**BÁO CÁO**

**Tìm hiểu về cơ sở dữ liệu.**

**1.** **Tìm hiểu về SQL**

SQL là viết tắt của Structured Query Language, nghĩa là ngôn ngữ truy vấn dữ liệu. Có thể coi SQL là ngôn ngữ chung mà bất cứ hệ thống cơ sở dữ liệu quan hệ (RDBMS) nào cũng phải đáp ứng, điển hình như: Oracle Database, SQL Server, MySQL…

Khi doanh nghiệp cần một hệ thống để quản trị nhân viên hoặc khách hàng, họ phải thiết kế ra một cơ sở dữ liệu để lưu trữ thông tin. Nếu chỉ lưu trữ ở dạng giấy hoặc excel thì sẽ chứa nhiều rủi ro như bị mất, sửa, xóa…

SQL giúp quản lý hiệu quả và truy vấn thông tin nhanh hơn, giúp bảo trì thông tin dễ dàng hơn.

Ví dụ:  trước đây, bệnh viện thường lưu trữ thông tin bệnh án của bệnh nhân bằng cách viết tay trên hồ sơ giấy. Sau đó, cất giữ hồ sơ trong kho.

Khi cần tìm kiếm hoặc thêm/xóa/sửa thông tin nào đó, em phải mất rất nhiều thời gian để lục lại hồ sơ. Đó là chưa kể đến một số trường hợp sau khi thêm hoặc sửa thông tin, hồ sơ sẽ không còn hợp lệ.

Trong khi, nếu lưu trữ thông tin vào một hệ thống cơ sở dữ liệu, em chỉ cần gõ một câu lệnh SQL ngắn là đã có thể trích xuất được thông tin em cần. Việc thêm/xóa/sửa cũng được thực hiện một cách dễ dàng, nhanh chóng.

**2.** **Tìm hiểu MySQL**

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 như các công ty như Twitter, Facebook, Google thu thập hàng terabyte dữ liệu người dùng mỗi ngày.

Cơ sở dữ liệu NoSQL là viết tắt của "Không chỉ SQL" hoặc "Không phải SQL".

RDBMS truyền thống sử dụng cú pháp SQL để lưu trữ và truy xuất dữ liệu để có thêm thông tin chi tiết. Thay vào đó, một hệ thống cơ sở dữ liệu NoSQL bao gồm một loạt các công nghệ cơ sở dữ liệu có thể lưu trữ dữ liệu có cấu trúc, bán cấu trúc, không có cấu trúc và đa hình.

**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 | NoSQL databases có thể mở rộng theo chiều ngang |
| 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, nhất quán, cách ly và độ bền) 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 |

**3. Tìm hiểu về JOIN trong MySQL**

- Lệnh Join (tham gia):

Lệnh join giúp lấy dữ liệu từ hai hoặc nhiều bảng cơ sở dữ liệu. Các bảng có liên kết với nhau bằng khoá chính hoặc khoá ngoại.

+) Tham gia chéo (Cross join):

Tham gia giao chéo là một hình thức tham gia đơn giản nhất nó khớp với từng hàng từ một bảng cơ sở dữ liệu với tất cả các hàng khác.

Nói cách khác, nó cung cấp cho chúng ta sự kết hợp của từng hàng của bảng đầu tiên với tất cả các bản ghi trong bảng thứ hai.

**-) Inner Join**

Inner Join được sử dụng để kết hợp các hàng từ cả hai bảng thỏa mãn điều kiện đã cho.

Outer Join cho tất cả kết quả các bản ghi khớp từ cả hai bảng.

Nó có thể phát hiện các bản ghi không có kết quả khớp trong bảng đã tham gia. Nó cho giá trị **NULL** các bản ghi của bảng đã tham gia nếu không tìm thấy kết quả khớp.

**-) Left Join**

Left Join cho kết quả tất cả các hàng từ bảng bên trái ngay cả khi không tìm thấy hàng phù hợp trong bảng bên phải. **Trường hợp không tìm thấy kết quả phù hợp trong bảng bên phải, kết quả sẽ là Null.**

**-) Right Join**

Right Join trái ngược với Left Join. Right Join cho tất cả kết quả các cột từ bảng bên phải ngay cả khi không tìm thấy hàng phù hợp trong bảng bên trái. Trường hợp không tìm thấy kết quả phù hợp trong bảng bên trái, kết quả là Null

Bạn có thể hiểu tại sao chúng ta sử dụng “Join” khi chúng ta thực hiện cùng một tác vụ chạy truy vấn rồi chứ. Đặc biệt nếu bạn có một số kinh nghiệm về lập trình cơ sở dữ liệu, bạn có thể chạy từng truy vấn một, sử dụng kết quả của từng truy vấn liên tiếp. Nhưng bằng cách sử dụng “Join”, bạn có thể hoàn thành công việc bằng cách chỉ sử dụng một truy vấn với bất kỳ tham số tìm kiếm nào. Mặt khác, **MySQL có thể đạt được hiệu năng tốt hơn.** Chỉ cần sử dụng truy vấn duy nhất thay vì chạy nhiều truy vấn điều đó giúp chúng ta sẽ giảm được chi phí . Thay vào đó, sử dụng nhiều truy vấn sẽ dẫn đến việc chuyển nhiều dữ liệu hơn giữa MySQL và các ứng dụng (phần mềm). Hơn nữa, nó đòi hỏi nhiều thao tác dữ liệu hơn trong ứng dụng.

**Chúng ta có thể đạt được hiệu suất ứng dụng và MySQL tốt hơn bằng cách sử dụng Lệnh Join.**

### ****Tóm lược****

* Lệnh Join cho phép chúng tôi kết hợp dữ liệu từ nhiều hơn một bảng vào một tập kết quả.
* “Join” có hiệu suất tốt hơn so với các truy vấn con
* Inner Join chỉ trả về các hàng đáp ứng các tiêu chí nhất định.
* Outer Join có thể trả về các hàng nơi không tìm thấy kết quả khớp. Các hàng ko có được trả về với kết quả là NULL.
* Các kiểu của lệnh Join bao gồm  Inner, Left, Outer, Right Outer, Cross JOin
* Mệnh đề thường được sử dụng trong các hoạt động của lệnh Join là “On” và mệnh đề “Using” yêu cầu các cột khớp phải cùng tên.
* “Join” cũng có thể được sử dụng trong các mệnh đề khác, chẳng hạn như**GROUP BY, WHERE, SUB QUERIES, AGGREGATE FUNCTIONS**

**4) Tìm hiểu về VIEW**

* View là các truy vấn SELECT được lưu lại như là một table và sau đó ta có thể truy vấn dữ liệu từ view như thực hiện với table.
* Một số lưu ý đối với câu lệnh SELECT trong view:
  + Không sử dụng biến local, user hay session
  + Những lệnh dịch sẵn cũng không thể sử dụng
  + Bảng tạm hay các view khác cũng không được sử dụng làm nguồn dữ liệu cho câu truy vấn này.
  + Không thể tạo trigger cho view.

##### Cú pháp

CREATE VIEW view\_name AS

SELECT statement

Ví dụ:

CREATE VIEW address\_people AS

SELECT

a.name AS name,

a.age AS age,

c.name AS province\_name

FROM people a

JOIN district b ON a.district\_id = b.id

JOIN province c ON b.id\_province = c.id

##### Ưu nhược điểm

* Ưu điểm
  + An ninh – Cho phép người quản trị CSDL (DBA) cung cấp cho người sử dụng chỉ những dữ liệu cần thiết
  + Đơn giản hóa – Các view cũng có thể sử dụng để ẩn và sử dụng lại các truy vấn phức tạp => rất tiện với những câu truy vấn còng kềnh dài hàng trang A4
* Nhược điểm:
  + Tốn thêm tài nguyên trên Server (memory, process)

**5) Tìm hiểu về Function/Procedure**

* Là đoạn chương trình kịch bản (programming scripts) với các câu lệnh SQL nhúng (embedded SQL) được lưu dưới dạng đã được biên dịch và thi hành thực tiếp bởi MySQL server.
* SP cho phép lưu trữ các logic ứng dụng trên CSDL.

##### Cú pháp

* Tạo hàm/thủ tục

CREATE FUNCTION name ([parameterlist]) RETURNS datatype [options] sqlcode

CREATE PROCEDURE name ([parameterlist]) [options] sqlcode

Ví dụ:

DELIMITER $$

CREATE PROCEDURE count\_people\_with\_age(IN age\_in INT, OUT quantity INT)

BEGIN

SELECT \*

FROM people

WHERE age >= age\_in;

SELECT FOUND\_ROWS() INTO quantity;

END

* DELIMITER $$: Giải phóng để cho phép sử dụng ‘;’ trong thủ tục
* Gán giá trị cho biến
  + Sử dụng SET hoặc SELECT INTO.
* Gọi thủ tục:

Call count\_people\_with\_age (18,@people\_count);

Select @ people \_count;

* Xóa Hàm/Thủ tục:

DROP FUNCTION [IF EXISTS] name

DROP PROCEDURE [IF EXISTS] name

* Hiện thị thông tin

SHOW FUNCTION STATUS;

SHOW PROCEDURE STATUS;

SHOW FUNCTION STATUS LIKE 'repeat%';

SHOW PROCEDURE STATUS LIKE 'film%';

SHOW CREATE PROCEDURE 'name';

##### **Một số cấu trúc điều khiển dùng trong Hàm, thủ tục:**

* Mệnh đề IF THEN

IF condition THEN

commands;

[ELSE IF condition THEN

commands;]

[ELSE

commands;]

END IF;

* Mệnh đề REPEAT UNTIL

[loopname:]

REPEAT commands;

UNTIL condition

END REPEAT

[loopname];

* Mệnh đề CASE

CASE expression

WHEN value1 THEN commands;

[WHEN value2 THEN commands;]

[ELSE commands;]

END CASE;

* Mệnh đề WHILE

[loopname:]

WHILE condition

DO commands;

END WHILE

[loopname];

##### **Con trỏ (cursor) trong hàm/thủ tục**

* MySQL hỗ trợ con trỏ trong hàm/thủ tục, trigger
* Con trỏ được sử dụng để duyệt các dòng kết quả trả lại bởi truy vấn, và xử lý từng dòng riêng lẻ
* Cú pháp Con trỏ:

DECLARE cursor\_name CURSOR FOR SELECT\_statement;

OPEN cursor\_name;

* Lấy ra từng dòng và chuyển tới dòng kết quả tiếp theo sử dụng lệnh FETCH

FETCH cursor\_name INTO variable list;

CLOSE cursor\_name;

##### **Xử lý lỗi thông qua Handlers**

* Luôn có khả năng hàm/thủ tục gặp lỗi trong khi thi hành các lệnh SQL. MySQL cung cấp kỹ thuật xử lý lỗi thông qua handler
* Một handler cần định nghĩa sau khai báo các biến, con trỏ và điều kiện, nhưng trước các lệnh SQL
* Cú pháp Handler:

DECLARE type HANDLER FORcondition1, condition2, condition3, ... statement;

* type: CONTINUE hoặc EXIT
* condition(s): Các điều kiện mà handler sẽ được gọi : Not found, SqlException,..
* statement: Câu lệnh sẽ thi hành khi có điều kiện xảy ra

##### **Ưu nhược điểm**

* Ưu điểm
  + Giảm dư thừa mã chương trình: Các đoạn mã tương tự trong các ứng dụng như thêm, cập nhật có thể lưu ở phía CSDL
  + Bảo trì: Nếu có sự thay đổi trong CSDL, mã lệnh cần thay đổi có thể xác định trong các SP
  + An ninh CSDL tốt hơn: Trong các ứng dụng an ninh cao, với hàm/thủ tục có thể kiểm soát truy cập dữ liệu và đưa ra các qui định an ninh tập trung
* Nhược điểm
  + Thiếu tính khả chuyển (Lack of Portability)  
    SP khó chuyển từ một DBMS sang một DBMS khác. Yêu cầu lập trình và kiểm thử lại đáng kể
  + Tải DB Server  
    Sử dụng hàm/thủ tục nhiều có thể gây quá tải MySQL server
  + Hạn chế ngôn ngữ lập trình  
    Lập trình SP không phong phú như các nền tảng phát triển khác như Java hay PHP  
    Lưu ý: Các phiên bản tương lai của MySQL có thể cung cấp các giao diện cho phép tạo các hàm/thủ tục sử dụng các ngôn ngữ bên ngoài, ví dụ Java

**6) Trigger**

* Triggers là quá trình tự động thi hành các lệnhSQL hoặc hàm/thủ tục sau hoặc trước các lệnh INSERT, UPDATE, hoặc DELETE.
* Các ứng dụng có thể bao gồm: lưu lại thay đổi hoặc cập nhật dữ liệu các bảng khác.
* Trigger chạy sau mỗi câu lệnh cập nhật bảng do đó có thể thêm tải với CSDL

##### Cú pháp

* Tạo trigger:

CREATE TRIGGER name BEFORE | AFTER INSERT | UPDATE | DELETE ON tablename

FOR EACH ROW sql-code

* Xóa trigger:

DROP TRIGGER tablename.triggername

* Sửa, xem thông tin trigger:

ALTER TRIGGER, SHOW CREATE TRIGGER, hoặc SHOW TRIGGER STATUS

* Để hiển thị các trigger gắn với 1 bảng dữ liệu:

SELECT \* FROM Information\_Schema.Trigger

WHERE Trigger\_schema = 'database\_name' AND Event\_object\_table = 'table\_name';

* Cú pháp lệnh bên trong tương tự hàm/thủ tục
* Trong trigger, mã lệnh có thể truy cập các cột của bản ghi hiện tại
  + OLD.columnname trả lại nội dung của bản ghi trước khi bị thay đổi hoặc xóa (UPDATE, DELETE)
  + NEW.columnname trả lại nội dung của bản ghi mới hoặc bản ghi thay thế (INSERT, UPDATE)

**7) Transaction**

Có thể hiểu Transaction là một tiến trình xử lý có xác định điểm đầu và điểm cuối, được chia nhỏ thành các operation (phép thực thi) , tiến trình được thực thi một cách tuần tự và độc lập các operation đó theo nguyên tắc hoặc tất cả đều thành công hoặc một operation thất bại thì toàn bộ tiến trình thất bại. Nếu việc thực thi một operation nào đó bị fail đồng nghĩa với việc dữ liệu phải rollback về trạng thái ban đầu.

Có thể lấy ví dụ về 1 Transaction đơn giản nhất là tiến trình cài đặt phần mềm hoặc gỡ bỏ phần mềm. Việc cài đặt được chia thành các bước, thực hiện tuần tự từ đầu đến cuối, nếu toàn bộ các bước thực thi thành công đồng nghĩa với việc tiến trình cài đặt hoặc gỡ bỏ phần mềm thành công và ngược lại, một phép thất bại thì tiến trình phải rollback lại tức sẽ không có bất kỳ thay đổi nào trên máy tính.

**Property của Transaction trong MySQL**

Transaction có 4 property chuẩn và thường được biết đến với tên lược danh là ACID:

* **Atomicity:** bảo đảm rằng tất cả hoạt động bên trong đơn vị công việc được hoàn thành một cách thành công; nếu không, transaction bị ngừng ở điểm thất bại, và các hoạt động trước được trao trả về trạng thái trước đó.
* **Consistency:** bảo đảm rằng Database thay đổi một cách chính xác trạng thái theo một transaction đã được ký thác thành công.
* **Isolation:** cho các transaction khả năng hoạt động một cách độc lập và không liên quan đến nhau.
* **Durability:** bảo đảm rằng kết quả hoặc tác động của một transaction, mà đã được ký thác, vẫn còn tồn tại trong trường hợp hệ thống thất bại.

Trong MySQL, các Transaction bắt đầu với lệnh BEGIN WORK và kết thúc với hoặc một lệnh COMMIT hoặc lệnh ROLLBACK. Các lệnh SQL giữa các lệnh bắt đầu và kết thúc cấu tạo nên Transaction.

**Lệnh COMMIT và ROLLBACK trong MySQL**

Hai từ khóa **Commit** và **Rollback** được sử dụng chủ yêu cho Transaction trong MySQL.

* Khi một Transaction được hoàn thành, lệnh COMMIT nên được thông báo để mà tất cả thay đổi tới các bảng liên quan sẽ có hiệu quả.
* Nếu một lỗi xuất hiện, một lệnh ROLLBACK nên được thông báo để trả bảng đã tham chiếu trong Transaction về trạng thái trước đó.

Bạn có thể điều khiển cách vận hành của một Transaction bằng việc thiết lập biến session gọi là **AUTOCOMMIT**. Nếu AUTOCOMMIT được thiết lập là 1 (mặc định), thì mỗi lệnh SQL (bên trong một Transaction hoặc không) được xem như là một Transaction đã kết thúc và được ký thác theo mặc định khi nó kết thúc. Khi AUTOCOMMIT được thiết lập là 0, thì bằng việc thông báo lệnh SET AUTOCOMMIT=0, các chuỗi lệnh theo sau hoạt động như một Transaction và không có hoạt động nào được ký thác tới khi một lệnh COMMIT tường minh được thông báo.

**Các loại bảng bảo vệ Transaction trong MySQL**

Bạn không thể sử dụng các Transaction một cách trực tiếp, bởi vì chúng không an toàn và không được bảo vệ. Nếu bạn muốn sử dụng các Transaction trong lập trình MySQL, thì bạn cần tạo các bảng theo một cách đặc biệt. Có nhiều loại bảng hỗ trợ cho các Transaction nhưng loại phổ biến nhất là InnoDB.

Sự hỗ trợ cho các bảng InnoDB yêu cầu một tham số biên dịch cụ thể khi biên dịch MySQL từ nguồn. Nếu phiên bản MySQL của bạn không có hỗ trợ cho InnoDB, bạn yêu cầu nhà cung cấp dịch vụ Internet để xây dựng một phiên bản MySQL với sự hỗ trợ cho loại bảng InnoDB hoặc tải và cài đặt MySQL-Max Binary Distribution cho Windows hoặc Linux/UNIX và làm việc với loại bảng này trong môi trường phát triển.

### 8) Khái niệm về chỉ mục (index) trong mysql

* Chỉ mục (Index) là bảng tra cứu đặc biệt mà Database Search Engine có thể sử dụng để tăng nhanh thời gian và hiệu suất thu thập dữ liệu. Hiểu đơn giản, một chỉ mục là một con trỏ tới dữ liệu trong một bảng. Một chỉ mục trong một Database là tương tự như một chỉ mục trong Mục lục của cuốn sách.
* Ví dụ, nếu bạn muốn tham chiếu tất cả các trang trong một cuốn sách về một chủ đề nào đó, đầu tiên bạn nghĩ ngay đến mục lục của nó, mà liệt kê tất cả các chương, chủ đề theo thứ tự và sau đó được tham chiếu tới một hoặc nhiều trang cụ thể. Khi đã có mục lục của cuốn sách, việc tìm kiếm trang sách đó sẽ nhanh chóng và đỡ tốn công hơn rất nhiều so với việc bạn phải lật từng trang.