:30 - 9:30'
    when right(ClassCode,1)='H' then '9:30 - 11:30'
    when right(ClassCode,1)='I' then '13:30 - 15:30'
    when right(ClassCode,1)='K' then '15:30 - 17:30'
    when right(ClassCode,1)='L' then '17:30 - 19:30'
    when right(ClassCode,1)='M' then '19:30 - 21:30'
    ELSE 'unknown'
  END
```

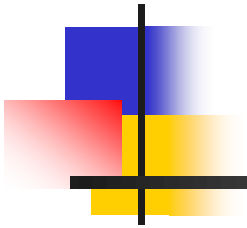
Case – Example 2

```
SELECT ProductNumber, Name, 'Price Range' =
   CASE
      WHEN ListPrice = 0 THEN 'not for resale'
      WHEN ListPrice < 50 THEN 'Under $50'
      WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Under $250'
      WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Under
    $1000'
       ELSE 'Over $1000'
   END
FROM Product
ORDER BY ProductNumber ;
```



Case – Example 3

```
update books set [type]=
    case
        when Totalpage<100 then 'thin'
        when Totalpage between 100 and
  1000 then 'normal'
        when TotalPage>1000 then 'thick'
    end
```



Functions in Transact-SQL 1-7

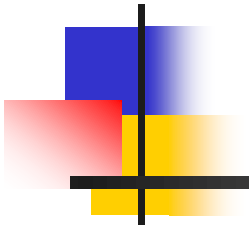
- A function is a set of Transact-SQL statements that is used to perform some task.
- SQL functions work on the data or group of data to return a required value.
- Transact-SQL contains the following categories of functions:
  - Aggregate Functions
  - Conversion Functions
  - Date Functions
  - Mathematical Functions
  - System Functions
  - Ranking Functions

Functions in Transact-SQL 2-7

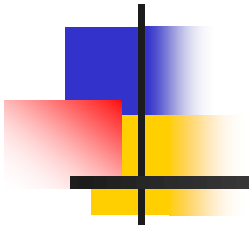
- Aggregate Functions

Function	Return Value	Example
SUM(col_name)	Returns the summation of a series of values	SELECT SUM(OrderQty) AS Total FROM PurchaseOrderDetail
AVG(col_name)	Returns average of a series of values	SELECT AVG(UnitPrice * OrderQty) AS AveragePrice FROM PurchaseOrderDetail
COUNT	Returns the number of records in a table that meet a given criteria	SELECT COUNT(*) AS 'Number of Large Orders' FROM PurchaseOrderDetail WHERE OrderQty > 100
MAX(col_name)	Returns the maximum value from a series of values	SELECT MAX(OrderQty * UnitPrice) AS 'Largest Order' FROM PurchaseOrderDetail
MIN(col_name)	Returns the minimum value from a series of values	SELECT MIN(OrderQty * UnitPrice) AS 'Smallest Order' FROM PurchaseOrderDetail

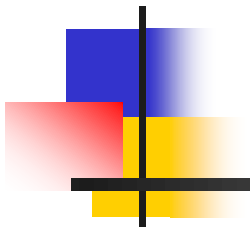
assist with the summarization of large volumes of data.



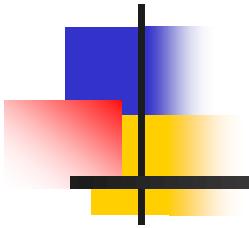
- SUM([ALL | DISTINCT] expression)
- AVG([ALL | DISTINCT] expression)
- COUNT([ALL | DISTINCT] expression)
- COUNT(*)
- MAX(expression)
- MIN(expression)
- where
  - SUM and AVG function works with numerical expression
  - SUM, AVG, COUNT, MIN and MAX function ignores NULL values when it calculates
  - COUNT(*) function does not ignore NULL values.



- SELECT AVG(diemlan1) FROM diemthi
- SELECT MAX(YEAR(GETDATE())-YEAR(ngaysinh)), MIN(YEAR(GETDATE())-YEAR(ngaysinh)), AVG(YEAR(GETDATE())-YEAR(ngaysinh)) FROM sinhvien WHERE noisinh=’HÀ NỘI’
- SELECT b.malop,b.tenlop,COUNT(a.masv) AS siso FROM sinhvien a inner join lop b on b.malop=a.malop GROUP BY b.malop,b.tenlop



- SELECT a.masv,a.hodem+’ ‘+a.ten as hoten,
  sum(c.diemlan1*b.sodvht)/sum(b.sodvht) as dtb FROM (sinhvien a inner join diemthi c on a.masv=c.masv) inner join monhoc b on c.mamonhoc=b.mamonhoc GROUP BY a.masv,a.hodem,a.ten HAVING sum (c.diemlan1*b.sodvht) / sum(b.sodvht)>=5 order by a.ten,a.hodem



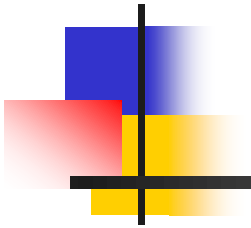
COMPUTE

- COMPUTE function_name(column_name) [,..., function_name (column_name)]

  [BY column_list]

  where:

  - *Function_name* includes SUM, AVG, MIN, MAX and COUNT.
  - When to use **COMPUTE** with **BY, it must have ORDER BY**
  - [Column_list] in **BY, they have to** appear in **ORDER BY**



- SELECT * FROM Suppliers COMPUTE count(*)
- SELECT * FROM Suppliers ORDER BY *SupplierID* COMPUTE count(*) BY *SupplierID*

```
SELECT khoa.makhoa,
    tenkhoa, malop, tenlop
FROM khoa,lop WHERE
    khoa.makhoa=
    lop.makhoa
ORDER BY khoa.makhoa
COMPUTE COUNT(malop)
    BY khoa.makhoa
```

	makhoa	tenkhoa	malop	tenlop
1	CNTT	Cong nghe thong tin	D05CNTT1	lop tin 1
2	CNTT	Cong nghe thong tin	D05CNTT2	lop tin2
3	CNTT	Cong nghe thong tin	D05CNTT3	lop tin 3

	cnt
1	3

	makhoa	tenkh...	malop	tenlop
1	DT	Dien tu	D05DT1	lop dien tu 1

	cnt
1	1

	makhoa	tenkhoa	malop	tenlop
1	QTKD	Quan tri kinh doanh	D05QTKD1	lop quan tri 1
2	QTKD	Quan tri kinh doanh	D05QTKD2	lop quan tri 2

	cnt
1	2

	makhoa	tenkhoa	malop	tenlop
1	VT	Vien thong	D05VT1	lop vien thong 1
2	VT	Vien thong	D05VT2	lop vien thong 2

	cnt
1	2

Functions in Transact-SQL 3-7

- Date Functions
  - The table shows the valid Dateparts in SQL Server 2005:

Datepart	Abbreviation	Values
Hour	hh	0-23
Minute	Mi	0-59
Second	Ss	0-59
Millisecond	Ms	0-999
Day of Year	Dy	1-366
Day	Dd	1-31
Week	wk	1-53
Weekday	dw	1-7
Month	mm	1-12
Quarter	qq	1-4
Year	yy	1753-9999

  - They are useful to perform calculations based on dates. Date functions help extract the day, month, and year of a date value, so that each date can be manipulated separately.
  - DATEPART refers to the part of the date that is to be extracted and used.

Functions in Transact-SQL 4-7

- The table shows the functions used for Dateparts in SQL Server 2005:

Function	Return Values	Example
GETDATE()	Current system date	SELECT GETDATE()
DATEADD(datepart,number,date)	Date with the specified number added to the specified date part	SELECT DATEADD(mm,4,'01/01/99') - returns 05/01/99 in the current date format
DATEDIFF(datepart,date1,date2)	Difference between the specified datepart of the two dates	SELECT DATEDIFF(mm,'01/01/99','05/01/99') - returns 4
DATENAME(datepart,date)	A character string for the datepart in the date	SELECT DATENAME(dw,'01/01/2000') - returns Saturday
DATEPART(datepart,date)	An integer for the specified datepart of the date	SELECT DATEPART(day,'01/15/2000') - returns 15

Functions in Transact-SQL 5-7

The table shows the Mathematical functions in SQL Server 2005:

Function	Return Values	Example
ABS(num_expr)	Absolute value of a numeric expression	SELECT ABS(-43) return 43
CEILING(num_expr)	Smallest integer less than, or equal to the specified numeric expression	SELECT CEILING(43.5) returns 44
FLOOR(num_expr)	Largest integer less than or equal to the specified expression	SELECT FLOOR(43.5) returns 43
POWER(num_expr,y)	Value of numeric expression to the power of y	SELECT POWER(5,2) returns 25
ROUND(num_expr,length)	Numeric expression rounded off to the specified integer length	SELECT ROUND(43.543,1) returns 43.500
SIGN(num_expr)	+1 for positive numbers, -1 for negative numbers, and 0 for zeros	SELECT SIGN(-43) returns -1
SQRT(float_expr)	Square root of the specified approximate float expression	SELECT SQRT(9) returns 3

- Mathematical Functions
  - Mathematical functions perform algebraic operations on numeric values.

Functions in Transact-SQL 6-7

- System Functions

The table shows the System functions in SQL Server 2005:

Function	Return Values
DB_ID([‘database_name’])	Database identification number
DB_NAME([database_id])	Database name
HOST_ID()	Identification number of the workstation
HOST_NAME()	Workstation name
ISNULL(expr,value)	Null expressions replaced with a specified value
OBJECT_ID(‘obj_name’)	Database object identification number
OBJECT_NAME(object_id)	Database object name
USER_ID([‘user_name’])	User’s database identification number
USER_NAME([user_id])	User’s database username

eturning metadata or configuration settings.

ISNULL(expr,value)

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	16.2	67	15
2	Mascarpone	23	22	
3	Gorgonzola	45	19	20

- SELECT ProductName,UnitPrice*(UnitsInStock+UnitsOnOrder) FROM Products
- In the example above, if any of the "UnitsOnOrder" values are NULL, the result is NULL.
- SELECT ProductName,UnitPrice*(UnitsInStock+ ISNULL(UnitsOnOrder,0)) FROM Products

Functions in Transact-SQL 7-7

- Ranking Functions
  - Using Ranking functions, many tasks, like creating arrays, generating sequential numbers, finding ranks, and so on can be implemented in an easier and faster way.

The table shows the ranking functions in SQL Server 2005:

Function	Return Values
ROW_NUMBER()	Returns the number of a row within a result set starting with 1 for the first row
DENSE_RANK()	Returns the rank of rows in a result set without gaps in ranking

ROW_NUMBER() & DENSE_RANK()

- ROW_NUMBER ( ) OVER ([ ])
- SELECT ROW_NUMBER() OVER (ORDER BY Age) AS [Row Number by Age],
  FirstName,
  Age
  FROM Person

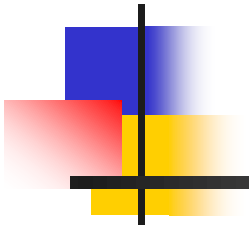
- SELECT DENSE_RANK() OVER (ORDER BY Age) AS [Dense Rank by Age],
  FirstName,
  Age
  FROM Person

Result by ROW_NUMBER()

Row Number by Age	FirstName	Age
1	Jonh	21
2	Mary	22
3	Bin	22
4	Jenry	34

Result by DENSE_RANK()

Row Number by Age	FirstName	Age
1	Jonh	21
2	Mary	22
2	Bin	22
3	Jenry	34



Summary

- Transact-SQL is an extension of the language defined in the SQL standards published by International Organization for Standardization (ISO) and American National Standards Institute (ANSI).
- Data definition language statements are used to build and modify the structure of tables and other objects such as view, trigger, stored procedure and so on.
- Data manipulation language is used to select, insert, update, and delete data in the objects defined with Data definition language.
- Data control language is used to control permissions on database objects.