**BÁO CÁO EX03\_MYSQL**

1. SQL VÀ NOSQL
2. SQL là gì?

* SQL (Structured Query Language) là ngôn ngữ được sử dụng để truy vấn và quản lý cơ sở dữ liệu quan hệ (Relational Database Management System – RDBMS). SQL được phát triển nhằm cung cấp các công cụ và câu lệnh để tương tác với cơ sở dữ liệu, bao gôm việc truy vấn, chèn, cập nhật và xóa dữ liệu.
* Một số đặc điểm chính của SQL:

+ Ngôn ngữ query: SQL sử dụng cú pháp đơn giản để truy vấn cơ sở dữ liệu. Các câu lệnh SELECT, INSERT, UPDATE và DELETE được sử dụng để truy vấn và thao tác dữ liệu.

+ Cơ sở dữ liệu quan hệ: SQL được thiết kế để làm việc với cơ sở dữ liệu quan hệ, trong đó dữ liệu được tổ chức thành các bảng có quan hệ với nhau thông qua khóa ngoại.

+ Tính nhất quán (Consistency): SQL hỗ trợ tính nhất quán của cơ sở dữ liệu thông qua các ràng buộc (constraints) như khóa chính (primary key), khóa ngoại (foreign key) và ràng buộc kiểu dữ liệu (data type constraints).

+ Truy cập nhật dữ liệu: SQL cho phép truy cập dữ liệu từ cơ sở dữ liệu bằng cách sử dụng các câu lệnh SELECT. Người dùng có thể chỉ định các điều kiện (mệnh đề WHERE) để lấy ra các bản ghi thỏa mãn yêu cầu.

+ Cập nhật dữ liệu: SQL cung cấp các câu lệnh INSERT, UPDATE và DELETE để thêm mới cập nhật và xóa dữ liệu trong cơ sở dữ liệu.

+ Xử lý dữ liệu tổng hợp: SQL cung cấp các câu lệnh GROUP BY, HAVING và các hàm tổng hợp như (SUM, COUNT, AVG, MAX, MIN) để xử lý dữ liệu tổng hợp và trích xuất thông tin thống kê tử cơ sở dữ liệu.

+ Đặc trưng hướng đối tượng: Một số RDBMS hỗ trợ các đặc trưng hướng đối tượng (Object-Oriented) như lưu trữ các kiểu dữ liệu phức tạp, phương thức và kế thừa.

* SQL là một công cụ quan trọng trong việc làm việc với cơ sở dữ liệu và rất phổ biến trong lĩnh vực phát triển phần mềm và quản lý dữ liệu.

1. NoSQL là gì?

* NoSQL (Not only SQL) là một ngôn ngữ dùng để đề cập đến các quan hệ quản trị cơ sở dữ liệu không tuân theo mô hình quan hệ truyền thống của SQL và thường đươc sử dụng để xử lý dữ liệu phi quan hệ, phân tán và lưu trữ lớn.
* Một số đặc điểm chính của NoSQL:

+ Mô hình dữ liệu linh hoạt: NoSQL cho phép lưu trữ và xử lý các dạng dữ liệu đa dạng, bao gồm dữ liệu có cấu trúc, bán cấu trúc và không có cấu trúc. Nó không yêu cầu sự tuân theo một mô hình qua hệ cụ thể nào và có thể thích nghi với các mô hình dữ liệu khác nhau.

+ Khả năng mở rộng dễ dàng: Các hệ quản trị cơ sở dữ liệu NoSQL thường được thiết kế để mở rộng theo chiều ngang, cho phép thêm các máy chủ mới để tăng khả năng xử lý và lưu trữ dữ liệu. Điều này giúp đáp ứng tốt với các ứng dụng có lưu lượng truy vấn lớn và dữ liệu phân tán.

+ Tốc độ và hiệu xuất cao: NoSQL thường được tối ưu hóa để đạt được hiệu suất cao và xử lý các tác vụ truy vấn nhanh chóng. Điều này đặc biệt hữu ích trong các ứng dụng có yêu cầu thời gian thực hoặc cần xử lý dữ liệu lớn.

+ Khả năng xử lý lỗi: Các hệ quản trị cơ sở dữ liệu NoSQL thường có khả năng tự phục hồi và chịu lỗi tốt hơn trong trường hợp một máy chủ hoặc thành phần của hệ thống gặp sự cố. Điều này giúp đảm bảo tính sẵn sàng cao và giảm thiểu thời gian chết ứng dụng.

+ Phân tán và đa vùng: Các hệ quản trị cở sở dữ liệu NoSQL thường hỗ trợ việc phân tán dữ liệu trên nhiều máy chủ và vị trí vật lý khác nhau. Điều này cho phép lưu trữ dữ liệu theo mô hình đa vùng, giúp cải thiện khả năng sẵn sàng và chống chịu lỗi.

+ Không có ràng buộc schema: NoSQL cho phép lưu trữ dữ liệu mà không yêu cầu một schema (cấu trúc) cụ thể trước. Điều này cho phép dữ liệu có cấu trúc khác nhau được lưu trữ trong cùng một hệ thống, và giúp dễ dàng thay đổi cấu trúc dữ liệu mà không gây ra sự gián đoạn trong ứng dụng.

* Tuy NoSQL có những đặc điểm và lợi ích riêng, việc sử dụng nó cần phải xem xét kỹ lưỡng và dựa trên yêu cầu cụ thể của dự án, bởi vì nó cũng có nhược điểm và hạn chế nhất định trong một số tình huống sử dụng.

1. CÁC CÂU LỆNH VÀ HÀM TRONG SQL
2. JOIN là gì?

* Join được sử dụng để kết hợp các bản ghi từ hai hay nhiều bảng trong cơ sở dữ liệu. Join là một phương tiện để kết hợp các trường từ hai bảng bằng cách sử dụng các giá trị chung của mỗi bảng.
* Các kiểu Join trong SQL:
* INNER JOIN: trả về các bản ghi có giá trị khớp trị hai bảng.
* LEFT JOIN: trả về tất cả các bản ghi từ bên trái và mọi bản ghi phù hợp từ bảng bên phải. Nếu không khớp, giá trị Null sẽ được hiển thị cho các cột của bảng bên phải.
* RIGHT JOIN: trả về tất cả các bản ghi bên phải và mọi bản ghi phù hợp từ bảng bên trái. Nếu không khớp, giá trị Null sẽ được hiển thị cho cột của bảng bên trái.
* FULL JOIN: là sự kết hợp của cả LEFT JOIN và RIGHT JOIN, trả về các bản ghi có giá trị khớp trong cả hai bảng, cũng như bất kỳ các bản ghi nào chưa khớp từ một trong hai bảng.

1. Function là gì?

* Function là một đối tượng trong cơ sở dữ liệu sử dụng trong các câu lệnh SQL, được biên dịch sẵn và lưu trong cơ sở dữ liệu nhằm mục đích thực hiện xử lý nào đó như tính toán phức tạp và trả về kết quả là giá trị nào đó.
* Đặc điểm:
* Luôn trả về giá trị.
* Gồm 2 loại: Function hệ thống và Function do người dùng tự định nghĩa.
* Function người dùng tự định nghĩa gồm 2 loại:
* Scalar-valued: trả về giá trị vô hướng của các kiểu dữ liệu T-SQL.
* Table-valued: trả về bảng, là kết quả của một hoặc nhiều lệnh.

1. Stored Procedure là gì?

* Stored Procedure là một khối mã lưu trữ trong cơ sở dữ liệu, chứa một tập hợp các câu lệnh SQL và logic xử lý, có thể được gọi và thực thi từ ứng dụng và gọi trực tiếp từ cơ sở dữ liệu.
* Ưu điểm:

+ Hiệu xuất: Stored Procedure thường được biên dịch và lưu trữ trong cơ sở dữ liệu, giúp giảm thiểu việc truyền thông qua mạng và tăng tốc độ khi thực thi.

+ Tái sử dụng: Stored Procedure có thể được gọi từ nhiều ứng dụng khác nhau và giúp giảm việc lặp đi lặp lại.

+ Bảo mật: Stored Procedure có thể được phân quyền riêng biệt, giúp kiểm soát quyền truy cập vào dữ liệu và bảo vệ an toàn dữ liệu.

* Nhược điểm:

+ Khó bảo trì: thủ tục lưu trữ có thể trở nên phức tạp và khó hiểu, đặc biệt khi có nhiều tác vụ và logic phức tap.

+ Khó di động: Stored Procedure có thể phụ thuộc vào cấu trúc của hệ quản trị cơ sở dữ liệu cụ thể, khiến việc di chuyển sang hệ thống khác trở nên khó khăn.

1. Trigger là gì?

* Trigger là một loại đối tượng trong cơ sở dữ liệu, được kích hoạt tự động khi có sự thay đổi dữ liệu trong bảng liên quan.
* Ưu điểm:

+ Tự động hóa: Trigger tự đông hóa kích hoạt và thực thi các tác vụ khi có thay đổi dữ liệu, giúp đảm bảo tính nhất quán và hành động tự động dữ trên sự kiện.

+ Kiểm soát dữ liệu: Trigger cho phép kiểm tra và giám sát dữ liệu để áp dụng các quy tác kinh doanh hoặc thực hiện các hành đông cần thiết

+ Audit: Trigger có thể được sử dụng để ghi lại thông tin kiểm tra, theo dõi và gỡ lỗi các hoạt động trên cơ sở dữ liệu.

* Nhược điểm:

+ Phức tạp: Khi số lượng trigger và logic kích hoạt tăng, việc quản lý và hiểu rõ hệ thống trở nên khó khăn.

+ Hiệu suất: Sử dụng trigger có thể gây ảnh hưởng đến hiệu suất của cơ sở dữ liệu, đặc biệt khi có nhiều trigger được kích hoạt cùng một lúc.

1. Transaction là gì?

* Transaction (giao dịch) là một tập hợp các hoạt động cơ sở dữ liệu được thực hiện như một đơn vị nguyên tử, tức là nếu một phần của transaction gặp lỗi, toàn bộ transaction sẽ bị hủy bỏ và cơ sở dữ liệu trở lại trạng thái ban đầu trước khi giao dịch bắt đầu. Ngược lại nếu tất cả các hoạt động trong giao dịch thành công, thì giao dịch được xác nhận và các thay đổi sẽ được áp dụng vào cơ sở dữ liệu.
* Một transaction bao gồm:

+ Bắt đầu transaction (BEGIN): đánh dấu sự bắt đầu của một giao dịch.

+ Thực hiện các hoạt động (INSERT, UPDATE, DELETE): thay đổi dữ liệu trong cơ sở dữ liệu.

+ Đánh dấu lưu ý (COMMIT): áp dụng các thay đổi vào cơ sở dữ liệu và xác nhận giao dịch thành công.

+ Hủy bỏ transaction (ROLLBACK): hủy bỏ tất cả các thay đổi và đưa cơ sở dữ liệu trở lại trạng thái ban đầu.

* Ưu điểm:

+ Tính toàn vẹn: transaction đảm bảo tính vẹn toàn của dữ liệu trong quá trình thực hiện các thay đổi, đảm bảo rằng dữ liệu không bị mất mát hoặc bị lỗi.

+ Tính nhất quán: transaction đảm bảo rằng dữ liệu nằm trong cơ sở dữ liệu luôn đạt được trạng thái nhất quán, tức là tuân thủ các ràng buộc và quy tác.

+ Tự động phục hổi: trong trường hợp xảy ra lỗi, giao dịch có thể được hủy bỏ và cơ sở dữ liệu trở lịa trạng thái ban đầu.

* Nhược điểm:

+ Overhead hiệu năng: Việc thực hiện giao dịch có thể tạo ra overhead hiệu năng nhất định, bởi vì nó đòi hỏi thêm thời gian và tài nguyên để kiểm soát và ghi lại các thay đổi.

+ Không thể phục hồi hoàn toàn: mặc dù transaction đảm bảo tính nhất quán và toàn vẹn của dữ liệu, nhưng nó không phục hồi hoàn toàn trong mọi trường hợp, ví dụ như mất điện hoặc hỏng đĩa cứng.

1. Index là gì?

* Index trong SQL là một cấu trúc dữ liệu được sử dụng để tăng tốc độ truy xuất dữ liệu trong cơ sở dữ liệu. Index giúp tìm kiếm và truy xuất dữ liệu một cách nhanh chóng bằng cách tạo ra một bản định danh cho các cột được chỉ định.
* Khi một index được tạo trên một cột, hệ quản trị cơ sở dữ liệu sẽ được tạo ra một bảng từ điển bổ sung cho cột đó. Bảng từ điển này lưu trữ giá trị của cột và các con trỏ đến các hàng dữ liệu tương ứng. Khi truy vấn yêu cầu tìm kiếm hoặc sắp xếp dữ liệu trên cột đã được tạo index, hệ quản trị cơ sở dữ liệu sẽ được sử dụng bảng từ điển để tỉm kiếm nhanh chóng các giá trị tương ứng và truy xuất dữ liệu từ các hàng tương ứng.
* Các ưu điểm của việc sử dụng index trong SQL bao gồm:

+ Tăng tốc độ truy vấn: Index giúp tìm kiếm và truy xuất dữ liệu nhanh hơn, đặc biệt đối với các bảng có kích thước lớn hoặc có nhiều dữ liệu.

+ Cải thiện hiệu suất hệ thống: Việc sử dụng index giảm thời gian xử lý truy vấn, giúp hệ thống hoạt động mượt mà và đáp ứng được nhu cầu truy xuất dữ liệu cao.

+ Tăng hiệu quả tìm kiếm: Index cho phép tìm kiếm dữ liệu theo các điều kiện hoặc tiêu chí cụ thể, giúp dễ dàng và nhanh chóng lấy ra các bản ghi phù hợp.

* Tuy nhiên, việc sử dụng index cũng có một số nhược điểm:

+ Tăng tải cho việc cập nhật dữ liệu: Khi có các thao tác cập nhật dữ liệu (như chèn, cập nhật hoặc xóa), index phải được cập nhật theo, điều này có thể làm chậm quá trình cập nhật và tăng tải cho hệ thống.

+ Chi phí lưu trữ: Index tiêu tốn không gian lưu trữ để lưu trữ bảng từ điển và các con trỏ, đặc biệt khi có nhiều index trên cùng một bảng.

+ Cần cân nhắc thiết kế và sử dụng: Việc thiết kế và sử dụng index phải được cân nhắc cẩn thận để đảm bảo hiệu quả và tránh tình trạng index quá nhiều hoặc không cần thiết.

1. Bài Tập
2. Tạo các bảng table

**CREATE** **TABLE** Kho (

Id **INT** **PRIMARY** **KEY** **AUTO\_INCREMENT**,

MaKho **VARCHAR**(50) **NOT** **NULL**,

Ten **VARCHAR**(100) **NOT** **NULL**,

DiaDiem **VARCHAR**(100) **NOT** **NULL**,

NgayTao **DATETIME** **NOT** **NULL**,

NgaySua **DATETIME** **NOT** **NULL**

);

**CREATE** **TABLE** LoaiDanhMuc (

Id **INT** **PRIMARY** **KEY** **AUTO\_INCREMENT**,

MaLoaiDanhMuc **VARCHAR**(50) **NOT** **NULL**,

Ten **VARCHAR**(100) **NOT** **NULL**,

MoTa **TEXT**,

NgayTao **DATETIME** **NOT** **NULL**,

NgaySua **DATETIME** **NOT** **NULL**

);

**CREATE** **TABLE** SanPham (

Id **INT** **PRIMARY** **KEY** **AUTO\_INCREMENT**,

MaSanPham **VARCHAR**(50) **NOT** **NULL**,

DanhMuc **INT**,

Kho **INT**,

Ten **VARCHAR**(100) **NOT** **NULL**,

MoTaSanPham **TEXT**,

DuongDanAnh **VARCHAR**(200),

SoLuongSanPham **INT**,

SoLuongBan **INT**,

NgayTao **DATETIME** **NOT** **NULL**,

NgaySua **DATETIME** **NOT** **NULL**,

**FOREIGN** **KEY** (DanhMuc) **REFERENCES** LoaiDanhMuc(Id),

**FOREIGN** **KEY** (Kho) **REFERENCES** Kho(Id)

);

1. Thêm sửa xóa kho, sản phẩm, loại danh mục theo Id

**INSERT** **INTO** Kho (Id, MaKho, Ten, DiaDiem, NgayTao, NgaySua)

**VALUES** (1,"K001","Kho 1","Hà Nam",**NOW**(),**NOW**());

**INSERT** **INTO** Kho (Id, MaKho, Ten, DiaDiem, NgayTao, NgaySua)

**VALUES** (2,"K002","Kho 3","Hà Nam",**NOW**(),**NOW**());

**INSERT** **INTO** LoaiDanhMuc (Id, MaLoaiDanhMuc, Ten, MoTa, NgayTao, NgaySua)

**VALUES** (1,"DM001","Danh mục 1","Mô tả 1",**NOW**(),**NOW**());

**INSERT** **INTO** LoaiDanhMuc (Id, MaLoaiDanhMuc, Ten, MoTa, NgayTao, NgaySua)

**VALUES** (2,"DM002","Danh mục 2","Mô tả 2",**NOW**(),**NOW**());

**INSERT** **INTO** SanPham (Id, MaSanPham, DanhMuc, Kho, Ten, MoTaSanPham, DuongDanAnh, SoLuongSanPham, SoLuongBan, NgayTao, NgaySua)

**VALUES** (1, 'SP001', 1, 1, 'Sản phẩm 1', 'Mô tả sản phẩm 1', 'duongdananh1.jpg', 10, 5, **NOW**(), **NOW**());

**INSERT** **INTO** SanPham (Id, MaSanPham, DanhMuc, Kho, Ten, MoTaSanPham, DuongDanAnh, SoLuongSanPham, SoLuongBan, NgayTao, NgaySua)

**VALUES** (2, 'SP002', 2, 2, 'Sản phẩm 2', 'Mô tả sản phẩm 2', 'duongdananh2.jpg', 20, 10, **NOW**(), **NOW**());

**UPDATE** Kho **SET** Ten = "Kho 2", NgaySua = **NOW**() **WHERE** Id = 1;

**UPDATE** LoaiDanhMuc **SET** Ten = "Danh mục 2", NgaySua = **NOW**() **WHERE** Id = 1;

**UPDATE** SanPham **SET** Ten = "Sản phẩm 2", NgaySua = **NOW**() **WHERE** Id = 1;

**DELETE** **FROM** Kho **WHERE** Id = 1;

**DELETE** **FROM** LoaiDanhMuc **WHERE** Id = 1;

**DELETE** **FROM** SanPham **WHERE** Id = 1;

1. Lọc các sản phẩm có chứa từ ‘Điện thoại’ và thuộc loại danh mục có tên là ‘Apple’

**SELECT** \* **FROM** SanPham **WHERE** Ten **LIKE** "%Điện thoại%" **AND** DanhMuc **IN** (

**SELECT** Id **FROM** LoaiDanhMuc **WHERE** Ten = "Apple"

);

1. Đếm số lượng sản phẩm trong mỗi loại danh mục, sắp xếp theo thứ tự giảm dần

**SELECT** LoaiDanhMuc.Ten, **COUNT**(SanPham.Id) **AS** SoLuongSanPham **FROM** LoaiDanhMuc

**LEFT** **JOIN** SanPham **ON** LoaiDanhMuc.Id = SanPham.DanhMuc

**GROUP** **BY** LoaiDanhMuc.Id, LoaiDanhMuc.Ten

**ORDER** **BY** SoLuongSanPham **DESC**;

1. Xóa danh mục đồng thời xóa luôn các sản phẩm thuộc danh mục đó (Có sử dụng transaction)

**START** **TRANSACTION**;

**DELETE** **FROM** SanPham **WHERE** DanhMuc **IN** (

**SELECT** Id **FROM** LoaiDanhMuc **WHERE** Ten = "Danh mục 1"

);

**DELETE** **FROM** LoaiDanhMuc **WHERE** Ten = "Danh mục 1";

**COMMIT**;

1. Procedure lấy 10 sản phẩm có số lượng bán nhiều nhất

DELIMITER //

**CREATE** **PROCEDURE** GetTopProducts ()

**BEGIN**

**SELECT** \* **FROM** SanPham **ORDER** **BY** SoLuongBan **DESC**

**LIMIT** 10;

**END/**/

DELIMITER;

**CALL** GetTopProducts();