

**Khoa Thống kê – Tin học**

# **NGÔN NGỮ SQL & T-SQL**

***Biên soạn: Cao Thị Nhâm***

**Đà Nẵng, tháng 09 năm 2018**

## MỤC LỤC

1	PHẦN I. NGÔN NGỮ SQL .....	5
1.1	1.1. Giới thiệu chung.....	5
1.1.1	Ngôn ngữ SQL .....	5
1.1.2	Các khái niệm cơ bản trong cơ sở dữ liệu.....	5
1.1.3	Các nhóm lệnh.....	6
1.1.4	1Cơ sở dữ liệu sử dụng trong tài liệu .....	7
1.2	1Các lệnh DDL .....	9
1.2.1	Các kiểu dữ liệu cơ bản.....	9
1.2.2	Lệnh tạo bảng (CREATE TABLE).....	10
1.2.3	Lệnh sửa bảng (ALTER TABLE).....	13
1.2.4	1Các lệnh DDL khác.....	14
1.3	Các lệnh DML.....	14
1.3.1	Thêm dữ liệu .....	14
1.3.2	Sửa dữ liệu .....	15
1.3.3	Xóa dữ liệu .....	15
1.4	Lệnh DQL (truy vấn dữ liệu) .....	16
1.4.1	Quy tắc viết lệnh .....	16
1.4.2	Cú pháp tổng quát .....	16
1.4.3	Các phép toán .....	17
1.4.4	Phép toán logic .....	21
1.4.5	Sắp xếp dữ liệu .....	22
1.4.6	Một số hàm cơ bản .....	23
1.4.7	Nối bảng .....	28
1.5	Truy vấn lồng (subquery).....	31
2	Phần II. T-SQL .....	32
2.1	T-SQL là gì .....	32
2.2	Biến .....	32
2.2.1	Khai báo biến .....	33

2.2.2	Gán giá trị.....	33
2.2.3	Một số biến toàn cục thường dùng.....	34
2.3	Các phép toán.....	35
2.4	Các lệnh điều khiển.....	35
2.4.1	Rẽ nhánh.....	35
2.4.2	Vòng lặp WHILE .....	36
2.5	Con trỏ.....	37
2.6	Ngoại lệ.....	39
2.7	Thủ tục .....	40
2.7.1	Cú pháp tạo thủ tục .....	41
2.7.2	Cú pháp gọi thủ tục .....	42
2.7.3	Sửa và xóa thủ tục .....	42
2.8	Hàm.....	43
2.8.1	Hàm vô hướng.....	43
2.8.2	Hàm inline .....	44
2.8.3	Hàm khai báo tường minh.....	44
2.8.4	Gọi hàm.....	45
2.8.5	Xóa và sửa hàm .....	45
2.9	Trigger.....	45
2.9.1	Trigger là gì.....	45
2.9.2	Cú pháp tạo trigger.....	45
2.9.3	Bảng “ma thuật” (Magic tables) – bảng logic.....	46
3	Phần III. Transaction .....	<b>Error! Bookmark not defined.</b>
3.1.1	Tại sao cần transaction? .....	53
3.1.2	Khai báo và sử dụng giao dịch.....	53
4	Phần IV. Quản lý tương tranh .....	<b>Error! Bookmark not defined.</b>
4.1	Các vấn đề thường gặp khi có tương tranh.....	55
4.1.1	Dữ liệu bẩn (Dirty reads) .....	55
4.1.2	Dữ liệu đọc không lặp lại (Non-Repeatable Reads).....	56

---

4.1.3	Dữ liệu ảo (Phantom) .....	56
4.2	Giải quyết tương tranh .....	56
4.2.1	Read Uncommitted.....	57
4.2.2	Read Committed.....	58
4.2.3	Repeatable read .....	59
4.2.4	Serializable .....	60
4.2.5	Snapshot .....	61
4.2.6	Phạm vi áp dụng.....	61

## CHƯƠNG 1. NGÔN NGỮ SQL

### 1.1 1.1. Giới thiệu chung

#### 1.1.1 Ngôn ngữ SQL

**SQL (Structured Query Language** - ngôn ngữ truy vấn mang tính cấu trúc) là một loại ngôn ngữ máy tính phổ biến để tạo, sửa, và lấy dữ liệu từ một hệ quản trị cơ sở dữ liệu quan hệ.

Giữa những năm 1970, một nhóm các nhà phát triển tại trung tâm nghiên cứu của IBM tại San Jose phát triển hệ thống cơ sở dữ liệu "Hệ thống R" dựa trên mô hình của Codd. Structured English Query Language, viết tắt là "SEQUEL" (tạm dịch là "Ngôn ngữ truy vấn tiếng Anh có cấu trúc"), được thiết kế để quản lý và truy lục dữ liệu được lưu trữ trong Hệ thống R. Sau này, tên viết tắt SEQUEL được rút gọn thành SQL để tránh việc tranh chấp nhãn hiệu (từ SEQUEL đã được một công ty máy bay của UK là Hawker-Siddeley đăng ký). Mặc dù SQL bị ảnh hưởng bởi công trình của tiến sĩ Codd nhưng nó không do tiến sĩ Codd thiết kế ra. Ngôn ngữ SEQUEL được thiết kế bởi Donald D. Chamberlin và Raymond F. Boyce tại IBM, và khái niệm của họ được phổ biến để tăng sự chú ý về SQL.

SQL được thừa nhận là tiêu chuẩn của ANSI (American National Standards Institute) vào năm 1986 và ISO (International Organization for Standardization) năm 1987. ANSI đã công bố cách phát âm chính thức của SQL là "ess kyoo ell", nhưng rất nhiều các chuyên gia cơ sở dữ liệu nói tiếng Anh vẫn gọi nó là sequel.

Tiêu chuẩn SQL đã trải qua một số phiên bản:

Năm	Tên	Tên khác	Chú giải
1986	SQL-86	SQL-87	Được công bố đầu tiên bởi ANSI. Được phê chuẩn bởi ISO năm 1987.
1989	SQL-89		Thay đổi nhỏ.
1992	SQL-92	SQL2	Thay đổi lớn.
1999	SQL:1999	SQL3	
2003	SQL:2003		

#### 1.1.2 Các khái niệm cơ bản trong cơ sở dữ liệu

### 1.1.3 Các nhóm lệnh

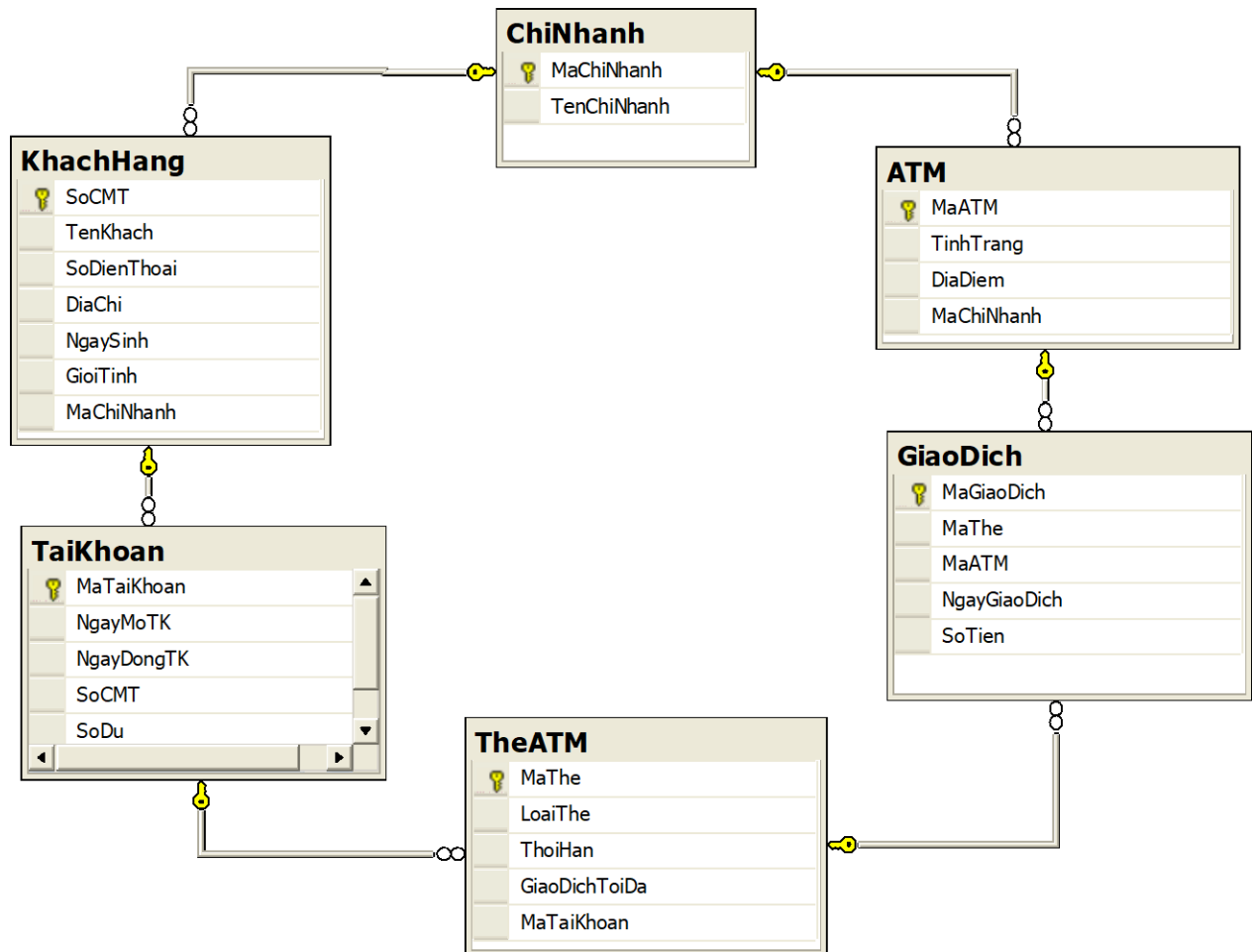
Nhóm lệnh	Lệnh	Chú giải
DQL (Data Query Language)	SELECT	Đây là lệnh phổ dụng nhất. Dùng để lấy dữ liệu trong cơ sở dữ liệu.
DML (Data Manipulation Language)	INSERT UPDATE DELETE	Là những lệnh dùng để thay đổi dữ liệu có trong các bảng của cơ sở dữ liệu
DDL (Data Definition Language)	CREATE ALTER DROP	Là những lệnh dùng để tạo ra, thay đổi hoặc xóa bỏ cấu trúc dữ liệu (bảng)
Transaction Control	COMMIT ROLLBACK SAVE POINT	Dùng để quản lý các giao dịch trong cơ sở dữ liệu.
DCL (Data Control Language)	GRANT REVOKE	Cấp quyền hoặc hủy quyền của người dùng trên đối tượng của cơ sở dữ liệu.

Trong nội dung tài liệu này chỉ xin đề cập tới DQL, DML và DDL. Ngôn ngữ DCL tham khảo thêm ở phần Quản lý người dùng.

Một số chú ý khi viết lệnh SQL:

- Với collation mặc định, KHÔNG phân biệt chữ HOA, chữ thường trong cú pháp viết lệnh SQL và tên bảng.

### 1.1.4 1 Cơ sở dữ liệu sử dụng trong tài liệu



Thông tin chi tiết về các bảng như sau:

Bảng CHINHANH

Name	Null	Type
MACHINHANH	NOT NULL	CHAR(10)
TENCHINHANH		NVARCHAR (30)

Bảng KHACHHANG

Name	Null	Type
SOCMT	NOT NULL	CHAR(9)
TENKHACH		NVARCHAR (50)
SODIENTHOAI		VARCHAR (15)
DIACHI		NVARCHAR (100)
NGAYSINH		DATE
GIOITINH		NUMERIC (1,0)
MACHINHANH		CHAR(10)

Bảng TAIKHOAN

Name	Null	Type
MATAIKHOAN	NOT NULL	CHAR(13)
NGAYMOTK		DATE
NGAYDONGTK		DATE
SOCMT		CHAR(9)
SODU		NUMERIC(12,0)

Bảng ATM

Name	Null	Type
MAATM	NOT NULL	CHAR(10)
TINHTRANG		CHAR(1)
DIADIEM		NVARCHAR (50)
MACHINHANH		CHAR(10)

Bảng THEATM

Name	Null	Type
MATHE	NOT NULL	CHAR(10)
LOAITHE		CHAR(1)
THOIHAN		DATE
GIAODICHTOIDA		NUMERIC (10,0)
MATAIKHOAN		CHAR(13)

Bảng GIAODICH



Name	Null	Type
MAGIAODICH	NOT NULL	CHAR(10)
MATHE		CHAR(10)
MAATM		CHAR(10)
NGAYGIAODICH		DATE
SOTIEN		NUMERIC(10,0)

## 1.2 1 Các lệnh DDL

### 1.2.1 Các kiểu dữ liệu cơ bản

Cấu trúc lưu trữ logic cơ bản của SQL Server là bảng. Mỗi bảng có các cột và các hàng. Mỗi cột có kiểu dữ liệu khác nhau. SQL Server cung cấp một số kiểu dữ liệu có sẵn. Tài liệu này chỉ đề cập đến những kiểu dữ liệu thường dùng.

Loại	Kiểu dữ liệu
Chữ	CHAR, NCHAR, VARCHAR, NVARCHAR
Số	NUMERIC, FLOAT, INT
Thời gian	DATE, TIME, DATETIME
Các loại khác	LONG, CLOB, ROWID, ...

Sau đây là mô tả chi tiết cho một số kiểu dữ liệu thường dùng.

#### CHAR(n), NCHAR(n)

Đây là kiểu dữ liệu dạng chuỗi có độ dài cố định. Với  $n$  là độ dài của chuỗi hay độ dài của cột,  $n \in [1, 8000]$  đối với CHAR và  $n \in [1, 4000]$  đối với NCHAR. Khi người dùng nhập vào chuỗi có độ dài nhỏ hơn độ dài đã khai báo thì SQL Server tự động thêm dấu cách (khoảng trống) phía đuôi chuỗi để đảm bảo độ dài luôn luôn cố định. Trong trường hợp độ dài chuỗi người dùng nhập vào lớn hơn giá trị  $n$ , SQL Server sẽ báo lỗi.

CHAR(n) dùng để lưu trữ kí tự dưới dạng ASCII, NCHAR(n) dùng để lưu kí tự dưới dạng UNICODE.

#### VARCHAR(n), NVARCHAR(n)

Đây là kiểu dữ liệu dạng chuỗi có độ dài không cố định. Với  $n$  là độ dài của chuỗi hay độ dài của cột,  $n \in [1, 8000]$  đối với VARCHAR và  $n \in [1, 4000]$  đối với NVARCHAR. VARCHAR và NVARCHAR không có độ dài mặc định, chính vì vậy, khi người dùng quên khai báo độ dài thì SQL Server sẽ báo lỗi.

VARCHAR dùng để lưu trữ kí tự dưới dạng ASCII, NVARCHAR dùng để lưu kí tự dưới dạng UNICODE.

**NUMERIC(n, m)**

Đây là dữ liệu kiểu số. Trong cách định nghĩa kiểu **NUMERIC**,  $n$  là số các chữ số trước dấu thập phân,  $m$  là số các chữ số sau dấu thập phân.

**DATE**

Dùng để lưu trữ ngày tháng

**TIME**

Dùng để lưu trữ thời gian

**DATETIME**

Dùng để lưu trữ ngày tháng và thời gian

**1.2.2 Lệnh tạo bảng (CREATE TABLE)****1.2.2.1 Quy tắc đặt tên bảng**

- Không quá 50 ký tự
- Tên bảng bắt đầu bằng chữ cái
- Tên bảng/cột không dùng những cụm từ đã có sẵn trong SQL Server (ví dụ: NUMERIC)
- Tên cột phải duy nhất trong bảng

**1.2.2.2 Cú pháp tạo bảng**

Cú pháp tổng quát để tạo bảng như sau:

```
CREATE TABLE tên_bảng(  
    Tên_Cột_1 Kiểu_Dữ_Liệu,  
    Tên_Cột_2 Kiểu_Dữ_liệu,  
    ..... ,  
    primary key (Tên_Cột_X, Tên_Cột_Y, ...),  
    foreign key (Tên_Cột_Z) references Tên_Bảng_Liên_Kết,  
    .....  
)
```

Ví dụ tạo bảng TAIKHOAN như sau:

```
CREATE TABLE TaiKhoan(  

```

```

        MaTaiKhoan    char(13) ,
        NgayMoTK      date,
        NgayDongTK    date,
        SoCMT         char(9) ,
        primary key (MaTaiKhoan) ,
        foreign key (SoCMT) references KhachHang
    )

```

### 1.2.2.3 1.2.1.3. Ràng buộc dữ liệu

Các ràng buộc dữ liệu dùng để kiểm tra sự đúng đắn và tính toàn vẹn của dữ liệu. Ràng buộc gồm các loại sau:

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

#### *Ràng buộc NOT NULL*

Là ràng buộc không cho phép cột chứa giá trị NULL (để trống).

Trong ví dụ dưới đây, cột **TenChiNhanh** không được phép để trống.

```

CREATE TABLE ChiNhanh(
    MaChiNhanh    char(10) primary key,
    TenChiNhanh   nvarchar(30) NOT NULL
)

```

Chú ý: trong trường hợp khóa chính chỉ có 1 cột, có thể khai báo từ khóa PRIMARY KEY ngay ở dòng khai báo cột như ví dụ trên.

#### *Ràng buộc UNIQUE*

Ràng buộc này quy định giá trị của một cột (hoặc một số cột) phải là duy nhất (không cho phép giá trị trùng lặp).

Ví dụ 1:

```
CREATE TABLE KháchHang(  
    SoCMT          char(9) primary key,  
    TenKhach       nvarchar(50),  
    SoDienThoai    varchar(15),  
    DiaChi         nvarchar(100),  
    NgaySinh       date,  
    GioiTinh       numeric(1,0),  
    MaChiNhanh     char(10),  
    foreign key(MaChiNhanh) references ChiNhanh,  
    CONSTRAINT UNQ_SDT UNIQUE (SoDienThoai)  
)
```

Trong ví dụ này, cột **SoDienThoai** có giá trị duy nhất.

Ví dụ 2:

```
CREATE TABLE KháchHang(  
    SoCMT          char(9) primary key,  
    TenKhach       nvarchar(50),  
    SoDienThoai    varchar(15),  
    DiaChi         nvarchar(100),  
    NgaySinh       date,  
    GioiTinh       numeric(1,0),  
    MaChiNhanh     char(10),  
    foreign key(MaChiNhanh) references ChiNhanh,  
    CONSTRAINT UNQ_SDT UNIQUE (SoDienThoai, TenKhach)  
)
```

Trong ví dụ này, tổ hợp cột **SoDienThoai** và **TenKhach** có giá trị duy nhất.

*Ràng buộc PRIMARY KEY (khóa chính)*

Cũng đưa ra ràng buộc về tính duy nhất của giá trị trong cột giống như UNIQUE nhưng cao cấp hơn. Mỗi bảng chỉ có một khóa chính. Khóa chính được tạo thành từ một hoặc một số cột. Những cột đóng vai trò làm khóa chính không được phép chứa giá trị NULL.

Ví dụ: (xem ở mục Cú pháp tạo bảng)

*Ràng buộc FOREIGN KEY (khóa ngoại)*

Chỉ ra mối ràng buộc tham chiếu giữa bảng này với bảng khác.

Ví dụ: (xem ở mục Cú pháp tạo bảng)

*Ràng buộc CHECK*

Ràng buộc này dùng để kiểm tra miền giá trị của một cột xác định.

Ví dụ:

```
CREATE TABLE TaiKhoan(  
    MaTaiKhoan char(13) primary key,  
    NgayMoTK date,  
    NgayDongTK date,  
    SoCMT char(9),  
    foreign key (SoCMT) references KhachHang,  
    CONSTRAINT ck_D check(NgayMoTK < NgayDongTK)  
)
```

Trong ví dụ trên:

- **NgayMoTK** phải diễn ra trước **NgayDongTK**.
- **ck\_D** là tên ràng buộc do người dùng tự đặt.

### 1.2.3 Lệnh sửa bảng (ALTER TABLE)

Lệnh này dùng để thay đổi cấu trúc của bảng.

#### 1.2.3.1 Thêm cột cho bảng

Cú pháp

```
ALTER TABLE tên_bảng ADD Tên_cột Kiểu_dữ_liệu
```

Ví dụ:

```
ALTER TABLE ChiNhanh add DiaChi nvarchar(50)
```

### 1.2.3.2 1.2.3.3. Thay đổi kiểu dữ liệu của cột

Cú pháp

```
ALTER TABLE ten_bang ALTER COLUMN tên_cột kiểu_dữ_liệu_mới
```

### 1.2.3.3 1.2.3.4. Xóa cột

Cú pháp

```
ALTER TABLE Tên_bảng DROP COLUMN tên_cột
```

### 1.2.3.4 1.2.3.5. Thêm ràng buộc

Cú pháp:

```
ALTER TABLE Tên_bảng ADD CONSTRAINT tên_ràng_buộc  
loại_ràng_buộc..
```

Ví dụ:

```
ALTER TABLE taikhoan ADD CONSTRAINT ck_D CHECK(NgayMoTK <  
NgayDongTK)
```

### 1.2.3.5 1.2.3.6. Xóa ràng buộc

Cú pháp:

```
ALTER TABLE Tên_bảng DROP CONSTRAINT tên_ràng_buộc
```

## 1.2.4 1. Các lệnh DDL khác

### 1.2.4.1 Xóa bảng

Cú pháp:

```
DROP TABLE tên_bảng
```

Một số chú ý:

- Khi DROP TABLE thì xóa cấu trúc và toàn bộ dữ liệu của bảng.

## 1.3 Các lệnh DML

### 1.3.1 Thêm dữ liệu

Để thêm một bản ghi cho bảng ta dùng lệnh **INSERT**.

Cú pháp 1 (dùng để thêm giá trị cho tất cả các cột trong bảng):

```
INSERT INTO Tên_bảng VALUES (gt1, gt2, ...)
```

Chú ý: Thứ tự của **gt1**, **gt2** phải tương ứng với thứ tự của các cột trong bảng.

Cú pháp 2 (dùng để thêm giá trị cho một số cột trong bảng):

```
INSERT INTO Tên_bảng (cột1, cột2, ...) VALUES (gt1, gt2, ...)
```

Chú ý: Thứ tự của **gt1**, **gt2** phải tương ứng với **cột1**, **cột2**.

Cú pháp 3 (lấy giá trị từ bảng khác để thêm mới cho bảng).

```
INSERT INTO Tên_bảng (cột1, cột2, ...)
SELECT cotx, coty, ...
FROM...WHERE...
```

Chú ý: thứ tự của **cotx**, **coty** phải tương ứng với thứ tự của **cột1**, **cột2**.

### 1.3.2 Sửa dữ liệu

Cú pháp:

```
UPDATE Tên_bảng
SET cột1 = giá_trị1, ..., cộtn = giá_trị_n
[WHERE điều_kiện]
```

Ví dụ:

```
UPDATE kháchhang
SET diachi = 'Cau Giay - Ha Noi'
WHERE socmt = '151413097'
```

### 1.3.3 Xóa dữ liệu

Xóa từng bản ghi (row) dùng lệnh DELETE. Cú pháp như sau:

```
DELETE Tên_bảng WHERE [điều_kiện]
```

Ví dụ: xóa những khách hàng có tên bắt đầu bằng chữ T.

```
DELETE KháchHang WHERE TenKhach LIKE 'T%'
```

Xóa toàn bộ dữ liệu trong bảng dùng lệnh TRUNCATE. Cú pháp như sau:

```
TRUNCATE TABLE Tên_bảng
```

Chú ý: Trong trường hợp cần xóa toàn bộ dữ liệu trong bảng, hãy dùng lệnh TRUNCATE để có tốc độ thực hiện nhanh hơn.

## 1.4 Lệnh DQL (truy vấn dữ liệu)

### 1.4.1 Quy tắc viết lệnh

- Không phân biệt chữ HOA, thường trong câu lệnh truy vấn
- Câu truy vấn có thể viết trên nhiều dòng
- Các từ khóa không được viết tắt hay phân cách trên nhiều dòng

### 1.4.2 Cú pháp tổng quát

```
SELECT [ALL/DISTINCT] danh_sách_cột
FROM {table_name | view_name}
[WHERE điều_kiện]
[GROUP BY danh_sách_cột_1]
[HAVING điều_kiện_lọc]
[ORDER BY danh_sách_cột_2 [ASC | DESC]]
```

Một câu truy vấn **bắt buộc** phải có mệnh đề SELECT...FROM. Các mệnh đề khác là tùy chọn.

Ví dụ: Hiển thị danh sách khách hàng.

```
select socmt, tenkhach, sodienthoai, diachi
from khachhang
```

Kết quả trả về như sau:

	SOCMT	TENKHACH	SODIENTHOAI	DIACHI
1	151413097	Ly Nhan Dao	0988333666	Duc Tho - Ha Tinh
2	151011097	Tran Duc Nghia	0988333663	Ly Nhan - Ha Nam
3	252433097	Nguyen Thi Hang	0988333623	Giao Thuy - Nam Dinh
4	353423196	Le Hong Nam	0978453676	Tinh Gia - Thanh Hoa
5	251718097	Dinh Hong Nam	0912338766	Dong Da - Ha Noi
6	971063137	Ho Viet Duan	0932333098	Thanh Xuan - Ha Noi
7	123456789	Dinh Chi Long	0955356666	Vinh Yen - Vinh Phuc
8	856382748	Tran Hong Nam	0901098612	Quan 1 - HCM
9	104875748	Nguyen Ba Dao	0902334554	Luc Ngan - Bac Giang
10	184656387	Ngo The Vinh	0915123123	Luc Ngan - Bac Giang



### 1.4.3 Các phép toán

#### 1.4.3.1 Các phép toán học

Các phép toán học bao gồm: +, -, \*, /. Các phép toán này thường được dùng ở mệnh đề SELECT, WHERE, HAVING.

Ví dụ:

```
select socmt, sodu, sodu + 50000
from taikhoan;
```

Kết quả trả về như sau:

	SOCMT	SODU	SODU+50000
1	151413097	12000000	12050000
2	151011097	44500000	45000000
3	252433097	120000	170000
4	353423196	50000	100000
5	251718097	120000	170000
6	971063137	0	50000
7	123456789	400000	450000
8	856382748	50000	100000
9	104875748	0	50000
10	184656387	510000	560000

Chú ý: các phép toán toán học chỉ thay đổi kết quả hiển thị, còn giá trị thực trong bảng vẫn không thay đổi.

#### 1.4.3.2 1.4.3.2. Đặt lại tên cột

Thông thường kết quả trả về của các câu truy vấn sẽ hiển thị tên cột giống như tên cột của các bảng trong cơ sở dữ liệu. Nếu muốn đặt lại tên cột cho rõ nghĩa hơn có thể dùng dùng pháp sau:

```
SELECT tên_cột_1 "Tên cột 1 mới", tên_cột_2 "Tên cột 2 mới"
FROM...
WHERE...
...
```

Tên cột mới đặt trong dấu nháy kép (" ") và có thể đặt tiếng Việt có dấu, có khoảng cách.

Ví dụ:

```
select      socmt, tenkhach "Tên khách hàng",
           sodienthoai "Số điện thoại", diachi "Địa chỉ"
```

from khachhang;

Kết quả trả về như sau:

	SOCMT	Tên khách hàng	Số điện thoại	Địa chỉ
1	151413097	Ly Nhan Dao	0988333666	Duc Tho - Ha Tinh
2	151011097	Tran Duc Nghia	0988333663	Ly Nhan - Ha Nam
3	252433097	Nguyen Thi Hang	0988333623	Giao Thuy - Nam Dinh
4	353423196	Le Hong Nam	0978453676	Tinh Gia - Thanh Hoa
5	251718097	Dinh Hong Nam	0912338766	Dong Da - Ha Noi
6	971063137	Ho Viet Duan	0932333098	Thanh Xuan - Ha Noi
7	123456789	Dinh Chi Long	0955356666	Vinh Yen - Vinh Phuc
8	856382748	Tran Hong Nam	0901098612	Quan 1 - HCM
9	104875748	Nguyen Ba Dao	0902334554	Luc Ngan - Bac Giang
10	184656387	Ngo The Vinh	0915123123	Luc Ngan - Bac Giang

### 1.4.3.3 1.4.3.3. Phép so sánh

Phép so sánh gồm: >, >=, <, <=, = và <>(!=). Các phép toán này thường được dùng ở mệnh đề WHERE hoặc HAVING, dùng để giới hạn kết quả trả về.

Cú pháp:

Tên\_Cột <phép\_so\_sánh> Giá\_trị

Ví dụ: Hiển thị những tài khoản có số dư nhiều hơn 10.000.000

```
select socmt, sodu
from taikhoan
where sodu > 10000000
```

Kết quả trả về như sau:

	SOCMT	SODU
1	151413097	12000000
2	684756584	150000000
3	067463836	1234000000
4	104875748	19077700

### 1.4.3.4 1.4.3.4. Các phép toán liên quan đến chuỗi

#### LIKE/NOT LIKE

Phép toán LIKE dùng để tìm kiếm các giá trị kiểu chuỗi một cách gần đúng. Thường được dùng ở mệnh đề WHERE.

Phép toán LIKE dùng 2 kí tự đại diện:

- %: đại diện cho mọi chuỗi kí tự (kể cả chuỗi NULL)
- [...]: đại diện cho một kí tự bất kì có trong danh sách đã liệt kê (khác NULL)

- `[^...]`: đại diện cho một ký tự bất kỳ không có trong danh sách đã liệt kê (khác NULL)

Cú pháp:

Tên\_cột    LIKE/NOT LIKE    'mẫu kí tự'

Ví dụ: Tìm những khách hàng có tên bắt đầu bằng từ *T* và kết thúc bằng chữ *m*.

```
select tenkhach
from khachhang
where tenkhach like 'T%m';
```

Kết quả trả về như sau:

	TENKHACH
1	Tran Hong Nam

### Phép ghép chuỗi (+)

Phép toán này dùng để nối hai chuỗi hoặc giá trị của 2 cột với nhau.

Ví dụ:

```
select 'Ông/bà ' + tenkhach + ' có ngày sinh là: ' + cast
( ngaysinh)
from khachhang;
```

Kết quả:

	'ÔNG/BÀ'  TENKHACH  'CÓ NGÀY SINH LÀ: '  NGAYSINH
1	Ông/bà Ly Nhan Dao có ngày sinh là: 12-JAN-78
2	Ông/bà Tran Duc Nghia có ngày sinh là: 17-MAY-90
3	Ông/bà Nguyen Thi Hang có ngày sinh là: 02-JAN-76
4	Ông/bà Le Hong Nam có ngày sinh là: 08-SEP-66
5	Ông/bà Dinh Hong Nam có ngày sinh là: 19-DEC-60
6	Ông/bà Ho Viet Duan có ngày sinh là: 30-DEC-84
7	Ông/bà Dinh Chi Long có ngày sinh là: 01-JAN-84
8	Ông/bà Tran Hong Nam có ngày sinh là: 12-AUG-83
9	Ông/bà Nguyen Ba Dao có ngày sinh là: 29-APR-82
10	Ông/bà Ngo The Vinh có ngày sinh là: 05-JUN-80

#### 1.4.3.5 1.4.3.5. Phép toán IN/NOT IN

Phép toán này dùng để kiểm tra xem cột nào đó có giá trị nằm trong danh sách giá trị cho trước hay không. Thường được dùng ở mệnh đề WHERE hoặc HAVING. Dùng cho mọi kiểu dữ liệu (số, chuỗi, date, ...).

Cú pháp:

Tên\_cột IN/NOT IN (giá\_trị\_1, giá\_trị\_2, ...)

Ví dụ: Tìm khách hàng thuộc chi nhánh HN00000001, HN00000003, HN00000007.

```
select tenkhach, machinhanh
from khachhang
where machinhanh in ('HN00000001', 'HN00000003', 'HN00000007')
```

Kết quả:

	TENKHACH	MACHINHANH
1	Ly Nhan Dao	HN00000001
2	Tran Duc Nghia	HN00000001
3	Nguyen Thi Hang	HN00000003
4	Le Hong Nam	HN00000003
5	Dinh Hong Nam	HN00000003
6	Ho Viet Duan	HN00000001
7	Dinh Chi Long	HN00000001
8	Tran Hong Nam	HN00000001
9	La Quang Vinh	HN00000003
10	Hoang A Na	HN00000001

#### 1.4.3.6 1.4.3.5. Phép toán BETWEEN/NOT BETWEEN

Phép toán này dùng để kiểm tra xem giá trị của một cột có thuộc vào một đoạn cho trước hay không. Thường được dùng ở mệnh đề WHERE hoặc HAVING.

Cú pháp:

Tên\_cột **BETWEEN/NOT BETWEEN** giá\_trị\_1 **AND** giá\_trị\_2

Ví dụ: Tìm những khách hàng sinh từ năm 1978 đến năm 1982.

```
select tenkhach, ngaysinh
from khachhang
where ngaysinh between '1-JAN-1978' and '31-DEC-1982'
```

Kết quả:

	TENKHACH	NGAYSINH
1	Ly Nhan Dao	12-JAN-78
2	Nguyen Ba Dao	29-APR-82
3	Ngo The Vinh	05-JUN-80
4	La Quang Vinh	12-JAN-80
5	Ma Van Kháng	10-OCT-78

### 1.4.3.7 1.4.3.6. Phép toán IS NULL/IS NOT NULL

Phép toán này dùng để kiểm tra xem một ô nào đó đã nhập giá trị hay chưa. Những ô chưa nhập giá trị sẽ có giá trị NULL.

Ví dụ: Tìm những tài khoản chưa đóng.

```
select socmt, mataikhoan, ngaydongtk
from taikhoan
where ngaydongtk is null
```

Kết quả:

	SOCMT	MATAIKHOAN	NGAYDONGTK
1	151413097	151413097000	(null)
2	151011097	151011097210	(null)
3	353423196	353423196011	(null)
4	251718097	251718097123	(null)
5	971063137	971063137987	(null)
6	123456789	123456789098	(null)
7	856382748	856382748596	(null)
8	184656387	184656387564	(null)

Chú ý: Không bao giờ được dùng phép toán = và <> đối với giá trị NULL.

### 1.4.4 Phép toán logic

Dùng để kết hợp các mệnh đề điều kiện. Các phép toán logic bao gồm: AND, OR, NOT

- AND: chỉ trả về kết quả khi và chỉ khi 2 mệnh đề đúng
- OR: trả về kết quả khi một trong 2 mệnh đề đúng.
- NOT: phủ định

Cú pháp:

<mệnh\_đề\_1> AND/OR [[NOT]mệnh\_đề\_2] AND/OR [[NOT]mệnh\_đề\_3]...

Ví dụ: Tìm những khách hàng không sống ở Hà Nội và sinh trước năm 1980.

```
select tenkhach, ngaysinh, diachi
from khachhang
where diachi not like '%Ha Noi' and ngaysinh < '1-JAN-1980'
```

Kết quả:

	TENKHACH	NGAYSINH	DIACHI
1	Ly Nhan Dao	12-JAN-78	Duc Tho - Ha Tinh
2	Nguyen Thi Hang	02-JAN-76	Giao Thuy - Nam Dinh
3	Le Hong Nam	08-SEP-66	Tinh Gia - Thanh Hoa
4	Hoang A Na	01-MAY-71	Can Loc - Ha Tinh
5	Tran Thi Lan	20-APR-77	Hai Hau - Nam Dinh

### 1.4.5 Sắp xếp dữ liệu

Dùng mệnh đề ORDER BY để sắp xếp dữ liệu trả về của câu truy vấn. Mệnh đề ORDER BY luôn luôn đứng ở vị trí cuối cùng trong câu truy vấn. Có thể sắp xếp theo nhiều cột, cột nào gần mệnh đề ORDER BY hơn sẽ có mức độ ưu tiên khi sắp xếp cao hơn. Chỉ định cách sắp xếp bằng từ ASC (tăng dần) hoặc DESC (giảm dần).

Cú pháp:

```
SELECT...
```

```
FROM...
```

```
...
```

```
ORDER BY tên_cột_1 ASC/DESC, tên_cột_2 ASC/DESC, ...
```

Ví dụ: Hiển thị danh sách khách hàng theo chiều tăng dần của mã chi nhánh và giảm dần của tên.

```
select machinhanh, tenkhach
from khachhang
order by machinhanh, tenkhach desc
```

Kết quả:

	MACHINHANH	TENKHACH
1	HN00000001	Tran Hong Nam
2	HN00000001	Tran Duc Nghia
3	HN00000001	Ly Nhan Dao
4	HN00000001	Hoang A Na
5	HN00000001	Ho Viet Duan
6	HN00000001	Dinh Chi Long
7	HN00000002	Ngo The Vinh
8	HN00000002	Ma Van Kháng
9	HN00000003	Nguyen Thi Hang
10	HN00000003	Le Hong Nam

Chú ý: nếu không có từ khóa ASC/DESC đặt sau tên cột thì SQL Server mặc định sắp xếp tăng dần.

## 1.4.6 Một số hàm cơ bản

### 1.4.6.1 Hàm xử lý chuỗi

Tên hàm	Giải thích
LEN(tên_cột)	Tính độ dài chuỗi
UPPER(tên_cột)	Chuyển thành chữ HOA
LOWER(tên_cột)	Chuyển thành chữ thường
LTRIM(tên_cột)	Cắt các dấu cách bên trái chuỗi (đầu chuỗi)
RTRIM(tên_cột)	Cắt các dấu cách bên phải chuỗi (cuối chuỗi)
TRIM(tên_cột)	Cắt các dấu cách ở đầu và cuối chuỗi
SUBSTRING(tên_cột, vị_trí_bắt_đầu, độ_dài_chuỗi_con)	Cắt chuỗi con từ một cột cho trước
LEFT(tên_cột, n)	Cắt n kí tự đầu chuỗi
RIGHT(tên_cột, n)	Cắt n kí tự cuối chuỗi

Ví dụ: hiển thị 5 kí tự đầu tiên trong địa chỉ của khách hàng và viết HOA.

```
select tenkhach, upper(substring(diachi, 1, 5))
from khachhang
```

Kết quả:

1	Ly Nhan Dao	DUC T
2	Tran Duc Nghia	LY NH
3	Nguyen Thi Hang	GIAO
4	Le Hong Nam	TINH
5	Dinh Hong Nam	DONG
6	Ho Viet Duan	THANH
7	Dinh Chi Long	VINH
8	Tran Hong Nam	QUAN
9	Nguyen Ba Dao	LUC N
10	Ngo The Vinh	LUC N

### 1.4.6.2 1.4.5.2. Hàm xử lý số

Tên hàm	Giải thích
FLOOR(n)	Lấy số nguyên lớn nhất nhỏ hơn hoặc bằng n
ROUND(n, m)	Làm tròn số n đến m đơn vị (m: số các chữ số sau dấu thập phân)
SQRT(n)	Tính căn bậc 2 của n

**1.4.6.3 1.4.5.3. Hàm xử lý ngày tháng**

Tên hàm	Giải thích
DAY (d)	Trả về ngày trong tháng
MONTH (d)	Trả về tháng trong năm
YEAR (d)	Trả về năm
DATEPART (part, d)	<p>Trả về một thành phần của ngày tháng. part nhận một trong số các tùy chọn sau:</p> <ul style="list-style-type: none"> <li>yy, yyyy: năm</li> <li>qq, q: quý</li> <li>mm, m: tháng</li> <li>dy, y: ngày trong năm</li> <li>dd, d: ngày trong tháng</li> <li>wk, ww: tuần</li> <li>dw: ngày trong tuần</li> <li>hh: giờ</li> <li>mi: phút</li> <li>ss: giây</li> <li>ms: mili giây</li> <li>mcs: micro giây</li> </ul>
DATENAME (part, d)	Trả về tên gọi của thành phần thời gian (ví dụ: January)
GETDATE ()	Trả về thời gian hiện tại (datetime)
DATEDIFF (part, start_date, end_date)	End date – start date theo đơn vị part
DATEADD (part, value)	Cộng/trừ thời gian

Ví dụ: lấy ra năm hiện tại.

```
select year(getdate())
```

Kết quả:

	(No column name)
1	2018

**1.4.6.4 1.4.5.4. Hàm rẽ nhánh**

Dùng để xử lý các rẽ nhánh trong lệnh truy vấn.

**CASE...WHEN**

Dạng 1:

```
CASE tên_cột WHEN giá_trị_1 THEN kết_quả_1
```



```

        WHEN giá_trị_2 THEN kết_quả_2
        ...
        ELSE kết_quả_ngoại_lệ
    END

```

Dạng 2:

```

CASE WHEN <so_sánh_1> THEN kết_quả_1
      WHEN <so_sánh_2> THEN kết_quả_2
      ...
      ELSE kết_quả_ngoại_lệ
END

```

Ví dụ: Hiển thị danh sách khách hàng và giới tính (Nam, nữ).

```

select tenkhach, case gioitinh when 1 then 'Nam'
                        when 0 then 'Nu'
                        else 'Khong xac dinh' end
from khachhang

```

Kết quả:

STT	TENKHACH	GIỚI TÍNH
1	Ly Nhan Dao	Nam
2	Tran Duc Nghia	Nam
3	Nguyen Thi Hang	Nu
4	Le Hong Nam	Nu
5	Dinh Hong Nam	Nam
6	Ho Viet Duan	Nam
7	Dinh Chi Long	Nam
8	Tran Hong Nam	Nam
9	Nguyen Ba Dao	Nam
10	Ngo The Vinh	Nam

#### 1.4.6.5 1.4.5.5. Hàm gộp/nhóm

Tên hàm	Giải thích
SUM(tên_cột)	Tính tổng. Chỉ áp dụng cho cột có kiểu dữ liệu số
COUNT(*) / COUNT(tên_cột)	COUNT(*) : đếm số bản ghi COUNT(tên_cột) : đếm các giá trị khác NULL

AVG(tên_cột)	Tính giá trị trung bình. Chỉ áp dụng cho cột có kiểu số
MIN(tên_cột)	Đối với dữ liệu số, date: tìm giá trị nhỏ nhất Đối với dữ liệu chuỗi: tìm giá trị nhỏ nhất theo thứ tự trong từ điển.
MAX(tên_cột)	Ngược lại với hàm MIN

Có thể dùng các hàm này để tác động lên toàn bộ dữ liệu hoặc các nhóm dữ liệu. Các hàm này chỉ dùng ở mệnh đề **SELECT** hoặc **HAVING**.

### Tác động lên toàn bộ dữ liệu

Ví dụ 1: Tính tổng số dư của các tài khoản.

```
select sum(sodu) from taikhoan
```

Kết quả:

	SUM(SODU)
1	1447046900

Ví dụ 2: Tính tuổi trung bình của các khách hàng quê ở Hà Nội.

```
select avg(year(getdate()) - year(ngaysinh))
from khachhang
where diachi like '%Ha Noi'
```

Kết quả:

1	36.5
---	------

### Tác động lên các nhóm dữ liệu

Muốn dùng hàm gộp cho từng nhóm dữ liệu thì dùng thêm mệnh đề **GROUP BY**. Cú pháp như sau:

```
SELECT hàm_gộp(tên_cột), danh_sách_cột
FROM ...
WHERE...
GROUP BY danh_sách_cột_1;
```

Với, danh\_sách\_cột\_1 và danh\_sách\_cột có các cột giống nhau nhưng thứ tự có thể khác nhau.

**Chú ý:** khi có danh\_sách\_cột xuất hiện cùng với các hàm gộp thì bắt buộc phải dùng mệnh đề **GROUP BY**.

Ví dụ: Thống kê số lượng khách hàng theo từng chi nhánh.

```
select count(socmt), machinhanh
from khachhang
group by machinhanh
```

Kết quả:

	1	COUNT(SOCMT)	1	MACHINHANH
1		6		HN00000001
2		3		HN00000004
3		2		HN00000002
4		2		HN00000005
5		4		HN00000003

### Lọc dữ liệu sau khi dùng hàm gộp

Sau khi dùng hàm gộp, muốn lọc dữ liệu thì dùng mệnh đề **HAVING** sau **GROUP BY**. Cú pháp như sau:

```
SELECT hàm_gộp(tên_cột), danh_sách_cột
FROM ...
WHERE...
GROUP BY danh_sách_cột_1
HAVING <điều_kiện_lọc>
```

Ví dụ: Hiển thị tuổi trung bình của khách hàng theo chi nhánh, chỉ hiển thị những chi nhánh có số lượng khách hàng nhiều hơn 2.

```
select machinhanh,
       avg(year(getdate()) - year(ngaysinh)) "TuoiTB"
from khachhang
group by machinhanh
having count(socmt) >= 3
```

Kết quả:

[illegible]

Chú yí:

- Các hàm gộp **không được phép** dùng ở mệnh đề WHERE nhưng vẫn **dùng được** ở mệnh đề HAVING.

#### 1.4.6.6 1.4.5.6. Hàm chuyển kiểu dữ liệu

Tên hàm	Giải thích
CAST (giá_trị AS tên_kiểu)	Chuyển giá_trị thành kiểu nào đó
CONVERT(tên_kiểu, giá_trị)	

### 1.4.7 Nối bảng

Muốn lấy dữ liệu từ nhiều bảng, hãy sử dụng đến các phép nối (JOIN). Dưới đây là một số loại phép nối thường dùng.

#### 1.4.7.1 Nổi trong

Phép nối trong là phép nối chỉ trả về những bản ghi thỏa mãn điều kiện nối.

Cú pháp:

```
SELECT ...
FROM bảng_1 JOIN bảng_2 ON <điều_kiện_nối>
WHERE...
...
```

Ví dụ: Hiển thị danh sách khách hàng và số lượng tài khoản tương ứng của họ.

```
select tenkhach, count(mataikhoan)
from khachhang join taikhoan on khachhang.socmt = taikhoan.socmt
group by tenkhach
```

### Kết quả:

STT	TENKHACH	STT	COUNT(MATAIKHOAN)
1	Tran Thi Lan		1
2	Dinh Hong Nam		1
3	Ho Viet Duan		1
4	La Quang Vinh		2
5	Ly Nhan Dao		2
6	Tran Hong Nam		2
7	Nguyen Ba Dao		3
8	Ma Van Khanh		1
9	Ngo The Vinh		1
10	Tong Van Tinh		2

#### 1.4.7.2 1.4.6.2. Nối ngoài

Phép nối ngoài trả về các bản ghi thỏa mãn điều kiện nối và không thỏa mãn điều kiện nối. Những bản ghi không thỏa mãn điều kiện nối sẽ để giá trị NULL.

Có 3 loại nối ngoài:

#### Nối trái (LEFT OUTER JOIN)

Phép nối trái lấy các bản ghi của bảng bên **trái** nối với các bản ghi ở bảng bên **phải**. Những bản ghi thỏa mãn điều kiện nối sẽ giữ giá trị như bình thường. Những bản ghi không thỏa mãn điều kiện nối sẽ để giá trị NULL.

Minh họa cho phép nối trái giữa bảng NHANVIEN và PHONGBAN

MaNV	TenNV	MaPB
01	Thành	01
02	An	02
03	Phát	03

(NHANVIEN)

MaPB	TenPB
01	Marketing
03	Sales
04	HR
05	IT

(PHONGBAN)

Kết quả:

MaNV	TenNV	MaPB	MaPB	TenPB
01	Thành	01	01	Marketing
02	An	02	NULL	NULL
03	Phát	03	03	Sales

Cú pháp:

```
SELECT ...
FROM bảng_1 LEFT OUTER JOIN bảng_2 ON <điều_kiện_nối>
WHERE...
```

...

#### Nối phải (RIGHT OUTER JOIN)

Phép nối phải lấy các bản ghi của bảng bên **phải** nối với các bản ghi ở bảng bên **trái**. Những bản ghi thỏa mãn điều kiện nối sẽ giá về giá trị như bình thường. Những bản ghi không thỏa mãn điều kiện nối sẽ để giá trị NULL.

Minh họa cho phép nối phải giữa bảng NHANVIEN và PHONGBAN

MaNV	TenNV	MaPB
01	Thành	01
02	An	02
03	Phát	03

(NHANVIEN)

MaPB	TenPB
01	Marketing
03	Sales
04	HR
05	IT

(PHONGBAN)

Kết quả:

MaPB	TenPB	MaNV	TenNV	MaPB
01	Marketing	01	Thành	01
03	Sales	03	Phát	03
04	HR	NULL	NULL	NULL
05	IT	NULL	NULL	NULL

Cú pháp:

```
SELECT ...
FROM bảng_1 RIGHT OUTER JOIN bảng_2 ON <điều_kiện_nối>
WHERE...
...
```

### Nối đầy đủ (FULL OUTER JOIN)

Là hợp của phép nối trái và nối phải (những bản ghi giống nhau chỉ hiển thị một lần).

Minh họa cho phép nối phải giữa bảng NHANVIEN và PHONGBAN

Kết quả:

MaNV	TenNV	MaPB	MaPB	TenPB
01	Thành	01	01	Marketing
02	An	02	NULL	NULL
03	Phát	03	03	Sales
NULL	NULL	NULL	04	HR
NULL	NULL	NULL	05	IT

Cú pháp:

```
SELECT ...  
FROM bảng_1 FULL OUTER JOIN bảng_2 ON <điều_kiện_nối>  
WHERE...  
...
```

## 1.5 Truy vấn lồng (subquery)

<đang cập nhật>

## CHƯƠNG 2. T-SQL

### 2.1 T-SQL là gì

Transact-SQL (còn gọi là T-SQL) là một ngôn ngữ lập trình database hướng thủ tục độc quyền của Microsoft sử dụng trong SQL Server.

Ngôn ngữ thủ tục được thiết kế để mở rộng khả năng của SQL trong khi có khả năng tích hợp tốt với SQL. Một số tính năng như các biến địa phương và xử lý chuỗi/dữ liệu được thêm vào.

Ngôn ngữ T-SQL có thể dùng để định nghĩa bảng, chèn, xóa, cập nhật và truy cập dữ liệu trong bảng.

T-SQL là ngôn ngữ khá mạnh có đề cập đến kiểu dữ liệu, đối tượng tạm thời, các thủ tục hệ thống và các thủ tục mở rộng.

T-SQL còn có khả năng xử lý trên mẫu tin, xử lý có điều kiện, điều khiển giao tác, xử lý lỗi và biệt lệ.

T-SQL tổ chức theo từng khối lệnh, một khối lệnh có thể lồng bên trong một khối lệnh khác, một khối lệnh bắt đầu bởi BEGIN và kết thúc bởi END, bên trong khối lệnh có nhiều lệnh.

Cấu trúc khối lệnh:

```
1 BEGIN
2     -- Khai báo biến
3     -- Các câu lệnh T-SQL
4 END;
```

### 2.2 Biến

Biến là một đối tượng có thể lưu giữ một giá trị dữ liệu. Dữ liệu có thể được chuyển đến câu lệnh T-SQL bằng việc sử dụng biến cục bộ.

Xét về phạm vi, biến có thể được phân thành 2 loại: biến cục bộ và toàn cục.

- **Biến cục bộ:** Trong T-SQL biến cục bộ được tạo và được sử dụng cho việc lưu trữ tạm thời trong khi câu lệnh SQL được thực hiện. Tên của biến cục bộ phải bắt đầu với dấu '@'



- **Biến toàn cục:** Biến toàn cục là biến được định nghĩa và xử lý bởi hệ thống. Biến toàn cục trong SQL Server được bắt đầu với 2 dấu '@'. Giá trị của các biến này có thể được truy lục bằng câu truy vấn SELECT đơn giản

Xét về cấu trúc dữ liệu, biến có thể phân thành 2 loại:

- **Biến vô hướng:** dữ liệu dạng nguyên thủy: số (int, numeric,...), chữ (char, varchar,...), thời gian(date, time,...)
- **Biến bảng:** dữ liệu có cấu trúc như một bảng

## 2.2.1 Khai báo biến

### 2.2.1.1 Biến vô hướng

```
DECLARE @<variable name> <variable type>[= <value>][,
    @<variable name> <variable type>[= <value>][,
    @<variable name> <variable type>[= <value>]]
```

Ví dụ:

```
declare @num numeric(2,0), @msg nvarchar(150)
```

### 2.2.1.2 Biến bảng

```
DECLARE @<variable name> TABLE (
    <column spec> [, . . .]
)
```

Ví dụ:

```
declare @tb table (id numeric(2,0),
    na varchar(100))
```

Chú ý: thực hiện các thao tác thêm, sửa, xóa, lấy dữ liệu trên biến bảng bằng cách dùng các lệnh INSERT, UPDATE, DELETE, SELECT như làm việc với bảng thông thường.

### 2.2.1.3 Quy tắc đặt tên biến

- Luôn bắt đầu bởi @ (biến toàn cục bắt đầu bởi @@)
- Tên không quá 50 ký tự
- Nên đặt theo kiểu camelCase
- Không sử dụng các ký tự đặc biệt

## 2.2.2 Gán giá trị

### 2.2.2.1 Biến vô hướng

Có thể khởi tạo giá trị cho biến ngay khi khai báo.

Ví dụ:

```
Declare @iTest int = 1
```

Ngoài ra, để gán giá trị cho biến có thể dùng một trong hai cách sau:

- **Dùng lệnh SET**

```
SET @variable_name = value/expression
```

- **Dùng lệnh SELECT**

```
SELECT @var1 = col1/exp1, @var2 = col2/exp2,...  
FROM ...
```

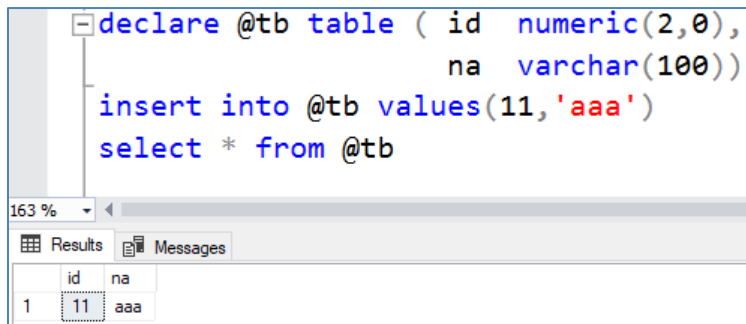
So sánh SET và SELECT

SET	SELECT
1 lệnh SET, chỉ gán giá trị cho 1 biến	1 lệnh SELECT, gán cho nhiều biến
	Gán giá trị ngay trong query

### 2.2.2.2 Biến bảng

Sử dụng các lệnh INSERT, UPDATE, DELETE, SELECT như làm việc với bảng thông thường.

Ví dụ:



### 2.2.3 Một số biến toàn cục thường dùng

Tên biến	Mục đích
@@ERROR	Mã lỗi của câu lệnh thực thi gần nhất
@@IDENTITY	

@@ROWCOUNT	Số lượng bản ghi chịu tác động của câu lệnh gần nhất
@@SERVERNAME	Tên server
@@TRANCOUNT	

## 2.3 Các phép toán

Giống như các phép toán trong SQL

## 2.4 Các lệnh điều khiển

### 2.4.1 Rẽ nhánh

Có thể điều khiển rẽ nhánh bằng: IF...ELSE hoặc CASE..WHEN

#### 2.4.1.1 IF...ELSE

Cú pháp:

```
IF <điều kiện 1>
    Khối lệnh 1;
[ELSE IF <điều kiện 2>
    Khối lệnh 2;
]
....
[ELSE
    Khối lệnh n + 1;
]
```

Ví dụ:

```
--IF ...ELSE
declare @type char(1)
select @type = t_type
from transactions
where t_Date = '2011-12-27' and t_time = '7:45' and ac_no = '1000000041'

if @type = '1'
    print N'Gửi tiền'
else
    print N'Rút tiền'
```

#### 2.4.1.2 CASE...WHEN

Nên dùng trong trường hợp có nhiều rẽ nhánh.

Cú pháp:

Cách 1: dùng cho điều kiện là phép so sánh =

```
CASE <input expression>
WHEN <when expression> THEN <result expression>
[...n]
[ELSE <result expression>]
END
```

Cách 2: dùng cho điều kiện có chứa mọi loại so sánh

```
CASE
WHEN <Boolean expression> THEN <result expression>
[...n]
[ELSE <result expression>]
END
```

Ví dụ:

```
----CASE WHEN
declare @type char(1), @msg nvarchar(20)

select @type = t_type
from transactions
where t_Date = '2011-12-27' and t_time = '7:45' and ac_no = '1000000041'

set @msg = case @type when '1' then N'Gửi tiền'
                  when '0' then N'Rút tiền'
                  else 'NA'
            end

print @msg
```

**Cách 1**

```
----CASE WHEN
declare @type char(1), @msg nvarchar(20)

select @type = t_type
from transactions
where t_Date = '2011-12-27' and t_time = '7:45' and ac_no = '1000000041'

set @msg = case when @type = '1' then N'Gửi tiền'
                  when @type = '0' then N'Rút tiền'
                  else 'NA'
            end

print @msg
```

**Cách 2**

## 2.4.2 Vòng lặp WHILE

Cú pháp:

```
WHILE condition
BEGIN
```

```
-- ...statements...  
END;
```

Trong vòng lặp WHILE bạn có thể sử dụng BREAK để thoát ra khỏi vòng lặp.

Sử dụng lệnh CONTINUE để bỏ qua các dòng lệnh trong khối WHILE và ở bên dưới nó, để tiếp tục một vòng lặp mới.

Ví dụ:

```
--WHILE  
declare @var int  
set @var = 1  
  
while @var <= 100  
begin  
    print @var  
    set @var = @var + 1  
  
end
```

## 2.5 Con trỏ

Cursor là kiểu biến có cấu trúc, cho phép xử lý dữ liệu gồm nhiều dòng. Với việc dịch chuyển con trỏ, có thể truy cập tất cả các dòng dữ liệu.

Những trường hợp cần dùng con trỏ:

- Khi cần duyệt qua từng dòng dữ liệu của một bảng để thao tác với chúng.
- **Chỉ dùng khi không còn cách nào khác** để cho ra kết quả tương tự vì tốc độ thực thi của cursor trong hầu hết trường hợp đều chậm hơn các lệnh T-SQL khác do chúng thường sử dụng nhiều tài nguyên SQL Server.

***Không nên dùng con trỏ khi câu lệnh của bạn tác động đến bảng có số lượng lớn records.***

### **5 bước sử dụng con trỏ:**

#### **1. Khai báo**

```
DECLARE tên_cursor CURSOR FOR (câu_lệnh_truy_vấn)
```

**Chú ý:**

- Sau từ khóa DECLARE là tên biến cursor, thường bắt đầu bằng 'cs\_' hoặc 'cursor' để phân biệt loại biến
- Sau từ khóa FOR là câu lệnh SELECT, chỉ lấy các trường cần thao tác.

## 2. Mở con trỏ

**OPEN** tên\_cursor

## 3. Xử lý dữ liệu

Đọc từng dòng dữ liệu (cần sử dụng vòng lặp ở phần này)

**FETCH NEXT FROM** tên\_cursor **INTO** danh\_sách\_biến

**Chú ý:** sau câu lệnh INTO là danh sách các biến mà khi đọc một dòng, thứ tự các biến đặt tương ứng với thứ tự các cột xuất hiện trong câu lệnh truy vấn của con trỏ.

Do phải đọc nhiều dòng dữ liệu nên cần phải sử dụng tới vòng lặp. Dùng biến toàn cục @@**FETCH\_STATUS** để điều khiển sự kết thúc của vòng lặp.

Các giá trị của @@**FETCH\_STATUS** như sau:

- 0: lấy dữ liệu thành công (*chỉ cần quan tâm giá trị này thôi*)
- -1: bị lỗi trong khi lấy dữ liệu hoặc đã hết dữ liệu để lấy
- -2: dữ liệu lấy được nhưng bị missing
- -9: con trỏ không thực hiện việc lấy dữ liệu

## 4. Đóng con trỏ

**CLOSE** tên\_cursor

## 5. Hủy con trỏ

**DEALLOCATE** tên\_cursor

Bước này để hủy con trỏ hoàn toàn → bộ nhớ được giải phóng.

**Ví dụ:**

```

declare @name nvarchar(100), @ad nvarchar(150)
declare csTest cursor for (select cust_name, cust_ad from customer
                           where cust_ad like N'%Đà Nẵng%')
open csTest --open cursor
while @@FETCH_STATUS = 0 --điều kiện duy trì vòng lặp
begin
    print @name + '---' + @ad
    fetch next from csTest into @name, @ad -- đổ dữ liệu từng dòng vào biến
end
close csTest -- đóng con trỏ
deallocate csTest -- hủy con trỏ

```

Kết quả:

Messages	
Nguyễn Quang Công Minh---	HÒA SON, HÒA VANG, ĐÀ NẴNG
Nguyễn Lê Minh Quân---	K79/4 THANH THỦY, HẢI CHÂU, ĐÀ NẴNG
Đặng Nhật Phong---	K907 NGUYỄN LUONG BẮNG- QUẬN LIÊU CHIỂU- ĐÀ NẴNG
Hứa Văn Đại---	TỔ 27A, NẠI HIÊN ĐÔNG, SON TRÀ, TP. ĐÀ NẴNG
Lê Anh Huy---	412 CÁCH MẠNG THÁNG 8, QUẬN CẨM LỆ, ĐÀ NẴNG
Hồ Trần Nhật Khánh---	193/12 NÚI THÀNH, PHƯỜNG HÒA CƯỜNG BẮC, HẢI CHÂU, ĐÀ NẴNG
Phan Công Diễm---	TỔ 19 - PHƯỜNG PHƯỚC MỸ - SON TRÀ - ĐÀ NẴNG
Nguyễn Xuân Anh---	67 Trần Văn Du, Mỹ An, Ngũ Hành Sơn Đà Nẵng
Lê Nguyễn Hoàng Văn---	SỐ 31 ĐƯỜNG HÒA MINH 4, LIÊN CHIỂU, ĐÀ NẴNG
Nguyễn Văn Hoàng Long---	TỔ 45 ĐA PHƯỚC II, HÒA KHÁNH BẮC, LIÊN CHIỂU, ĐÀ NẴNG
Trần Phước Đạt---	11 THỨC TỄ, THANH KHÊ, TP. ĐÀ NẴNG

## 2.6 Ngoại lệ

Đây là phần quan trọng, tuy nhiên rất nhiều người bỏ quên không đọc nó ☺

Hiểu một cách đơn giản, ngoại lệ là cơ chế các ngôn ngữ lập trình cho phép bạn xử lý nếu lỗi xảy ra. T-SQL cũng có cơ chế xử lý ngoại lệ thông qua **TRY...CATCH**.

Cú pháp:

**BEGIN TRY**

--các lệnh T-SQL

**END TRY**

**BEGIN CATCH**

-- các lệnh T-SQL xử lý cho trường hợp lỗi

**END CATCH**

Một số hàm cung cấp thông tin lỗi:

STT	TÊN HÀM	CHỨC NĂNG
1	ERROR_NUMBER()	Trả lại mã lỗi (dưới dạng số)
2	ERROR_SEVERITY()	Trả lại mức độ nghiêm trọng của lỗi
3	ERROR_STATE()	Trả lại trạng thái của lỗi (dưới dạng số)
4	ERROR_PROCEDURE()	Trả lại tên của Stored Procedure hoặc tên của Trigger đã phát sinh lỗi
5	ERROR_LINE()	Trả lại vị trí dòng lệnh đã phát sinh ra lỗi.
6	ERROR_MESSAGE()	Trả lại thông báo lỗi dưới hình thức văn bản (text)

Ví dụ:

```

begin try
    declare @name nvarchar(100), @ad nvarchar(150)
    declare csTest cursor for (select cust_name, cust_ad from customer
                               where cust_ad like N'%Đà Nẵng%')

    open csTest --open cursor
    while @@FETCH_STATUS = 1/0 --điều kiện duy trì vòng lặp
    begin
        print @name + '---' + @ad
        fetch next from csTest into @name, @ad -- đổ dữ liệu từng dòng vào biến
    end
    close csTest -- đóng con trỏ
    deallocate csTest -- hủy con trỏ
end try
begin catch
    print N'Lỗi: ' + convert(varchar, ERROR_NUMBER()) + ERROR_MESSAGE()
end catch

```

Kết quả:

```

Messages
Lỗi: 8134Divide by zero error encountered.

```

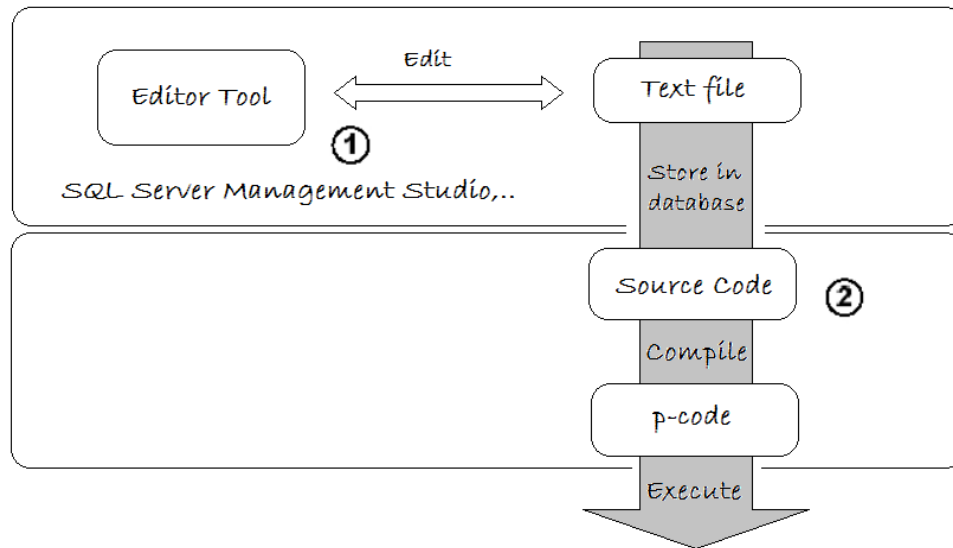
## 2.7 Thủ tục

Một nhóm các lệnh T-SQL thực hiện chức năng nào đó có thể được gom lại trong một thủ tục (procedure) nhằm làm tăng khả năng xử lý, khả năng sử dụng chung, tăng tính bảo mật và an toàn dữ liệu, tiện ích trong phát triển.

Thủ tục có thể được lưu giữ ngay trong database như một đối tượng của database, sẵn sàng cho việc tái sử dụng. Vì vậy thủ tục còn được gọi là Store procedure. Với các Store procedure, ngay khi lưu giữ Store procedure, chúng đã được biên dịch thành dạng p-code vì thế có thể nâng cao khả năng thực hiện.

Các bước thực hiện thủ tục minh họa như hình dưới đây.





Thủ tục không trả về giá trị trực tiếp như hàm (function). Tuy nhiên nó có thể có 0 hoặc nhiều tham số đầu ra.

Đặc điểm của thủ tục:

- Mỗi thủ tục có một tên duy nhất
- Có các tham số đầu vào và đầu ra
- Có thể sử dụng lại nhiều lần

Ưu điểm:

- Động
- Nhanh hơn Transact-SQL
- Giảm thiểu bandwidth
- Bảo mật

### 2.7.1 Cú pháp tạo thủ tục

```

-- procedure_name: Tên thủ tục
-- argument: Tên tham số
-- mode: Loại tham số: INPUT hoặc OUTPUT, mặc định là INPUT
-- datatype: Kiểu dữ liệu của tham số
-- Chú ý: Với thủ tục các tham số có thể đặt trong dấu () hoặc
không cần thiết.

CREATE PROCEDURE <procedure_name>
[
    argument1 datatype1 [mode1] ,
    argument2 datatype2 [mode2] ,
    ...
]
AS
BEGIN
    -- Khai báo biến sử dụng
  
```

```

-- Nội dung của thủ tục.
END;

-- Hoặc:

CREATE PROCEDURE <procedure_name>
(
    [
        argument1 datatype1 [mode1] ,
        argument2 datatype2 [mode2] ,
        ...
    ]
)
AS
BEGIN
    -- Khai báo biến sử dụng
    -- Nội dung của thủ tục.
END;

```

Ví dụ:

```

create proc spTest @word nvarchar(100), @count int output
as
begin
    set @count = (select count(*) from customer where cust_name like '%' + @word + '%')
end

```

### 2.7.2 Cú pháp gọi thủ tục

Đối với thủ tục không có tham số:

**EXEC** tên\_thủ\_tục

Đối với thủ tục có tham số:

**EXEC** tên\_thủ\_tục tên\_biến\_1[output], tên\_biến\_2[output],...

Ví dụ:

```

declare @t int
exec spTest 'inh', @t output
print @t

```

Messages

6

### 2.7.3 Sửa và xóa thủ tục

Để sửa thủ tục, hãy thay từ khóa **CREATE** trong cú pháp ở trên thành **ALTER**.

Để xóa thủ tục:

**DROP PROC** tên\_thủ\_tục

## 2.8 Hàm

Hàm là tập hợp các lệnh SQL được biên dịch, lưu trữ trong cơ sở dữ liệu và được sử dụng như một đơn vị độc lập.

So sánh hàm và thủ tục:

THỦ TỤC	HÀM
Là tập hợp các lệnh SQL có chức năng thực hiện một công việc nào đó	
Giá trị trả về thông qua tham số	Trả về một giá trị cho tên hàm
Trả về nhiều giá trị	Chỉ trả về một giá trị (vô hướng/có hướng)
	Có thể nhúng trong các câu truy vấn
	Không chứa các lệnh chỉnh sửa dữ liệu (INSERT, UPDATE, DELETE,...) bên trong hàm

Có 3 loại hàm: vô hướng, inline và khai báo tường minh

### 2.8.1 Hàm vô hướng

Giá trị trả về của hàm là vô hướng (int, char, date, ...)

```

CREATE FUNCTION function_Name ([@param data_type,...])
RETURNS data_type
AS
Begin
    [Declare variables for processing]
    {Transact-SQL statements}
    RETURN ...
End

```

Cú pháp:

**Chú ý:** bắt buộc phải sử dụng dấu (..) sau function\_Name

Ví dụ:

```
create function fTest ( @word nvarchar)
returns int
as
begin
    declare @count int = 0
    set @count = (select count(*) from customer where cust_name like '%' + @word + '%')
    return @count
end
```

### 2.8.2 Hàm inline

Nội dung hàm chỉ chứa **MỘT câu truy vấn** và giá trị trả về có kiểu **TABLE**

Cú pháp:

```
CREATE FUNCTION function_Name ([@param data_type,...])
RETURNS table
AS
RETURN query_statement
```

**Chú ý:** bắt buộc phải sử dụng dấu (..) sau function\_Name

Ví dụ:

```
create function fTest1 ( @word nvarchar)
returns table
as
return select count(*) as A from customer where cust_name like '%' + @word + '%'
```

### 2.8.3 Hàm khai báo tường minh

Hàm trả về dữ liệu kiểu TABLE và có thể có nhiều lệnh bên trong thân hàm.

Cú pháp:

```
CREATE FUNCTION function_Name ([@param data_type,...])
RETURN @var_name table ( col_name_1 data_type,
                        col_name_2 data_type,...)
AS
BEGIN
    [Declare variables for processing]
    {Transact-SQL statements}
    RETURN ...
END
```

**Chú ý:** bắt buộc phải sử dụng dấu (..) sau function\_Name

Ví dụ:

### 2.8.4 Gọi hàm

Giống như cách gọi các hàm khác trong T-SQL.

Ví dụ:

```
select dbo.fTest('inh')
```

### 2.8.5 Xóa và sửa hàm

Để sửa hàm hãy thay thế từ khóa **CREATE** trong cú pháp trên bằng từ khóa **ALTER**.

Để xóa hàm, dùng cú pháp sau:

```
DROP FUNCTION tên_hàm
```

## 2.9 Trigger

### 2.9.1 Trigger là gì

Là một thủ tục đặc biệt, tự động kích hoạt (chạy) khi gặp sự kiện tương ứng. Trigger thường được sử dụng để kiểm tra ràng buộc trên nhiều quan hệ (nhiều bảng/table) hoặc trên nhiều dòng (nhiều record) của bảng.

Đặc điểm:

- Tự động: Trigger sẽ tự động được gọi. Trigger sẽ ngay tức được kích hoạt mỗi khi có sự thay đổi dữ liệu trên bảng dữ liệu.
- Không trả về dữ liệu sau khi chạy trigger.
- Trigger có những hiệu lực ít bị hạn chế hơn so với ràng buộc giá trị nghĩa là có thể ràng buộc tham chiếu đến những cột của những bảng dữ liệu khác.
- Sử dụng database trigger có thể gây rối, khó khăn cho việc bảo trì và phát triển hệ thống lớn. Vì thế, ta chỉ sử dụng database trigger khi thật cần thiết.

### 2.9.2 Cú pháp tạo trigger

```
CREATE TRIGGER tên_trigger ON tên_bảng  
{FOR|AFTER|INSTEAD OF} {DELETE, INSERT, UPDATE}  
  
AS  
  
BEGIN  
  
    câu_lệnh_sql
```

**END**

Trong đó:

- **FOR|AFTER|INSTEAD OF**: thời điểm chạy trigger (chỉ được chọn 1 trong 3 loại này)
  - **FOR**: trigger chạy đồng thời với lệnh SQL
  - **AFTER**: trigger chạy sau khi đã hoàn thành lệnh SQL
  - **INSTEAD OF**: thay vì chạy câu lệnh SQL sẽ chuyển hướng sang làm việc khác. Ví dụ: khi ra lệnh xóa một bản ghi trong bảng **CUSTOMER** bằng lệnh **DELETE FROM CUSTOMER...** trigger sẽ chuyển sang chạy một lệnh **UPDATE** trạng thái khách hàng thay vì xóa luôn bản ghi đó.
- **DELETE, INSERT, UPDATE**: loại hành động kích hoạt trigger. Có thể chọn nhiều hơn một hành động.

### 2.9.3 Bảng “ma thuật” (Magic tables) – bảng logic

Khi trigger được kích hoạt do hành động INSERT, UPDATE hoặc DELETE xảy ra, hai bảng đặc biệt sẽ được tạo ra để phục vụ cho hoạt động của trigger, đó là: INSERTED và DELETED. Những bảng này có cấu trúc giống như bảng đang bị tác động bởi câu lệnh INSERT, UPDATE hoặc DELETE (tạm gọi là bảng làm việc).

- INSERTED: chứa bản copy của những bản ghi mới được thêm vào bảng làm việc
- DELETED: chứa bản copy của những bản ghi bị xóa khỏi bảng làm việc


Để hiểu rõ hơn về hai bảng này, hãy cùng phân tích từng hành động.

#### 2.9.3.1 INSERT trigger

Được kích hoạt khi có hành động INSERT lên bảng. Khi hành động INSERT xảy ra, bản ghi mới được thêm vào cả hai bảng: bảng làm việc (bảng đang được thêm mới bản ghi) và một bảng tạm có tên là INSERTED.

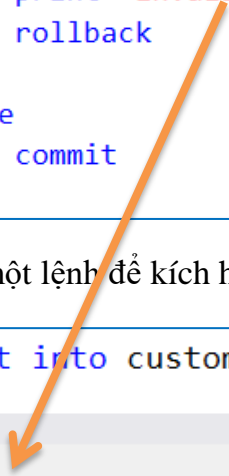
Ví dụ:

```
create trigger tgCheckName
on Customer
for insert
as
begin
    declare @temp nvarchar(100)
    set @temp = (select cust_name from inserted)
    if len(@temp) <= 3
    begin
        print 'invalid value'
        rollback
    end
    else
        commit
end
```



Chạy thử một lệnh để kích hoạt trigger


```
insert into customer(cust_id, cust_name) values('999999','a')
```



Messages

invalid value

Msg 3609, Level 16, State 1, Line 18  
The transaction ended in the trigger. The batch has been aborted.



### Phân tích ví dụ:

Ở ví dụ trên,

- Bảng bị tác động là **CUSTOMER**
- “thời điểm” kích hoạt là “ON” → trigger chạy cùng lúc với lệnh **INSERT**
- Sự kiện kích hoạt: **INSERT**

Khi trigger chạy, một bảng có tên **INSERTED** được tạo ra và có cấu trúc giống hệt bảng **CUSTOMER**.

Tại thời điểm này, bản ghi được thêm mới ở hình 2 được lưu ở cả 2 bảng **CUSTOMER** và **INSERTED**.

Khi kiểm tra dữ liệu, dữ liệu trong bảng **INSERTED** được lấy ra để kiểm tra (không lấy ở bảng **CUSTOMER**). Nếu hợp lệ thì lưu dữ liệu vào bảng (dùng lệnh **COMMIT**), nếu không hợp lệ thì hủy lệnh **INSERT** (dùng lệnh **ROLLBACK**).

### 2.9.3.2 DELETE trigger

Được kích hoạt khi có hành động **DELETE** lên bảng. Khi hành động **DELETE** xảy ra, bản ghi bị xóa được xóa khỏi bảng làm việc và thêm vào bảng **DELETED**.

Ví dụ:

```

alter trigger tgChkDel
on transactions
for delete
as
begin
    declare @id varchar(10)
    select @id = ac_no from deleted
    print N'Bạn không được phép xóa giao dịch của account: ' + @id
    rollback
end

```

Chạy thử lệnh kích hoạt

```

delete from transactions where t_date = '2011-12-27' and t_time = '07:45'

```

Messages

Bạn không được phép xóa giao dịch của account: 1000000041  
 Msg 3609, Level 16, State 1, Line 12  
 The transaction ended in the trigger. The batch has been aborted.

Kiểm tra xem dữ liệu có bị xóa không?

```

select * from transactions where t_date = '2011-12-27' and t_time = '07:45'

```

Results

t_id	t_type	t_amount	t_date	t_time	ac_no
0000000201	0	1752000	2011-12-27	07:45:00.0000000	1000000041

Bản ghi không bị xóa

Khi trigger chạy, một bảng có tên **DELETED** được tạo ra và có cấu trúc giống hệt bảng **TRANSACTIONS**.

Tại thời điểm này, bản ghi bị xóa ở bảng **TRANSACTIONS** và thêm mới vào bảng **DELETED**.

Khi kiểm tra dữ liệu, dữ liệu trong bảng **DELETED** được lấy ra để kiểm tra (không lấy ở bảng **TRANSACTIONS**). Vì hệ thống không cho xóa dữ liệu trong bảng này nên đưa ra thông báo vào hủy hành động xóa bằng lệnh **ROLLBACK**.



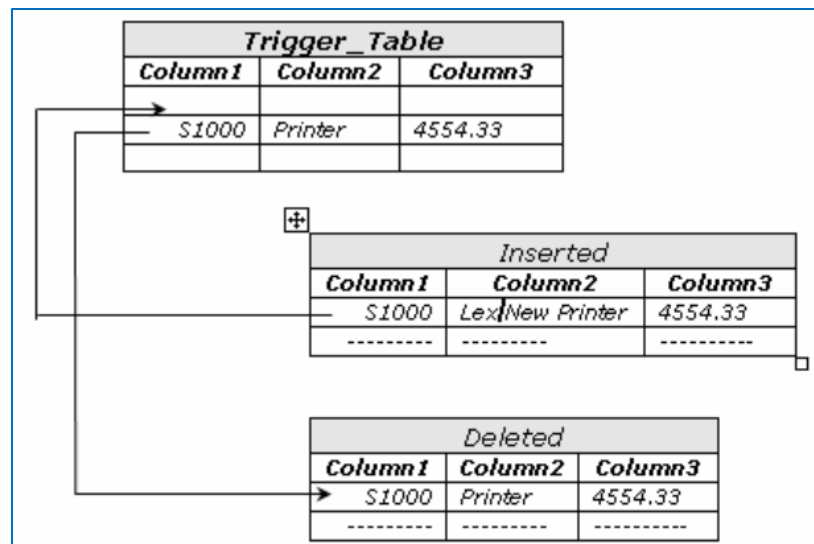
### 2.9.3.3 UPDATE trigger

Khi trigger chạy, bảng **DELETED** và **INSERTED** được tạo ra và có cấu trúc giống hệt bảng **TRANSACTIONS**.

Bản ghi bị update có:

- Giá trị mới: lưu ở bảng **INSERTED**
- Giá trị cũ: lưu ở bảng **DELETED**

Hình sau đây minh họa cách thức lưu trữ dữ liệu của 2 bảng **INSERTED** và **DELETED**



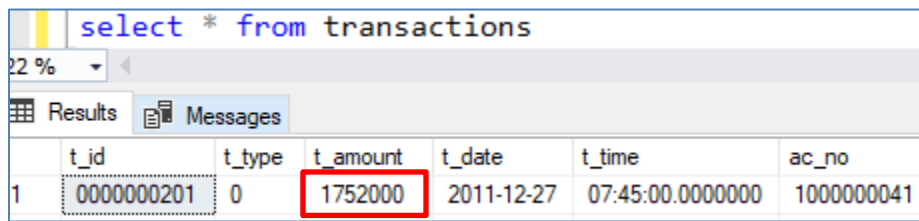
Ví dụ:

```

create trigger tA
on transactions
after update
as
begin
    declare @a numeric(15), @b numeric(15), @ac varchar(12)
    select @a = t_amount from inserted
    select @b = t_amount, @ac = ac_no from deleted
    print @a
    print @b
    print @ac
    rollback
end

```

Dữ liệu ví dụ trước khi kích hoạt trigger:



```
select * from transactions
```

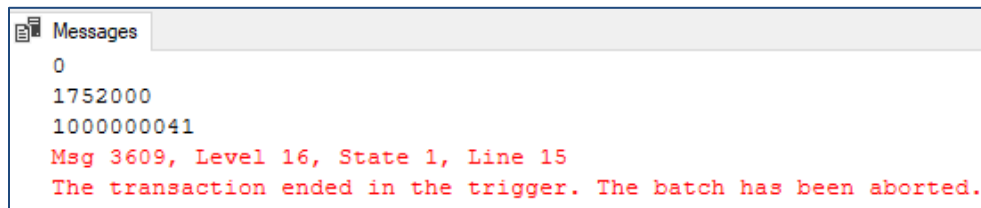
	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	0	1752000	2011-12-27	07:45:00.0000000	1000000041

Dữ liệu trước khi  
INSERT vào bảng  
**TRANSACTIONS**

Kích hoạt trigger bằng một lệnh INSERT:

```
update transactions
set t_amount = 0
where t_id = '0000000201'
```

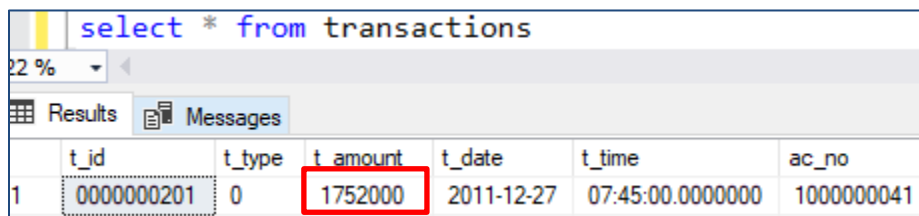
Kiểm tra trigger chạy có đúng không?



```

Messages
0
1752000
1000000041
Msg 3609, Level 16, State 1, Line 15
The transaction ended in the trigger. The batch has been aborted.

```



```
select * from transactions
```

	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	0	1752000	2011-12-27	07:45:00.0000000	1000000041

Dữ liệu sau khi  
INSERT vào bảng  
**TRANSACTIONS**

## 2.10 Index

### 2.10.1 Khái niệm

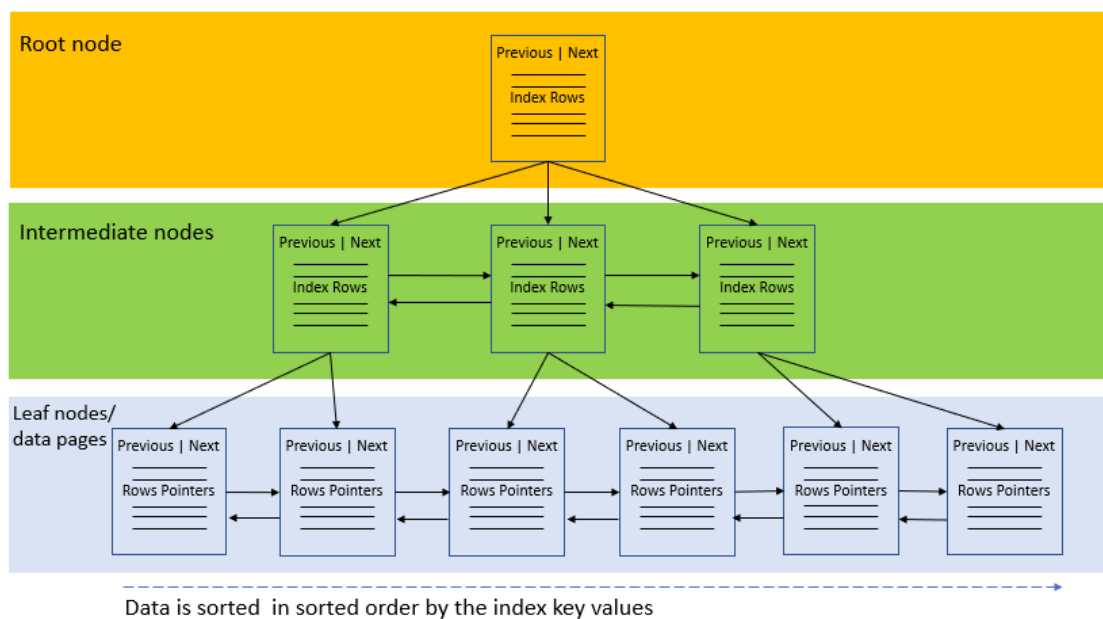
Index là một cấu trúc liên kết với một bảng hoặc một view dùng để tăng tốc độ truy vấn dữ liệu. Index chứa các khóa được tạo từ một hay nhiều cột và được lưu trữ dưới dạng B-tree.

Index chia thành 2 loại:

- *Clustered index.*
  - Trong một bảng chỉ có tối đa một clustered index.
  - Các **bản ghi** trong bảng được lưu trữ và sắp xếp theo thứ tự của giá trị khóa (cột giá trị khóa được xác định trong lúc định nghĩa index)
  - Tốc độ đọc dữ liệu nhanh hơn non-clustered index

- Không cần thêm dung lượng ổ đĩa để lưu trữ index
- Nếu bảng có clustered index thì các thao tác thêm, sửa, xóa chậm hơn
- *Non-clustered index.*
  - Trong một bảng có thể có nhiều non-clustered index
  - Không làm ảnh hưởng tới thứ tự lưu trữ của các bản ghi trong bảng. Bởi, với dạng index này, SQL Server tạo ra một cấu trúc dữ liệu để trỏ tới các bản ghi trong bảng.
  - Các thao tác thêm, sửa, xóa trên bảng sẽ nhanh hơn so với clustered index.
  - Cần thêm dung lượng ổ đĩa để lưu trữ cấu trúc index.

Hình dưới đây mô tả cho non-clustered index.



### Chú ý:

- Khi khai báo PRIMARY KEY thì mặc định SQL Server sẽ tạo một clustered index trên khóa chính đó
- Khi khai báo UNIQUE constraint thì mặc định SQL Server sẽ tạo một non-clustered index trên cột đó.

## 2.10.2 Khai báo index

### a. Cú pháp tạo index:

**CREATE UNIQUE CLUSTERED/NONCLUSTERED INDEX** Tên\_index **on** tên\_bảng(danh\_sách\_cột)

Trong đó:

- **UNIQUE**: xác định giá trị trên cột phải duy nhất (không bắt buộc)
- **CLUSTERED/NONCLUSTERED**: xác định kiểu index (không bắt buộc). Nếu không có tham số này thì mặc định là non-clustered index.

Ví dụ:

```
create index idx_name on customer(cust_name)
```

*Giải thích ví dụ:*

Ví dụ trên tạo một non-clustered index trên cột **cust\_name** của bảng **customer**.

#### **b. Cú pháp xóa index**

```
DROP INDEX tên_index ON tên_bảng
```

Ví dụ:

```
DROP INDEX IDX_NAME ON CUSTOMER
```

### **2.10.3 Một số chú ý khi dùng index**

Mặc dù sử dụng INDEX nhằm mục đích để nâng cao hiệu suất của Database, nhưng đôi khi nên tránh dùng chúng. Dưới đây là một số trường hợp cần xem xét để quyết định có nên sử dụng INDEX hay không:

- Không nên sử dụng trong các bảng nhỏ, ít bản ghi.
- Không nên sử dụng Index trong bảng mà các hoạt động UPDATE, INSERT xảy ra thường xuyên với tần suất lớn.
- Không nên sử dụng cho các cột mà chứa một số lượng lớn giá trị NULL.
- Không nên dùng Index cho các cột mà thường xuyên bị sửa đổi.

**Cần bổ sung minh chứng: có index thì chạy nhanh hơn**

## CHƯƠNG 3. GIAO TÁC

### 3.1 Tại sao cần transaction?

Giao dịch (Transaction) là một khái niệm quan trọng trong SQL. Hãy xem một tình huống:

Một giao dịch trong ngân hàng, người A chuyển cho người B một khoản tiền 100\$, khi đó trong Database xảy ra 2 thao tác:

- Trừ tiền của người A đi 100\$
- Cộng tiền vào cho người B 100\$.

Điều gì sẽ xảy ra nếu chỉ có 1 thao tác thành công?

Xem một ví dụ khác:

Khi bạn thêm một sinh viên vào một lớp học bạn cập nhật lại số của lớp học. Nếu việc chèn thông tin sinh viên không thành công mà số lại được cộng thêm 1, tính toán vện của dữ liệu bị hỏng.

Giao dịch được coi là thành công nếu tất cả các đơn vị lệnh thành công. Ngược lại một trong các đơn vị lệnh bị lỗi, toàn bộ giao dịch cần phải được trả về (rollback) trạng thái ban đầu.

### 3.2 Tính chất của giao tác

Giao tác có tính chất **ACID**.

- **Tính nguyên tử (Atomicity).** Một giao tác là một đơn vị công việc không thể phân chia. Có nghĩa là: các hành động trong một giao tác hoặc là thực hiện được tất cả hoặc là không thực hiện được bất cứ hành động nào.
- **Tính nhất quán (Consistency).** Khi một giao tác kết thúc (thành công hay thất bại), CSDL phải ở trạng thái nhất quán (đảm bảo mọi ràng buộc toàn vẹn dữ liệu). Một giao tác đưa CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.
- **Tính cô lập (Isolation):** Một giao tác khi thực hiện sẽ không bị ảnh hưởng bởi các giao tác khác thực hiện đồng thời với nó.
- **Tính bền vững (Durability):** Mọi thay đổi trên CSDL được ghi nhận bền vững vào thiết bị lưu trữ dù có sự cố có thể xảy ra.

### 3.3 Khai báo và sử dụng giao dịch

*Các lệnh liên quan:*

- Bắt đầu transaction:
  - `begin tran / begin transaction`
- Hoàn tất transaction:
  - `commit/ commit tran / commit transaction`
- Quay lui transaction (Rollback transaction):
  - `rollback / rollback tran / rollback transaction`
- Đánh dấu một savepoint trong transaction:
  - `save transaction tên_của_savepoint`
- Biến @@trancount: cho biết số transaction hiện đang thực hiện (chưa được kết thúc với rollback hay commit) trong connection hiện hành.

### Chú ý:

- Lệnh `rollback tran + tên_của_savepoint` có tác dụng quay lui (rollback) giao dịch đến vị trí đặt savepoint tương ứng (không có tác dụng kết thúc transaction), các khóa (locks) đã được thiết lập khi thực hiện các thao tác nằm trong phần bị rollback sẽ được mở ra (unlock).
- Khi khai báo transaction tường minh, phải đảm bảo rằng sau đó nó được `rollback` hoặc `commit` **tường minh**, nếu không, transaction sẽ tiếp tục tồn tại và chiếm giữ tài nguyên, ngăn trở sự thực hiện của các transaction khác.
- Lệnh `rollback` chỉ có tác dụng quay lui các giao dịch trên cơ sở dữ liệu (`insert`, `delete`, `update`). Các câu lệnh khác, chẳng hạn lệnh gán, sẽ không bị ảnh hưởng bởi lệnh `rollback`.

## 3.4 Ví dụ

## CHƯƠNG 4. QUẢN LÝ TƯƠNG TRANH

Trong ngành khoa học máy tính, **tương tranh** là một tính chất của các hệ thống bao gồm **các tính toán được thực thi trùng nhau** về mặt thời gian, trong đó các tính toán chạy đồng thời có thể chia sẻ các tài nguyên dùng chung.

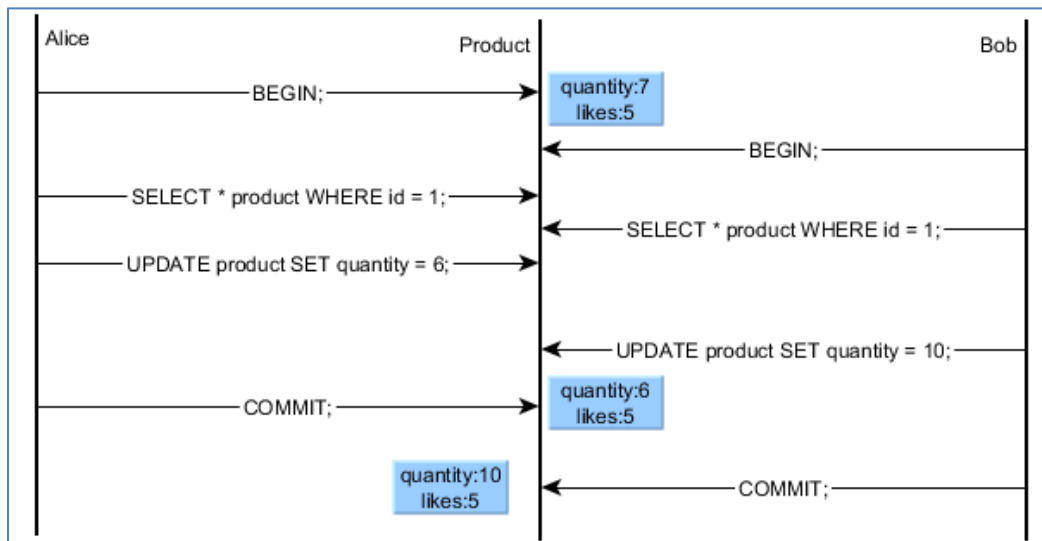
Trong hệ quản trị CSDL, tương tranh xảy ra khi có nhiều người cùng sử dụng CSDL một lúc.

### 4.1 Các vấn đề thường gặp khi có tương tranh

#### 4.1.1 Lost updates

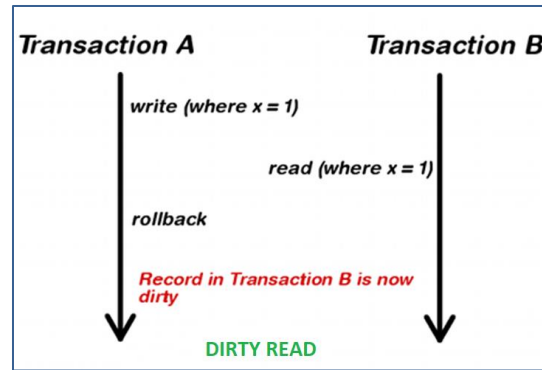
Hiện tượng lost updates xảy ra khi cùng một thời điểm có nhiều hơn một giao dịch cùng thực hiện cập nhật một cột của một bản ghi trong bảng. Kết quả là cập nhật của giao tác trước sẽ bị ghi đè bởi giao tác sau.

Hãy cùng phân tích ví dụ sau đây:

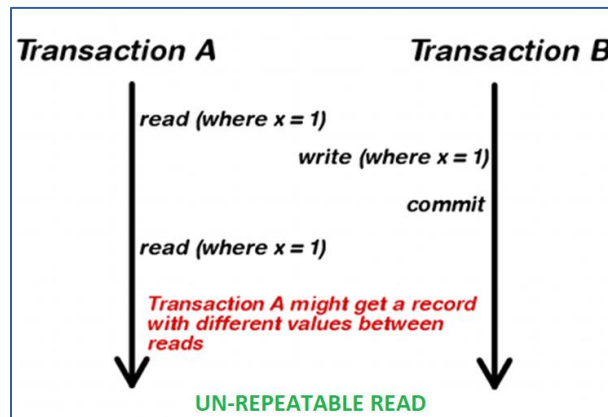


Hai giao tác xuất phát từ hai user Alice và Bob. Ban đầu, cả hai giao tác đều nhìn thấy số lượng sản phẩm là 7 (quantity: 7).

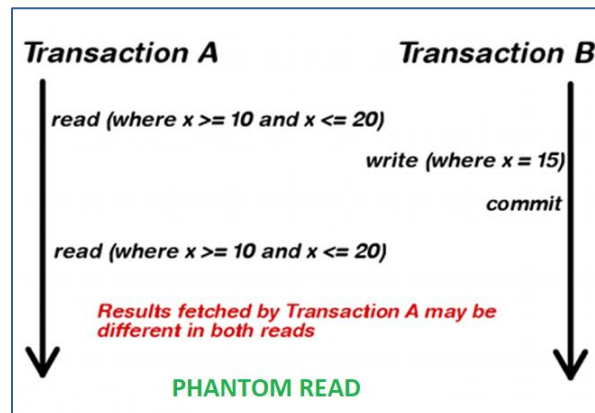
#### 4.1.2 Dữ liệu bẩn (Dirty reads)



#### 4.1.3 Dữ liệu đọc không lặp lại (Non-Repeatable Reads)



#### 4.1.4 Dữ liệu ảo (Phantom)



### 4.2 Giải quyết tương tranh

Để giải quyết các vấn đề tương tranh trong CSDL, MS SQL Server đưa ra các cấp độ cô lập.



Isolation level là một thuộc tính của transaction, qui định mức độ cô lập của dữ liệu mà transaction có thể truy nhập vào khi dữ liệu đó đang được cập bởi một transaction khác. Khi một transaction cập nhật dữ liệu đang diễn ra, một phần dữ liệu sẽ bị thay đổi (ví dụ một số bản ghi của bảng được sửa đổi hoặc bị xóa bỏ, một số được thêm mới), vậy các transaction hoặc truy vấn khác xảy ra đồng thời và cùng tác động vào các bản ghi đó sẽ diễn ra thế nào? Chúng sẽ phải đợi đến khi transaction đầu hoàn thành hay có thể thực hiện song song, kết quả dữ liệu nhận được là trong khi hay sau khi cập nhật? Bạn có thể điều khiển những hành vi này thông qua việc đặt isolation level của từng transaction. SQL Server cung cấp các mức isolation level sau xếp theo thứ tự tăng dần của mức độ cô lập của dữ liệu: Read Uncommitted, Read Committed, Repeatable Read, Serializable và Snapshot (từ phiên bản 2005 mới có Snapshot).

#### 4.2.1 Read Uncommitted

Khi transaction thực hiện ở mức này, các truy vấn vẫn có thể truy nhập vào các bản ghi đang được cập nhật bởi một transaction khác và nhận được dữ liệu tại thời điểm đó mặc dù dữ liệu đó chưa được commit (uncommitted data). Nếu vì lý do nào đó transaction ban đầu rollback lại những cập nhật, dữ liệu sẽ trở lại giá trị cũ. Khi đó transaction thứ hai nhận được dữ liệu sai. Hãy tìm hiểu qua ví dụ sau:

Mở đồng thời 2 cửa sổ soạn thảo query trên Microsoft SQL Server Management Studio (tạo 2 giao dịch chạy song song).

Cửa sổ thứ nhất gõ lệnh sau:

```
begin transaction
update transactions set t_amount = 100 where t_id = '0000000201'
waitfor delay '00:00:10' --chờ 10 giây
rollback
```

Trên cửa sổ thứ 2 gõ lệnh sau:

```
set transaction isolation level read uncommitted
select * from transactions where t_id = '0000000201'
```

#### ***Cách chạy lệnh và quan sát hiện tượng:***

Thực hiện lệnh ở cửa sổ thứ nhất, sau đó nhanh chóng chuyển sang cửa sổ thứ 2 để chạy lệnh.

Kết quả thu được khi chạy lệnh ở cửa sổ thứ 2:

Results		Messages				
	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	100	2011-12-27	07:45:00.0000000	1000000041

Chờ sau 10 giây, chạy lại lệnh ở cửa sổ thứ 2 ta thu được một kết quả khác:

Results		Messages				
	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	-1	2011-12-27	07:45:00.0000000	1000000041

### ***Giải thích hiện tượng:***

Khi đặt cấp độ cô lập của giao dịch là read **uncommitted** có nghĩa là: các tiến trình **đọc** không cần đợi đến khi tiến trình ghi hoàn tất mà có thể lấy dữ liệu ra được ngay. Nói nôm na là yêu cầu đọc của nó là “tôi không cần biết dữ liệu có đang được cập nhật hay không, hãy cho tôi dữ liệu hiện có ngay tại thời điểm này”. Do đó, khi chạy lệnh lần một ở cửa sổ thứ 2, chúng ta thu được dữ liệu đang trong quá trình xử lý, chưa phải là final. Sau 10 giây, transaction ở cửa sổ thứ nhất mới hoàn tất, lúc đó dữ liệu bị undo về trạng thái ban đầu, đây mới là dữ liệu final.

### ***Nhận xét***

Đây là cấp độ cô lập thấp nhất, nó cho phép độ tương tranh cao nhất (nhiều giao tác cùng làm việc) tuy nhiên sẽ gặp phải vấn đề dữ liệu bản trong quá trình truy vấn dữ liệu. Thường những hệ thống đòi hỏi tính tương tranh cao mà không cần quan tâm đến vấn đề đọc dữ liệu “sạch” hay “bẩn” thì nên thiết lập mức độ này.

### **4.2.2 Read Committed**

Đây là mức cô lập mặc định, nếu không đặt gì cả thì transaction sẽ hoạt động ở mức này. Transaction sẽ không đọc được dữ liệu đang được cập nhật mà phải **đợi đến khi việc cập nhật thực hiện xong**. Vì thế nó tránh được dirty read như ở mức trên. Giờ hãy sửa lại đoạn lệnh ở cửa sổ thứ hai thành:

Ví dụ:

Hãy sửa lệnh ở cửa sổ thứ 2 thành:

```
set transaction isolation level read committed
select * from transactions where t_id = '0000000201'
```

Tương tự như trên, hãy chạy lệnh ở cửa sổ thứ nhất và nhanh chóng chuyển sang cửa sổ thứ 2 để chạy lệnh.

**Hiện tượng:** Phải chờ sau 10 giây mới nhìn thấy kết quả.

Results		Messages				
	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	-1	2011-12-27	07:45:00.0000000	1000000041

### Giải thích

Do chúng ta thiết lập lại chế độ cô lập là **committed** nên lệnh ở cửa sổ thứ 2 phải chờ cho transaction ở cửa sổ thứ nhất chạy xong nó mới được phép đọc dữ liệu.

### Nhận xét

Cấp độ cô lập này giải quyết được vấn đề: dirty read và **phantom**.

### 4.2.3 Repeatable read

Mức cô lập này hoạt động như mức **read commit** nhưng nâng thêm một nấc nữa bằng cách ngăn không cho transaction ghi vào dữ liệu đang được đọc bởi một transaction khác cho đến khi transaction khác đó hoàn tất.

### Ví dụ:

Trở lại hai cửa sổ:

Cửa sổ thứ nhất gõ lệnh sau:

```
begin transaction
select * from transactions where t_id = '0000000201'
waitfor delay '00:00:10'
select * from transactions where t_id = '0000000201'
commit
```

Cửa sổ thứ 2 gõ lệnh sau:

```
set transaction isolation level repeatable read
update transactions set t_amount = 100 where t_id = '0000000201'
select * from transactions where t_id = '0000000201'
```

### Chạy lệnh:

Chạy lệnh ở cửa sổ thứ nhất, sau đó chạy lệnh ở cửa sổ thứ 2. Quan sát kết quả ở 2 cửa sổ 1 và 2.

### Hiện tượng:

Hai lệnh select ở cửa sổ 1 cho cùng kết quả và cửa sổ 2 phải đợi đến khi cửa sổ 1 hoàn tất mới được thực hiện.

Kết quả của cửa sổ 1:

	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	100	2011-12-27	07:45:00.0000000	1000000041

	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	100	2011-12-27	07:45:00.0000000	1000000041

Kết quả thu được ở cửa sổ 2:

	t_id	t_type	t_amount	t_date	t_time	ac_no
1	0000000201	1	150	2011-12-27	07:45:00.0000000	1000000041

### ***Giải thích:***

Ở cấp độ cô lập này, dữ liệu bị tác động bởi transaction trong cửa sổ 1 bị khóa, không cho câu lệnh ở cửa sổ 2 sửa đổi. Sau khi transaction ở cửa sổ 1 thực hiện xong, lúc đó lệnh ở cửa sổ 2 mới được phép thực hiện.

### ***Nhận xét:***

Cấp độ cô lập này giải quyết được vấn đề phantom.

#### **4.2.4 Serializable**

Mức isolation này tăng thêm một cấp nữa và khóa toàn bộ dải các bản ghi có thể bị ảnh hưởng bởi một transaction khác, dù là UPDATE/DELETE bản ghi đã có hay INSERT bản ghi mới.

### ***Ví dụ:***

Cửa sổ 1:

```
set transaction isolation level serializable
begin tran
select * from bank
waitfor delay '00:00:10'
select * from bank
commit
```

Cửa sổ 2:

```
insert into bank values('001','Vietinbank','124 Tran Phu')
select * from bank
```

**Chạy lệnh:**

Chạy lệnh ở cửa sổ 1 và chuyển sang cửa sổ 2 để chạy lệnh.

**Hiện tượng:**

Cả hai cửa sổ bị treo cho đến khi cửa sổ 1 thực hiện xong. Câu lệnh SELECT ở hai cửa sổ trả về kết quả giống nhau.

Results Messages			
	b_id	b_name	b_ad
1	001	Vietinbank	124 Tran Phu
2	BFTVVNVX07	Ngân hàng Công thương Việt Nam	03 Lý Thái Tổ, Hà Nội, Việt Nam
3	VB001	Vietcombank Thái Bình	12 Lý Bôn - tp Thái Bình

#### 4.2.5 Snapshot

Mức độ này cũng đảm bảo độ cô lập tương đương với Serializable, nhưng nó hơi khác ở phương thức hoạt động. Khi transaction đang select các bản ghi, nó không khóa các bản ghi này lại mà tạo một bản sao (snapshot) và select trên đó. Vì vậy các transaction khác insert/update lên các bản ghi đó không gây ảnh hưởng đến transaction ban đầu. Tác dụng của nó là giảm blocking giữa các transaction mà vẫn đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên cái giá kèm theo là cần thêm bộ nhớ để lưu bản sao của các bản ghi, và phần bộ nhớ này là cần cho mỗi transaction do đó có thể tăng lên rất lớn. Để thiết lập isolation mức này cần đặt lại option của database:

```
ALTER DATABASE TestDB
SET ALLOW_SNAPSHOT_ISOLATION ON
```

**Nhận xét:**

Tốn bộ nhớ.

#### 4.2.6 Phạm vi áp dụng

Các mức cô lập từ 1 – 4 kể trên tăng theo thứ tự mức độ cô lập dữ liệu, giúp tăng tính toàn vẹn dữ liệu và nhất quán của transaction. Đồng thời nó cũng tăng thời gian chờ lẫn nhau của các transaction. Khi càng lên mức cao, đòi hỏi về tính toàn vẹn dữ liệu càng cao và càng có nhiều tình huống một transaction ngăn không cho các transaction khác truy

nhập vào dữ liệu mà nó đang thao tác. Do đó nó càng tăng tình trạng locking và blocking trong database (ngoại trừ với snapshot thì tăng lượng bộ nhớ cần sử dụng). Hiệu năng của hệ thống do đó bị giảm đi. Thông thường, mức cô lập read committed (mức mặc định) là phù hợp trong đa số các ứng dụng. Có thể một vài chức năng quan trọng (ví dụ chức năng ở trang admin update dữ liệu có ảnh hưởng đến toàn hệ thống) cần tính toàn vẹn cao và phải chọn mức cô lập cao hơn. Hoặc có những chức năng cần ưu tiên tốc độ thực hiện và có thể chấp nhận một chút dữ liệu không nhất quán thì có thể đặt xuống mức read uncommitted. Bảng dưới đây tóm tắt các tính năng của từng mức cô lập.

Mức cô lập	Lost update	Dirty read	Nonrepeatable read	Phantom read
<b>Read Uncommitted</b>	Yes	Yes	Yes	Yes
<b>Read Committed</b>	Yes	No	Yes	Yes
<b>Repeatable read</b>	No	No	No	Yes
<b>Serializable</b>	No	No	No	No
<b>Snapshot</b>	No	No	No	No

**Chú thích:**

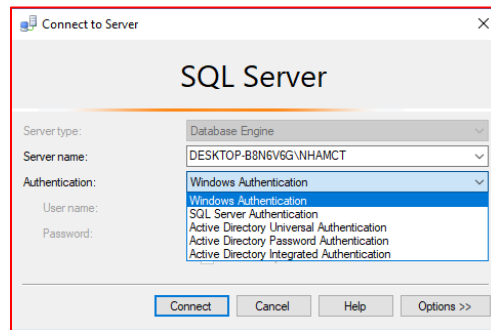
- YES: xảy ra hiện tượng
- NO: không xảy ra hiện tượng

## CHƯƠNG 5. TÍNH NĂNG BẢO MẬT TRONG SQL SERVER 2016

### 5.1 Quản lý người dùng

#### 5.1.1 Xác thực người dùng

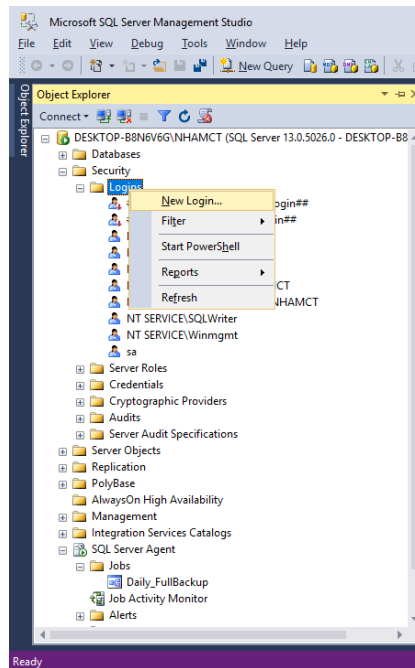
Xác thực người dùng (authentication) là việc xác định danh tính người dùng. Về cơ bản, SQL Server 2016 cung cấp 5 phương thức xác thực người dùng.



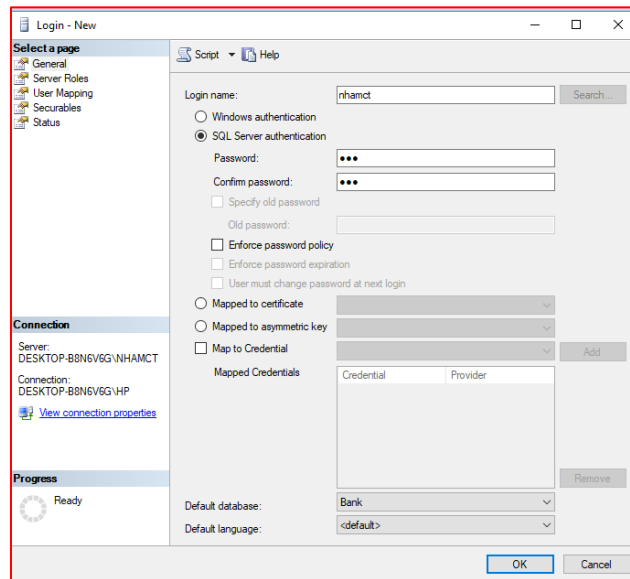
- Windows Authentication
- SQL Server Authentication
- Active Directory Universal Authentication
- Active Directory Password Authentication
- Active Directory Integrated Authentication

#### 5.1.2 Login

Để kết nối tới cơ sở dữ liệu, người dùng cần thiết lập một tài khoản (trong SQL Server 2016 gọi là Login).

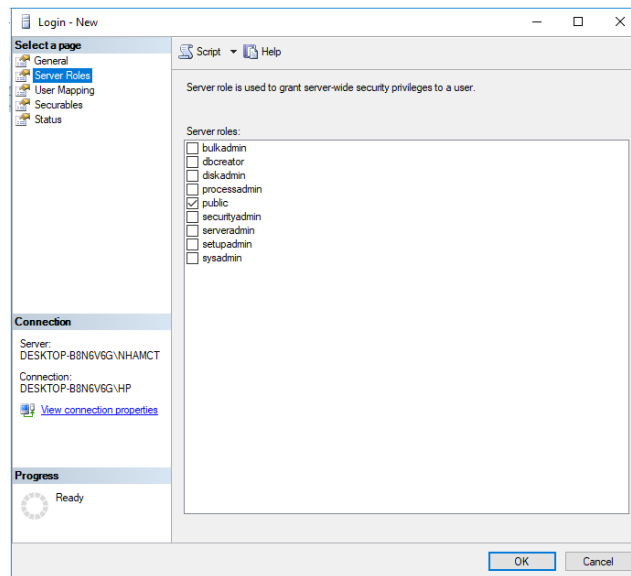


Bước 1. Thiết lập thông tin cơ bản về login bao gồm: tên đăng nhập (login name), phương thức xác thực (windows, SQL server, Mapped to certificate, mapped to asymmetric key,...). Thường sẽ chọn SQL Server authentication.



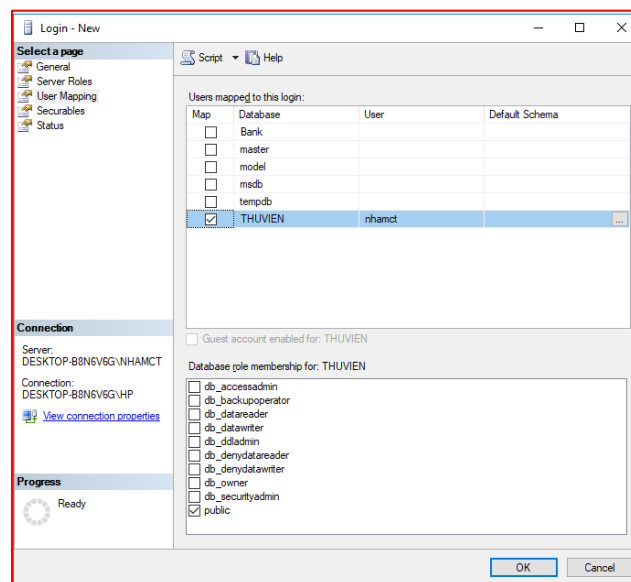
Bước 2. Thiết lập quyền trên server (server roles)





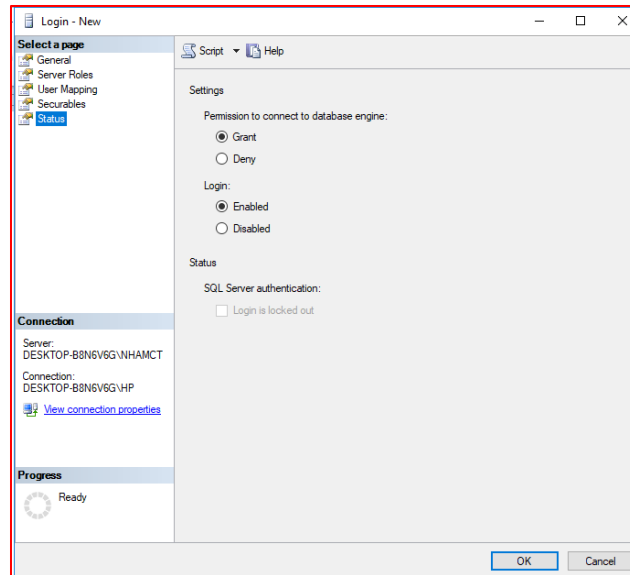
Hãy nhớ nguyên tắc: đặt ít quyền nhất có thể cho mỗi người dùng.

Bước 3: Thiết lập quyền trên cơ sở dữ liệu (user mapping)



Chọn tên cơ sở dữ liệu mà người dùng sẽ làm việc và quyền của họ trên cơ sở dữ liệu.

Bước 4: Thiết lập trạng thái



## 5.2 Bảo mật dữ liệu

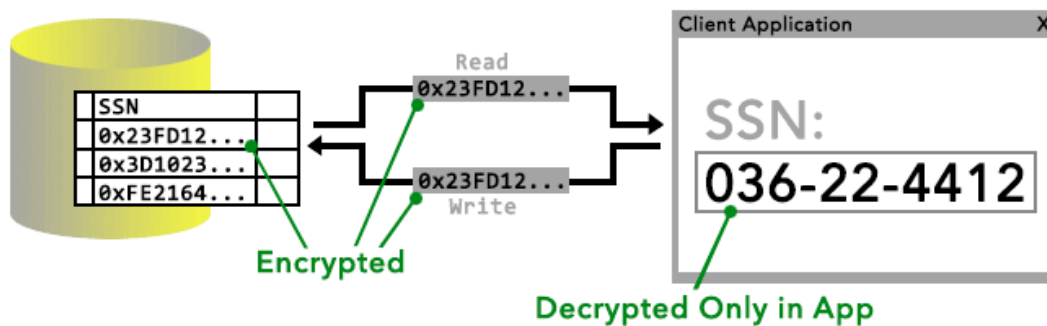
<https://www.microsoft.com/en-us/sql-server/data-security>

SQL Server 2016 cung cấp cơ chế bảo mật dữ liệu như sau:

- Always Encrypted
- Row-Level Security
- Dynamic Data Masking
- Transparent Data Encryption
- SQL Server Audit

### 5.2.1 Always Encrypted (mã hóa cấp độ cột)

Là kỹ thuật dùng để bảo vệ dữ liệu nhạy cảm như số thẻ tín dụng, mật khẩu,...



Một số thuật ngữ quan trọng:

- Column Master Key: đây là một khóa dùng để mã hóa cột. Phải có ít nhất một khóa để thực hiện mã hóa.
- Column Encryption Key
- Column-level encryption setting
  - Deterministic
  - Randomized

### **5.2.2 Thực hiện mã hóa cột**

## CHƯƠNG 6. SAO LƯU VÀ PHỤC HỒI

### 6.1 Sao lưu (backup)

#### 6.1.1 Tổng quan

SQL Server 2012 cung cấp 2 loại sao lưu (backup):

- *Full backup (sao lưu đầy đủ)*: thực hiện backup toàn bộ các đối tượng, bảng hệ thống và dữ liệu trong cơ sở dữ liệu.
- *Differential backup (sao lưu những khác biệt)*: chỉ backup dữ liệu thay đổi kể từ lần full backup trước đó. Loại backup này tốn ít thời gian hơn full backup.
- *Transaction log backup*: lưu trữ những thay đổi kể từ transaction log backup trước đó và loại trừ đi những transaction đã được commit vào cơ sở dữ liệu.
- *File and file group backup*: thực hiện backup file hoặc nhóm file theo yêu cầu của người quản trị.
- *Copy-only backup*: tương tự như full backup hoặc transaction log backup, tuy nhiên loại backup này không làm thay đổi thứ tự backup. Copy-only backup không được dùng để làm cơ sở cho differential backup.

Người quản trị có thể lựa chọn nhiều loại backup cho chiến lược backup của mình, ví dụ: đối với cơ sở dữ liệu có sự thay đổi dữ liệu thường xuyên có thể thực hiện full backup hàng ngày và differential backup theo từng giờ. Chiến lược backup phụ thuộc vào cách thức phục hồi dữ liệu.

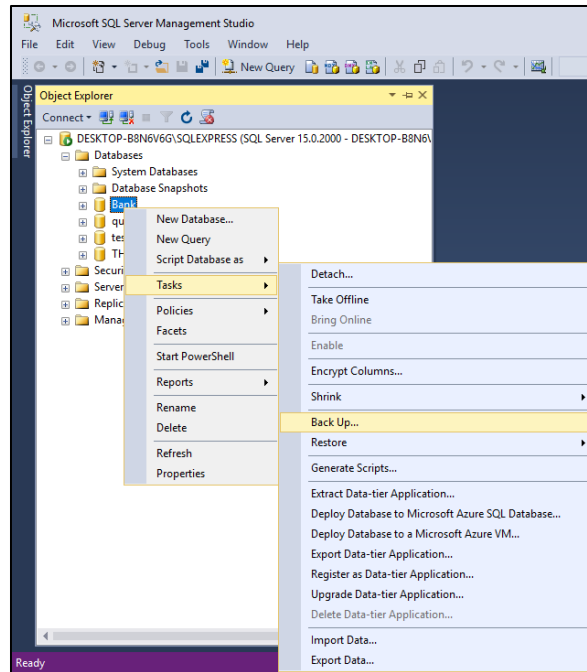
Một số hạn chế trong khi thực hiện backup:

- Không thể backup khi dữ liệu đang offline.
- Những lệnh thay đổi cấu trúc cơ sở dữ liệu như ALTER DATABASE (ADD FILE hoặc REMOVE FILE) sẽ chỉ được thực hiện sau khi lệnh backup chạy xong.

#### 6.1.2 Thực hiện backup

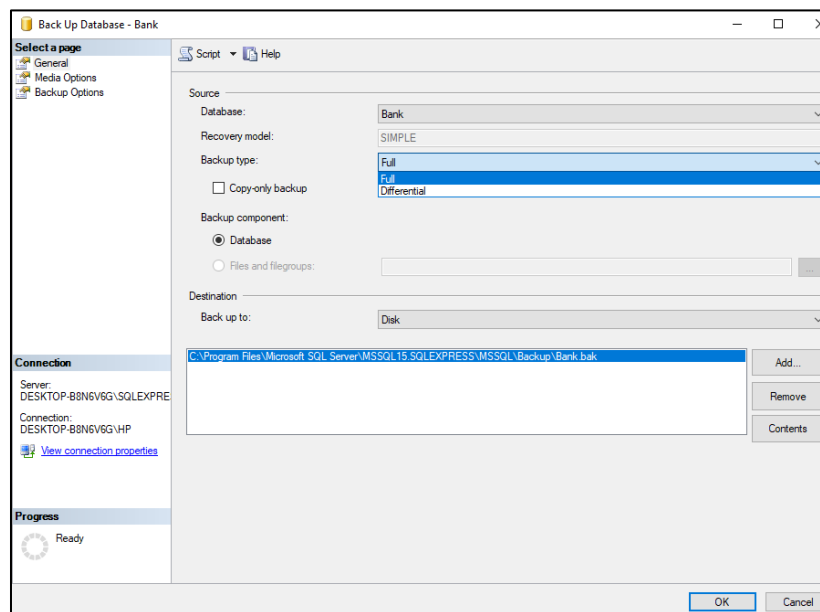
Các thao tác thực hiện backup thực hiện trên SQL Server Management Studio như sau:

Bước 1: Lựa chọn thao tác backup



Hình 6-1 Lựa chọn thao tác backup

## Bước 2: Thiết lập các tùy chọn backup



Hình 6-2 Thiết lập tùy chọn backup

Một số chú ý:

- Database: lựa chọn tên cơ sở dữ liệu backup
- Backup type: có thể lựa chọn Full hoặc Differential (chỉ lựa chọn differential khi đã thực hiện ít nhất một lần full trước đó).

- Backup To: có thể lựa chọn lưu bản backup vào ổ đĩa cứng hoặc lưu lên một máy tính khác trên internet.

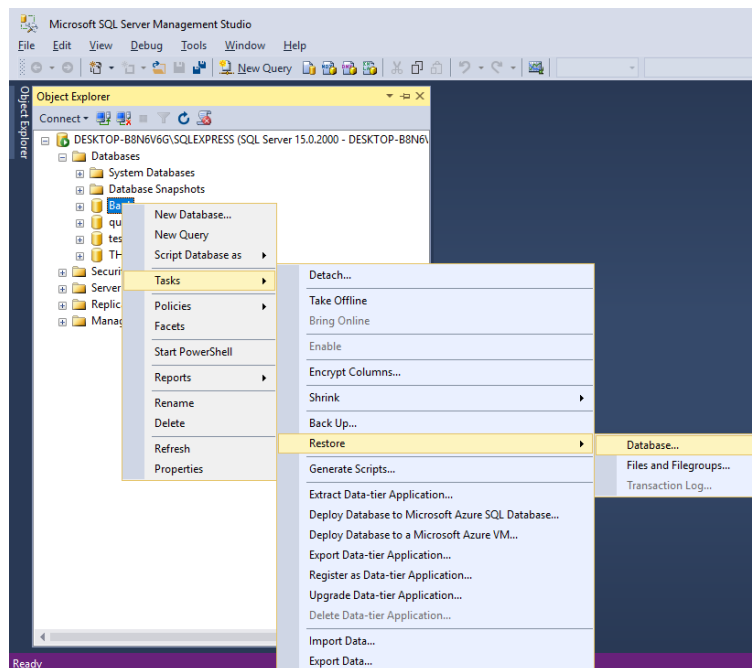
### 6.1.3 Quản lý lịch sử backup

## 6.2 Phục hồi dữ liệu

Tùy chọn phục hồi dữ liệu tùy thuộc vào loại backup đã thực hiện trước đó. Ví dụ: trước đó chưa thực hiện transaction log backup thì không thể thực hiện phục hồi dạng transaction log được.

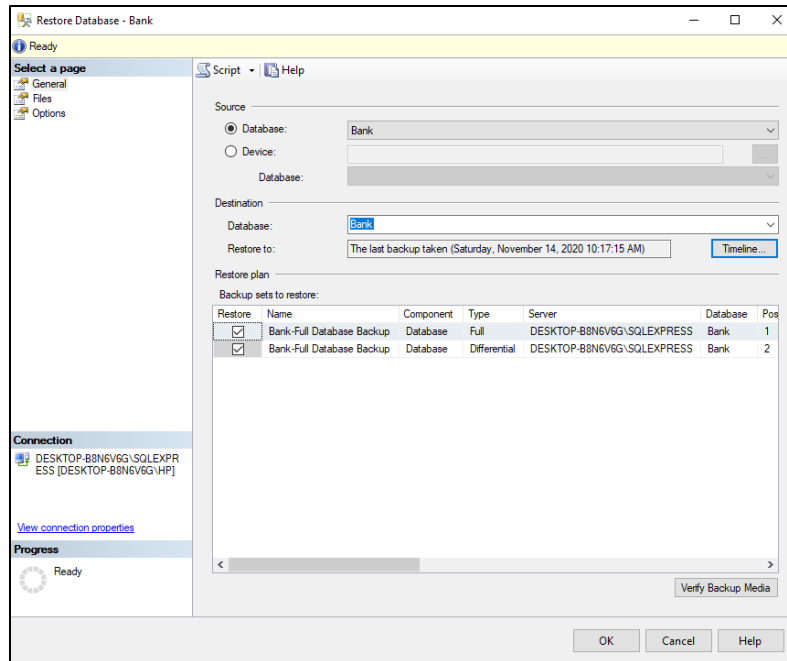
Các bước thực hiện sao lưu như sau:

Bước 1: Lựa chọn loại phục hồi



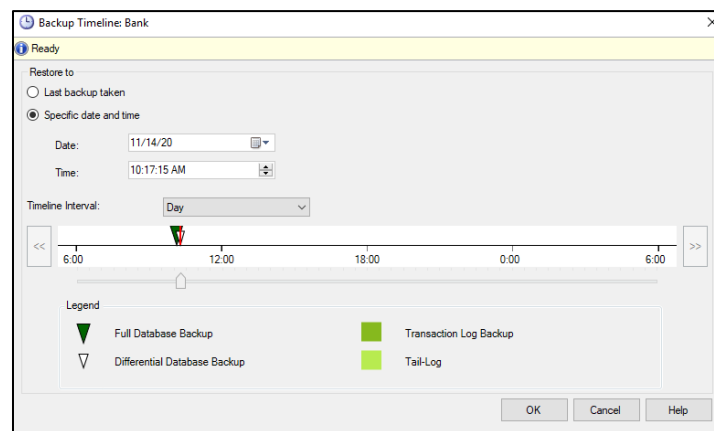
Hình 6-3 Lựa chọn loại phục hồi

Bước 2: Thiết lập tùy chọn phục hồi



Hình 6-4 Thiết lập tùy chọn phục hồi

Trong cửa sổ này, người dùng sẽ nhìn thấy các bản backup và có thể lựa chọn phục hồi dữ liệu về một thời điểm nhất định trong khoảng thời gian kể từ khi thực hiện backup (như hình dưới).



Hình 6-5 Lựa chọn thời gian phục hồi

## CHƯƠNG 7. THIẾT LẬP THỜI GIAN THỰC HIỆN CÔNG VIỆC

Trong quản trị cơ sở dữ liệu, có nhiều việc cần được thực hiện vào khung giờ nhất định hàng ngày, hàng tuần hoặc hàng tháng hay thậm chí là thực hiện vào một thời điểm nhất định, ví dụ như việc backup.

Có hai đối tượng cần lưu ý khi thiết lập thời gian thực hiện: Job và Step.

- Step: một lệnh nào đó cần được thực hiện. Thường nằm trong Job, chứ không đứng độc lập.
- Job: bao gồm một hoặc nhiều step.

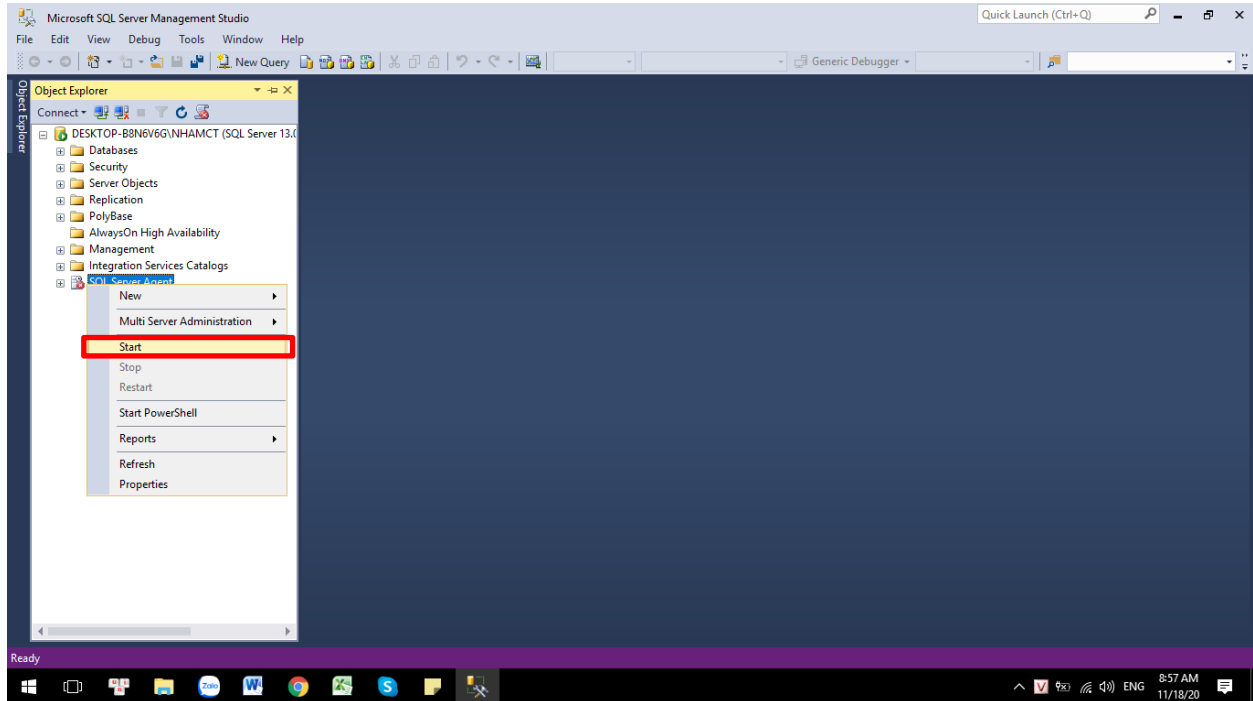
Job thường thiết lập để chạy:

- Thường kì: hàng ngày, hàng tuần, hàng tháng
- Một thời điểm xác định
- Sau một khoảng thời gian xác định (ví dụ: cứ 10 phút chạy 1 lần)
- Khi CPU ở trạng thái idle
- Khi SQL Server Agent bắt đầu chạy
- Khi cần đưa ra phản hồi cho một thông báo.

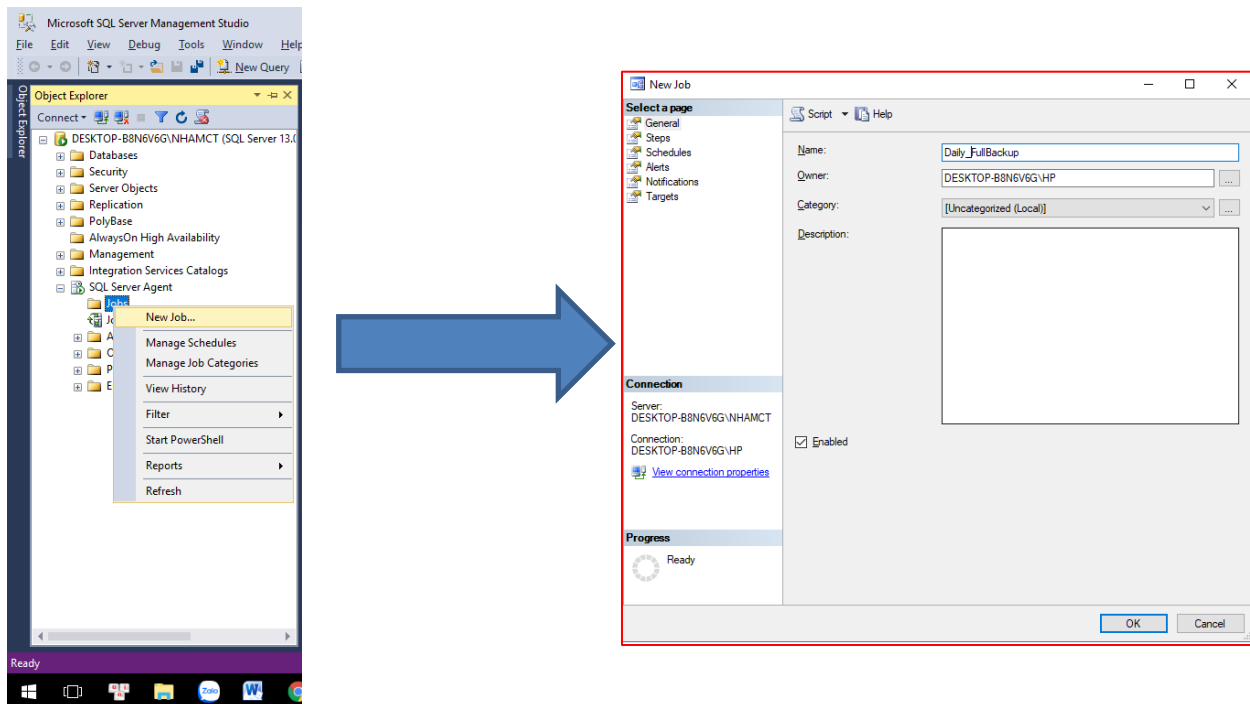
Sử dụng SQL Server Management Studio để thiết lập Job như sau:

Bước 1: Bật SQL Server Agent

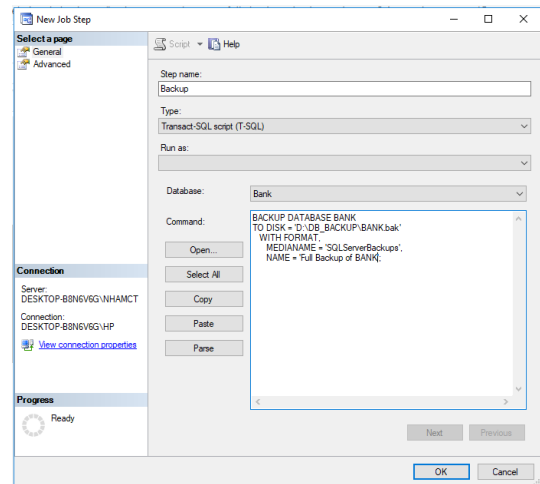
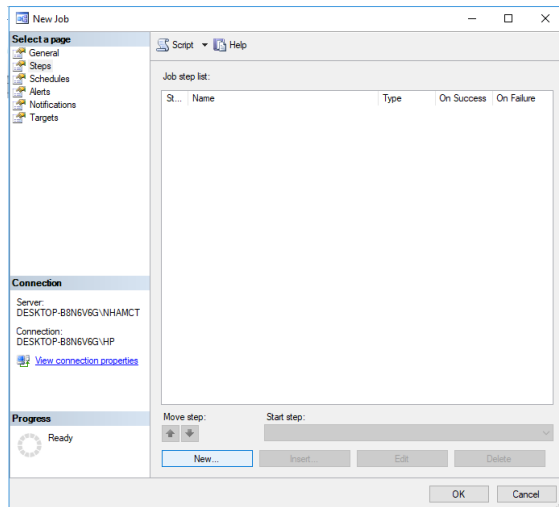




## Bước 2: Tạo job mới



## Bước 3: Tạo các step trong job



#### Bước 4: Thiết lập thời gian

