**4. Khái niệm cơ sở dữ liệu quan hệ:**

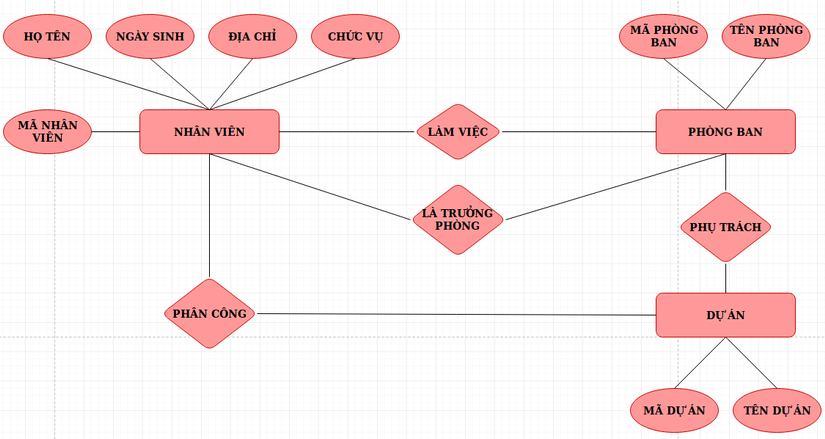
- cơ sở dữ liệu là cách thức tổ chức lưu trữ dữ liệu hiệu quả để đảm bảo việc nhập và khai thác dữ liệu nhanh và chính xác.

- Cơ sở dữ liệu quan hệ tổ chức dữ liệu theo các bảng và có quan hệ với nhau đê giảm thiểu sự dư thừa dữ liệu đồng thời vẫn đảm bảo sự hiệu quả trong lưu trữ và truy xuất dữ liệu

**5. Khái niệm về Entity relationship ?**

- là một mô hình được sử dụng rộng rãi trong các bản thiết kế dữ liệu ở mức khái niệm , được xây dựng dựa trên việc nhận thức thế giới thực thông qua tập các đối tượng được gọi là các thực thể và các mối quan hệ giữa các đối tượng này

EX: mô hình quan hệ của tập các đối tượng là **Nhân viên** , **Phòng ban** , **Dự án**



**6. thực hành tạo ERD**

- thực hành tạo ERD thông qua tool [**https://erdplus.com/**](https://erdplus.com/)

**7. Các loại Relationship?**

- có 3 loại relationship:

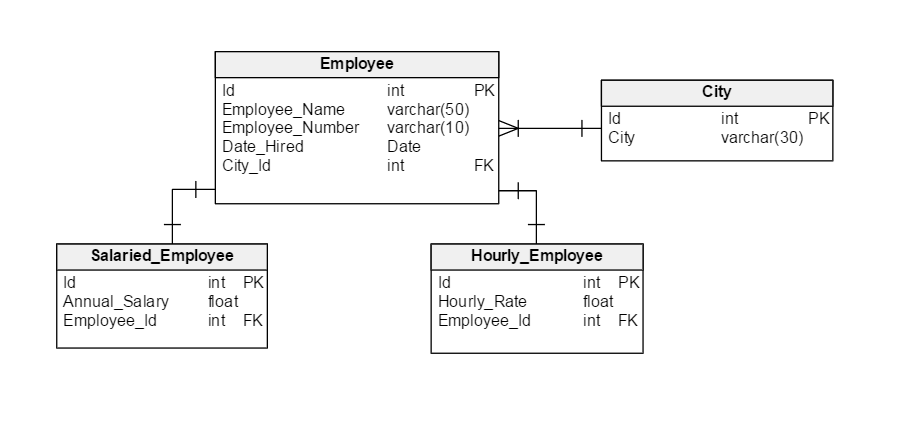
+ **quan hệ 1 – 1**: Trọng quan hệ này mỗi bảng chỉ có 1 và chỉ 1 bản ghi tương ứng với nhau mà thôi

+ **quan hệ 1 – n** : Là quan hệ phổ biến nhất trong cơ sở dữ liệu , trong quan hệ này 1 bản ghi ở bảng này sẽ có nhiều bản ghi tương ứng ở bảng kia

+ **quan hệ n – n**: trong mối quan hệ này một bản ghi ở bảng này tương ứng có nhiều bản ghi ở bảng kia và ngược lại

**8.Relational database schema:**

- mô hình quan hệ thực thể(ERD) thể hiện ý tưởng, nội dung của 1 database, nhưng nó lại không thể hiện được cách sắp xếp cụ thể của cơ sở dữ liệu .Một ERD tốt sẽ rất gần với DB thực tế. Nhưng ta vẫn cần có kiến thức về Relational Schema để hiểu được thiết kế thật sự của của 1 database. Cụ thể, ERD chính là hình vẽ của 1 chiếc xe, còn Relational Schema là bản vẽ chi tiết của chiếc xe đó.

EX : 

**9.Nắm được “rule” chuyển đổi từ ERD sang Relational Database schema?**

Rule:

Bước 1: chuyển mỗi loại thực thể thành một loại quan hệ tương ứng

Chuyển các mối kết hợp 1-1 gom 2 thực thể thành một thực thể

Các mối kết hợp 1-N lấy khóa bên thực thể nhiều chuyển thành khóa ngoại

Các mối quan hệ N-N hình thành một loại quan hệ mới

Bước 2: Kiểm tra lại dạng chuẩn của các mối quan hệ.

**10. Đã thực hành chuyển đổi từ ERD sang Relational DB Schema ?**

- đã thực hành chuyển đổi

**11. Các kiểu dữ liệu trong SQL Server?**

- có 3 kiểu dữ liệu trong SQL Server : kiểu dữ liệu dạng kí tự, dạng số và dạng ngày tháng

**+ kiểu dữ liệu dạng kí tự:**

|  |  |  |
| --- | --- | --- |
| **Cú pháp kiểu dữu liệu** | **Kích thước tối đa** | Giải thích |
| VARCHAR(kich\_thuoc) | Tối đa 8000 kí tự | Kích thước là số kí tự lưu trữ  Độ dài cố định  Thêm dấu cách về bên để bù phần trống cho đủ số kí tự  Không chưa kí tự unicode |
| VARCHAR(kich\_thuoc) hoặc VARCHAR(toi\_da) | Tối đa 8000 kí tự hoặc theo số tối đa. | Kích thước là số kí tự lưu trữ  Độ dài tùy biến  Nếu chỉ định là toi\_da thì tối đa là 2G |
| TEXT | Tối đa 2G | Độ dài tùy biến  Không chứa kí tự unicode |
| NChar(kick\_thuoc) | Tối đa 4000 kí tự | Độ dài cố định  Kí tự Unicode |
| NVARCHAR(kich\_thuoc) hoặc NVARCHAR(toi\_da) | Tối đa 4000 kí tự theo số tối đa | Kich\_thuoc là số kí tự lưu trữ.  Độ dài tùy biến.  Nếu số toi\_da được chi định thì số kí tự tối đa là 2GB.  Kí tự Unicode |
| NTEXT | Tối đa 1.073.741.823 | Độ dài tùy biến  Kí tự Unicode |
| BINARY(kich\_thuoc) | Tối đa 8000 kí tự | kich\_thuoc là số kí tự lưu trữ  Độ dài cố định  Thêm các dấu cách để bù phần trống cho đủ số kí tự.  Dữ liệu nhị phân |
| VARBINARY(kich\_thuoc) hoặc VARBINARY(toi\_da) | Tối đa 8000 kí tự hoặc theo số tối đa | kich\_thuoc là số kí tự lưu trữ  Độ dài tùy biến  Nếu chỉ định là toi\_da thì thì tối đa là 2GB. |
| IMAGE | Kích thước tối đa là 2G | Độ dài tùy biến  Dữ liệu nhị phân |

**+Kiểu dữ liệu dạng số:**

|  |  |  |
| --- | --- | --- |
| **Cú Pháp kiểu dữ liệu** | **Kích thước tối đa** | **Giải thích** |
| BIT | Số nguyên 0,1,null |  |
| TINYINT | Từ 0 đến 255 |  |
| SMALLINT | Từ -32768 đến 32767 |  |
| INT | -2,147,483,648 đến 2,147,483,647 |  |
| BIGINT | từ -9,223,372,036,854,775,808 đến 9,223,372,036,854,775,807 |  |
| DECIMAL(m,d) | m mặc định là 18 nếu không được chỉ định cụ thể  d mặc định là 0 nếu không được chỉ định cụ thể | m là tổng số lượng các số còn d là số lượng các số nằm sau dấu phẩy |
| DEC(m , d) | m mặc định là 18 nếu không được chỉ định  d mặc định là 0 nếu không được chỉ định cụ thể | m là tổng số lượng các số còn d là số lượng các số nằm sau dấu phẩy |
| NUMERIC(m,d) | m mặc định là 18 nếu không được chỉ định  d mặc định là 0 nếu không được chỉ định cụ thể | m là tổng số lượng các số còn d là số lượng các số nằm sau dấu phẩy. |
| FLOAT(n) | Số dấu phẩy động n mặc định là 53 nếu không được chỉ định cụ thể | N là số lượng của số bit lưu trữ trong một kí hiệu hóa học |
| REAL | Tương đương với FLOAT(24) |  |
| SMALLMONEY | từ - 214,748.3648 đến 214,748.3647 |  |
| MONEY | từ -922,337,203,685,477.5808 đến 922,337,203,685,477.5807 |  |

**+Kiểu dữ liệu dạng ngày tháng/ thời gian trong SQL Server**

|  |  |  |
| --- | --- | --- |
| **Cú pháp kiểu dữ liệu** | **Kích thước tối đa** | **Giải thích** |
| DATE | Giá trị từ ‘0001-01-01’ đến ‘9999-12-31’ | Hiển thị dưới dạng ‘YYYY-MM-DD’ |
| DATETIME | Ngày lấy từ '1753-01-01 00:00:00' to '9999-12-31 23:59:59'.  Giờ lấy từ '00:00:00' to '23:59:59:997' | Hiển thị dưới dạng ‘YYYY-MM-DD’  Hh:mm:ss[.số giây thập phân]’ |
| DATETIME2(Chính xác tới số thập phân của giây) | giá trị lấy từ '0001-01-01' đến '9999-12-31'.  Thời gian lấy từ '00:00:00' đến '23:59:59:9999999 | hiển thị dưới dạng 'YYYY-MM-DD hh:mm:ss[.số giây thập phân]' |
| SMALLDATETIME | giá trị lấy từ '1900-01-01' đến '2079-06-06'.  Thời gian lấy từ '00:00:00' đến '23:59:59'. | Hiển thị dưới dạng ‘YYYY-MM-DD’  hh:mm:ss |
| TIME | giá trị lấy từ '00:00:00.0000000' đến '23:59:59.9999999'.  Ngày lấy từ '0001-01-01' đến '9999-12-31'. | Hiển thị dưới dạng ‘YYYY-MM-DD’  hh:mm:ss[.nnnnnn] |
| DATETIMEOFFSET(chính xác tới sô thập phân của giây) | giá trị thời gian lấy từ '00:00:00' đến '23:59:59:9999999'.  Múi giờ lấy từ -14:00 đến +14:00 | Hiển thị dưới dạng YYYY-MM-DD  hh:mm:ss[.nnnnnnn]  [{+|-} hh:mm] |

**12.Khái niệm DDL (Data Definition Language) ?**

- DDL là ngôn ngữ định nghĩa dữ liệu gồm các câu lệnh: CREATE, ALTER, DROP, RENAME

- Khác với DML: DML là viết tắt của Ngôn ngữ Thao tác Dữ liệu (Data Manipulation Language): INSERT, UPDATE và DELETE là các câu lệnh DML.

**13: các câu lệnh Create, Alter ,Drop:**

* Create tạo mới 1 DB hoặc 1 Table.

+ cú pháp lệnh Create - tạo Database**: Create database ten\_database**

+ cú pháp lệnh Create - tạo Table: **CREATE TABLE ten\_bang(**

**cot1 kieu\_du\_kieu,**

**cot2 kieu\_du\_kieu,**

**cot3 kieu\_du\_kieu,**

**.....**

**cotN kieu\_du\_kieu,**

**PRIMARY KEY( mot hoac nhieu cot )**

**);**

* Alter: lệnh ALTER TABLE được dùng để thêm cột, chỉnh sửa cột, xóa cột, đổi tên cột hoặc đổi tên bảng.
* Drop: xóa cả bảng khỏi Database

Drop còn được sử dụng trong câu lệnh Alter để xóa cột ra khỏi Table

**14: Đã thực hành với create ,alter, drop.**

* Đã thực hành:

**15: khái niệm schema?**

* Schema là một khái niệm mới được đưa vào SQL Server từ phiên bản 2005, nó là một namespace dùng để gom nhóm các table có chung một đặc điểm nào đó đễ dễ dàng quản lý. Nếu bạn không sử dụng schema trong CSDL thì nó sẽ lấy schema mặc định là dbo.

**16: đã tạo được schema và thực hành table với schema đó**.

**17:khái niệm table, cách tạo 1 table:**

* Một bảng (table) thuộc 1 database. Mỗ table bao gồm tên table, danh sách cột ,(column), khóa chính, khóa ngoại
* Cách tạo 1 Table: sử dụng câu lệnh Create table như câu số 13.

**18: Foreign key và Primary key.**

* Foreign key được dung để tăng tính tham chiếu trong cơ sở dữ liệu SQL Server. Khóa ngoại nghĩa là giá trị trong bảng này phải xuất hiện trong bảng khác. DÙng để liên kết 2 bảng (khoá ngoại của thằng này là khoá chính của thằng khác). Một bảng có 1 khoá chính và nhiều khoá ngoại
* Primary key: khóa chính là một trường hoặc kết hợp nhiều trường và được xác định là một bản ghi duy nhất. Không trường nào trong khóa chính được chứa giá trị NULL. Một bảng chỉ có duy nhất một khóa chính

**21: các kiểu ràng buộc:**

* NOT NULL: Đảm bảo dữ liệu của cột không được nhận giá trị NULL.
* DEFAULT: cung cấp một giá trị mặc định cho một cột trong trường hợp dữ liệu dữ liệu của cột không được nhập vào hay không được xác định.
* UNIQUE: Đảm bảo rằng dữ liệu của cột là duy nhất, tất cả giá trị trong trong một cột là khác nhau không trùng lặp
* Primary Key: Dùng để thiết lập khóa chính trên bảng, giá trị cột làm khóa chính phải phải là duy nhất, tất cả các giá trị trong một cột là khác nhau và không trùng lặp
* FOREIGN Key: Dùng để thiết lập khóa ngoại chính trên bảng, tham chiếu đến bảng khác thông qua giá trị của cột được liên kết. Giá trị của cột được liên kết phải là duy nhất trong bảng kia
* Check: Đảm bảo tất cả các giá trị trong một cột thỏa mãn một số điều kiện
* INDEX: Dùng để tạo và lấy dữ liệu từ cơ sở dữ liệu một cách nhanh chóng
* Xóa bỏ ràng buộc: Bất kể ràng buộc nào được tạo cũng có thể bị xóa bỏ bằng cách sử dụng lệnh Alter Table với tùy chọn **DROP CONSTRAINT**

EX: Ví dụ, để drop Constraint khóa chính trong bảng NHANVIEN, bạn có thể sử dụng lệnh sau đây**: ALTER TABLE NHANVIEN DROP CONSTRAINT NHANVIEN\_PK;**

**22:**

* Đã thực hành với các ràng buộc NOT NULL, CHECK, UNIQUE

**23:**

* Đã thực hành với PRIMARY KEY, DEAFAULT, FOREIGN KEY

**24: Constraints Scope (table level and column level)?**

* Ràng buộc có thể được chỉ định cho các cột riêng lẻ như 1 phần của đặc tả cột (ràng buộc cấp cột) hoặc cho các nhóm cột như 1 phần của định nghĩa bảng (ràng buộc cấp bảng).
* ???????????????????????? bổ sung sau

**25: từ khóa IDENTITY?**

* Giống như trong MySQL thì chúng ta có thuộc tính **auto\_increment** dùng để xác định 1 cột nào đó tăng tự động, SQL Server thì chúng ta có thuộc tính **IDENTITY**
* Cú pháp của **IDENTITY** như sau: IDENTITY (seed, increment)

Trong đó: seed là giá trị được gán cho record đầu tiên lúc insert

Increment: là giá trị tăng lên sau mỗi lần insert

**26: thực hành tạo table với bảng sử dụng IDENTITY?**

* Đã thực hành

**27: từ khóa Truncate**?

* Dùng để xóa tất cả bản ghi từ 1 bảng trong SQL Serve. Lệnh này hoạt động tương tự như DELETE nhưng không có mệnh đề WHERE
* Sự khác biệt giữa TRUNCATE, DELETE và DROP

+ DELETE: xóa 1 hay tất cả các dòng trong 1 bảng theo 1 điều kiện nhất định, dữ liệu có thể phục hồi lại

+TRUNCATE: Xóa toàn bộ các dòng của bảng, giải phóng bộ nhớ và không thể phục hồi lại. Chỉ xóa dữ liệu, không xóa bảng khỏi Database

+ DROP: xóa luôn bảng và dữ liệu của bảng khỏi Database và không thể phục hồi lại.

**28: Thực hành với từ khóa Truncate?**

* Cú pháp cơ bản của lênh TRUNCATE: **TRUNCATE TABLE ten\_bang;**

**29: khái niệm index trong sql?**

* Chỉ mục(index) trong SQL là bảng tra cứu đặc biêt mà công cụ tìm kiếm cơ sở dữ liệu có thể sử dụng để tăng nhanh thời gian và hiệu suất truy xuất dữ liệu.
* Chỉ mục trong Database có ý nghĩa tương tự như các mục xuất hiện trong Mục lục của một cuốn sách

**30: Mục đích phải sử dụng chỉ mục index?**

* Để định vị và truy cập nhanh nhất vào dữ liệu trong các bảng database
* Là 1 cách tối ưu hiệu suất truy vấn database bằng việc giảm lượng truy cập vào bộ nhớ khi thực hiện truy vấn

**31: Tạo index trên table?**

* Cú pháp như sau:

**CREATE INDEX ten\_index ON ten\_bang;**

**32: Phân loại index, điểm khác nhau giữa các loại đó?**

* **Single-Column Index** được tạo cho duy nhất 1 cột trong bảng. Cú pháp cơ bản như sau:

**CREATE INDEX ten\_index**

**ON ten\_bang (ten\_cot);**

* **Unique Index** là chỉ mục duy nhất, được sử dụng để tăng hiệu suất và đảm bảo tính toàn vẹn dữ liệu. Một chỉ mục duy nhất không cho phép chèn bất kỳ giá trị trùng lặp nào được chèn vào bảng. Cú pháp cơ bản như sau.

**CREATE UNIQUE INDEX ten\_index**

**ON ten\_bang (ten\_cot);**

* **Composite Index** là chỉ mục kết hợp dành cho hai hoặc nhiều cột trong một bảng. Cú pháp cơ bản của nó như sau:

**CREATE INDEX ten\_index**

**ON ten\_bang (cot1, cot2);**

* **Implicit Index (Index ngầm định**) là chỉ mục mà được tạo tự động bởi Database Server khi một bảng được tạo. Các Index ngầm định được tạo tự động cho các ràng buộc Primary key và các ràng buộc Unique.

**33: Thực hành sử dụng index?**

**34: Khái niệm view?**

* 1 khung nhìn VIEW là 1 bảng ảo trong cơ sở dữ liệu có nội dung được định nghĩa thông qua một câu lệnh SQL nào đó. Một view bao gồm các hàng và cột giống như 1 bảng thực. Các trường trong 1 khung nhìn là các trường từ 1 hoặc nhiều bảng thực.

**35: Tại sao phải sử dụng view?**

* Sử dụng View trong quản lý database sẽ có những ưu thế sau:

+ Đơn giản hóa các truy vấn phức tạp: View giúp đơn giản hóa những câu truy vấn phức tạp cho những lần sử dụng sau, bởi khi dữ liệu ở bảng chính thay đổi thì trong View cũng sẽ được thay đổi theo nên vấn đề đồng bộ dữ liệu rất chính xác.

+ Làm giảm độ phức tạp tính toán: Giả sử bạn phải viết cùng một công thức cho mọi truy vấn. Hoặc bạn có một truy vấn có logic phức tạp. Để làm cho logic này nhất quán trên các truy vấn thì có thể sử dụng View để lưu trữ phép tính.

+ Phân quyền và bảo mật: Với tính chất chỉ Xem chứ không được thay đổi dữ liệu, nên khi bạn muốn cho ai đó xem thì bạn có thể cho họ xem View thay vì xem Table.Ngoài ra, nó là 1 table nên bạn có thể giới hạn phân quyền cho table một cách dễ dàng.

**36: Thực hành tạo, sử dụng và xóa view?**

* Tạo và sử dụng view. Chỉ mục thứ 32
* Xóa View. Câu lệnh như sau

**DROP VIEW ten\_view;**

**37: Qui tắc đặt tên và qui tắc viết code trong SQL?**

* **Qui tắc:**

+ Tên không được có số ở đầu. Ví dụ: 1sst, 2ab, 3xyz, …

+ Tên được phép chứa dấu cách, nhưng phải đặt trong cặp []. Ví dụ như [ho ten], [ngày sinh], [db sinh viên], [môn học] … là hợp lệ

+ tên được phép có chứa kí tự có dấu. Ví dụ như họ tên, ngàysinh, dbsinhvien, mônhọc, …

+ tên được phép trùng với từ khóa nhưng phải đặt trong cặp []. Ví dụ như [ho ten], [ngày sinh], [db sinh viên], [môn học] …

+ tên không được chưa dấu ‘@’ ở đầu, vì dấu ‘@’ thể hiện cho biến

+ tên không được chứa phép toán. Ví dụ, n\*m, !m, a/b,… là sai.

**38: Áp dụng qui tắc đặt tên và qui tắc viết code vào trong thực hành?**

**39: Khái niệm DML:**

* DML là viết tắt của Ngôn ngữ Thao tác Dữ liệu (Data Manipulation Language): INSERT, UPDATE và DELETE là các câu lệnh DML.

**40: Câu lệnh Insert INTO:**

* Câu lệnh *insert into*: là câu lệnh dùng để thêm 1 bản ghi mới vào bảng trong cơ sở dữ liệu

**41: thực hành chèn dữ liệu vào bảng sử dụng câu lệnh INSERT?**

* Cú pháp như sau:

**INSERT INTO TEN\_BANG (cot1, cot2, cot3, ...cotN)**

**VALUES (gia\_tri1, gia\_tri2, gia\_tri3, ...gia\_triN);**

**42: Câu lệnh Update?**

* là truy vấn dùng để chỉnh sửa những bản ghi đã tồn tại trong bảng. Bạn có thể sử dụng mệnh đề WHERE để thêm điều kiện chỉnh sửa nếu không muốn toàn bộ các bản ghi đều bị thay đổi

**43: thực hành chỉnh sửa dữ liệu sử dụng câu lệnh UPDATE?**

* Cú pháp câu lệnh như sau:
* UPDATE *table\_name*  
  SET *column1*=*value1*,*column2*=*value2*, ...  
  WHERE *condition*;

**44: Câu lệnh DELETE:**

* Câu lệnh DELETE được dùng để xóa các bản ghi hiện có trong bảng

**45: Thực hành với câu lệnh DELETE:**

* cú pháp lệnh DELETE như sau:

DELETE FROM *table\_name*WHERE *condition*;

**46: So sánh DELETE, TRUNCATE với DROP:**

* DELETE chỉ xóa bản ghi ra khỏi Table, có thể khôi phục lại được
* TRUNCATE xóa toàn bộ bản ghi ra khỏi TABLE, và không khôi phục lại được
* DROP: xóa luôn table ra khỏi Database

**47: Hiểu tác dụng và các thành phần trong câu lệnh SELECT?**

* Các thành phần trong câu lệnh SELECT:

+ FROM: danh sách các table hoặc query /view [as alas]

+ where: điều kiện, lọc bản ghi

+ Group By: danh sách các thuộc tính \_gom nhóm

+ Having: điều kiện tính toán nhóm

+ Order By: Danh sách các thuộc tính \_sắp xếp [ASC|DESC]

**48: Thực hành truy vấn dữ liệu sử dụng SELECT?**

**49: Syntax ALIAS (từ khóa AS)?**

* **ALIAS** trong SQL là cách đổi tên một bảng hoặc một cột tạm thời bằng cách đặt tên khác. Việc sử dụng các bí danh bảng (table alias) là để đổi tên một bảng trong một câu lệnh SQL cụ thể. Việc đổi tên là một thay đổi tạm thời và tên bảng thật sự không thay đổi trong cơ sở dữ liệu.
* Trường hợp bắt buộc phải sử dụng AS: Trường hợp ta tính toán số lượng các bản ghi theo yêu cầu thì những giá trị này phải được lưu trong 1 cột mới có bí danh được ta đặt thông qua từ khóa AS.

**50: Thực hành với ALIAS:**

* Cú pháp AS:

**SELECT column1, column2....**

**FROM table\_name AS alias\_name**

**WHERE [condition];**

**51: Câu lệnh Group by?**

* Trong SQL, lệnh SQL được dùng kết hợp với lệnh SELECT để sắp xếp dữ liệu từ nhiều bản ghi đồng nhất vào trong các nhóm
* Lệnh GROUP BY thường được sử dụng đi kèm với các function COUNT (), MAX (), MIN (), SUM (), AVG () để nhóm tập hợp kết quả theo 1 hoặc nhiều cột.

**52: Thực hành với lệnh GROUP BY:**

* Cú pháp của lệnh GROUP BY:

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *condition*  
GROUP BY *column\_name(s)*

**53: Mục đích của HAVING trong câu truy vấn:**

* HAVING là câu lệnh điều kiện trả kết quả đối chiếu cho nhóm được tạo bởi mệnh đề GROUP BY.

**54: Đã thực hành với câu lệnh HAVING trong câu truy vấn**

* Cú pháp thực hiện:

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *condition*  
GROUP BY *column\_name(s)*HAVING *condition*

**55: So sánh HAVING với WHERE**

* **Sự khác nhau cơ bản của 2 mệnh đề này chính là ở chức năng mà nó đảm nhận**
* WHERE là mệnh đề điều kiện trả về kết quả đối chiếu cho từng bản ghi
* HAVING là mệnh đề điều kiện trả về kết quả đối chiếu cho nhóm được tạo bởi mệnh đề GROUP BY

**56: Từ khóa ORDER BY**

* Sắp xếp bản ghi theo chiều tăng dần hoặc giảm dần. Mặc định là sắp xếp tăng dần, để sắp xếp giảm dần ta dùng từ khóa DESC.

**57: Đã thực hành với từ khóa ORDER BY**

* Cú pháp câu lệnh:

ORDER BY *column1, column2, ...*ASC|DESC;

**58: Sử dụng một số hàm conversion functions cơ bản:**

* Convert functions:

+ Cú pháp: CONVERT (kieudulieu(do\_dai), bieuthuc, dinh\_dang);

*Kieudulieu*: tên kiểu dữ liệu mới mà biểu thức sẽ được chuyển đổi sang

*Dodai* (không bắt buộc): độ dài kiểu dữ liệu cho kết quả của char, varchar, nchar, nvarchar, binary hoặc image.

*Dinh dang* (không bắt buộc): là một con số chỉ định việc định dạng cho việc chuyển đổi dữ liệu từ dạng ngày sang dạng chuỗi.

**59: Một số hàm DATE and Time Function cơ bản:**

* **ADDDATE ()**: Thêm một thời gian vào date.

Cú pháp: có 2 dạng là

ADDDATE (date, interval expr unit)

ADDDATE (expr, day)

1: ADDDATE (expr, day)

Ex: SELECT ADDDATE ('2019-01-02', 31); => results = 2019 – 02 – 02

2: ADDDATE (date, interval expr unit)

Ex: SELECT ADDDATE ('2019-01-02', INTERVAL 31 DAY); => results = 2019 – 02 – 02

* **ADDTIME ()**: Thêm 1 khoảng thời gian vào time

ADDTIME () trong SQL có dạng: **ADDTIME (expr1, expr2).**

Hàm này cộng giá trị expr2 vào expr1 và trả kết quả Expr1: time || datetime

Expr2: time

EX: SELECT ADDTIME ('2018-12-31 23:59:59.999999','1 1:1:1.000002'); => results = 2019 – 01 – 02 01:01:01.000001

* **DATEDIFF ()**: trả về chênh lệch giữa 2 giá trị thời gian

ở trong SQL DATEDIFF () có dạng DATEDIFF (expr1, expr2). Giá trị chênh lệch thời gian được xác định dựa trên 2 khoảng thời gian được chỉ định là expr1 và expr2. Chúng phải có định dạng là hoặc datetime. Time chênh lệch = expr1 - expr2;

EX: SELECT DATEDIFF('2019-12-31 23:59:59','2019-12-30'); => results = 1

* **DATEPART() :** trả về một giá trị thời gian của đối số truyền vào , có thể là ngày, tháng, năm, quí, giờ, phút, giây, mili giây. Giá trị trả về là kiểu số nguyên (int)

Cú pháp sử dụng DATEPART(): DATEPART(dangthoigian, thoigian);

EX: SELECT DATEPART(year, '2019/04/28');

Result: 2019

* Còn nhiều hàm khác. Link tham khảo: <https://quantrimang.com/ham-xu-ly-date-time-trong-sql-phan-1-162457>

**60: Những hàm Aggregate ?**

* SUM() dùng để tính tổng các giá trị của cột nào đó

SUM(column-name)

* AVG() dùng để tính giá trị trùng bình của của các giá trị trong 1 cột nào đó

AVG(column\_name)

* COUNT() dùng để đếm số bản ghi lấy được từ câu lệnh SELECT

COUNT(\* hoặc column\_name)

* MAX() dùng để lấy giá trị lớn nhất của cột nào đó

MAX(column\_name)

* MIN() dùng để lấy giá trị nhỏ nhất của cột nào đó

MIN(column\_name)

**61: Thực hành sử dụng các hàm Aggregate?**

**62: Hiểu toán tử UNION và UNION ALL?**

* UNION: Gộp kết quả 2 lệnh SELECT và loại bỏ đi những kết quả bị trùng

Cú Pháp: SELECT statement1

UNION

SELECT statement2

Điều kiện để gộp được:

+Tên của các column phải giống nhau

+Thứ tự của các column phải giống nhau

+Tổng các column phải giống nhau

* UNION ALL: Có cú pháp và tác dụng y hệt như UNION, chỉ có điều là lệnh UNION ALL sẽ giữ lại toàn bộ giữ liệu

Cú pháp: SELECT statement1

UNION ALL

SELECT statement2

Điều kiện để gộp được y hệt như UNION

**63: Thực hành sử dụng UNION và UNION ALL ?**

**64: Lệnh SELECT INTO:**

* Sao chép dữ liệu từ một bảng vào 1 bảng mới

Cú pháp:

SELECT \*

INTO bảng\_mới [IN database\_khác]

FROM bảng\_cũ

WHERE điều\_kiện; -> sao chép toàn bộ dữ liệu từ bảng cũ sang bảng mới

SELECT cột\_1, cột\_2, cột\_3, ...

INTO bảng\_mới [IN database\_khác]

FROM bảng\_cũ

WHERE điều\_kiện; -> chỉ sao chép một số cột

**65: Thực hành sử dụng SELECT INTO:**

**66: Hiểu lệnh INSERT INTO SELECT:**

* Câu lệnh INSERT INTO SELECT sao chép dữ liệu từ một bảng và chèn nó vào một bảng khác

Điều kiện :

+INSERT INTO SELECT yêu cầu các kiểu dữ liệu trong các bảng nguồn và đích phù hợp

+ các bản ghi hiện có trong bảng đích không bị ảnh hưởng

* Cú pháp:

+ INSERT INTO bảng\_2

SELECT \* FROM bảng\_1

WHERE điều\_kiện;

**67: Thực hành câu lệnh INSERT INTO SELECT:**

**68: Hàm cast:**

* Hàm CAST trong SQL Server chuyển đổi một biểu thức từ một kiểu dữ liệu này sang kiểu dữ liệu khác. Nếu chuyển đổi không thành công, CAST sẽ báo lỗi, ngược lại nó sẽ trả về giá trị chuyển đổi tương ứng.

Cú Pháp:

**96: CROSS JOIN:**

* Khi ta thực hiện phép CROSS JOIN hay có một cái tên khác là Cartesian JOIN, mỗi hàng của bảng thứ nhất sẽ được kế hợp với mỗi hàng của bảng thứ hai. VÌ thế mà số lượng các hàng trong bảng mới được tạo ra bằng tích số lượng giữa các hàng
* Cú pháp của CROSS JOIN là:

SELECT \* FROM [BANG1] CROSS JOIN [BANG2]

Hoặc

SELECT \* FROM [ BANG1] , [BANG2]

**97: Thực hành với CROSS JOIN**:

**98: Khái niệm self join?**

* SELF JOIN trong SQL được sử dụng để thực hiện phép Join trên cùng 1 bảng bằng cách nối 1 bảng với chính nó, coi như nó là 2 bảng và thay thế tên tạm thời ít nhất 1 bảng
* Cú pháp:
* SELECT a.ten\_cot, b.ten\_cot...  
  FROM bang1 a, bang1 b  
  WHERE a.cot\_chung = b.cot\_chung;

**99: đã thực hành với self join:**

**100: thực hành với 3 loại join:**

**101: Khái niệm Sub-query?**

* Sub-query trong SQL là truy vấn trong 1 truy vấn SQL khác và được nhúng trong mệnh đề WHERE

**102: khi nào/Lợi ích khi sử dụng Sub-query:**

* Cái này mình k tìm hiểu được luôn nha các bạn

**103: Phân loại Sub-query?**

* Truy vấn lồng phân cấp:

+ mệnh đề WHERE của truy vấn con không tham chiếu đến thuộc tính của các bảng trong mệnh đề FROM của truy vấn cha

+ truy vấn con được thực hiện trước truy vấn cha

* Truy vấn lồng tương quan:

+ Mệnh đề WHERE của truy vấn con tham chiếu đến thuộc tính của các bảng trong mệnh đề FROM của truy vấn cha

+ Truy vấn con được thực hiện nhiều lần, mỗi lần ứng với một bộ truy vấn cha.

**104: Thực hành truy vấn với Sub-query**

**105: Khái niệm Common Table Expressions**

* CTE (Common Table Expressions) có thể được xem như 1 bảng chwua dữ liệu tạm thời tương tự như 1 bảng dẫn xuất (derived table) ở chỗ nó không được lưu trữ như 1 đối tượng và chỉ kéo dài trong suốt thời gian của câu truy vấn. Không giống như bảng dẫn xuất, CTE có thể tự tham chiếu đến bản thân của nó và có thể tham chiếu nhiều lần trong 1 câu truy vấn.

**106: Thực hành sử dụng CTE?**

* Cú pháp sử dụng CTE:

WITH expression\_name [ ( column\_name [,...n] ) ]

AS

(

CTE\_query\_definition

)

SELECT

FROM expression\_name;

**107: Một số hàm ranking ();**

* ROW\_NUMBER (): dùng để xếp hạng tập kết quả 1 cách tuần tự, mà không quan tâm đến các giá trị giống nhau.

Cú pháp:

SELECT ROW\_NUMBER ()

OVER (ORDER BY Tên\_cột asc/desc) AS Bí\_danh,

Các\_cột FROM Tên\_bảng;

* DENSE\_RANK (): Hàm DENSE\_RANK () dùng để xếp hạng các giá trị của 1 cột theo qui tắc các giá trị giống nhau sẽ có hạng lớn hơn 1 đơn vị so với hạng của các giá trị ngay trước nó.

Cú pháp:

SELECT DENSE\_RANK ()

OVER (ORDER BY Tên\_cột asc/desc) AS Bí\_danh,

Các\_cột FROM Tên\_bảng;

* RANK (): hàm RANK () dùng để xếp hạng các giá trị của một cột theo quy tắc giống hàm DENSE\_RANK (), nhưng khác ở 1 điểm đó là hạng của giá trị nằm ngay sau các giá trị giống nhau đó cộng với số lượng các giá trị giống nhau.

Cú pháp:

SELECT RANK ()

OVER (ORDER BY Tên\_cột asc/desc) AS Bí\_danh,

Các\_cột FROM Tên\_bảng;

**108: Thực hành Paging với Row\_Number ();**

**109: Cách comment trong SQL:**

Trong SQL chúng ta có 3 cách sau:

* **Single line comment (comment 1 dòng):** bắt đầu và kết thúc trên cùng 1 dòng

EX:

-- single line comment

-- another comment

SELECT \* FROM Customers;

* **Multi-line comments (comment nhiều dòng):** được bắt đầu và kết thúc ở các dòng khác nhau

EX:

\* multi line comment

another comment \*/

SELECT \* FROM Customers;

* **Inline comment (comment trên cùng dòng code):** loại comment đứng chung với đoạn mã SQL, có cú pháp tương tự multi-line comments.

EX:

SELECT \* FROM /\* Customers; \*/

**110: Quy tắc đặt tên định danh (bảng, biến, hàm, thủ thục, ..) trong SQL?**

* 1. Sử dụng một kí tự alphabet ở đầu tên, không sử dụng các kí tự "\_" gạch dưới, hoặc số ở đầu tên biến.
* 2. Giới hạn việc viết tắt quá nhiều trong tên biến hoặc đối tượng. Điều này dễ gây nhầm lẫn.
* 3. Giới hạn việc sử dụng các từ viết tắt hai lần trong tên.
* 4. Đặt tên mang ý nghĩa đúng với nội dung và tính chất của biến hoặc đối tượng.
* 5. Không sử dụng khoảng trắng(space) hoặc các kí tự đặc biệt trong tên như \*,$,#...
* 6. Tên bảng thường không cần bắt đầu bằng prefix "tbl" hay "tb" và tên bảng nên để ở dạng số ít như "Student" thay vì "Students". Tên bảng nên sử dụng Pascal Case tức là viết hoa chữ cái đầu của mỗi từ trong tên như SaleDepartment.
* 7. Nguyên tắc đặt tên cột.
* \* Nên đặt tên cột theo kiểu Pascal, hạn chế dùng theo kiểu camelCase( kiểu lạc đà - tức là chữ đầu không viết hoa mà viết thường còn các chữ sau có thể viết hoa chữ cái đầu của mỗi từ như myDocument).
* \* Đặt tên khóa chính nên có hậu tố Id thường thì công thức sẽ là tên bảng + "Id".
* \* Đối với khóa ngoại thì nên có tên giống với khóa chính mà nó tham chiếu tới. Ví dụ khóa chính của bảng Staff là "StaffId" và bảng [Class] có khóa ngoại đến bảng Staff thì nó có tên là "StaffId".
* \* Không nên sử dụng các tiến tố trước tên cột.
* \* Tên nên chỉ ra loại dữ liệu mà nó chứa ví dụ bạn có 1 trường dữ liệu lưu việc sinh viên có đi học vào ngày nào đó hay không thì thay vì đặt tên biến là Absent bạn có thể đặt là IsAbsent với kiểu dữ liệu là bit.
* \* Tên cột phải có ý nghĩa và không nên viết tắt quá nhiều nhưng bạn phải lưu ý đến độ dài của tên cột nên nằm trong khoảng dưới 30 chars là tốt nhất nhưng giới hạn của nó là 50 chars.
* \* Tên cột có thể chứa số nhưng không chứa các kí tự đặc biệt.
* 8. Đặt tên cho index thì nên bắt đầu bằng "IX\_", đặt tên cho unique column thì nên bắt đầu là "UN\_" hoặc "U/N"
* 9. Đặt tên cho ràng buộc(Constraint)
* \* Công thức: loại ràng buộc+ tên bảng + "\_"+tên cột
* \* Một số prefix cho các loại ràng buộc.
* o. Primary Key: Pk ví dụ PkProduct\_Id.
* o. Foreign Key: Fk ví dụ FkOrder\_ProductId.
* o. Check: Ck ví dụ CkStudent\_Age.
* o. Unique: Un ví dụ UnContact\_Phone.
* 10. Đặt tên cho view.
* \* Nên bắt đầu bằng "vw".
* \* Tên nên có ý nghĩa và nên là sự kết hợp của các bảng mà nó join lại với nhau. Ví dụ bạn tạo view join bảng Product và bảng Category thì lúc tạo view nó sẽ tên là vwProductCategory.
* 11. Đặt tên cho thủ tục.
* \* Nên bắt đầu bằng "sp".
* \* Đặt tên theo mục đích sử dụng của thủ tục đó ví dụ thủ tục tạo product mới spProductCreate hay xóa product spProductDelete.
* 12. Đặt tên cho các hàm.
* \* Nên bắt đầu bằng "fn".
* \* Tên hàm nên có ý nghĩa là như những động từ chỉ chức năng mà nó sẽ thực hiện như fnGetTime...
* 13. Đặt tên cho trigger.
* \* Nên bắt đầu bằng "tr".
* \* Nếu trigger thực hiện nhiều chức năng thì tên nên chứa ý nghĩa thực hiện những chức năng đó ví dụ: trProductInsertUpdate.
* 14. Đặt tên cho dữ liệu tự định nghĩa.
* \* Nên bắt đầu bằng "ud".
* \* Không chứa các kí tự đặc biệt, chỉ nên chứa chữ và số.
* 15. Đặt tên cho biến.
* \* Tên biến không nên quá dài(giới hạn là 50 chars).
* \* Tên biến phải bắt đầu bằng @ và không sử dụng @@ vì đây là cách đặt tên cho biến toàn cục của SQL.
* \* Tên biến nên đặt tên theo kiểu camelCase. Ví dụ @totalMark.
* \* Không sử dụng các kí tự đặc biệt.

**111: Khai báo biến (variables) trong SQL?**

* Biến(variables) dùng để lưu trữ các giá trị tạm thời trong quá trình thực thi thuật toán.
* Cú pháp khi báo biến trong SQL

DECLARE

@variable\_name datatype [ = initial\_value],

@variable\_name datatype [ = initial\_value],

...;

EX:

DECLARE @string VARCHAR (50);

lệnh DECLARE này khai báo một biến có tên @string với kiểu dữ liệu là VARCHAR có độ dài là 50 kí tự

Để gán giá trị cho biến @string ta sử dụng câu lệnh SET

SET @string = ‘Loi’

**112: Control – Of Flow trong SQL SERVER?**

* Các thành phần Control – Of Flow

+ Như BEGIN…END, BREAK, CONTINUE, GOTO, IF…ELSE, RETURN, WHILE….

Comments Chú Thích

**113: thực hành sử dụng lệnh IF, WHILE:**

**114 + 115: USER STORE PROCEDURE?**

* STORE PROCEDURE cho phép tập hợp nhiều câu lệnh SQL trong 1 chương trình con. Giống như các ngôn ngữ khác, STORE PROCEDURE cũng cho phép truyền tham số.
* Khai báo

CREATE PROCEDURE sp\_name ([parameter […,...]])

* Lợi ích của STORE PROCEDURE:

|  |  |
| --- | --- |
| **Lợi ích** | **Giải thích** |
| Module hóa | Bạn chỉ cần viết Store Procedure 1 lần, sau đó có thể gọi nó nhiều lần trong ứng dụng |
| Hiệu suất | Stored Procedure thực thi mã nhanh hơn và giảm tải băng thông.  - Thực thi nhanh hơn: Stored Procedure sẽ được biên dịch và lưu vào bộ nhớ khi được tạo ra. Điều đó có nghĩa rằng nó sẽ thực thi nhanh hơn so với việc gửi từng đoạn lệnh SQL tới SQL Server. Vì nếu bạn gửi từng đoạn lệnh nhiều lần thì SQL Server cũng sẽ phải biên dịch lại nhiều lần, rất mất thời gian so với việc biên dịch sẵn.  - Giảm tải băng thông: Nếu bạn gửi nhiều câu lệnh SQL thông qua network đến SQL Server sẽ ảnh hưởng tới hiệu suất đường truyền. Thay vì gửi nhiều lần thì bạn có thể gom các câu lệnh SQL vào 1 Stored Procedure và chỉ phải gọi đến 1 lần duy nhất qua network. |
| Bảo mật | Trong SQL Server có các tác vụ cấp cao mà người dùng bình thường không thể truy cập vào được. Bằng việc cung cấp các Stored Procedure đã truy cập tới các tác vụ này cho người dùng thường thì không sao hết. Vì làm vậy thì người dùng thường sẽ truy cập giántiếp mà không ảnh hưởng tới vấn đề bảo mật của SQL Server. |

**116: Lấy giá trị trả về khi sử dụng STORE PROCEDURE?**

* Để tạo tham số OUTPUT thì bạn vẫn sử dụng cách tạo như 1 tham số bình thường, có điều là bạn phải thêm từ khóa OUTPUT vào vị trí cuối cùng của khai báo.  
  EX:

CREATE PROCEDURE uspFindProductByModel (

@model\_year SMALLINT,

@product\_count INT OUTPUT

) AS

BEGIN

SELECT

product\_name,

list\_price

FROM

production. products

WHERE

model\_year = @model\_year;

SELECT @product\_count = @@ROWCOUNT;

END;

* Trong đó:

@@ROWCOUNT là biến toàn cục lưu trữ tổng số record trả về

SELECT @product\_count = @@ROWCOUNT; là gán tổng số record vào tham số OUTPUT

**117: Thực hành sử dụng Procedure:**

**118: User - Defined Function?**

* Hàm do người dùng định nghĩa (user defined functions - UDFs) là một đối tượng mới được bổ sung của phiên bản SQL Server 2000. Trước hết, bạn cần biết rằng UDFs mang đầy đủ các ý nghĩa và tính chất của một hàm như đa số các ngôn ngữ lập trình khác, có nghĩa là một UDFs là một chương trình con đảm trách một xử lý nào đó với đặc tính là sẽ nhận các tham số đầu vào và trả về một giá trị kết quả xử lý tính toán được dựa trên các tham số đầu vào đã nhận
* Không chỉ UDFs mà tất cả các hàm nói chung (các phiên bản SQL Server trước đây cung cấp các hàm được cài đặt sẵn như getdate (), object\_name (), …) có thể phân thành 2 nhóm:
  + hàm xác định(deterministic)
  + hàm không xác định (non - deteministic).

**119: How many types of UDFs:**

* Như ở trên đã nói có 2 kiểu cho UDFs là các hàm xác định và các hàm không xác định
* Cụ thể như sau:

+ Các hàm xác định sẽ luôn luôn trả về cùng giá trị nếu giá trị các tham số được cung cấp (truyền vào) là như nhau.

+ Các hàm không xác định có thể tạo ra các kết quả khác biệt tại mỗi thời điểm chúng được gọi thực hiện, ngay cả khi giá trị các tham số được cung cấp vẫn không thay đổi

**120: Thực hành sử dụng function User Scalar – valued Function?**

Tạo 1 bảng mcninvoices (trả về 1 bản ghi)

* **create** **table** mcninvoices
* (
* invoiceid **int** **not null** identity **primary** **key**,
* vendorid **int** notnull,
* invoiceno **varchar** (15),
* invoicetotal money,
* paymenttotal money,
* creadittotal money
* )

Insert giá trị vào bảng:

* **insert** **into** mcninvoices **values** (20,'e001',100,100,0.00)
* **insert** **into** mcninvoices **values** (21,'e002',200,200,0.00)
* **insert** **into** mcninvoices **values** (22,'e003',500,0.00,100)
* **insert** **into** mcninvoices **values** (23,'e004',1000,100,100)
* **insert** **into** mcninvoices **values** (24,'e005',1200,200,500)

Tạo thêm 1 bảng mcnvendors

* create **table** mcnvendors
* (
* vendorid **int**,
* vendorname **varchar** (15),
* vendorcity **varchar** (15),
* vendorstate **varchar** (15)
* )

Insert giá trị vào bảng:

1. **insert** **into** mcnvendors **values** (20,'vipendra','noida','up')
2. **insert** **into** mcnvendors **values** (21,'deepak','lucknow','up')
3. **insert** **into** mcnvendors **values** (22,'rahul','kanpur','up')
4. **insert** **into** mcnvendors **values** (23,'malay','delhi','delhi')
5. **insert** **into** mcnvendors **values** (24,'mayank','noida','up')

* Viết hàm user-defined (Scalar-valued function)
* **CREATE** **FUNCTION** fnbal\_invoice ()
* **RETURNS** MONEY
* **BEGIN**
* **RETURN** (**SELECT** SUM (invoicetotal-paymenttotal-creadittotal)
* **FROM** dbo. mcninvoices
* **WHERE** invoicetotal-paymenttotal-creadittotal > 0)
* **END**
* Sử dụng function:

PRINT ‘Out standing balance of company is: $ ‘+ CONVERT (VARCHAR, fnbal\_invoice () , 1)

**121: thực hành sử dụng function User Table-valued Function?**

* là một hàm do người dùng định nghĩa trả về dữ liệu của một loại bảng. Kiểu trả về của **User Table-valued Function** là một bảng, do đó, bạn có thể sử dụng hàm định trị bảng giống như cách bạn sử dụng một bảng.
* EX:

CREATE FUNCTION udfProductInYear (

@model\_year INT

)

RETURNS TABLE

AS

RETURN

SELECT

product\_name,

model\_year,

list\_price

FROM

production. products

WHERE

model\_year = @model\_year;

* Sử dụng function: Vì là hàm có giá trị trả về là 1 bảng nên là ta có thể sử dụng function này như là 1 bảng

SELECT

\*

FROM

udfProductInYear (2017);

* + Lệnh select trên trả về các bản ghi là 1 table mà các bản ghi có giá trị model\_year là 2017

**122: Sự khác biệt giữa Store procedure và User-define function**:

|  |  |
| --- | --- |
| **Stored procedure** | **Users define function** |
| Có thể trả về giá trị 0, 1 hoặc nhiều giá trị | Trả về giá trị duy nhất hoặc là 1 bảng |
| Có thể có các tham số đầu vào và ra | Chỉ có thể có các tham số đầu vào |
| Có thể gọi được hàm | K thể gọi được procedure |
| Không thể sử dụng trong câu lệnh select vì nó có thể trả về nhiều tập kết quả nên nó không phù hợp để sử dụng câu select | Hàm có thể được nhúng trong câu lệnh select |
| Không thể sử dụng các câu lệnh Where/Having/Select | Có thể sử dụng các câu lệnh Where/Having/Select |
| Một ngoại lệ có thể được xử lý bằng Try-catch | Không thể sử dụng try-catch trong hàm để xử lý ngoại lệ |
| Có thể sử dụng được Transaction | Không thể sử dụng được Transactions |

**123: Trigger?**

* Trigger là 1 đối tượng được liên kết với bảng (table), được kích hoạt khi có những sự kiện (thêm, xóa, sửa…) xảy ra trong bảng.

Tạo 1 Trigge:

Create Trigger trigger\_name

[before|after] trigger event

ON table\_name FOR EACH ROW

Trigger\_stmt

Trong đó:

* + Trigger\_event: xác định loại sự kiện (Insert, update, delete)
  + trigger\_stmt: các lệnh được gọi khi trigger được kích hoạt, để sử dụng nhiều lệnh, dùng cấu trúc BEGIN…END giống như STORE PROCEDURE

**124: Khi nào sử dụng Trigger?**

* Trigger thường được sử dụng để kiểm tra ràng buộc (check constraints) trên nhiều quan hệ (nhiều bảng/table) hoặc trên nhiều dòng (nhiều record) của bảng.
* Ngoài ra, việc sử dụng trigger để chương trình có những hàm chạy ngầm nhằm phục vụ những TH hữu hạn và thường không sử dụng cho mục đích kinh doanh hoặc giao dịch
* Ngăn chặn việc xóa những dự liêu quan trọng. (có thể dùng back up các dữ liệu quan trọng sang 1 table khác phòng khi mà dữ liệu bị xóa ngoài ý muốn)

**125**: **How many types of Trigger?**

* Có 2 loại trigger: DDL (Data Definition Language) và DML (Data Manipulation Language)

+ DDL:   
Với loại trigger này được kích hoạt khi có thực hiện những lệnh sau: Create, alter và Drop. Như Create\_Table, create\_View, drop\_Table, drop\_View và Alter\_Table.

+ DML trigger: lại chia thành 2 loại:

* + AFTER TRIGGER: được thực thi sau khi hành động của câu lệnh INSERT, UPDATE, hoặc DELETE
  + INSTEAD OF Triggers: Trigger này sẽ báo cho bộ thực thi của csdl thực hiện trigger thay vì thực hiện câu lệnh. Ví dụ: Một Trigger Insert thực thi khi xuất hiện một sự kiện thay vì sử dụng câu lệnh insert giá trị vào bảng đó.

**126: 2 bảng Inserted và delete:**

* Khi thực hiện một Trigger thì SQL tự động tạo ra 2 bảng Inserted và Deleted trong bộ nhớ chính và cục bộ cho mỗi Trigger, có nghĩa là khi áp dụng Trigger trên bảng nào thì bảng Inserted và Deleted sẽ được sử dụng riêng cho đó bảng đó.
* Cấu trúc 2 bảng Inserted và Deleted được tạo ra sẽ giống hệt cấu trúc của bảng mà Trigger đang thực thi và chúng chỉ tồn tại trong thời gian Trigger đó thực thi mà thôi.

**Inserted** là bảng chứa các dòng dữ liệu vừa được Insert hay Update vào bảng mà Trigger đang thực thi.

**Deleted** là bảng chứa các dòng dữ liệu mới được xóa khỏi bảng bằng thao tác Delete hay Update.

Khi thực hiện thao tác **Update**, thì đồng nghĩa với việc sẽ xóa những dòng dữ liệu cũ và thêm những dòng dữ liệu mới, khi đó tác Update sẽ vừa đồng thời thêm dữ liệu là các dòng mới vào 2 bảng Inserted và Deleted

**129: enable và disable trigger?**

* Khi ta muốn vô hiệu hóa tất cả các trigger hiện tại trên database để insert dữ liệu thì sẽ sử dụng enable và disable trigger. Disable khi insert, còn enable khi đã insert xong
* 1: Enable tất cả các trigger trên một bảng Table
* ENABLE TRIGGER ALL ON TableName;
* SQL
* 2. Disable tất cả các trigger trên một Table
* DISABLE TRIGGER ALL ON TableName;
* SQL
* 3. Enable tất cả các trigger trên một Database
* ENABLE TRIGGER ALL On DATABASE;
* SQL
* 4. Disable tất cả các trigger trên một Database
* DISABLE TRIGGER ALL ON DATABASE;
* SQL
* 5. Enable tất cả các trigger trên một Server
* ENABLE TRIGGER ALL ON SERVER;
* SQL
* 6. Disable tất cả các trigger trên một Server
* DISABLE TRIGGER ALL ON SERVER;

**130: So sánh trigger với store procedure?**

|  |  |
| --- | --- |
| **Trigger** | **Procedure** |
| Một Trigger sẽ thực thi tự động mỗi khi có các sự kiện INSERT, DELETE, UPDATE xảy ra trong một table. | Một Procedure sẽ được thi khi khi được gọi thông qua lệnh exec, EXECUTE, hay đơn giản procedure\_name |
| Không thể gọi một trigger trong một trigger khác. | Có thể gọi một procedure trong một procedure khác. |
| Cú pháp: CREATE TRIGGER TRIGGER\_NAME | Cú pháp: CREATE PROCEDURE PROCEDURE\_NAME |
| Các mệnh đề trong transaction như COMMIT, ROLLBACK, SAVEPOINT đều không được sử dụng trong trigger | Tất cả các mệnh đề trong transaction như COMMIT, ROLLBACK, SAVEPOINT đều được sử dụng trong procedure. |
| Trigger được sử dụng để duy trì tính toàn vẹn của dữ liệu bằng cách thực thi các mã lệnh trên các dòng dữ liệu mỗi khi xảy ra các hoạt động có thể ảnh hưởng đến dữ liệu. | Procedure được dùng để thực thi các công việc cụ thể được người dùng định nghĩa. |
| Trigger không có giá trị trả về cũng như các tham số đầu vào. | Hoạt động như một hàm, procedure có thể nhận các tham số và trả về các kết quả. |

**131: Transaction:**

* là tiến trình thực hiện một nhóm các câu lệnh SQL. Các câu lệnh này được thực thi một cách tuần tự và độc lập. Một Transaction được thực hiện thành công khi tất cả câu lệnh đều thành công, khi đó tất cả các thay đổi dữ liệu được thực hiện trong Transaction được lưu vào cơ sở dữ liệu. Tuy nhiên, nếu chỉ một trong số đó thất bại thì toàn bộ tiến trình sẽ thất bại, đồng nghĩa với việc dữ liệu phải rollback về trạng thái ban đầu (dữ liệu được khôi phục về trạng thái trước khi thực hiện Transaction).
* Thuộc tính của Transaction:

1. **Atomicity - Tính bảo toàn**: nguyên tắc "all or nothing", đảm bảo rằng tất cả các câu lệnh trong nhóm lệnh được thực thi thành công. Nếu không, Transaction bị hủy bỏ tại thời điểm thất bại và tất cả các thao tác trước đó được khôi phục về trạng thái cũ đồng nghĩa với việc không có gì thay đổi về mặt dữ liệu.
2. **Consistency - Tính nhất quán**: đảm bảo rằng cơ sở dữ liệu thay đổi chính xác các trạng thái khi một transaction được thực thi thành công.
3. **Isolation - Tính độc lập**: cho phép các Transaction hoạt động độc lập và minh bạch với nhau.
4. **Durability - Tính bền vững:** đảm bảo rằng kết quả của một transaction được xác định, không có chuyện dữ liệu của Transaction sau khi thực thi có thể chuyển lại trạng thái dữ liệu lúc trước khi thực hiện.

* **Các lệnh sử dụng để điều khiển transaction:**

**COMMIT**: để lưu các thay đổi.

**ROLLBACK**: để quay trở lại trạng thái trước khi có thay đổi.

**SAVEPOINT**: tạo các điểm (point) bên trong các nhóm Transaction để ROLLBACK, tức là để quay trở lại điểm trạng thái đó.

**SET TRANSACTION**: đặt một tên cho một Transaction.

**132**: **Tại sao phải sử dụng Transaction?**

* Vì đảm bảo tính toàn vẹn dữ liệu khi xảy ra cập nhật (insert, update, delete…). Một Transaction bao gồm nhiều lệnh cập nhật, nó đảm bảo tất cả cập nhật đều được thực hiện thành công, hoặc trong TH là 1 lệnh gặp sự cố thì tất cả các lệnh khác đều bị hủy bỏ. Khi đó dữ liệu trở về trạng thái như trước khi xảy ra transaction.
  + Transaction ngăn chặn tình huống dữ liệu bị cập nhật nửa chừng, trong đó một phần được cập nhật, một phần được bỏ qua.

Ví dụ như việc chuyển tiền giữa 2 tài khoản ngân hàng. A chuyển tiền cho B 2 tỉ. như vậy ta phải thực hiện 2 việc là: trừ A 2 tỉ và cộng cho B 2 tỉ. Nếu như k sử dụng transaction thì khi xảy ra sự cố thì tài khoản A bị mất đi 2 tỉ và tài khoản B k được cộng thêm tiền. Điều này sẽ được khắc phục khi sử dụng transaction. Nó sẽ đảm bảo rằng:

* + Cả 2 tài khoản sẽ được cập nhật với số tiền tương ứng hoặc là không thực hiện giao dịch.

**133: Thực hành sử dụng transaction:**

**134: Khái niêm ACID?**

* **ACID** (viết tắt **của Atomicity, Consistency, Isolation, Durability**) là một khái niệm cơ sở dữ liệu mà các chuyên gia thường tìm kiếm khi đánh giá các cơ sở dữ liệu và kiến trúc ứng dụng. Đối với một cơ sở dữ liệu đáng tin cậy tất cả bốn thuộc tính cần đạt được.
* **Atomicity** là một đề xuất tất cả hoặc không có gì. Tính chất này đảm bảo rằng khi một giao dịch liên quan đến hai hay nhiều xử lý, hoặc là tất cả các xử lý được thực hiện hoặc không có xử lý được thực hiện.
* **Consistency** đảm bảo rằng một giao dịch không bao giờ được thông qua cơ sở dữ liệu của bạn trong tình trạng dở dang. Tính chất này, hoặc là tạo ra toàn bộ trạng thái mới hoặc rollback tất cả các xử lý để về trạng thái ban đầu, nhưng không bao giờ thông qua cơ sở dữ liệu trong trạng thái dở dang.
* **Isolation** giữ giao dịch tách rời nhau cho đến khi chúng đã hoàn tất. Tính chất này đảm bảo rằng hai hoặc nhiều giao dịch không bao giờ được trộn lẫn với nhau, tạo ra dữ liệu không chính xác và không phù hợp.
* **Durability** đảm bảo rằng cơ sở dữ liệu sẽ theo dõi các thay đổi cấp phát trong một cách mà các máy chủ có thể phục hồi từ một sự kết thúc bất thường. Tính chất này đảm bảo rằng trong trường hợp thất bại hay dịch vụ khởi động lại các dữ liệu có sẵn trong trước khi gặp lỗi.