**SQL Server**

# **Khái niệm SQL**

SQL là gì?

SQL viết tắt của từ Structured Query Language (ngôn ngữ truy vấn có cấu trúc). Ngôn ngữ SQL được dùng để tương tác với cơ sở dữ liệu, từ truy vấn lấy thông tin, đến cập nhật (thêm mới, cập nhật thông tin mới, xóa ...).

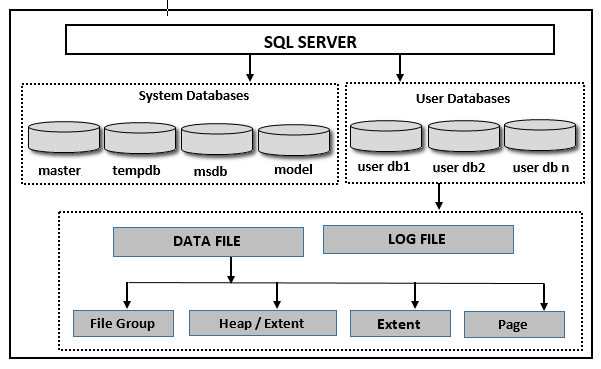
Với SQL bạn có thể:

* Chèn, cập nhật, xóa các dòng dữ liệu
* Tạo ra cơ sở dữ liệu mới
* Lấy thông tin CSDL ...

Ngôn ngữ SQL là một tiêu chuẩn ANSI (American National Standards Institute), có một số sự khác nhau nhỏ cho từng phiên bản của ngôn ngữ này.

Hầu hết các hệ quản trị CSDL quan hệ hiện nay đều hỗ trợ các lệnh chính của SQL

# **Cấu trúc của SQL Server**



SQL Server bao gồm 5 cơ sở dữ liệu (CSDL) hệ thống (system databases) và một hay nhiều user database. Các system databases bao gồm:

* **Cơ sở dữ liệu Master**: Chứa tất cả những thông tin cấp hệ thống (system-level information) bao gồm thông tin về các cơ sở dữ liệu, bảng, quyền người dùng, cấu hình hệ thống, và các đối tượng khác liên quan đến hệ thống.
* **Cơ sở dữ liệu Tempdb**: Chứa tất cả những table hay stored procedure được tạm thời tạo ra trong quá trình làm việc bởi user hay do bản thân SQL Server engine. Các table hay stored procedure này sẽ biến mất khi khởi động lại SQL Server hay khi ta disconnect
* **Cơ sở dữ liệu Model**: Database này đóng vai trò như một bảng mẫu (template) cho các database khác. Nghĩa là khi một user database được tạo ra thì SQL Server sẽ copy toàn bộ cấu trúc của system objects (tables, stored procedures…) từ Model database sang database mới vừa tạo
* **Cơ sở dữ liệu Msdb**: được sử dụng cho SQL Server Agent để lập lịch các công việc và các cảnh báo (schedule alerts and jobs)

### Cơ sở dữ liệu Master

Chứa thông tin về cấu hình, thiết lập và các đối tượng khác của một hệ thống quản trị cơ sở dữ liệu (DBMS).

* Thông tin về cấu hình máy chủ và cơ sở dữ liệu
* Quản lý bảo mật và phân quyền
* Quản lý các kết nối tới cơ sở dữ liệu
* Quản lý các tài khoản người dùng và vai trò
* Nhật ký hoạt động và các cảnh báo

### Cơ sở dữ liệu Tempdb

Lưu trữ các bảng tạm thời và các đối tượng khác khi các câu lệnh T-SQL của người dùng được thực thi. Mặc định của temporary sẽ bị xóa khỏi hệ thống khi một phiên làm việc chấm dứt. Không có gì khác so với một table bình thường. Tên của bảng tạm phải bắt đầu bằng dấu thăng #.

* Lưu trữ các bảng tạm thời
* Lưu trữ các biến tạm thời
* Hỗ trợ tính toán song song: Khi các truy vấn được thực thi song song, Tempdb cung cấp một không gian lưu trữ chung để lưu trữ các bảng tạm thời và đối tượng khác được sử dụng bởi các tiến trình song song này

### Cơ sở dữ liệu Model

Cơ sở dữ liệu quan hệ (Relational Database), là một tập hợp các bảng (Tables) có liên kết với nhau thông qua các khóa ngoại (Foreign Keys).

### Cơ sở dữ liệu Msdb

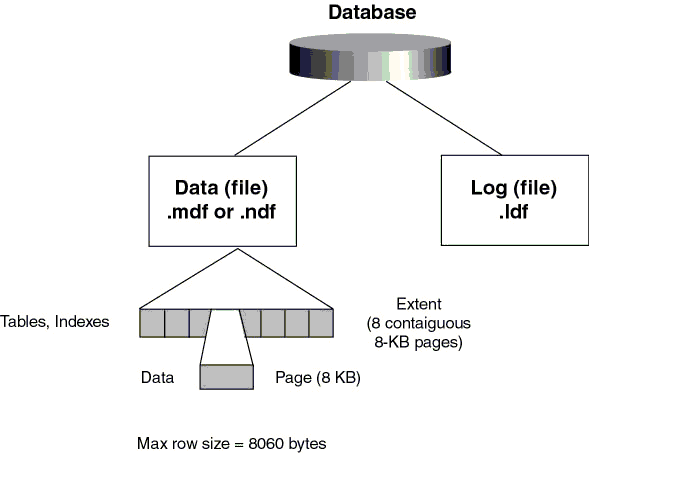
Lưu trữ thông tin về các giao dịch, bảo trì, backup và restore, và các công việc lập lịch trong SQL Server. Các đối tượng quan trọng trong Msdb bao gồm các bảng như sysjobs, sysjobhistory, sysjobsteps, và sysjobschedules, các stored procedure như sp\_start\_job và sp\_help\_job, và các hàm như fn\_trace\_gettable và fn\_get\_sql.

* Quản lý các công việc lập lịch, bao gồm backup, restore, thực thi các câu lệnh T-SQL, và các tác vụ khác
* Lưu trữ lịch sử các hoạt động của các công việc, bao gồm thành công hay thất bại, thời gian thực thi, và các thông tin khác.
* Cung cấp thông tin về các bản sao lưu (backup) và phục hồi (restore) của cơ sở dữ liệu.
* Cung cấp thông tin về các tập tin log và tập tin kiểm tra của cơ sở dữ liệu.

### Cơ sở dữ liệu Resource

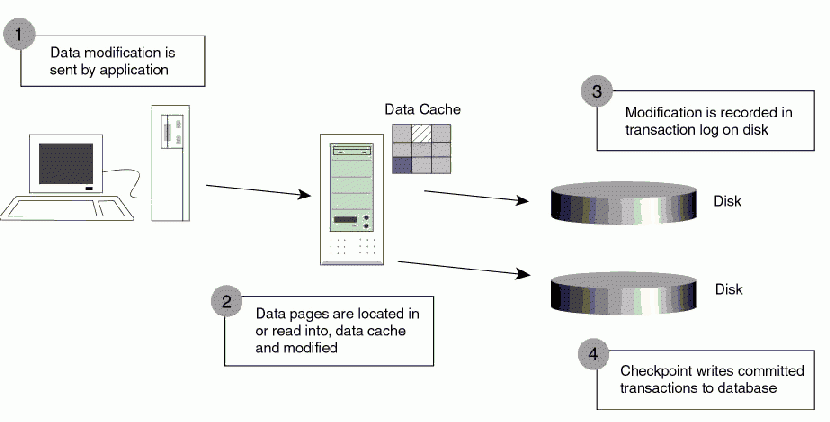
* Các hàm mật mã: được sử dụng để mã hóa mật khẩu trong SQL Server.
* Stored procedures: được sử dụng để thực hiện các tác vụ quản lý hệ thống như backup, restore, và kiểm tra sao lưu (backup) của cơ sở dữ liệu.
* Các trigger hệ thống: được sử dụng để theo dõi và xử lý các sự kiện quan trọng như sự kiện đăng nhập vào hệ thống hoặc thay đổi trạng thái của một database.

# **Cấu trúc vật lý của một CSDL SQL Server**



* Primary data file: (thường có phần mở rộng .mdf): đây là file chính chứa data và những system tables.
* Secondary data file (thường có phần mở rộng .ndf): đây là file phụ thường chỉ sử dụng khi database được phân chia để chứa trên nhiều dĩa.
* Transaction log file (thường có phần mở rộng .ldf): đây là file ghi lại tất cả những thay đổi diễn ra trong một database và chứa đầy đủ thông tin để có thể roll back hay roll forward khi cần.

### Nguyên tắc hoạt động của Transaction Log trong SQL Server



Quá trình này diễn ra như sau: đầu tiên khi có một sự thay đổi dữ liệu như Insert, Update, Delete được yêu cầu từ các ứng dụng, SQL Server sẽ tải (load) trang dữ liệu (data page) tương ứng vào bộ nhớ (vùng bộ nhớ này gọi là data cache), sau đó dữ liệu trong data cache được thay đổi (những trang bị thay đổi còn gọi là dirty-page). Tiếp theo mọi sự thay đổi đều được ghi vào transaction log file cho nên người ta gọi là write-ahead log. Cuối cùng thì một quá trình gọi là Check Point Process sẽ kiểm tra và viết tất cả những transaction đã được hoàn tất (commited) vào đĩa cứng (flushing the page).

Transaction Log có nhiều chức năng quan trọng trong SQL Server, bao gồm

* Khôi phục dữ liệu: Transaction Log cho phép khôi phục lại các hoạt động thay đổi dữ liệu trong trường hợp có sự cố xảy ra. Chẳng hạn như, nếu máy chủ của bạn bị sập hoặc bị mất dữ liệu, bạn có thể sử dụng Transaction Log để khôi phục lại các thay đổi và đưa cơ sở dữ liệu của bạn về trạng thái trước khi sự cố xảy ra.
* Replication: Transaction Log cũng được sử dụng để sao chép dữ liệu giữa các máy chủ SQL Server khác nhau. Các thay đổi dữ liệu được ghi vào Transaction Log có thể được sao chép qua mạng và áp dụng vào các cơ sở dữ liệu khác.
* Recovery model: Transaction Log cũng liên quan đến khả năng phục hồi của cơ sở dữ liệu. Cấu hình Recovery model cho phép bạn quản lý cách mà Transaction Log được quản lý và ghi lại các thay đổi dữ liệu.

# **D. Chuẩn hóa dữ liệu trong SQL**

**Chuẩn hóa dữ liệu là gì?**

Chuẩn hóa là một kỹ thuật thiết kế cơ sơ dữ liệu tổ chức các bảng theo cách làm giảm sự dư thừa và phụ thuộc của dữ liệu

Nó chia các bảng lớn thành các bảng nhỏ hơn và liên kết bằng các quan hệ

**Khóa chính**

Một khóa là một giá trị được sử dụng để xác định một bản ghi trong một bảng duy nhất. Một key có thể là một cột đơn hoặc kết hợp nhiều cột. Trong một bảng chỉ có 1 khóa chính duy nhất

Khóa chính là là một trường hoặc tập hợp các trường trong cơ sở dữ liệu mà giá trị của nó là định danh duy nhất mỗi bản ghi trong bảng

Các thuộc tính của khóa chính:

* Khóa chính không thể NULL
* Giá trị khóa chính là duy nhất
* Các giá trị khóa chính hiếm khi được thay đổi
* Khóa chính phải được cung cấp giá trị khi một bản ghi mới được thêm vào

**Khóa ngoại**

Khóa ngoại là khóa dùng để tham chiếu khóa chính của bảng khác dùng để kết nối các bảng

* Khóa ngoại có thể có tên khác với khóa chính của nó
* Nó đảm bảo các hàng trong một bảng có các hàng tương ứng trong bảng khác
* Khóa ngoại không phải là duy nhất
* Khóa ngoại có thể NULL

**Khóa tổng hợp**

Khóa tổng hợp là khóa chính gồm nhiều cột được sử dụng để xác định một bản ghi duy nhất

**Các dạng chuẩn hóa dữ liệu**

1. **Quy tắc 1NF**

* Mỗi ô trong bảng nên chứa một giá trị đơn (không phải một danh sách)
* Mỗi dòng (bản ghi) cần phải là duy nhất

Ví dụ:

Thiết kế vi phạm chuẩn NF1:

|  |  |  |  |
| --- | --- | --- | --- |
| **Customer ID** | **First Name** | **Surname** | **Telephone Number** |
| 123 | Pooja | Singh | 555-861-2025, 192-122-1111 |
| 456 | San | Zhang | (555) 403-1659 Ext. 53; 182-929-2929 |
| 789 | John | Doe | 555-808-9633 |

Trong ví dụ trên, thiết kế này đã vi phạm chuẩn NF1 vì có tới 2 giá trị nằm trong cột Telephone Number

Để đưa về dạng chuẩn NF1, ta cần chia ra các số điện thoại thành các bản ghi khác nhau và đảm bảo các hàng không chứa nhiều hơn 1 số điện thoại

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Customer Name** | | | | **Customer ID** | **First Name** | **Surname** | | 123 | Pooja | Singh | | 456 | San | Zhang | | 789 | John | Doe | | |  |  |  | | --- | --- | --- | | **Customer Telephone Number** | | | | **Id** | **Customer ID** | **Telephone Number** | | 1 | 123 | 555-861-2025 | | 2 | 123 | 192-122-1111 | | 3 | 456 | (555) 403-1659 Ext. 53 | | 4 | 456 | 182-929-2929 | | 5 | 789 | 555-808-9633 | |

1. **Quy tắc 2NF**

* Đáp ứng quy tắc 1NF
* Mọi thuộc tính không phải là khóa trong bảng phải phụ thuộc vào toàn bộ khóa chính
* VD: VD: Employees(**EmpID**,EmpName,DepartmentName, DepartmentLocation)

🡪 DepartmentName phụ thuộc vào DepartmentLocation vì nếu địa điểm của 1 phòng thay đổi thì thông tin phòng sẽ thay đổi cho tất cả employee thuộc phòng đó

🡪 Tách thành 2 bảng Employees(**EmpID**, EmpName, *DepartmentID*) và Departments(**DepartmentID**, DepartmentName, DepartmentLocation)

Bảng Order

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **id** | customer\_id | product\_id | customer\_name | product\_name | Quantity |
| 1 | 1 | 1 | Long | Laptop | 1 |
| 2 | 2 | 2 | Thanh | Phone | 2 |
| 3 | 2 | 1 | Thanh | Laptop | 3 |

Bảng trên kia tuân thủ theo 1NF vì không có thuộc tính đa trị hay thuộc tính lặp nhưng vẫn chưa đạt chuẩn 2NF vì customerName phụ thuộc vào customerID chứ không phải orderID (khóa chính), tương tự đối với productName phụ thuộc vào productID.

Để xử lý trường hợp này, ta tách bảng trên thành các bảng nhỏ hơn như sau:

Bảng Customer

|  |  |
| --- | --- |
| Id | customer\_name |
| 1 | Long |
| 2 | Thanh |

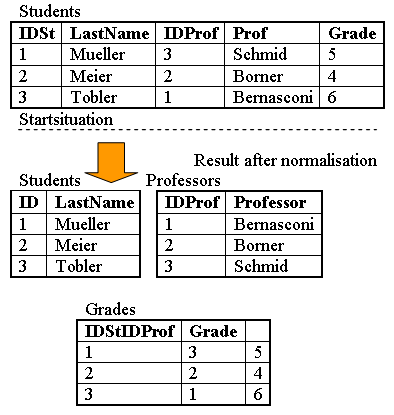
Bảng Product

|  |  |
| --- | --- |
| Id | product\_name |
| 1 | Laptop |
| 2 | Phone |

Bảng Order

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Customer\_id | product\_id | Quantity |
| 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 2 |
| 3 | 2 | 2 | 3 |

VD2:



*Bảng trong ví dụ này ở dạng chuẩn đầu tiên (1NF) vì tất cả các thuộc tính đều có giá trị đơn. Nhưng nó vẫn chưa có trong 2NF. Nếu sinh viên 1 rời trường đại học và bộ dữ liệu bị xóa, thì chúng tôi sẽ mất tất cả thông tin về giáo sư Schmid, vì thuộc tính này hoàn toàn phụ thuộc vào khóa chính IDSt.*

1. **Quy tắc 3NF**

* Đáp ứng quy tắc 2NF
* Các thuộc tính không khóa không được phụ thuộc bắc cầu vào khóa chính (xảy ra khi một thuộc tính không phụ thuộc trực tiếp vào khóa chính của bảng, nhưng lại phụ thuộc vào một thuộc tính khác qua một hoặc nhiều phụ thuộc chức năng)

Ví dụ:

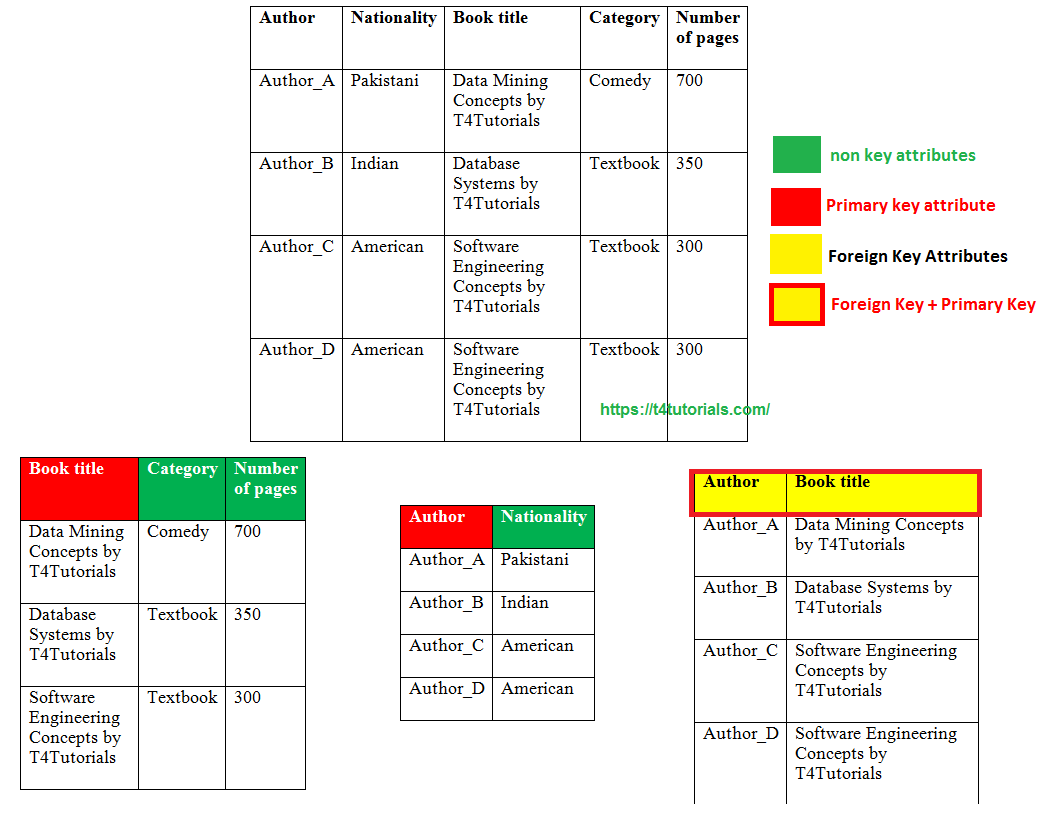


*Trong bảng trên đang thiết kế vi phạm quy chuẩn 3NF do street, city and state phụ thuộc vào zip mà không phụ thuộc vào Student\_id nên xảy ra sự phụ thuộc bắc cầu.*

1. **Quy tắc BCNF**

* Đáp ứng quy tắc 3NF (1NF: không có thuộc tính lặp , 2NF: Phải tồn tại phụ thuộc chức năng đầy đủ(Full functional dependency) và tránh phụ thuộc một phần(*partial dependency*), 3NF: Tránh phụ thuộc bắc cầu)
* Với mọi sự phụ thuộc X -> Y, X phải là siêu khóa (super key)

Ví dụ:



1. Quy tắc 4NF

Quy tắc 4NF là quy tắc khi 1 CSDL đáp ứng yêu cầu của BCNF và nó không được chứa nhiều hơn một phụ thuộc đa trị.

1. Quy tắc 5NF

Một bảng là 5th Normal Form chỉ khi là 4NF và không thể bị phân tách thành các bảng nhỏ hơn mà không bị mất dữ liệu.

1. Quy tắc 6NF

Trong các dạng chuẩn hóa dữ liệu thì 6NF hay còn được gọi là 6th Normal Form chưa có quy chuẩn nhưng đang được thảo luận bởi những chuyên gia về CSDL.

# **E. Các câu lệnh SQL Cơ Bản**

## **1. Các Câu Lệnh Định Nghĩa Dữ Liệu**

### 1.1 Create

- Là câu lệnh được sử dụng để tạo mới một đối tượng trong cơ sở dữ liệu như các database, table, chế độ xem (view –1 bảng ảo lưu trữ kết quả từ lệnh select), thủ tục (procedure), chức năng (function), chỉ mục (index)…

VD: create database UserManagement;

use UserManagement;

create table users(

Id int identity(1,1) primary key, --cột id là khóa chính tự động tăng giá trị bắt đầu từ 1 và bước nhảy là 1

Fullname varchar(50) not null,

Email nvarchar(50) not null,

Phone nvarchar(50) not null

)

create table roles(

Id int identity(1,1) primary key,

Rolename varchar(50) not null

)

create table users\_roles(

User\_id int foreign key references users(Id),

Role\_id int foreign key references roles(Id)

)

VD2: create View view\_users as

select \* from users

where Email is not null

select \* from view\_users

### 1.2 Alter

- Lệnh ALTER được sử dụng để thay đổi cấu trúc của một đối tượng có sẵn trong cơ sở dữ liệu, chẳng hạn như bảng, cột, ràng buộc, v.v. Lệnh ALTER cho phép bạn thêm, sửa đổi hoặc xóa các thành phần của đối tượng đó.

- Cú pháp: Alter Table <Tên table> <Thực hiện các lệnh trên cột>

- VD:

alter table users drop column Phone; -- xóa cột

alter table users add Phone varchar(20) -- thêm cột vào bảng đã tồn tại

-- select all constraints trong bảng users\_roles

SELECT CONSTRAINT\_NAME

FROM INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS

WHERE TABLE\_NAME = 'users\_roles' AND CONSTRAINT\_TYPE = 'Foreign KEY';

-- xóa CONSTRAINTS foreign key trong bảng users\_roles tới users

ALTER TABLE users\_roles

DROP CONSTRAINT FK\_\_users\_rol\_\_User\_\_\_398D8EEE;

-- thêm constaint foreign key từ bảng users\_roles tới bảng users

ALTER TABLE users\_roles

ADD CONSTRAINT FK\_users\_roles\_User\_1 FOREIGN KEY (User\_id)

REFERENCES users(Id);

VD2: alter view view\_users as

select Fullname, Email from users

where Email is not null

### 1.3 Drop

- Là câu lệnh dùng để xóa các đối tượng trong cơ sở dữ liệu như database, table, View, index…

- Cú pháp:

Drop database database\_name;

Drop table table\_name;

Alter table table\_name Drop constraint constaint\_name;

Drop index index\_name on table\_name;

- VD:

drop database testDb;

drop table test;

alter table users\_roles Drop constraint FK\_users\_roles\_Role\_1;

drop view view\_users

## **2. Các Câu Lệnh Cập Nhật Dữ Liệu**

### 2.1 Insert

- Là lệnh dùng để chèn 1 hàng hoặc 1 số hàng vào bảng

- Cú pháp:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

*(Value1, value2 … là các giá trị tương ứng với các cột đã định nghĩa ở trên)*

- VD:

INSERT INTO users(Fullname, Email, Phone)

VALUES (N'Nguyễn Văn D', '[nvd@gmail.com](mailto:nvd@gmail.com)', '0123123123');

INSERT INTO users(Fullname, Phone)

VALUES

(N'Nguyễn Văn Đú','0123123123'),

(N'Nguyễn Văn Đá','0123123123'),

(N'Nguyễn Văn Đớ','0123123123')

### 2.2 Update

- Là lệnh để cập nhật dữ liệu trong 1 bảng

- Cú pháp:

UPDATE tên\_bảng

SET tên\_cột1 = giá\_trị\_mới1, tên\_cột2 = giá\_trị\_mới2, ...

WHERE điều\_kiện; (nếu không có điều kiện sẽ update lên tất cả dòng)

- VD:

update users set [Email='nvda@gmail.com](mailto:Email='nvda@gmail.com)' where Id = 5

update view\_users set Fullname = N'Nguyễn Văn A' where email = '[nva@gmail.com](mailto:nva@gmail.com)'

### 2.3 Delete

- Là lệnh dùng để xóa 1 hàng trong bảng

- Cú pháp:

DELETE FROM tên\_bảng WHERE điều\_kiện;

- VD: Delete from users where Id = 8

## **3. Truy Vấn Dữ Liệu**

- Khối câu lệnh phổ dụng: SELECT – FROM - WHERE

- VD:

-- lấy ra tất cả thông tin của mỗi bảng ghi trong bảng

select \* from users

-- Lấy ra các thông tin được khai báo trong bảng và đổi tên cột trong tập kết quả trả về

select Fullname as [Name], Email from users

-- Lấy ra một bản ghi bất kỳ

select \* from users where Id = 5

### 3.1 Một số từ khóa

#### **ORDER BY**

**-** kết hợp với SELECT để sắp xếp kết quả trả về theo theo 1 hoặc nhiều điều kiện (mặc định là sắp xếp tăng dần)

**- Cú pháp:**  **SELECT** column1, column2, ...

FROM table\_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...;

- VD: select \* from users order by Id, Fullname desc

#### **DISTINCT**

- kết hợp với SELECT để lấy các dòng kết quả duy nhất, loại bỏ dòng kết quả giống nhau

VD: lấy ra các Phone phân biệt trong bảng users

select distinct Phone from users

#### **LIMIT, TOP**

- kết hợp với SELECT sử dụng để giới hạn số lượng bản ghi trả về từ kết quả truy vấn

- Cú pháp MySQL: SELECT column list

FROM table\_name

LIMIT số-dòng-cần-lấy;

- Cú pháp SQL Server: SELECT TOP number \* FROM table\_name;

- Có thể sử dụng offset để lấy số lượng dòng bất kỳ trong bảng

SELECT \*

FROM table\_name

ORDER BY column\_name

OFFSET {start\_offset} ROWS

FETCH NEXT {number\_of\_rows} ROWS ONLY;

*({start\_offset}: số lượng dòng cần bỏ qua từ vị trí đầu tiên, {number\_of\_rows}: Số lượng dòng tiếp theo muốn lấy từ {start\_offset} + 1))*

VD: SELECT \*

FROM users

order by Id

OFFSET 1 ROWS

FETCH NEXT 3 ROWS ONLY;

#### **AND/OR/ON/IN**

**AND**: được sử dụng để kết hợp nhiều điều kiện và yêu cầu cả hai điều kiện phải đúng để mệnh đề toàn cục đúng.

VD: select \* from users where Id >3 AND Phone = '0123123123'

**OR:** được sử dụng để kết hợp nhiều điều kiện và yêu cầu ít nhất một trong các điều kiện phải đúng để mệnh đề toàn cục đúng.

VD: select \* from users where Id >3 OR Phone = '0123123123'

**IN**: được sử dụng để xác định một tập hợp các giá trị cho điều kiện và kiểm tra xem giá trị của một cột có nằm trong tập hợp đó hay không.

VD: SELECT \* FROM users WHERE Id IN (1,3,5)

**ON**: được sử dụng trong câu lệnh JOIN để xác định điều kiện liên kết giữa hai bảng.

VD:

SELECT \* FROM users join users\_roles on users.Id = users\_roles.User\_id

#### **AS, CONCAT, Toán tử số học**

**AS**: dùng để đặt tên cho 1 cột hoặc 1 bảng

VD: --đặt tên cho 1 cột

select name as Fullname from test

--đặt tên cho 1 bảng

select t.name, t.age from test as t

**CONCAT**: Dùng để nối chuỗi

VD: select concat(LastName,' ',FirstName) as FullName from Employees

**Toán tử số học**: + - \* / % power

select ProductName, UnitPrice + 1 as UpdatedPrice from Products

select ProductName, UnitPrice - 1 as UpdatedPrice from Products

select ProductName, UnitPrice \* 1 as UpdatedPrice from Products

select ProductName, UnitPrice / 1 as UpdatedPrice from Products

select ProductName, UnitPrice % 1 as UpdatedPrice from Products

select ProductName, power(UnitPrice,2) as UpdatedPrice from Products

#### **LIKE**

- Dùng để tìm kiếm dữ liệu theo mẫu điều kiện where trên các giá trị String

- VD:

--Tất các các giá trị bắt đầu bằng ký tự "m"

select \* from Employees where FirstName like 'm%'

--Tất cả giá trị kết thúc bằng ký tự "t"

select \* from Employees where FirstName like '%t'

--Bất kỳ giá trị nào có chứa "an" bên trong

select \* from Employees where FirstName like '%an%'

--Các giá trị có chứa "a" ở vị trí thứ 2

select \* from Employees where FirstName like '\_a%'

--Giá trị bắt dầu bằng "M" và kết thúc bằng "t"

select \* from Employees where FirstName like 'M%t'

#### **Case**

- Dùng để thực hiện điều kiện và trả về giá trị tương ứng dựa trên kết quả của điều kiện đó

- VD:

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select Fullname, Age,

case

when Age >= 18 and Age < 60 then 'Adult'

when Age >= 60 and Age <= 80 then 'Senior'

else 'Other'

end as Age\_group -- Đặt tên cột chứa kết quả trả về từ điều kiện

from tmpEmployeeData

**ISNULL():** Kiểm tra 1 giá trị có null hay không, nếu null thì return true và ta có thể thay thế giá trị null bằng 1 giá trị mặc định khác null

VD:

select EmployeeID, LastName,FirstName, ISNULL(Region, 'VN') as Region from Employees

### 3.2 Một số hàm

* **UPPER** chuyển chuỗi thành chữ HOA

select upper(FirstName) as UpperName from Employees

* **LOWER** chuyển chuỗi thành chữ thường

select lower(FirstName) as [Lower Name] from Employees

* **SQRT** căn bậc 2 của một số (SQLite không hỗ trợ)

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select sqrt(Age) as SQRTAge from tmpEmployeeData

* **AVG** giá trị trung bình của cột

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select avg(Age) as AvgAge from tmpEmployeeData

* **SUM** giá trị tổng của cột

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select sum(Age) as Total from tmpEmployeeData

* **ABS** trị tuyệt đối

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select abs(Age - 100) as Total from tmpEmployeeData

* **MIN / MAX** giá trị nhỏ nhất, lớn nhất

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select min(Age) as MinAge,max(Age) as MaxAge from tmpEmployeeData

* **COUNT** đếm số dòng

-- count số bản ghi có age > 60

with tmpEmployeeData as (

select CONCAT(LastName,' ',FirstName) as Fullname,

DATEDIFF(day, BirthDate, GETDATE())/ 365 as Age

from Employees

)

select count(\*) as NumberOfSenior from tmpEmployeeData where Age > 60

**Group by:** sử dụng để nhóm các bản ghi có cùng 1 trường dữ liệu nào đó để phục vụ cho việc tính toánCOUNT(), MAX(), MIN(), SUM(), AVG()

- VD:

-- đếm số lượng sản phẩm theo từng category

select CategoryID, count(ProductID) as Products from Products group by CategoryID

**Having:** lọc dữ liệu theo nhóm trả về từ câu lệnh group by

-- Lấy ra số các categoryID và số lượng sản phẩm tương ứng > 10

select CategoryID, count(ProductID) as Products

from Products

group by CategoryID

having count(ProductID) > 10

**Exists:** kiểm tra các giá trị trả về từ câu truy vấn ngoài có tồn tại trong câu truy vấn con hay không

-- Lấy ra thông tin khách hàng order trước '1996-10-01'

select \* from Customers where exists

(select \* from Orders

where Customers.CustomerID = Orders.CustomerID and Orders.OrderDate <= '1996-10-01')

*(Ban đầu chạy câu query ngoài sẽ lấy ra tất cả customer xong rồi dùng tập kết quả trả về kiểm tra với điều kiện trong câu sub-query và trả về kết quả cuối cùng)*

**Tương tự với Not exists:** trả về các gtri trả về từ câu truy vấn ngoài mà không tồn tại trong câu truy vấn con

-- Lấy ra thông tin khách hàng order sau '1996-10-01'

select \* from Customers where not exists

(select \* from Orders

where Customers.CustomerID = Orders.CustomerID and Orders.OrderDate <= '1996-10-01')

**ANY**: Toán tử "ANY" được sử dụng để so sánh một giá trị với một tập hợp và trả về TRUE nếu bất kỳ giá trị nào trong tập hợp đó khớp với giá trị được so sánh

-- lấy ra các sản phẩm có số lượng bán > 1000

select \* from Products where ProductID = any

(select ProductID

from [Order Details]

group by ProductID

having sum(Quantity) > 1000)

**ALL**: Toán tử "ALL" được sử dụng để so sánh một giá trị với một tập hợp và trả về TRUE nếu tất cả các giá trị trong tập hợp đó khớp với điều kiện so sánh

-- lấy ra các sản phẩm có số lượng bán <= 1000

select \* from Products where ProductID != ALL

(select ProductID

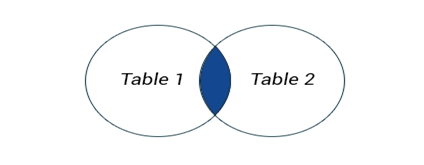
from [Order Details]

group by ProductID

having sum(Quantity) > 1000)

### 3.3 Các lệnh Join

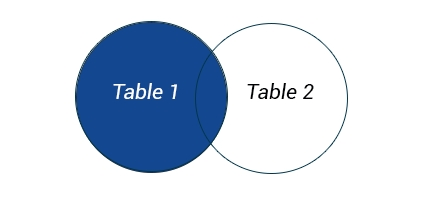
**Inner join**: Trả về các bản ghi từ hai hoặc nhiều bảng mà khớp với điều kiện kết nối.



VD: select \* from Orders o inner join [Order Details] od on o.OrderID = od.OrderID

*(lấy các bản ghi trong bảng Order sau đó kiểm tra trong Order\_Details xem có bản ghi nào thỏa mãn mệnh đề ON không. Nếu thỏa mãn thì hợp 2 bản ghi từ 2 bảng lại và trả về)*

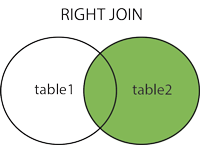
**Left join:** Mỗi dòng của bảng trái được lấy ra để kết hợp với bảng phải theo điều kiện mệnh đề ON, nếu bảng phải có các dòng phù hợp thì các dòng này được kết hợp với dòng được lấy ra của bảng trái để tạo kết quả trả về. Nếu có dòng bảng bên trái không tồn tại trong bảng bên phải thì vẫn trả về dòng bảng bên trái và các trường tương ứng của bảng bên phải sẽ nhận null



VD:select \* from Categories c left join Products p on c.CategoryID = p.CategoryID

*(bảng Categories có 2 bản ghi cuối không tồn tại trong bảng Products nên vẫn return và các trường tương ứng sẽ nhận giá trị null)*

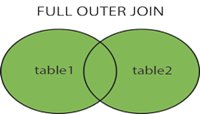
**Right join:** Trả về tất cả các bản ghi từ bảng phải (table2) và các bản ghi khớp từ bảng trái (table1) thỏa mãn mệnh đề ON. Nếu không có khớp, các giá trị NULL sẽ được trả về cho các cột của bảng trái.



VD: select \* from Categories right outer join Products on Products.CategoryID = Categories.CategoryID

*(Lấy ra tất cả tất cả bản ghi từ bảng Products và các bản ghi tương ứng trong* Categories *dựa trên trường CategoryID. Các bản ghi trong bảng Products không tồn tại trong Categories vẫn được trả về, các trường tương ứng trong bảng Categories sẽ nhận null)*

**Full outer join:** Trả về tất cả các hàng từ cả hai bảng trái và bảng phải. Các hàng khớp sẽ được hiển thị, còn không khớp sẽ trả về các giá trị NULL.



-VD: select \* from Categories full outer join Products on Products.CategoryID = Categories.CategoryID

**Cross join**: Từng bản ghi của bảng bên trái kết hợp với tất cả bản ghi bảng bên phải. Số bản ghi trả về bằng tích số bản ghi bảng trái \* số bản ghi bảng phải

- VD: select \* from Categories cross join Products

**Self join**: Là một loại JOIN kết hợp một bảng với chính nó, nhưng sử dụng các bí danh (alias) khác nhau để phân biệt các phiên bản của bảng.

- VD:

-- Tìm tất cả các product có cùng category với từng product trong bảng product

select p1.ProductName, p2.ProductName

from Products p1, Products p2

where p1.ProductID != p2.ProductID and p1.CategoryID = p2.CategoryID

### 3.4 Sử dụng Index truy vấn dữ liệu

#### 3.4.1. Định nghĩa

Index là một cấu trúc dữ liệu đặc biệt được xây dựng từ một hoặc nhiều cột trong bảng, giúp tìm kiếm và truy xuất dữ liệu một cách nhanh chóng. Sử dụng Index trong câu query giúp tăng hiệu suất truy vấn dữ liệu trong cơ sở dữ liệu.

#### 3.4.2. Phân loại

**1. Clustered Index (Chỉ mục cụm)**

- Clustered index sắp xếp dữ liệu theo giá trị của cột index. Vì vậy, các hàng trong bảng được tổ chức theo thứ tự của index.

- Clustered index xác định thứ tự vật lý của dữ liệu trong bảng. Mỗi bảng chỉ có thể có một Clustered index và Primary key chính là Clustered Index

- Khi tạo một Clustered index trên một cột, nó sẽ tự động tạo một bảng gốc mới sắp xếp theo cột index. Điều này có thể ảnh hưởng đến hiệu suất của các thao tác ghi dữ liệu vì cần phải cập nhật vị trí vật lý của hàng khi có thay đổi.

**2. Non-clustered Index (Chỉ mục phi cụm):**

- **Non-clustered Index** không xác định thứ tự vật lý của dữ liệu trong bảng. Một bảng có thể có nhiều **Non-clustered Index**.

- **Non-clustered Index** tạo ra một bảng mới chứa các cột index và tạo ra một con trỏ đến hàng gốc tương ứng trong bảng gốc.

- Cú pháp: Create (NONCLUSTERED) index index\_name on table\_name(column1, column2,…)

- VD: create index idx\_users\_Fullname on users(Fullname)

*(Nonclustered index sẽ tổ chức các giá trị của cột* Fullname *theo trình tự sắp xếp và liên kết với con trỏ đến các bản ghi tương ứng trong bảng users.)*

Khi truy vấn: select \* from users where full\_name = 'Full Name 999998'

(SQL Server sẽ kiểm tra xem có nonclustered index nào được tạo trên cột name hay không

Nếu có, sẽ duyệt danh sách con trỏ và tìm giá trị full\_name = 'Full Name 999998', sau đó trả về các bản ghi tương ứng trong bảng users có full\_name = 'Full Name 999998')

### 3.5 SQL Dates

**GETDATE**(): trả về ngày giờ hiện tại

VD: SELECT getdate();

**FORMAT**(date, pattern): định dạng date theo pattern

VD: SELECT FORMAT(GETDATE(), 'dd/MM/yyyy') AS FormattedDate;

**DATEADD**(datepart, number, date): Thêm số datepart (day, month, year) vào date

VD: select DATEADD(day, 2, GETDATE()) AS NewDate;

select DATEADD(day, -2, GETDATE()) AS NewDate;

select DATEADD(MONTH, -2, GETDATE()) AS NewDate;

**DATEDIFF**(datepart, startdate, enddate): tính số ngày/tháng/năm chênh lệch giữa 2 giá trị date

VD: SELECT DATEDIFF(day, '2023-06-10', GETDATE()) AS DayDiff;

SELECT DATEDIFF(MONTH, '2022-06-10', GETDATE()) AS MonthDiff;

-- tính số tuổi hiện tại

SELECT DATEDIFF(day, '2001-10-16', GETDATE()) / 365 AS Age;

Có thể đặt 1 hay nhiều bảng phụ bằng cú pháp:  
**With tableA as(query), tableB as(query),… (ngăn cách bởi dấu phảy. Câu truy vấn tổng không cần dấu phảy)**

🡪 câu truy vấn tổng sẽ thao tác các bảng phụ

VD: **chạy cả đoạn 3 câu**

--Bảng Employees(EmpId, EmpName), Departments(DepId, DepName),

--Employees\_Departments(Id, DepId, EmpId).

--Viết truy vấn in ra đầy đủ thông tin của các phòng ban có số

--lượng nhân viên nhiều nhất bao gồm: DepId, Depname, NumberOfEmp

with CountEmpOfDepartment as (

select d.DeptId, d.DeptName, count(\*) as NumberOfEmps

from Departments d join Employees\_Departments ed

on d.DeptId = ed.DeptId

group by d.DeptId, d.DeptName

),

MaxEmpsCounted as(

select max(NumberOfEmps) as MaxCount from CountEmpOfDepartment

)

select ce.DeptId, ce.DeptName, ce.NumberOfEmps

from CountEmpOfDepartment ce join MaxEmpsCounted me

on ce.NumberOfEmps = me.MaxCount

### 3.6 Stored Procedures, Trigger

#### 3.6.1 Stored Procedures

- Là một khối mã SQL đã được đặt tên và lưu trữ trong cơ sở dữ liệu giúp tăng hiệu suất (do chỉ biên dịch ở lần đầu tiên, các lần tiếp theo sẽ được thực thi luôn mà không cần biên dịch), tăng tính bảo mật (có thể chỉ định quyền thực thi Stored Procedures cho các người dùng cụ thể, điều này giúp kiểm soát và hạn chế quyền truy cập vào dữ liệu và giảm thiểu rủi ro của cuộc tấn công SQL Injection bởi việc tự động xử lý tham số truy vấn), tăng tính tải sử dụng và bảo trì mã(Thay vì lặp lại mã SQL trong từng ứng dụng, ta chỉ cần tạo một Stored Procedure và sử dụng nó ở nhiều nơi)

- Cú pháp:

-- Tạo Procedure

CREATE PROCEDURE *procedure\_name*  
AS

Begin  
 *sql\_statement*

*End;*

*-- Thực thi Procedure*

EXEC *procedure\_name*;

- VD:

CREATE PROCEDURE GetUsers

AS

BEGIN

SELECT \* FROM users;

END;

--

exec GetUsers

-- Tạo procedure chứa tham số đầu vào

CREATE PROCEDURE GetUsersById @UserId int

AS

BEGIN

SELECT \* FROM users where Id = @UserId;

END;

--

exec GetUsersById @UserId = 2

-- Tạo procedure chứa tham số đầu vào và đầu ra

CREATE PROCEDURE GetUsersById2 @UserId int, @Email varchar(50) output

AS

BEGIN

SELECT @Email = Email FROM users where Id = @UserId;

END;

--

Declare @OutputEmail varchar(50);

exec GetUsersById2 @UserId = 2, @Email = @OutputEmail output;

select @OutputEmail as Email

-- Tạo procedure cho câu lệnh insert, dùng try catch để xử lý tình huống lỗi

CREATE PROCEDURE InsertUser

@FullName NVARCHAR(50),

@Email varchar(50),

@Phone varchar(50)

AS

BEGIN

BEGIN TRY

-- Bắt đầu giao dịch

BEGIN TRANSACTION;

-- Thực hiện các câu lệnh INSERT và UPDATE

insert into users(Fullname,Email,Phone) values(@FullName,@Email,@Phone);

-- Kết thúc giao dịch

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

-- Xử lý lỗi và rollback giao dịch

ROLLBACK TRANSACTION;

PRINT ERROR\_MESSAGE();

END CATCH;

END;

--

exec InsertUser @Fullname = 'Trương Trọng Hòa',@Email = '[HoaTT32@fpt.com](mailto:HoaTT32@fpt.com)', @Phone = '0222333444'

exec GetUsers

#### 3.6.2 Trigger

- là một loại stored procedure đặc biệt (không có tham số) được thực thi một cách tự động khi có một sự kiện thay đổi dữ liệu(Insert,Delete, update)

- Cú pháp:

CREATE TRIGGER trigger\_name ON table\_name

{[AFTER], [INSTEAD OF]} {[INSERT],[UPDATE],[DELETE]}

AS

BEGIN

{sql\_statements}

END;

- Các loại trigger:

+ INSTEAD OF: Trigger này được kích hoạt trước khi một sự kiện xảy ra trên bảng. Ta có thể sử dụng trigger này để kiểm tra hoặc thay đổi dữ liệu trước khi nó được thêm, sửa đổi hoặc xóa trong bảng

VD: -- Trigger thực hiện xóa các bản ghi ở bảng quan hệ trước khi xóa bản ghi trong bảng chính

CREATE TRIGGER DeleteUserTrigger

ON users

INSTEAD OF delete

AS

BEGIN

-- Lưu trữ UserId được truyền vào

DECLARE @userId INT;

SET @userId = (SELECT Id FROM deleted);

-- Xóa các bản ghi liên quan trong bảng users\_roles

DELETE FROM users\_roles WHERE users\_roles.User\_id = @userId;

-- Xóa người dùng trong bảng users

DELETE FROM users WHERE Id = @userId;

END;

-- trigger kiểm tra email của user truyền vào trước khi update

CREATE TRIGGER UpdateUserTrigger ON users

INSTEAD OF update

AS

BEGIN

IF EXISTS(SELECT \* FROM inserted WHERE email LIKE '[%@gmail.com](mailto:%25@gmail.com)')

BEGIN

-- Nếu email chứa "[@gmail.com](mailto:@gmail.com)", thực hiện thêm người dùng

update users set Fullname = i.Fullname, Email = i.Email, Phone = i.Phone from users inner join inserted i on users.Id = i.Id

END

ELSE

BEGIN

-- Nếu email không chứa "[@gmail.com](mailto:@gmail.com)", rollback thao tác và hiển thị thông báo lỗi

ROLLBACK TRANSACTION;

PRINT N'Email phải chứa "[@gmail.com](mailto:@gmail.com)". Thao tác đã bị hủy.'

END;

END;

*(Trong trigger instead of UPDATE, bảng inserted chứa các bản ghi mới trước khi thực hiện update, và bảng deleted chứa các bản ghi cũ trước khi bị cập nhật)*

-- trigger kiểm tra email của user truyền vào trước khi insert

alter TRIGGER InsertUserTrigger ON users

INSTEAD OF insert

AS

BEGIN

IF EXISTS(SELECT \* FROM inserted WHERE email LIKE '[%@gmail.com](mailto:%25@gmail.com)')

BEGIN

-- Nếu email chứa "[@gmail.com](mailto:@gmail.com)", thực hiện thêm người dùng

INSERT INTO users (Fullname,Email,Phone)

SELECT Fullname,Email,Phone FROM inserted;

END

ELSE

BEGIN

-- Nếu email không chứa "[@gmail.com](mailto:@gmail.com)", rollback thao tác và hiển thị thông báo lỗi

ROLLBACK TRANSACTION;

PRINT N'Email phải chứa "[@gmail.com](mailto:@gmail.com)". Thao tác đã bị hủy.'

END;

END;

*(Trong trigger instead of Insert, bảng inserted chứa các bản ghi mới trước khi thực hiện insert)*

-- trigger hiển thị thông tin sau khi thay đổi dữ liệu trong bảng

alter TRIGGER UsersAuditTrigger ON users

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

-- Kiểm tra xem trigger được kích hoạt bởi INSERT, UPDATE hay DELETE

DECLARE @Action AS VARCHAR(10);

IF EXISTS(SELECT \* FROM inserted) AND EXISTS(SELECT \* FROM deleted)

SET @Action = 'UPDATED';

ELSE IF EXISTS(SELECT \* FROM inserted)

SET @Action = 'INSERTED';

ELSE IF EXISTS(SELECT \* FROM deleted)

SET @Action = 'DELETED';

-- In ra thông tin bảng Users sau khi có thay đổi

PRINT '----------------------------------------';

PRINT 'Action: ' + @Action;

SELECT \* FROM users;

PRINT '----------------------------------------';

END;

**Quy trình xử lý một câu lệnh SQL trong Database**

B1: Kiểm tra câu truy vấn đúng cú pháp, đúng ngữ nghĩa

B2: Kiểm tra câu truy vấn đã tồn tại trong Shared pool (nơi lưu trữ các câu lệnh SQL đã được phân tích (parsed) và tạo ra các kế hoạch thực hiện (execution plan) trước đây và các procedure và trigger…) chưa

B3:

+ Nếu không tồn tại câu lệnh truy vấn tương tự trong Shared pool:

🡺 Thực hiện quá trình hard parse (gây tiêu tốn tài nguyên):

- Phân tích tất cả các chiến lực thực thi có thể xảy ra (quét full bảng hay quét index, sử dụng index nào để quét, sử dụng kiểu quét index nào (range scan, fast scan, fast full scan…))

- Sau khi có tất cả chiến lược thực thi, hệ thống sẽ build lại từng bước chi tiết và chạy theo thứ tự từng bước chi tiết đó và trả về kết quả

+ Nếu đã tồn tại câu lệnh truy vấn trong Shared pool (sql query đã được thực hiện trước đó):

🡺 Thực hiện quá trình soft parse:

- Không cần phân tích các chiến lược thực thi và build từng bước mà sẽ sử dụng lại query plan đã tạo từ hard parse trước đó

🡪 Tốc độ truy vấn nhanh hơn và tiết kiệm tài nguyên.

BT: sử dụng subquery

Cho 2 bảng: Department: Id, Name

Employee: Id, DepartmentId, Name

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | Dev |
| 2 | Kế toán |
| 3 | Kinh doanh |

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **DepartmentId** |
| 1 | A | 1 |
| 2 | B | 2 |
| 3 | C | 1 |
| 4 | D | 3 |
| 5 | E | 2 |

Đề: Hãy lấy ra thông tin của các phòng có số lượng nhân viên nhiều nhất. Hiển thị Id, Name, Số lượng