**EX03\_MYSQL**

**BÁO CÁO THU HOẠCH**

1. **SQL và NoSQL**

**SQL**

SQL là viết tắt của Structured Query language, là ngôn ngữ chuẩn để xử lý Cơ sở dữ liệu quan hệ. Một cơ sở dữ liệu quan hệ xác định các mối quan hệ dưới dạng các bảng. Các ví dụ về cơ sở dữ liệu quan hệ SQL: MySQL, Oracle, MS SQL Server, Sybase,…

Lập trình SQL có thể được sử dụng hiệu quả để chèn, tìm kiếm, cập nhật, xóa các bản ghi cơ sở dữ liệu.

**NoSQL**

NoSQL là một DMS không quan hệ, không yêu cầu một lược đồ cố định, tránh các phép nối, và dễ dàng mở rộng. Cơ sở dữ liệu NoSQL được sử dụng cho các kho dữ liệu phân tán với nhu cầu lưu trữ dữ liệu khổng lồ. NoSQL được sử dụng cho dữ liệu lớn và các ứng dụng web thời gian thực.

Ví dụ về NoSQL: MongoDB, CouchDB, CouchBase, Cassandra, HBase, Redis, Riak, Neo4J

**So sánh SQL và NoSQL**

|  |  |  |
| --- | --- | --- |
| **Tham số** | **SQL** | **NoSQL** |
| Định nghĩa | Cơ sở dữ liệu SQL chủ yếu được gọi là RDBMS hoặc Cơ sở dữ liệu quan hệ | Cơ sở dữ liệu NoSQL chủ yếu được gọi là cơ sở dữ liệu không liên quan hoặc phân tán |
| Design for | RDBMS truyền thống sử dụng cú pháp và truy vấn SQL để phân tích và lấy dữ liệu để có thêm thông tin chi tiết. Chúng được sử dụng cho các hệ thống OLAP. | Hệ thống cơ sở dữ liệu NoSQL bao gồm nhiều loại công nghệ cơ sở dữ liệu khác nhau. Các cơ sở dữ liệu này được phát triển để đáp ứng nhu cầu trình bày cho sự phát triển của ứng dụng hiện đại. |
| Ngôn ngữ Query | Structured query language (SQL) | Không có ngôn ngữ query |
| Type | SQL databases là cơ sở dữ liệu dựa trên bảng | NoSQL databases có thể dựa trên tài liệu, cặp khóa-giá trị, cơ sở dữ liệu biểu đồ |
| Schema | SQL databases có lược đồ được xác định trước | NoSQL databases sử dụng lược đồ động cho dữ liệu phi cấu trúc. |
| Khả năng mở rộng | SQL databases có thể mở rộng theo chiều dọc (Tập trung phát triển phần cứng và hạ tầng) | NoSQL databases có thể mở rộng theo chiều ngang (phân tán và lưu trữ trên nhiều máy |
| Ví dụ | Oracle, Postgres, and MS-SQL. | MongoDB, Redis, , Neo4j, Cassandra, Hbase. |
| Phù hợp cho | Đây là 1 lựa chọn lý tưởng cho môi trường truy vấn phức tạp | Không phù hợp với truy vấn phức tạp |
| Lưu trữ dữ liệu phân cấp | SQL databases không thích hợp cho việc lưu trữ dữ liệu phân cấp. | Phù hợp hơn cho kho lưu trữ dữ liệu phân cấp vì nó hỗ trợ phương thức cặp khóa-giá trị. |
| Variations | Một loại có biến thể nhỏ | Nhiều loại khác nhau bao gồm các kho khóa-giá trị, cơ sở dữ liệu tài liệu và cơ sở dữ liệu đồ thị. |
| Năm phát triển | Nó được phát triển vào những năm 1970 để giải quyết các vấn đề với lưu trữ tệp phẳng | Được phát triển vào cuối những năm 2000 để khắc phục các vấn đề và hạn chế của SQL databases. |
| Open-source | Một sự kết hợp của mã nguồn mở như Postgres & MySQL, và thương mại như Oracle Database. | Open-source |
| Tính nhất quán | Nó phải được cấu hình cho sự nhất quán chặt chẽ. | Nó phụ thuộc vào DBMS như một số cung cấp tính nhất quán mạnh mẽ như MongoDB, trong khi những người khác cung cấp chỉ cung cấp sự nhất quán cuối cùng, như Cassandra. |
| Được sử dụng tốt nhất cho | RDBMS database là tùy chọn thích hợp để giải quyết các vấn đề về ACID. | NoSQL được sử dụng tốt nhất để giải quyết các vấn đề về tính khả dụng của dữ liệu |
| Tầm quan trọng | Nó nên được sử dụng khi hiệu lực dữ liệu là siêu quan trọng | Sử dụng khi nó quan trọng hơn để có dữ liệu nhanh hơn dữ liệu chính xác |
| Lựa chọn tốt nhất | Khi bạn cần hỗ trợ truy vấn động | Sử dụng khi bạn cần mở rộng quy mô dựa trên yêu cầu thay đổi |
| Hardware | Specialized DB hardware (Oracle Exadata, etc.) | Commodity hardware |
| Network | Highly available network (Infiniband, Fabric Path, etc.) | Commodity network (Ethernet, etc.) |
| Loại lưu trữ | Highly Available Storage (SAN, RAID, etc.) | Commodity drives storage (standard HDDs, JBOD) |
| Tính năng tốt nhất | Hỗ trợ đa nền tảng, Bảo mật và miễn phí | Dễ sử dụng, hiệu suất cao và công cụ linh hoạt. |
| Mô hình ACID và BASE | ACID (Atomicity, Consistency, Isolation và Durability) là một chuẩn cho RDBMS | Cơ bản (Về cơ bản có sẵn, trạng thái mềm, phù hợp cuối cùng) là một mô hình của nhiều hệ thống NoSQL |
| Performance | SQL hoạt động tốt và nhanh thì việc desgin tốt là cực kì quan trọng và ngược lại. | Nhanh hơn SQL NoSQL thì denormalized cho phép bạn lấy được tất cả thông tin về một item cụ thể với các codition mà không cần JOIN liên quan hoặc truy vấn SQL phức tạp. |
| Kết luận | Dự án đã có yêu cầu dữ liệu rõ ràng xác định quan hệ logic có thể được xác định trước. | Phù hợp với những dự án yêu cầu dữ liệu không liên quan, khó xác định, đơn giản mềm dẻo khi đang phát triển |

**Khi nào sử dụng SQL?**

* SQL là ngôn ngữ đơn giản nhất được sử dụng để giao tiếp với RDBMS
* Phân tích các phiên liên quan đến hành vi và tùy chỉnh
* Tạo trang tổng quan tùy chỉnh
* Nó cho phép bạn lưu trữ và lấy dữ liệu từ cơ sở dữ liệu một cách nhanh chóng
* Được ưu tiên khi bạn muốn sử dụng các phép nối và thực hiện các truy vấn phức tạp

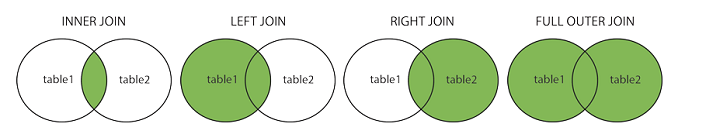
**Khi nào sử dụng NoSQL?**

* Khi không cần hỗ trợ ACID
* Khi mô hình RDBMS truyền thống không đủ
* Dữ liệu cần lược đồ linh hoạt
* Các ràng buộc và logic xác thực không bắt buộc phải được thực hiện trong cơ sở dữ liệu
* Muốn lưu dữ liệu một cách phân tán

1. **Phép toán join**

Trong Cơ sở dữ liệu quan hệ SQL có 4 phép toán JOIN cơ bản

* INNER JOIN: Trả về các bản ghi có những giá trị phù hợp trong cả hai bảng.
* LEFT (OUTER) JOIN: Trả về tất cả bản ghi từ bảng bên trái và bản ghi trùng với bảng bên phải.
* RIGHT (OUTER) JOIN: Trả về tất cả bản ghi từ bảng bên phải và bản ghi trùng với bản bên trái.
* FULL (OUTER) JOIN: Trả về tất cả bản ghi khi có một kết quả phù hợp trong bảng bên trái hoặc bên phải.

****

1. **Function**

Function là một đối tượng trong cơ sở dữ liệu (CSDL) sử dụng trong các câu lệnh SQL, được biên dịch sẵn và lưu trong CSDL 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

**Tạo function**

CREATE FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS <kiểu dữ liệu>

[WITH ENCRYPTION]

[AS]

BEGIN

[Thân của hàm]

RETURN <Biểu thức giá trị đơn>

END

**Thay đổi function**

ALTER FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS <kiểu dữ liệu> | TABLE

[WITH ENCRYPTION]

[AS]

BEGIN

[Thân của hàm]

RETURN <Biểu thức giá trị đơn> | Câu lệnh SQL

END

**Xóa function**

DROP FUNCTION [schema\_name.] <Tên function>

**Xem function**

EXEC sp\_helptext 'FunctionName'

1. **Procedure**

Procedure được định nghĩa như một tập các khai báo sql được lưu trữ ngay trong cơ sở dữ liệu (database) và sau đó, được triệu gọi bởi một program, một trigger hay thậm chí là một stored procedure khác.

**Tạo Procedure**

DELIMITER $$

CREATE PROCEDURE procedureName()

BEGIN

SELECT \* FROM products;

END; $$

DELIMITER

Giải thích:

* DELIMITER $$ dùng để phân cách bộ nhớ lưu trữ thủ tục Cache và mở ra một ô lưu trữ mới.
* CREATE PROCEDURE procedureName() dùng để khai báo tạo một Procedure mới, trong đó procedureName chính là tên thủ tục còn hai từ đầu là từ khóa.
* \*\*BEGIN và END; $$ \*\* dùng để khai báo bắt đầu của Procedure và kết thúc Procedure
* \*\*DELIMITER ; \*\* đóng lại ô lưu trữ

**Gọi Procedure**

CALL procedureName();

**Xóa Procedure**

DROP PROCEDURE IF EXISTS `procedureName`$$

**Các tham số trong Procedure**

Trong MYSQL thì sẽ tồn tại ba loại Tham số đó là tham số IN, tham số OUT và tham số INOUT

* IN: Đây là chế độ mặc định (nghĩa là nếu bạn không định nghĩa loại nào thì nó sẽ hiểu là IN).
* OUT: Chế độ này nếu như trong Procedure có tác động thay đổi thì nó sẽ thay đổi theo. Nhưng có điều đặc biệt là dù trước khi truyền vào mà bạn gán giá trị cho biến đó thì vẫn sẽ không nhận được vì mặc định nó luôn hiểu giá trị truyền vào là NULL.
* INOUT: Đây là sự kết hợp giữa IN và OUT. Nghĩa là có thể gán giá trị trước và có thể bị thay đổi nếu trong Procedure có tác động.

1. **Transaction**

Transaction trong SQL 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).

Tính chất của Transaction:

* Tính nguyên tử(Atomicity): Nếu transaction có nhiều phép tính thì tất cả phải được thực hiện. Nếu bất kỳ phép tính nào trong nhóm thất bại thì nó nên được rollback (trở lại trạng thái trước khi transaction bắt đầu).
* Tính nhất quán(Consistency): Chuỗi các phép tính phải nhất quán.
* Cô lập(Isolation): Những phép tính được thực hiện phải được phân lập từ những phép tính khác trên cùng một máy chủ hoặc trên cùng một cơ sở dữ liệu.
* Độ bền(Durability): Những phép tính được thực hiện trên cơ sở dữ liệu phải được lưu lại và lưu trữ trong cơ sở dữ liệu vĩnh viễn (commit data).

START TRANSACTION

[*transaction\_characteristic* [, *transaction\_characteristic*] ...]

*transaction\_characteristic*: {

WITH CONSISTENT SNAPSHOT

| READ WRITE

| READ ONLY

}

BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET autocommit = {0 | 1}

Giải thích:

* START TRANSACTION hoặc BEGIN: bắt đầu một giao dịch mới.
* COMMIT: cam kết giao dịch hiện tại, thực hiện các thay đổi của nó là vĩnh viễn.
* ROLLBACK: quay trở lại giao dịch hiện tại, hủy các thay đổi của nó.
* SET autocommit: tắt hoặc bật chế độ tự động gửi mặc định cho phiên hiện tại.

Ví dụ:

START TRANSACTION;

SELECT @A:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=@A WHERE type=1;

COMMIT;

1. **Index**

Index là một trong những yếu tố quan trọng nhất góp phần vào việc nâng cao hiệu suất của cơ sở dữ liệu. Index trong SQL tăng tốc độ của quá trình truy vấn dữ liệu bằng cách cung cấp phương pháp truy xuất nhanh chóng tới các dòng trong các bảng.

Index trong SQL Server được tạo ra trên các cột trong bảng hoặc View. Chúng cung cấp một phương pháp giúp bạn nhanh chóng tìm kiếm dữ liệu dựa trên các giá trị trong các cột. Ví dụ, nếu bạn tạo ra một Index trên cột khóa chính và sau đó tìm kiếm một dòng dữ liệu dựa trên một trong các giá trị của cột này, đầu tiên SQL Server sẽ tìm giá trị này trong Index, sau đó nó sử dụng Index để nhanh chóng xác định vị trí của dòng dữ liệu bạn cần tìm. Nếu không có Index, SQL Server sẽ thực hiện động tác quét qua toàn bộ bảng (table scan) để xác định vị trí dòng cần tìm.

Index trong SQL Server có thể tạo trên hầu hết các cột trong bảng hoặc View. Tuy nhiên chúng ta phải xem xét vì nó tốn bộ nhớ.

Index trong SQL được tạo thành từ một tập hợp các page (các Index Node) và chúng được tổ chức trong một cấu trúc có tên gọi là B-tree.

Cú pháp tạo Index

CREATE INDEX ten\_index ON ten\_bang;

**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:

Khi không cần sử dụng INDEX nữa có thể DROP theo cú pháp sau:

DROP INDEX ten\_index;