Lê Đức Quý

1. Định nghĩa
2. **SQL**: SQL (Structured Query Language SQL là ngôn ngữ bạn sử dụng để tương tác với cơ sở dữ liệu. Các câu lệnh SQL được sử dụng để thực hiện các tác vụ như cập nhật dữ liệu trên cơ sở dữ liệu hoặc truy xuất dữ liệu từ cơ sở dữ liệu. SQL có thể được sử dụng để chèn, tìm kiếm, cập nhật và xóa các bản ghi cơ sở dữ liệu; thực hiện nhiều hoạt động khác bao gồm tối ưu hóa và bảo trì cơ sở dữ liệu.

**Đặc điểm SQL:**

**Ngôn ngữ định nghĩa dữ liệu**

Ngôn ngữ định nghĩa dữ liệu (DDL) là các lệnh SQL thiết kế cấu trúc cơ sở dữ liệu. Các kỹ sư cơ sở dữ liệu sử dụng DDL để tạo và điều chỉnh các đối tượng cơ sở dữ liệu dựa trên các yêu cầu nghiệp vụ. Ví dụ: kỹ sư cơ sở dữ liệu sử dụng lệnh CREATE để tạo các đối tượng cơ sở dữ liệu như bảng, chế độ xem và chỉ mục.

**Ngôn ngữ truy vấn dữ liệu**

Ngôn ngữ truy vấn dữ liệu (DQL) bao gồm các lệnh hướng dẫn để truy xuất dữ liệu được lưu trữ trong cơ sở dữ liệu quan hệ. Các ứng dụng phần mềm sử dụng lệnh SELECT để lọc và trả về kết quả cụ thể từ một bảng SQL.

**Ngôn ngữ thao tác dữ liệu**

Các câu lệnh ngôn ngữ thao tác dữ liệu (DML) viết thông tin mới hoặc điều chỉnh các bản ghi hiện có trong cơ sở dữ liệu quan hệ. Ví dụ: một ứng dụng sử dụng lệnh INSERT để lưu trữ một bản ghi mới trong cơ sở dữ liệu.

**Ngôn ngữ kiểm soát dữ liệu**

Quản trị viên cơ sở dữ liệu sử dụng ngôn ngữ kiểm soát dữ liệu (DCL) để quản lý hoặc cấp quyền truy cập cơ sở dữ liệu cho người dùng khác. Ví dụ: họ có thể sử dụng lệnh GRANT để cho phép các ứng dụng nhất định thao tác với một hoặc nhiều bảng.

**Ngôn ngữ kiểm soát giao dịch**

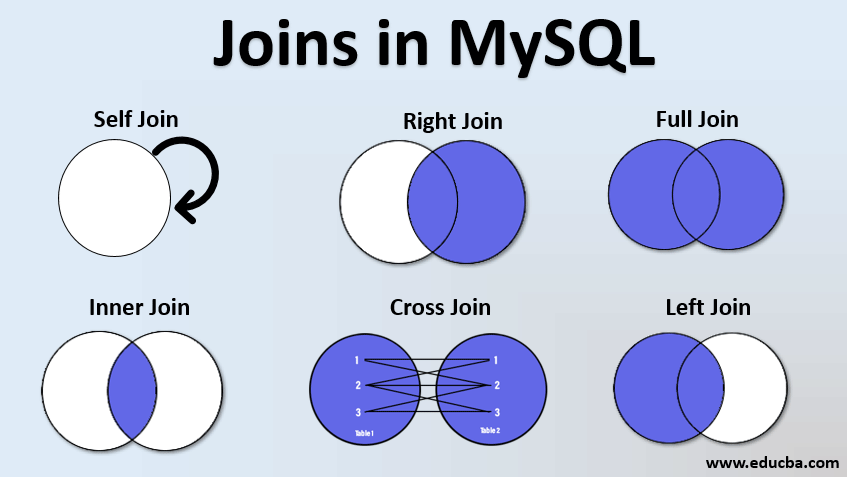
Công cụ quan hệ sử dụng ngôn ngữ kiểm soát giao dịch (TCL) để tự động thực hiện các thay đổi đối với cơ sở dữ liệu. Ví dụ: cơ sở dữ liệu sử dụng lệnh ROLLBACK để hoàn tác một giao dịch bị lỗi.

1. **NoSQL** ám chỉ đến những cơ sở dữ liệu không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mềm.

**Lý do sử dụng:**

* Tính linh hoạt cao: sơ đồ linh hoạt giúp quá trình phát triển nhanh chóng, cũng đảm bảo ở khả năng lặp lại cao hơn rất nhiều. Chính mô hình dữ liệu có khả năng linh hoạt cao sẽ biến cơ sở dữ liệu NoSQL trở thành sự lựa chọn hoàn hảo cho những dữ liệu ở dạng không được tổ chức thành cấu trúc, hay có cấu trúc chưa thực sự hoàn chỉnh.
* Có khả năng thay đổi ở quy mô: đặc trưng của NoSQL chính là việc được thiết kế với mục đích giúp tăng quy mô bằng cách dùng những cụm phần cứng đã được phân phối thay vì tăng ở quy mô thông qua việc bổ sung máy chủ mạnh mẽ, tốn kém nhiều chi phí.
* Sở hữu hiệu năng cao: việc được tối ưu hóa theo các mô hình dữ liệu cụ thể, cũng như các mẫu truy cập giúp NoSQL có thể tăng hiệu năng cao hơn so với việc chúng ta cố gắng đạt được mức độ chức năng tương tự, nhờ vào cơ sở dữ liệu quan hệ.
* Thực sự thiết thực: với NoSQL cung cấp những API và các kiểu dữ liệu thực sự thiết thực, được tiến hành xây dựng riêng cho từng mô hình dữ liệu tương ứng. Nó đem lại tính ứng dụng cao hơn rất nhiều trong quá trình sử dụng.

1. **JOIN** được sử dụng để kết hợp các hàng từ hai hoặc nhiều bảng dựa trên một cột có liên quan giữa chúng.



* **INNER JOIN:** Trả về các record có giá trị khớp trong cả hai bảng
* **LEFT JOIN:** Trả về tất cả các record từ bảng bên trái (table1) và mọi record phù hợp từ bảng bên phải (table2). Nếu không khớp, giá trị NULL sẽ được hiển thị cho các cột của bảng bên phải (table2).
* **RIGHT JOIN:** Trả về tất cả các record từ bảng bên phải (table2) và mọi record phù hợp từ bảng bên trái (table1). Nếu không khớp, giá trị NULL sẽ được hiển thị cho các cột của bảng bên trái (table1).
* **FULL JOIN:** Trả về các record có giá trị khớp trong cả hai bảng, cũng như bất kỳ record nào chưa khớp từ một trong hai bảng <==> LEFT JOIN union RIGHT JOIN
* **CROSS JOIN:** Trả về tích Descartes của hai bảng (nghĩa là mọi tổ hợp các record từ cả hai bảng)
* **SELF JOIN:** Trả về các record có giá trị khớp với điều kiện khi một bảng được join với chính nó

1. **Index** (hay chỉ mục) là một cấu trúc dữ liệu để tăng hiệu suất truy vấn của cơ sở dữ liệu. **Index** cho phép cơ sở dữ liệu thực hiện một số câu truy vấn có điều kiện nhanh hơn so với thông thường. Nhưng index cũng được lưu trên bộ nhớ và tiêu tốn không gian bộ nhớ và thời gian để tạo, cập nhật index nên khi sử dụng index cần phải suy xét kĩ.
2. **Stored Procedures:** là một tập các câu lệnh SQL nhằm thực thi tác vụ nào đó. Stored Procedure có thể nhận tham số đầu vào nhưng lại không bắt buộc phải trả về kết quả như Function (Hàm), tuy nhiên có thể trả về một lúc nhiều tham số đầu ra - điều mà Function không làm được.

**Cú pháp:**

**Tạo mới Stored Procedures – CREATE**

|  |
| --- |
| CREATE PROCEDURE procedure\_name (  [IN | OUT | INOUT] parameter\_name datatype[(length)]  )  BEGIN  statements;  END $$  -- Ký tự phân cách $$ có thể định nghĩa lại bằng khai báo DELIMITER <characters> ví dụ: DELIMITER $$ |

**Cú pháp gọi Stored Procedures – CALL**

|  |
| --- |
| CALL <procedure name> ([<List value input> , <List output: @out\_name>]); |

**Cú pháp xóa Strored Procedure:**

|  |
| --- |
| DROP PROCEDURE [IF EXISTS] procedure\_name; |

**Cú pháp xem danh sách các Strored Procedure:**

|  |
| --- |
| SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search\_condition]; |

1. **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.

**-Các kiểu Transaction:**

**Flat Transaction – Transaction ngang hàng:**

Việc chia các operation là ngang hàng nhau. Thực thi các operation là tuần tự từ trái sang phải hoặc từ trên xuống dưới.

**Nested Transaction – Transaction lồng nhau**

Việc thực thi các operation dựa theo nguyên tắc từ trong ra ngoài. Như vậy khi nhìn vào hình vẽ chúng ta thấy các operation ở dạng này có vẻ phụ thuộc vào nhau nhưng khi thực thi thì là độc lập theo nguyên tắc operation trong thực thi xong thì mới đến operation ngoài.

**Các thuộc tính của Transaction**

**Atomicity – tính đơn vị:**

Một transaction xác định ranh giới của nó rất rõ ràng, tức xác định điểm bắt đầu và kết thúc của tiến trình. Như vậy có thể coi nó như một đơn vị thực thi và đơn vị thực thi này thực hiện theo nguyên tắc “all or nothing”. Nghĩa là nếu một thành phần nào đó trong transaction thực thi hỏng (fail) thì đồng nghĩa với việc không có gì xảy ra tức không có gì thay đổi về mặt dữ liệu.

**Consistency – nhất quán:**

Dữ liệu nhất quán với transaction ở thời điểm bắt đầu và kết thúc. Nhất quán ở transaction là strong consistency. Để tìm hiểu kỹ hơn về tính nhất quán, xin đọc lại bài viết NoSQL.

**Isolation – độc lập:**

Nếu hai transaction thực thi cùng lúc thì nguyên tắc thực thi là thực thi độc lập. Nghĩa là một transaction không thể “nhìn thấy” một transaction khác. “Không nhìn thấy” ở đây là không tác động lẫn nhau, chủ yếu trên dữ liệu.

**Durability – bền vững:**

Dữ liệu của transaction sau khi thực thi xong được cố định, chính thức và bền vững. Nghĩa là những thay đổi đã được cố định, không có chuyện có thể chuyển lại trạng thái dữ liệu lúc trước khi thực hiện transaction.

1. **Index**

Index (hay chỉ mục) là một cấu trúc dữ liệu để tăng hiệu suất truy vấn của cơ sở dữ liệu. Index cho phép cơ sở dữ liệu thực hiện một số câu truy vấn có điều kiện nhanh hơn so với thông thường. Nhưng index cũng được lưu trên bộ nhớ và tiêu tốn không gian bộ nhớ và thời gian để tạo, cập nhật index nên khi sử dụng index cần phải suy xét kĩ.

Unique Index trên một bảng. Một Unique Index nghĩa là hai hàng không thể có cùng giá trị chỉ mục. Dưới đây là cú pháp để tạo một chỉ mục trên một bảng.

|  |
| --- |
| CREATE UNIQUE INDEX index\_name  ON table\_name ( column1, column2,...); |

1. **Trigger**

Trigger là một đối tượng được định danh trong CSDL và được gắn chặt với một sự kiện xảy ra trên một bảng nào đó (điều này có nghĩa là nó sẽ được tự động thực thi khi xảy ra một sự kiện trên một bảng). Các sự kiện này bao gồm: INSERT, UPDATE hay DELETE một bảng.

**Cú pháp tạo Trigger**

|  |
| --- |
| CREATE TRIGGER trigger\_name trigger\_time trigger\_event  ON table\_name  FOR EACH ROW  BEGIN  ...  END |

**Ưu nhược điểm của trigger**

**- Ưu điểm:**

– Dễ dàng kiểm tra tính toàn vẹn của csdl. – Trigger có thể bắt lỗi logic nghiệp vụ (business logic) ở mức csdl.

– Có thể dùng trigger là một cách khác để thay thế việc thực hiện những công việc hẹn theo giờ theo lịch.

– Trigger rất hiệu quả khi được sử dụng để kiểm soát những thay đổi của dữ liệu trong bảng.

**- Nhược điểm:**

– Trigger chỉ là một phần mở rộng của việc kiểm tra tính hợp lệ của dữ liệu chứ không thay thế được hoàn toàn công việc này.

– Trigger hoạt động ngầm ở trong csdl, không hiển thị ở tầng giao diện. Do đó, khó chỉ ra được điều gì xảy ra ở tầng csdl.

– Trigger thực hiện các update lên bảng dữ liệu vì thế nó làm gia tăng lượng công việc lên csdl và làm cho hệ thống chạy chậm