# Cơ sở dữ liệu

Cơ sở dữ liệu hay Database là tập hợp dữ liệu có tổ chức được lưu trữ và truy xuất trên thiết bị điện tử thông qua việc sử dụng hệ quản trị cơ sở dữ liệu (DBMS)

**MongoDB**

NoSQL:

* NoSQL là 1 dạng CSDL mã nguồn mở và được viết tắt bởi: None-Relational SQL hay có nơi thường gọi là Not-Only SQL.
* NoSQL được phát triển trên Javascript Framework với kiểu dữ liệu là JSON và dạng dữ liệu theo kiểu key và value.
* NoSQL ra đời như là 1 mảnh vá cho những khuyết điểm và thiếu xót cũng như hạn chế của mô hình dữ liệu quan hệ RDBMS (Relational Database Management System - Hệ quản trị cơ sở dữ liệu quan hệ) về tốc độ, tính năng, khả năng mở rộng,...
* NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng.
* NoSQL có thể mở rộng dữ liệu mà không lo tới những việc như tạo khóa ngoại, khóa chính, kiểm tra ràng buộc .v.v ...

MongoDB:

* MongoDB là một hệ quản trị cơ sở dữ liệu mã nguồn mở, là CSDL thuộc NoSql
* MongoDB là một database hướng tài liệu (document), các dữ liệu được lưu trữ trong document kiểu JSON thay vì dạng bảng như CSDL quan hệ nên truy vấn sẽ rất nhanh.
* So với RDBMS thì trong MongoDB **collection** ứng với **table**, còn **document** sẽ ứng với **row** , MongoDB sẽ dùng các document thay cho row trong RDBMS.
* Các collection trong MongoDB được cấu trúc rất linh hoạt, cho phép các dữ liệu lưu trữ không cần tuân theo một cấu trúc nhất định.
* Thông tin liên quan được lưu trữ cùng nhau để truy cập truy vấn nhanh thông qua ngôn ngữ truy vấn MongoDB

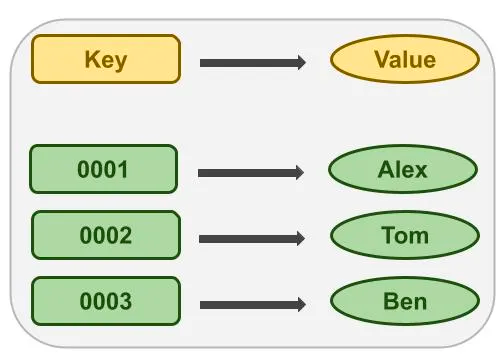
Ưu điểm của MongoDB

* Do MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ có các kích cỡ và các document khác nhau, linh hoạt trong việc lưu trữ dữ liệu, có thể insert thoải mái
* Dữ liệu trong MongoDB không có sự ràng buộc lẫn nhau, không có join như trong RDBMS nên khi insert, xóa hay update nó không cần phải mất thời gian kiểm tra xem có thỏa mãn các ràng buộc dữ liệu như trong RDBMS.
* MongoDB rất dễ mở rộng (Horizontal Scalability). Trong MongoDB có một khái niệm cluster là cụm các node chứa dữ liệu giao tiếp với nhau, khi muốn mở rộng hệ thống ta chỉ cần thêm một node với vào cluster:
* Trường dữ liệu “\_id” luôn được tự động đánh index (chỉ mục) để tốc độ truy vấn thông tin đạt hiệu suất cao nhất.
* Khi có một truy vấn dữ liệu, bản ghi được cached lên bộ nhớ Ram, để phục vụ lượt truy vấn sau diễn ra nhanh hơn mà không cần phải đọc từ ổ cứng.
* Hiệu năng cao: Tốc độ truy vấn (find, update, insert, delete) của MongoDB nhanh hơn hẳn so với các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS). Với một lượng dữ liệu đủ lớn thì thử nghiệm cho thấy tốc độ insert của MongoDB có thể nhanh tới gấp 100 lần so với MySQL.

Nhược điểm của MongoDB

* MongoDB không có tính chất ràng buộc như RDBMS nên khi thao tác cần cẩn thận
* Tốn bộ nhớ do dữ liệu lưu dưới dạng key-value, các collection chỉ khác về value do đó key sẽ bị lặp lại. Không hỗ trợ join nên dễ bị dữ thừa dữ liệu.

*Key-Value Database:*



Key-Value Database là một loại trong NoSQL database sử dụng phương thức đơn giản là key-value để lưu trữ dữ liệu, ở đó, dữ liệu được lưu trữ dưới dạng tập hợp của một cặp key-value. Mỗi key có một giá trị xác định duy nhất trên toàn bộ datbase, từ đó có thể lấy được value dựa theo key.

Về cơ bản key-value được coi là các bảng chỉ có 2 cột và không có tính năng joins như các bảng trong SQL. Các cơ sở dữ liệu dạng này thường đơn giản, kích thước nhỏ, lưu phân tán tại nhiều nơi và lưu ở trong RAM nhằm tăng tốc độ xử lý.

Do đặc tính tổ chức đơn giản nên Key-Value Database có thể dễ dàng phân mảnh và lưu trữ trên nhiều máy tính khác nhau, giúp cho việc mở rộng theo chiều ngang được thực hiện một cách dễ dàng và đáng tin cậy.

Key-Value Database được sử dụng nhiều trong các ứng dụng cần tốc độ truy xuất dữ liệu nhanh như lưu trữ phiên làm việc của người dùng trong môi trường phân tán, hay làm cache để truy xuất thông tin nhanh hơn. Một số Key-Value Database được sử dụng nhiều hiện nay như Amazon DynamoDB, Redis ...

*Document database:*

Document database là một loại database NoSQL lưu trữ dữ liệu dưới dạng tài liệu (document). Mỗi tài liệu có thể chứa nhiều trường dữ liệu và được lưu trữ dưới dạng JSON hoặc BSON hoặc XML . Các trường có thể có các loại dữ liệu khác nhau, chẳng hạn như số, chuỗi, mảng, và đối tượng.

DocuDB là một phần mở rộng của Key-Value, nơi các giá trị được lưu trữ trong các document có cấu trúc như XML hoặc JSON. Document giúp bạn dễ dàng ánh xạ các đối tượng trong hướng đối tượng sang database.

Document database được thiết kế để lưu trữ dữ liệu bán cấu trúc (dữ liệu không có cấu trúc cố định ). Điều này làm cho chúng phù hợp để lưu trữ dữ liệu từ các nguồn khác nhau, chẳng hạn như dữ liệu từ ứng dụng web, dữ liệu từ cảm biến, và dữ liệu từ hệ thống phân tán ...

*Ưu điểm:*

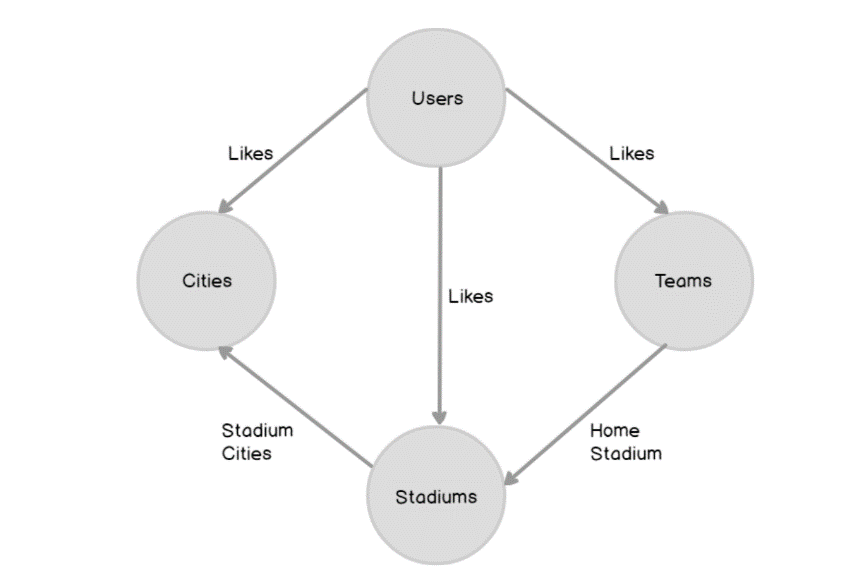
* **Document model:**Mô hình tài liệu là phổ biến, trực quan và cho phép phát triển phần mềm nhanh chóng.
* **Flexible schema:**Schema linh hoạt cho phép mô hình dữ liệu thay đổi khi yêu cầu của ứng dụng thay đổi.
* **Rich APIs and query languages:**Document database có API và ngôn ngữ truy vấn phong phú cho phép nhà phát triển dễ dàng tương tác với dữ liệu của họ.
* **Distributed and resilient:**Document database có thể dễ dàng mở rộng theo chiều ngang và có khả năng chịu tải và phục hồi nhanh sau sự cố.

*Nhược điểm:*

* Cơ sở dữ liệu document không hỗ trợ quan hệ. Mỗi document trong kho lưu trữ là độc lập và không có tính toàn vẹn quan hệ. Điều này giúp ta giảm rất nhiều bảng liên kết cũng như nested-joins như ở SQL. Chúng ta có thể truy cập thuộc tính lồng nhau nhiều mức thay vì phải joins các bảng 1 cách phức tạp.

Graph DB:

Graph database là một loại database NoSQL lưu trữ dữ liệu dưới dạng đồ thị. Đồ thị là một cấu trúc dữ liệu bao gồm các đỉnh (nút) được kết nối với nhau bằng các cạnh. Mỗi đỉnh có thể đại diện cho một thực thể, chẳng hạn như một người, một địa điểm, hoặc một sản phẩm. Mỗi cạnh có thể đại diện cho một mối quan hệ giữa hai thực thể, chẳng hạn như bạn bè, hàng xóm, hoặc mua bán.



Dữ liệu trong graph database được lưu dưới dạng các node. Mỗi node sẽ có 1 label, 1 số properties như một row trong SQL. Các node này được kết nối với nhau bằng các relationship. Graph database tập trung nhiều vào relationship giữa các node, áp dụng nhiều thuật toán duyệt node để tăng tốc độ.

Graph database được thiết kế để lưu trữ dữ liệu có nhiều mối quan hệ phức tạp. Điều này làm cho chúng phù hợp để lưu trữ dữ liệu từ các lĩnh vực như mạng xã hội, tài nguyên mạng, và phân tích dữ liệu, hệ thống gợi ý ...

# TableDB, Database engine

Cấu trúc dữ liệu dạng bảng (CSDL quan hệ) chính là RDBMS

So sánh RBDMS và NoSQL

***So sánh về cấu trúc dữ liệu***

* RDBMS: Sử dụng mô hình dữ liệu có cấu trúc với các bảng, hàng, cột và quan hệ giữa các bảng. Dữ liệu được lưu trữ trong các bảng được định rõ trước, theo một schema.
* NoSQL: Sử dụng mô hình dữ liệu không có cấu trúc hoặc có cấu trúc linh hoạt. Dữ liệu có thể lưu trữ dưới dạng tài liệu, cặp khóa-giá trị, đồ thị hoặc các cấu trúc khác mà không yêu cầu schema cứng.

***So sánh cách thể hiện quan hệ dữ liệu***

* RDBMS: Sử dụng quan hệ để kết nối các bảng và thể hiện mối quan hệ giữa dữ liệu thông qua khóa ngoại (foreign keys).
* NoSQL: Thể hiện mối quan hệ dữ liệu theo cách khác nhau hoặc không thể hiện (tùy thuộc vào loại NoSQL).

***So sánh khả năng mở rộng và hiệu suất***

* RDBMS: Thường khó mở rộng theo quy mô lớn và cần phải sử dụng các giải pháp phân tán phức tạp để đáp ứng các yêu cầu cao cấp về hiệu suất và khả năng chịu lỗi.
* NoSQL: NoSQL thường dễ dàng mở rộng ngang (horizontal scaling) và đáp ứng dễ dàng các yêu cầu về hiệu suất và khả năng chịu lỗi.

***Khả năng tích hợp dữ liệu***

* RDBMS: Thường yêu cầu phải tuân theo schema cố định, điều này có thể gây khó khăn trong việc tích hợp dữ liệu từ các nguồn khác nhau.
* NoSQL: Có khả năng linh hoạt hơn trong việc tích hợp dữ liệu từ nhiều nguồn khác nhau mà không cần phải áp dụng schema cứng.

***Loại dữ liệu và ứng dụng phù hợp***

* RDBMS: Phù hợp cho các ứng dụng có cấu trúc, như hệ thống quản lý khách hàng, tài khoản ngân hàng, và các ứng dụng yêu cầu tính nhất quán cao.
* NoSQL: Phù hợp cho các ứng dụng có cấu trúc linh hoạt hoặc không cố định, chẳng hạn như các ứng dụng truyền phương tiện xã hội, lưu trữ dữ liệu IoT, và ứng dụng đám mây.

***Tốc độ đọc và ghi***

* RDBMS: RDBMS thường phù hợp với ứng dụng yêu cầu đọc và ghi dữ liệu với tính nhất quán cao, đặc biệt trong các giao dịch và ứng dụng kinh doanh quan trọng. Tuy nhiên, hiệu suất có thể trở thành vấn đề khi cơ sở dữ liệu quá lớn hoặc khi có nhiều giao dịch cùng một lúc.
* NoSQL: NoSQL có thể cung cấp hiệu suất đọc và ghi cao, đặc biệt là trong trường hợp cần khả năng mở rộng theo chiều ngang. Dựa vào loại cơ sở dữ liệu NoSQL (tài liệu, khóa-giá trị, cột rộng, đồ thị), tốc độ có thể thay đổi. Ví dụ, cơ sở dữ liệu tài liệu thường cung cấp tốc độ ghi nhanh hơn so với cơ sở dữ liệu cột rộng.

***Truy vấn***

* RDBMS: RDBMS thường mạnh mẽ trong việc hỗ trợ truy vấn phức tạp bằng SQL. Nó thích hợp cho các ứng dụng cần truy vấn phức tạp và nhiều quan hệ dữ liệu.
* NoSQL: Các cơ sở dữ liệu NoSQL có các ngôn ngữ truy vấn riêng, và khả năng thực hiện truy vấn có thể khác nhau tùy theo loại cơ sở dữ liệu. Trong một số trường hợp, truy vấn cơ bản có thể được thực hiện nhanh chóng, nhưng truy vấn phức tạp có thể gặp khó khăn hơn.

***Báo cáo***

* RDBMS: RDBMS thường là lựa chọn ưa thích cho việc tạo báo cáo phức tạp và phân tích dữ liệu vì SQL hỗ trợ các truy vấn phức tạp và thao tác nâng cao. Các công cụ báo cáo truyền thống thường tương thích với RDBMS.
* NoSQL: NoSQL có thể gặp khó khăn trong việc tạo báo cáo phức tạp, đặc biệt là khi cơ sở dữ liệu không hỗ trợ các tính năng phân tích mạnh mẽ hoặc khi cần kết hợp dữ liệu từ nhiều nguồn khác nhau.

**Các kiểu dữ liệu của MySQL**

1. **Kiểu dữ liệu số**
   1. **Kiểu dữ liệu số nguyên**

* **Tinyint**
  + Độ dài (số byte): 1
  + Giá trị lưu trữ (có dấu): -128 **-** 127
  + Giá trị lưu trữ (không dấu): 0 **-** 255
* SMALLINT
  + Độ dài (số byte): 2
  + Giá trị lưu trữ (có dấu): -32768 **-** 32767
  + Giá trị lưu trữ (không dấu): 0 **-** 65535
* MEDIUMINT
  + Độ dài (số byte): 3
  + Giá trị lưu trữ (có dấu): -8388608 **-** 8388607
  + Giá trị lưu trữ (không dấu): 0 **-** 16777215
* INT
  + Độ dài (số byte): 4
  + Giá trị lưu trữ (có dấu): -2147483648 **-** 2147483647
  + Giá trị lưu trữ (không dấu): 0 **-** 4294967295
* BIGINT
  + Độ dài (số byte): 8
  + Giá trị lưu trữ (có dấu): -9223372036854775808 **-** 92233720368 54775807
  + Giá trị lưu trữ (không dấu): 0 **-** 184467440737 09551615
  1. **Kiểu dữ liệu số thực**
* FLOAT(M,D)

(M) là độ dài phần nguyên (số chứ số phần nguyên)

(D) là độ dài phần thập phân (số chữ số phần thập phân)

* + - Độ dài (số byte): 4
    - Giá trị lưu trữ (có dấu): -3.402823466E+38 **đến** -1.175494351E-38
    - Giá trị lưu trữ (không dấu): 1.175494351E-38 **đến** 3.402823466E+38
* DOUBLE(M,D)

(M) là độ dài phần nguyên (số chứ số phần nguyên)

(D) là độ dài phần thập phân (số chữ số phần thập phân)

Độ dài (số byte): 8

Giá trị lưu trữ (có dấu): -1.7976931348623157E+ 308 **đến** -2.2250738585072014E- 308

Giá trị lưu trữ (không dấu): 0 and 2.2250738585072014E- 308 **đến** 1.7976931348623157E+ 308

* 1. **Kiểu dữ liệu decimal và numberic**

Trong MySQL kiểu DECIMAL và NUMERIC lưu trữ chính xác các dữ liệu số, định dạng nhị phân.

Decimal (5,2): Nghĩa là nó có thể lưu trữ một giá trị có 5 chữ số trong đó có 2 số thập phân

1. **Kiểu dữ liệu Date & Time**
   1. **Kiểu dữ liệu Date**

Lưu trữ thông tin ngày tháng năm:

* Định dạng hiển thị: YYYY-MM-DD
* Phạm vi thời gian: **'1000-01-01 00:00:00'** to **'9999-12-31'**
  1. **Kiểu dữ liệu Datetime**

Lưu trữ thông tin ngày tháng năm và giờ phút giây

* Định dạng hiển thị: YYYY-MM-DD HH:MM:SS
* Phạm vi thời gian: '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
  1. **Kiểu dữ liệu TimeStamp**

Sử dụng khi bạn cần giá trị lưu trữ thời gian.

* Định dạng hiển thị: HH:MM:SS
* Phạm vi thời gian: '-838:59:59' to '838:59:59'

TimeStamp cũng có thể lưu ở dạng

Lưu ý: Phần thời gian có thể lớn bởi vì kiểu TIME có thể không chỉ mô tả thời gian của một ngày (Vốn chỉ có tối đa 24 giờ), mà nó có thể là thời gian trôi qua hoặc khoảng thời gian giữa hai sự kiện (Cái mà có thể lớn hơn 24h thậm trí có giá trị âm).

Sự khác biệt chính của DATETIME và TIMESTAMP là giá trị của TIMESTAMP được chuyển đổi từ múi giờ hiện tại sang UTC trong khi lưu trữ, và chuyển ngược trở lại từ UTC sang múi giờ hiện tại trong lúc lấy ra. Còn kiểu dữ liệu DATETIME thì không có gì thay đổi002E

* 1. **Kiểu dữ liệu Year**

Kiểu dữ liệu YEAR được sử dụng 1-byte để mô tả giá trị.

* YEAR(2)
  + Chỉ định rõ chiều rộng hiển thị là 2 ký tự
  + Định dạng hiển thị: Hiển thị 2 số cuối của năm
  + Phạm vi: **'0'** to **'99'**
* YEAR(4)
  + Chỉ định rõ chiều rộng hiển thị là 4 ký tự. Nếu không chỉ rõ chiều rộng mặc định là 4 ký tự
  + Định dạng hiển thị: YYYY
  + Phạm vi: **'1901'** to **'2155'**

1. **Kiểu chuỗi**
   1. **Char và Varchar**

Kiểu dữ liệu CHAR và VARCHAR là giống nhau, nhưng khác nhau ở cách chúng được lưu trữ và truy xuất. Chúng cũng khác nhau về chiều dài tối đa và giữ lại hay không khoảng trắng phía trước (gọi là trailing spaces).

* **CHAR**
  + Định dạng hiển thị: Khoảng trắng phía trước (Trailing spaces) được loại bỏ
  + Phạm vi các ký tự: **'0'** to **'255'**
* **VARCHAR**
  + Định dạng hiển thị: Giống như data được nhập và lưu trữ
  + Phạm vi các ký tự: **'0'** to **'65535'**
* **CHAR**: Lưu trữ chuỗi ký tự có độ dài cố định. Nếu bạn khai báo một cột **CHAR(10)**, nó sẽ luôn lưu trữ 10 ký tự, và nếu chuỗi ít hơn 10 ký tự, MySQL sẽ điền vào bằng các ký tự trắng.
* **VARCHAR**: Lưu trữ chuỗi ký tự có độ dài biến đổi. Nếu bạn khai báo một cột **VARCHAR(10)**, nó chỉ sử dụng bộ nhớ cần thiết để lưu trữ dữ liệu thực tế và không thêm các ký tự trắng để đạt đến độ dài đã khai báo.
* Cột CHAR có thể tiêu tốn nhiều bộ nhớ hơn so với VARCHAR, vì nó lưu trữ một lượng cố định của dữ liệu, thậm chí khi độ dài thực tế là ít hơn. VARCHAR thường ít tốn bộ nhớ hơn do không giữ các ký tự trắng dư thừa.
  1. **Binary và Varbinary**

Các kiểu dữ liệu BINARY và VARBINARY tương tự như CHAR và VARCHAR, ngoại trừ việc chúng có chứa các chuỗi nhị phân chứ không phải là chuỗi non-binary.

* **BINARY**
  + Định dạng hiển thị: Chứa các chuỗi nhị phân (Binary Strings)
  + Phạm vi các ký tự: **'0'** to **'255'**
* **VARBINARY**
  + Định dạng hiển thị: Chứa các chuỗi nhị phân (Binary Strings)
  + Phạm vi các ký tự: **'0'** to **'65535'**
  1. **Blob và text**

**BLOB**

Là một đối tượng nhị phân lớn (Binary Large OBject) có thể chứa một lượng lớn dữ liệu. Có 4 loại BLOB:

* TINYBLOB (Chiều dài tối đa là 255 ký tự)
* BLOB
* MEDIUMBLOB (Chiều dài tối đa là 16777215 ký tự)
* LONGBLOB (Chiều dài tối đa là 4294967295 ký tự)

**TEXT**

Lưu trữ giá trị được coi như một chuỗi các ký tự có mã hóa (character set). Có 4 loại TEXT tương ứng với bốn loại BLOB:

* TINYTEXT (Chiều dài tối đa là 255 ký tự)
* TEXT
* MEDIUMTEXT (Chiều dài tối đa là 16777215 ký tự)
* LONGTEXT (Chiều dài tối đa là 4294967295 ký tự)
  1. **Enum**

**Khi định nghĩa một trường kiểu này, tức là, ta đã chỉ ra một danh sách các đối tượng mà trường phải nhận (có thể là Null).**

# Index

**Khóa chính:** Khóa chính (hay ràng buộc khóa chính) được sử dụng để định danh duy nhất mỗi record trong table của cơ sở dữ liệu. Ngoài ra, nó còn dùng để thiết lập quan hệ 1-n (hay ràng buộc tham chiếu) giữa hai table trong cơ sở dữ liệu.

Dữ liệu (value) của field khóa chính phải có tính duy nhất. Và không chứa các giá trị Null.

**Khóa ngoại** của một table được xem như con trỏ trỏ tới khóa chính của table khác.

**Index** là một cấu trúc dữ liệu giúp xác định nhanh chóng các records trong table.

**Ưu điểm**

Ưu điểm của index là tăng tốc độ tìm kiếm records theo câu lệnh WHERE.

Không chỉ giới hạn trong câu lệnh SELECT mà với cả xử lý UPDATE hay DELETE có điều kiện WHERE.

**Nhược điểm**

- Khi sử dụng index thì tốc độ của những xử lý ghi dữ liệu (Insert, Update, Delete) sẽ bị chậm đi.

- Vì ngoài việc thêm hay update thông tin data thì MYSQL cũng cần update lại thông tin index của bảng tương ứng.

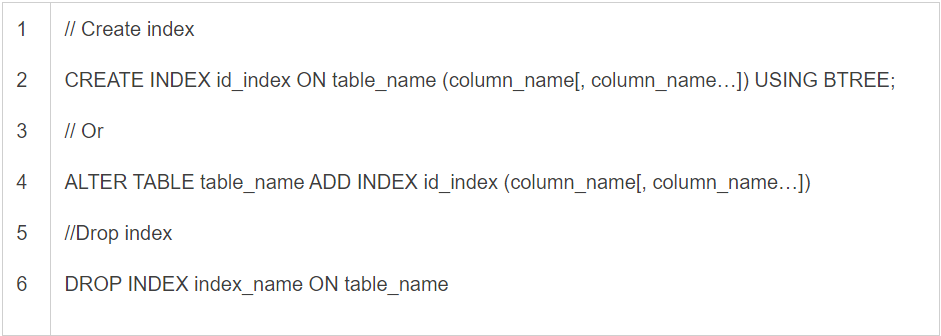
- Tốc độ xử lý bị chậm đi cũng tỷ lệ thuận với số lượng index được xử dụng trong bảng.

- Do vậy với những table hay có xử lý insert, update hoặc delete và cần tốc độ xử lý nhanh thì không nên được đánh index.

- Ngoài ra việc đánh index cũng sẽ tốn resource của server như thêm dung lượng cho CSDL.

B-Tree index:

Thông thường khi nói đến index mà không chỉ rõ loại index thì default là sẽ sử dụng B-Tree index.



**Các đặc điểm của B-Tree Index:**

– Dữ liệu index được tổ chức và lưu trữ theo dạng tree, tức là có root, branch, leaf.

Cách sắp xếp không phải theo dạng cây tìm kiếm nhị phân – Binary search tree vì số lá là mỗi node không bị giới hạn là 2.

– Giá trị của các node được tổ chức tăng dần từ trái qua phải.

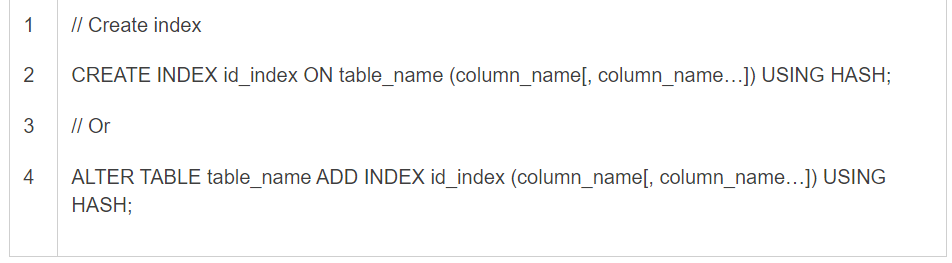
– B-Tree index được sử dụng trong các biểu thức so sánh dạng: =, >, >=, <, <=, BETWEEN và LIKE. ⇒ Có thể tối ưu tốt cho câu lệnh **ORDER BY**

– Khi truy vấn dữ liệu thì CSDL sẽ không scan dữ liệu trên toàn bộ bảng để tìm dữ liệu, việc tìm kiếm trong B-Tree là 1 quá trình đệ quy, bắt đầu từ root node và tìm kiếm tới branch và leaf, đến khi tìm được tất cả dữ liệu – thỏa mãn với điều kiện truy vấn thì mới dùng lại.

### **Hash Index**

Hash index dựa trên giải thuật **Hash Function (hàm băm)**. Tương ứng với mỗi khối dữ liệu (index) sẽ sinh ra một bucket key(giá trị băm) để phân biệt.

**Cú pháp:**



– Khác với B-Tree, thì Hash index chỉ nên sử dụng trong các biểu thức toán tử là = và <>. Không sử dụng cho toán từ tìm kiếm 1 khoảng giá trị như > hay < .

– Không thể tối ưu hóa toán tử **ORDER BY** bằng việc sử dụng Hash index bởi vì nó không thể tìm kiếm được phần từ tiếp theo trong Order.

– Hash có tốc độ nhanh hơn kiểu B-Tree.

**Các Storage Engine:**

**InnoDB**

Đây là Storage Engine mặc định trong MySQL 5.7. **InnoDB** là một Storage Engine transaction-safe (tuân thủ ACID) cho MySQL có các commit, rollback và khả năng khôi phục lỗi để bảo vệ dữ liệu người dùng. Row-level locking của InnoDB và kiểu nonlocking read của Oracle-style làm tăng sự đồng thời và hiệu suất của nhiều người dùng. InnoDB lưu trữ dữ liệu người dùng trong các clustered indexes để giảm I/O cho các truy vấn thông thường dựa trên các primary key. Để duy trì tính toàn vẹn của dữ liệu, InnoDB cũng hỗ trợ các ràng buộc toàn vẹn Foreign Key.

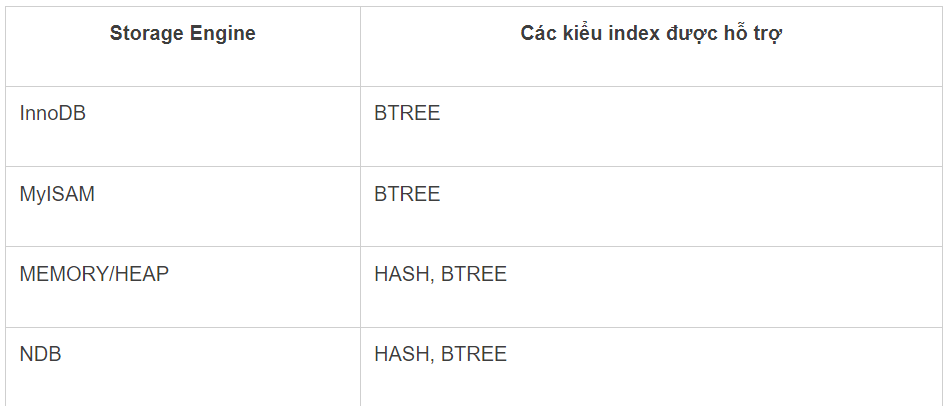
**MyISAM**

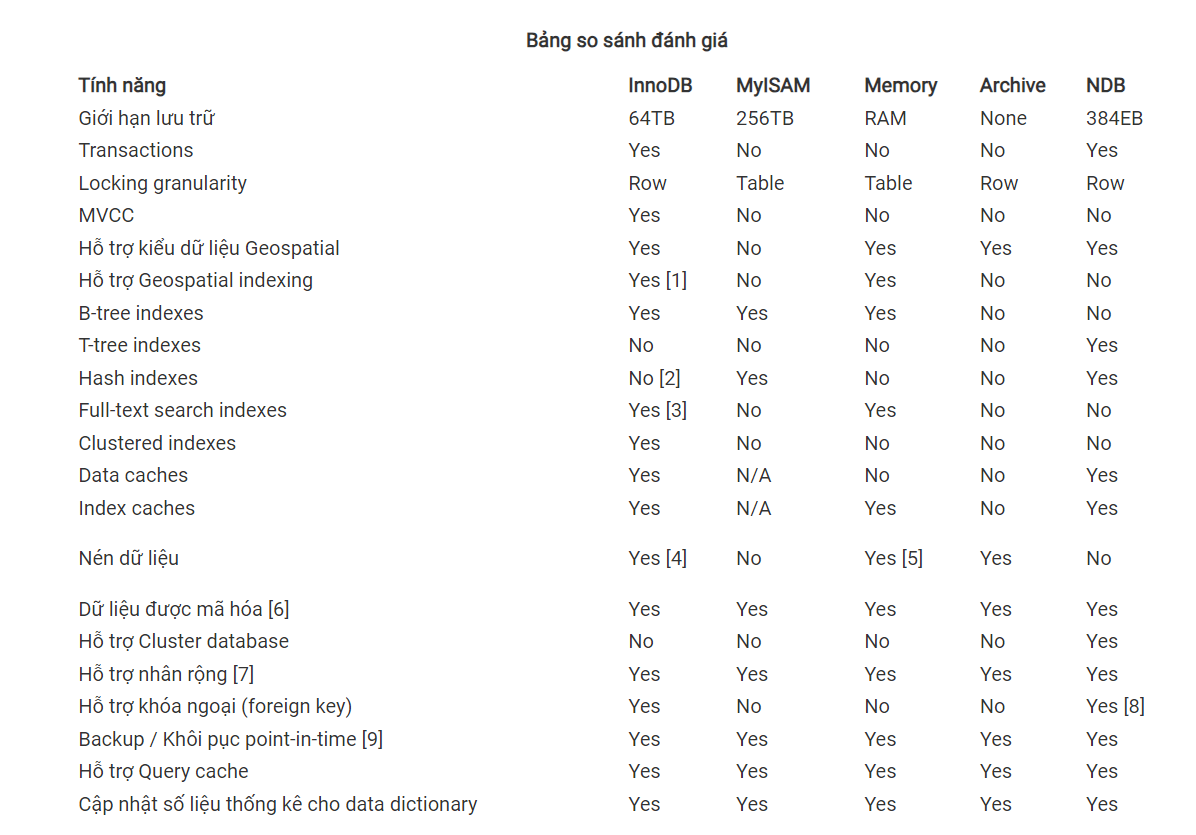
Table-level locking giới hạn hiệu suất read/write dữ liệu, vì vậy nó thường được sử dụng cho các công việc read-only hoặc read-mostly trong các cấu hình Web và lưu trữ dữ liệu.

**Memory**

Lưu trữ tất cả dữ liệu trong RAM, để truy cập nhanh trong các môi trường đòi hỏi tra cứu nhanh các dữ liệu không quan trọng. Engine này trước đây gọi là HEAP Engine. Storage Engine này đang sử dụng ít dần, do InnoDB với vùng bộ đệm cung cấp một cách mục đích chung và bền để giữ hầu hết hoặc tất cả dữ liệu trong memory, và NDBCLUSTER cung cấp tra cứu giá trị quan trọng nhanh cho các bộ dữ liệu phân tán lớn.

**Các kiểu index tương ứng trong storage engine**



****

1. InnoDB:

- InnoDB là storage engine mặc định cho MySQL từ phiên bản 5.5 trở đi.

- Hỗ trợ cả Clustered Index (Primary Key) và Non-Clustered Index.

- Dữ liệu và index được lưu trữ trong các file dữ liệu riêng biệt (bảng InnoDB thường được mô tả như có cấu trúc dạng cây B).

2. MyISAM:

- Mặc định cho MySQL trước phiên bản 5.5.

- Hỗ trợ Non-Clustered Index.

- Dữ liệu và index được lưu trữ trong các file riêng biệt.

- Thường nhanh hơn cho các truy vấn đọc nhưng không hỗ trợ giao dịch và các tính năng ACID.

3. MEMORY (HEAP):

- Lưu trữ toàn bộ bảng trong bộ nhớ.

- Hỗ trợ Non-Clustered Index.

- Index và dữ liệu được lưu trữ trong bộ nhớ RAM, nhanh nhưng có thể bị mất khi hệ thống tắt.

4. TokuDB:

- Một storage engine thay thế cho InnoDB với hiệu suất và nén dữ liệu tốt hơn.

- Sử dụng cấu trúc cây B+ và bảng có thể chia thành các phần nhỏ (partition) để tối ưu hiệu suất.

5.ARCHIVE:

- Dùng để lưu trữ và truy vấn dữ liệu lớn.

- Chỉ hỗ trợ Non-Clustered Index.

- Dữ liệu được lưu trữ dưới dạng file nén và thích hợp cho các tình huống lưu trữ lớn như lưu trữ log.

Việc đánh index cho tất cả các field có thể mang lại lợi ích tìm kiếm nhanh. Index giúp tăng tốc quá trình tìm kiếm dữ liệu trong bảng, đặc biệt là với các truy vấn SELECT. Tuy nhiên nó cũng có các nhược điểm

* Tăng kích thước bảng: Index sử dụng không gian lưu trữ, do đó, việc đánh index cho tất cả các cột có thể làm tăng kích thước bảng.
* Chậm khi cập nhật: Mọi lần cập nhật, thêm mới hoặc xóa dữ liệu sẽ ảnh hưởng đến các index liên quan, làm giảm hiệu suất của các thao tác này.
* Chọn lọc không hiệu quả: Nếu không sử dụng một index, việc đánh index cho nó có thể làm tăng overhead mà không cung cấp lợi ích đáng kể.
* Tiêu tốn tài nguyên

# Các câu truy vấn

**Lệnh replace:**

REPLACE INTO table\_name (column1, column2, ...)   
VALUES (value1, value2, ...);

REPLACE là một lệnh SQL dùng để thêm hoặc cập nhật dữ liệu trong bảng. Nếu bản ghi đã tồn tại dựa trên khóa chính hoặc khóa duy nhất, nó sẽ bị xóa và thêm một bản ghi mới. Nếu bản ghi chưa tồn tại, REPLACE sẽ thực hiện như lệnh INSERT.

Ví dụ:

REPLACE INTO users (id, username, email)

VALUES (1, 'john\_doe', 'john@example.com');

**Hàm Replace**

REPLACE(str, find\_string, replace\_with);

Ví dụ :

SELECT REPLACE('Hello World', 'World', 'MySQL');

🡪 Kết quả: Hello MySQL

REPLACE là một hàm chuỗi trong MySQL.

Nó được sử dụng để thay thế tất cả các xuất hiện của find\_string trong chuỗi str bằng replace\_with.

Trả về chuỗi mới.

REPLACE và hàm REPLACE có mục đích khác nhau. Lệnh REPLACE được sử dụng trong ngữ cảnh của một câu lệnh SQL để thao tác dữ liệu trong bảng. Trong khi đó, hàm REPLACE được sử dụng để thực hiện thay thế trong nội dung của một chuỗi.

**Loại trừ các bản ghi trùng nhau** •

Từ khoá DISTINCT

SELECT DISTINCT <DS cột >

FROM <DS bảng>

Ví dụ: đưa ra danh sách tên các khoa (Dept) tương ứng với các khoá học (Course). Mỗi giá trị chỉ hiện thị một lần

SELECT DISTINCT Dept

FROM Course

Tính tỷ lệ có gender = ‘F’ trong bảng employees

SELECT

COUNT(\*) AS total,

(SUM(gender = 'F') / COUNT(\*)) \* 100 AS female\_percentage

FROM employeesbackup.employees;

SELECT \* FROM employeesbackup.employees;

**Từ csdl, chọn ra phòng ban có nhiều nhân viên có titles = “engineer” nhất**

C1:

Select d.dept\_name, count(\*) count

from employees.departments as d

join employees.dept\_emp as de on d.dept\_no = de.dept\_no

join employees.employees as e on de.emp\_no = e.emp\_no

join employees.titles as t on t.emp\_no = e.emp\_no

where t.title= 'Engineer'

group by de.dept\_no

order by count desc

limit 1;

C2:

select de.dept\_no, count(\*) count

from dept\_emp de

join employees e on de.emp\_no = e.emp\_no

join titles t on e.emp\_no = t.emp\_no

where t.title = 'Engineer'

group by de.dept\_no

order by count desc

limit 1

**Cách sử dụng GROUP BY**

Lệnh Group by trong SQL Server được dùng để sắp xếp dữ liệu giống hệt nhau theo các nhóm với sự trợ giúp của một số hàm khác. Ví dụ, nếu một cột cụ thể có cùng giá trị với các hàng khác nhau, thì nó sẽ sắp xếp những hàng đó vào một nhóm.

Đặc điểm nổi bật của Group by trong SQL Server

* Mệnh đề Group by được dùng với lệnh Select.
* Trong truy vấn này, lệnh Group by được đặt sau mệnh đề WHERE.
* Trong truy vấn này, mệnh đề GROUP BY được đặt trước lệnh Having.

Cú pháp:

SELECT column1, column2,...columnN FROM table\_name

[WHERE]

[GROUP BY column1, column2...columnN]

[HAVING]

[ORDER BY]

Mệnh đề SELECT có thể bao gồm cột được dùng với SQL GROUP BY. Vì thế, để bao gồm các cột khác trong hàm GROUP BY, dùng các hàm tổng hợp như COUNT(), MAX(), MIN(), SUM(), AVG() với những cột đó.

**Đặc điểm nổi bật của GROUP BY trong SQL:**

* Được dùng để tạo thành nhóm các bản ghi.
* Đứng sau mệnh đề WHERE nếu có và trước HAVING.
* Có thể bao gồm một hoặc nhiều cột để tạo thành một hoặc nhiều nhóm dựa trên cột đó.
* Chỉ các cột GROUP BY có thể được dùng bao gồm trong mệnh đề SELECT. Để dùng các cột khác trong SELECT, dùng hàm tổng hợp.

***– Mệnh đề GROUP BY phải luôn theo sau các điều kiện thuộc mệnh đề WHERE, tức sau khi viết mệnh đề WHERE xong xuống dòng đến mệnh đề GROUP BY.– Mệnh đề ORDER BY luôn theo sau mệnh đề GROUP BY nếu được sử dụng ở trong cú pháp truy vấn dữ liệu.***

**VÍ DỤ CHO BẢNG:**

+------------+----------+-------------+-------+

|IDChuyenmuc | Muccon |Chuyenmuclon | Sobai |

+------------+----------+-------------+-------+

| 1 |SQL Server|Lap trinh | 101 |

| 2 |Facebook |Mang xa hoi | 152 |

| 3 |Python |Lap trinh | 101 |

| 4 |JavaScript|Lap trinh | 122 |

| 5 |Chrome |Web | 94 |

| 6 |Instagram |Mang xa hoi | 165 |

+------------+----------+-------------+-------+

Cú pháp mệnh đề GROUP BY trong SQL Server

SELECT

bieu\_thuc1, bieu\_thuc2,... bieu\_thuc\_n,

ham\_tong\_hop (bieu\_thuc)

FROM

ten\_bang

[WHERE dieukien]

GROUP BY

bieu\_thuc1,

bieu\_thuc2,...;

Tên biến hoặc giá trị biến:

* **bieu\_thuc1, bieu\_thuc2,... bieu\_thuc\_n:**Biểu thức không nằm bên trong hàm tổng hợp và phải nằm trong mệnh đề GROUP BY.
* **ham\_tong\_hop:**Có thể là các hàm như SUM, COUNT, MIN, MAX hoặc AVG.
* **ten\_bang:**Bảng cần lấy bản ghi từ đó, phải có ít nhất 1 bảng trong mệnh đề FROM.
* [**WHERE**](https://quantrimang.com/hoc/menh-de-where-trong-sql-server-147325)**dieukien:**Tùy chọn. Đây là điều kiện mà bản ghi phải đáp ứng để được chọn.

**Ví dụ - dùng hàm SUM**

SELECT Chuyenmuclon, SUM(Sobai) AS "Tong"

FROM [Quantrimang.com]

GROUP BY Chuyenmuclon;

Ví dụ này dùng hàm tính tổng SUM để trả về tên Chuyenmuclon và tổng số bài viết (Sobai) (theo tên chuyên mục lớn).

Vì bạn đưa ra 1 cột (Chuyenmuclon) trong lệnh [SELECT](https://quantrimang.com/hoc/lenh-select-trong-sql-server-147319) và không nằm trong hàm SUM nên sẽ phải dùng mệnh đề GROUP BY.

Khi đó ta có bảng kết quả trả về như sau:

+--------------+------+

| Chuyenmuclon | Tong |

+--------------+------+

|Lap trinh | 324 |

|Mang xa hoi | 317 |

|Web | 94 |

+--------------+------+

## **Ví dụ - dùng hàm COUNT**

SELECT COUNT(Chuyenmuclon), Chuyenmuclon

FROM [Quantrimang.com]

WHERE Chuyenmuclon = 'Lap trinh'

GROUP BY Chuyenmuclon;

Ở ví dụ này, hàm COUNT sẽ trả về số lượng Chuyenmuccon của Chuyenmuclon Lap trinh:

+------+--------------+

| | Chuyenmuclon |

+------+--------------+

| 3 | Lap trinh |

+------+--------------+

Nếu không muốn lấy tên cột mà chỉ cần trả về số lượng mục con, bạn chạy lệnh sau:

SELECT COUNT(Chuyenmuclon)

FROM [Quantrimang.com]

WHERE Chuyenmuclon = 'Lap trinh'

GROUP BY Chuyenmuclon;

Kết quả trả về sẽ chỉ có số 3.

## **Ví dụ - dùng hàm MIN**

SELECT Chuyenmuclon, MIN(Sobai) AS 'So luong it nhat'

FROM [Quantrimang.com]

GROUP BY Chuyenmuclon;

Mệnh đề GROUP BY dùng với hàm MIN ở trên sẽ trả về Chuyenmuclon và số lượng bài ít nhất của mục lớn đó.

CÁCH SỬ DỤNG ORDER BY:

Lệnh ORDER BY trong SQL được dùng để phân loại dữ liệu theo thứ tự tăng hoặc giảm dần, dựa trên một hoặc nhiều cột. Mệnh đề này có thể phân loại dữ liệu theo một cột hoặc nhiều cột. Phân loại theo nhiều cột có thể hữu ích khi bạn cần phân loại dữ liệu theo thứ bậc, chẳng hạn như phân loại theo bang, thành phố, rồi tới tên người.

ORDER BY được dùng với lệnh SQL SELECT và thường đứng sau WHERE, HAVING và GROUP BY.

Những điểm quan trọng về câu lệnh sắp xếp trong SQL:

* Một số database phân loại kết quả truy vấn mặc định theo thứ tự tăng dần.
* Để phân loại dữ liệu theo thứ tự tăng dần, bạn dùng từ khóa ASC.
* Để phân loại dữ liệu theo thứ tự giảm dần, dùng từ khóa DESC.

Ngoài việc phân loại bản ghi theo thứ tự tăng hoặc giảm dần, mệnh đề ORDER BY đều có thể phân loại dữ liệu trong bảng database theo thứ tự lập trình viên muốn.

## Cú pháp lệnh ORDER BY trong SQL

Cú pháp cơ bản của lệnh ORDER BY với câu lệnh SELECT sẽ như sau:

SELECT danhsach\_cot

FROM ten\_bang

[WHERE dieu\_kien]

[ORDER BY cot1, cot2, .. cotN] [ASC | DESC];

**Lưu ý**:

* Nếu không chọn cụ thể ASC hay DESC trong mệnh đề ORDER BY thì kết quả sẽ mặc định được sắp xếp theo thứ tự tăng dần, tương đương với [ORDER BY cot1, cot2, .. cotN] [ASC].

Giả sử bảng NHANVIEN có các bản ghi như sau:

+----+----------+-----+-----------+----------+

| ID | TEN |TUOI | DIACHI | LUONG |

+----+----------+-----+-----------+----------+

| 1 | Thanh | 32 | Haiphong | 2000.00 |

| 2 | Loan | 25 | Hanoi | 1500.00 |

| 3 | Nga | 23 | Hanam | 2000.00 |

| 4 | Manh | 25 | Hue | 6500.00 |

| 5 | Huy | 27 | Hatinh | 8500.00 |

| 6 | Cao | 22 | HCM | 4500.00 |

| 7 | Lam | 24 | Hanoi | 10000.00 |

+----+----------+-----+-----------+----------+

Ví dụ sau minh họa cách sắp xếp theo thứ tự tăng dần cho TEN và LUONG.

NOTE: nếu order by 2 thuộc tính thì nó sẽ sắp xếp thuộc tính đầu tiên, nếu thuộc tính đầu tiên có 2 bản ghi giống nhau thì order by tiếp đến thuộc tính thứ 2.

VD như dưới thì order ASC by TEN xong nếu có bản ghi có 2 TEN = NHAU thì order by tiếp đến LUONG

SELECT \* FROM NHANVIEN

ORDER BY TEN, LUONG;

Ví dụ trên trả về kết quả:

+----+----------+-----+-----------+----------+

| ID | TEN |TUOI | DIACHI | LUONG |

+----+----------+-----+-----------+----------+

| 6 | Cao | 22 | HCM | 4500.00 |

| 5 | Huy | 27 | Hatinh | 8500.00 |

| 7 | Lam | 24 | Hanoi | 10000.00 |

| 2 | Loan | 25 | Hanoi | 1500.00 |

| 4 | Manh | 25 | Hue | 6500.00 |

| 3 | Nga | 23 | Hanam | 2000.00 |

| 1 | Thanh | 32 | Haiphong | 2000.00 |

+----+----------+-----+-----------+----------+

### Câu lệnh ORDER BY với mệnh đề WHERE

Bạn có thể dùng mệnh đề WHERE với lệnh sắp xếp trong SQL để chỉ phân loại các hàng đáp ứng điều kiện cụ thể. Điều này có thể hữu ích khi bạn muốn phân loại một tệp dữ liệu phụ trong bảng dựa trên tiêu chí cụ thể.

Ví dụ:

Giả sử bạn đang cố gắng để truy xuất tất cả bảng ghi từ bảng CUSTOMERS mà tuổi của khách khàng là 25 và phân loại chúng theo thứ tự giảm dần dựa theo tên:

SELECT \* FROM CUSTOMERS

WHERE AGE = 25

ORDER BY NAME DESC;

Kết quả:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+-----------+----------+

### Câu lệnh ORDER BY với mệnh đề TOP

Bạn có thể dùng mệnh đề TOP với ORDER BY để hạn chế số lượng hàng bằng cách phân loại nó theo thứ tự tăng hoặc giảm dần.

Cú pháp:

SELECT TOP N column1, column2, ...

FROM table\_name

ORDER BY column\_name1 [ASC | DESC], column\_name2 [ASC | DESC], …

Ví dụ:

Giả sử bạn đang cố truy xuất 4 bản ghi hàng đầu từ bảng CUSTOMERS trên mức lương cơ bản và phân loại chúng theo thứ tự tăng dần dựa theo tên.

SELECT TOP 4 SALARY

FROM CUSTOMERS

ORDER BY NAME;

Kết quả:

+----------+

| SALARY |

+----------+

| 6500.00 |

| 8500.00 |

| 2000.00 |

| 1500.00 |

+----------+

**MỆNH DỀ DISTINCT**

Mệnh đề DISTINCT được dùng để loại bỏ trùng lặp trong bộ kết quả trả về. Mệnh đề DISTINCT chỉ có thể dùng trong lệnh [SELECT](https://quantrimang.com/hoc/lenh-select-trong-sql-server-147319).

Cú pháp mệnh đề DISTINCT

SELECT DISTRINCT “biểu thức”FROM “bảng”[WHERE “điều kiện”];

Tên biến hoặc giá trị biến

**“biểu thức”**: Cột hoặc giá trị tính toán mà bạn muốn lấy

**“bảng”**: Bảng dùng để lấy bản ghi. Phải có ít nhất 1 bảng trong mệnh đề FROM.

**WHERE “điều kiện”**: Tùy chọn. Điều kiện mà bản ghi phải đáp ứng để được chọn.

**Ví dụ - 1 biểu thức**

Ví dụ đơn giản nhất với mệnh đề DISTINCT trong SQL Server chỉ có 1 biểu thức.

SELECT DISTINCT hoFROM nhanvienWHERE nhanvien\_id >= 50;

**MỆNH ĐỀ UNION**

Toán tử UNION được dùng để kết hợp 2 bộ kết quả từ 2 hoặc nhiều lệnh [SELECT](https://quantrimang.com/hoc/lenh-select-trong-sql-server-147319). Nó sẽ xóa các hàng trùng trong các lệnh SELECT này.

Mỗi lệnh SELECT trong toán tử UNION phải có cùng số cột trong bộ kết quả với kiểu dữ liệu tương ứng.

Toán tử UNION ALL chọn các trường từ hai hoặc nhiều bảng tương tự như UNION. Tuy nhiên, khác UNION, UNION ALL không bỏ qua các trường trùng lặp.

UNION và UNION ALL là hai toán tử SQL cơ bản nhất được dùng trong database để kết hợp nhóm kết quả từ nhiều bảng. Khác với toán tử [UNION](https://quantrimang.com/hoc/toan-tu-union-trong-sql-server-148035), toán tử UNION ALL trả về tất cả các hàng từ truy vấn và không xóa các hàng trùng lặp.

### Cú pháp toán tử UNION

SELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien]UNIONSELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien];

### Tên biến hoặc giá trị biến

**bieu\_thuc1, bieu\_thuc2, … bieu\_thucn**: Cột hoặc giá trị tính toán muốn lấy.

**Bang**: Bảng muốn lấy bản ghi. Phải có ít nhất 1 bảng trong mệnh đề [FROM](https://quantrimang.com/hoc/menh-de-from-trong-sql-server-147321).

**WHERE dieu\_kien**: Tùy chọn. Điều kiện phải đáp ứng để bản ghi được chọn.

**Lưu ý:**

* 2 lệnh SELECT phải có cùng số biểu thức
* Số cột tương ứng trong mỗi lệnh SELECT phải có cùng kiểu dữ liệu
* Toán tử UNION xóa các hàng trùng nhau.
* **Ví dụ - trả về một trường thông tin**
* Ví dụ toán tử UNION trả về 1 trường từ nhiều lệnh SELECT (và các trường có cùng kiểu dữ liệu).
* SELECT sanpham\_idFROM sanphamUNIONSELECT sanpham\_idFROM hangtonkho;
* Ở ví dụ trên, nếu có sanpham\_id nào xuất hiện ở cả bảng sanpham và hangtonkho, thì sẽ chỉ còn 1 sanpham\_id xuất hiện trong bộ kết quả. Nếu không muốn loại bỏ bản ghi trùng, hãy dùng toán tử UNION ALL.

## Áp dụng UNION cho nhiều hơn hai truy vấn

Bạn có thể dùng UNION để kết hợp kết quả của nhiều truy vấn nếu cần. Ví dụ một bảng về các loài vật nuôi, bạn có thể làm như sau:

SELECT DogName AS PetName

FROM Dogs

UNION ALL

SELECT CatName

FROM Cats

UNION ALL

SELECT BirdName

FROM Birds;

### **Cú pháp toán tử UNION ALL**

SELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien]UNION ALLSELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien];

### Tên biến hoặc giá trị biến

**bieu\_thuc1, bieu\_thuc2, … bieu\_thucn**

Cột hoặc giá trị tính toán muốn lấy.

**bang**

Bảng muốn lấy bản ghi. Phải có ít nhất 1 bảng trong mệnh đề [FROM](https://quantrimang.com/hoc/menh-de-from-trong-sql-server-147321).

**WHERE dieu\_kien**

Tùy chọn. Điều kiện phải đáp ứng để bản ghi được chọn.

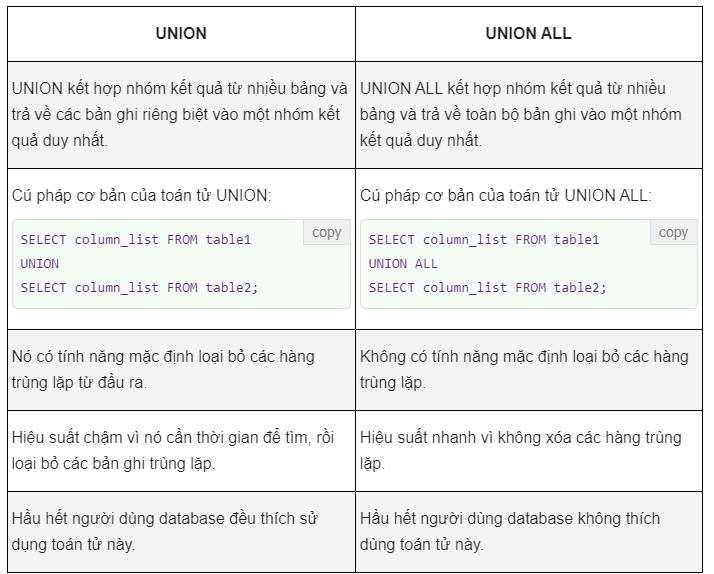
**Lưu ý:**

* 2 lệnh SELECT phải có cùng số biểu thức
* Số cột tương ứng trong mỗi lệnh SELECT phải có cùng kiểu dữ liệu
* Toán tử UNION không xóa các hàng trùng nhau.

**Ví dụ - trả về một trường thông tin**

SELECT sanpham\_idFROM sanphamUNION ALLSELECT sanpham\_idFROM hangtonkho;

Ví dụ này trả về nhiều sanpham\_id trong bộ kết quả nếu chúng có xuất hiện trong cả bảng sanpham và hangtonkho. Nếu muốn xóa các bản trùng lặp, hãy dùng toàn tử UNION.



CÂU LỆNH INTERSECT

Toán tử SQL INTERSECT kết hợp 2 lệnh lựa chọn và trả về chỉ dataset xuất hiện ở cả hai lệnh. Hiểu đơn giản, nó hoạt động như một giao điểm toán học. Về mặt toán học, giao điểm của A và B là dữ liệu chung của cả hai. Vì thế, khi cung cấp hai truy vấn lựa chọn, SQL INTERSECT sẽ chỉ trả về những hàng xuất hiện ở cả hai truy vấn SELECT. Tương tự toán tử UNION trong SQL, ngay cả toán tử INTERSECT cũng phải tuân thủ hai quy tắc để triển khai thành công:

* Số và thư tự của cột ở cả hai truy vấn phải giống nhau.
* Kiểu dữ liệu của cột tương ứng từ cả hai truy vấn lựa chọn phải tương thích với nhau.

### Cú pháp toán tử INTERSECT

SELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien]INTERSECTSELECT bieu\_thuc1, bieu\_thuc2, … bieu\_thucnFROM bang[WHERE dieu\_kien];

### Tên biến hoặc giá trị biến

**bieu\_thuc**: Cột hoặc giá trị mà bạn muốn so sánh giữa 2 lệnh SELECT. Chúng không nhất thiết phải nằm trong cùng 1 trường thông tin ở mỗi lệnh SELECT nhưng các cột tương ứng phải có dữ liệu giống nhau.

**Bang**: Bảng muốn lấy bản ghi từ đó. Phải có ít nhất 1 bảng trong mệnh đề [FROM](https://quantrimang.com/hoc/menh-de-from-trong-sql-server-147321).

**WHERE dieu\_kien**: Tùy chọn. Điều kiện phải đáp ứng để bản ghi được chọn.

**Ví dụ - với 1 biểu thức**

SELECT sanpham\_idFROM sanphamINTERSECT SELECT sanpham\_idFROM hangtonkho;

Ở ví dụ này, nếu sanpham\_id xuất hiện ở cả bảng sanpham và hangtonkho thì nó sẽ nằm trong bộ kết quả của INTERSECT.

Giờ hãy thêm điều kiện [WHERE](https://quantrimang.com/hoc/menh-de-where-trong-sql-server-147325) vào truy vấn này.

SELECT sanpham\_idFROM sanphamWHERE sanpham\_id >= 50INTERSECT SELECT sanpham\_idFROM hangtonkhoWHERE soluong > 0;

Bộ dữ liệu thứ 1 sẽ lọc và trả về các bản ghi nằm trong bảng sanpham và sanpham\_id lớn hơn hoặc bằng 50. Bộ dữ liệu thứ 2 sẽ lọc lấy từ bảng hangtonkho nếu số lượng lớn hơn 0.

**Ví dụ - dùng ORDER BY**

Dùng mệnh đề [ORDER BY](https://quantrimang.com/hoc/menh-de-order-by-trong-sql-server-147326) với truy vấn INTERSECT để sắp xếp kết quả.

SELECT nhacung\_id, nhacung\_tenFROM nhacungWHERE nhacung\_id > 500INTERSECT SELECT congty\_id, congty\_tenFROM congtyWHERE congty\_ten in (‘Apple’, ‘Microsoft’, ‘SQL Server’)ORDER BY 2;

Vì tên cột ở 2 lệnh SELECT khác nhau nên sẽ dễ hơn nếu tham chiếu tới cột trong mệnh đề ORDER BY bằng vị trí của chúng trong bộ kết quả. Ở ví dụ trên, ta lọc kết quả nhacung\_ten / congty\_ten theo thứ tự tăng dần qua cụm từ ORDER BY 2.

Vì nhacung\_ten / congty\_ten đứng thứ 2 trong bộ kết quả.

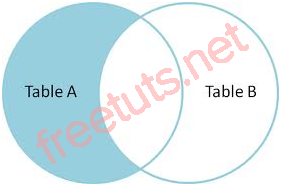
**NOTE: ORDER BY 2 tức là lấy sắp xếp theo cột thứ 2 trong các cột ở SELECT**

**TOÁN TỬ MINUS**

Trong Oracle toán tử MINUS có tác dụng ngược lại với INTERSECT.

Với toán tử INTERSECT sẽ hợp kết quả của 2 lệnh SELECT và chọn ra những record nào trùng nhau thì MINUS sẽ hợp lại và chọn ra những record chỉ có ở câu SELECT đầu tiên mà không có ở câu SELECT thứ hai.

Bạn hãy xem hình sau đây sẽ thấy rõ ràng hơn, phần màu xanh nhạt là phần được chọn bởi MINUS.



**CÚ PHÁP:**

SELECT column1 , column2 , ... columnN

FROM table\_name

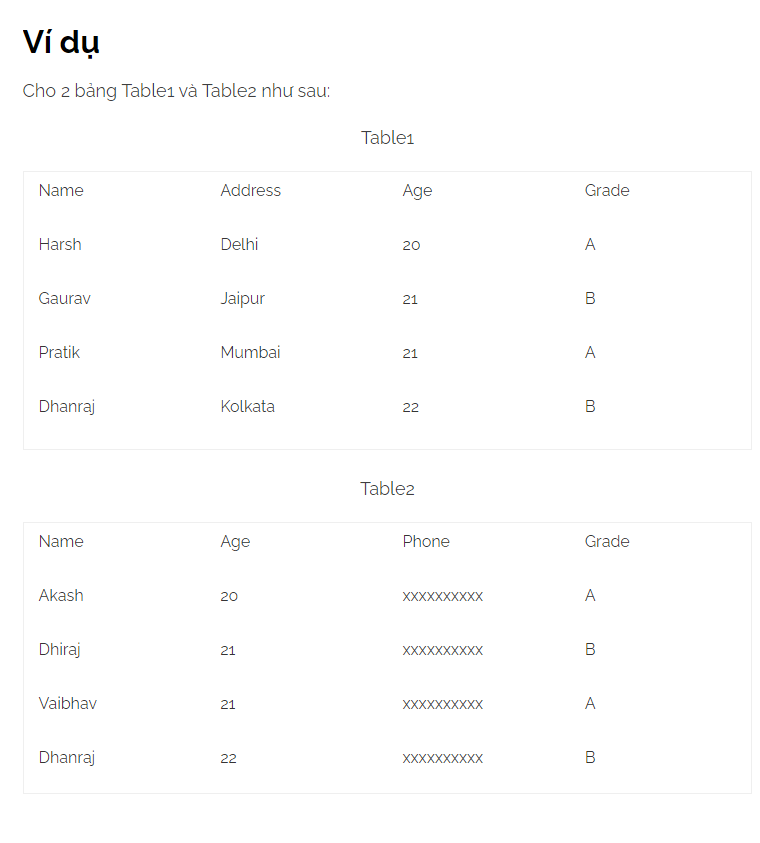
WHERE condition

MINUS

SELECT column1 , column2 , ... columnN

FROM table\_name

WHERE condition;

****

Ta dung câu lệnh:

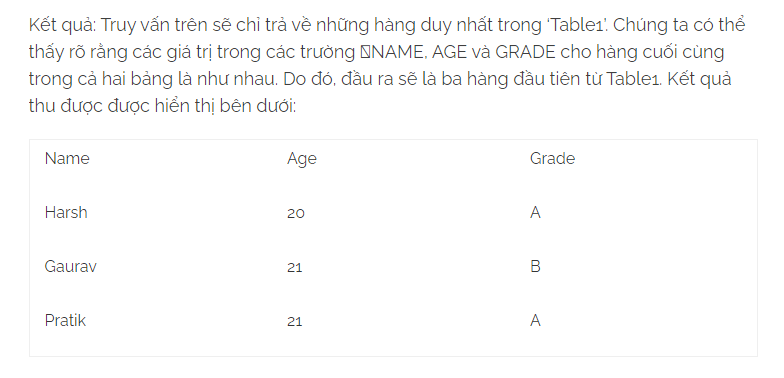
SELECT NAME, AGE , GRADE

FROM Table1

MINUS

SELECT NAME, AGE, GRADE

FROM Table2

****

# **TOÁN TỬ LIKE**

SQL Server “LIKE” là một toán tử logic xác định một chuỗi ký tự có khớp với mẫu được chỉ định hay không. Một mẫu có thể bao gồm các ký tự thông thường và ký tự đại diện. Toán tử LIKE được dùng trong mệnh đề WHERE của lệnh SELECT, UPDATE và DELETE để lọc các hàng dựa trên cách viết kết hợp mẫu.

Điều kiện LIKE trong [SQL Server](https://quantrimang.com/hoc/sql-server) (Transact-SQL) cho phép dùng wildcard (các kí hiệu đại diện) trong mệnh đề [WHERE](https://quantrimang.com/hoc/menh-de-where-trong-sql-server-147325) ở các lệnh [SELECT](https://quantrimang.com/hoc/lenh-select-trong-sql-server-147319), INSERT, UPDATE và DELETE, dùng để đối sánh mẫu.

### Cú pháp điều kiện LIKE

“biểu thức” LIKE “mẫu” [ ESCAPE ‘Escape\_Character’ ]

### Tên biến hoặc giá trị biến

**biểu thức**: Biểu thức kí tự như cột hoặc trường thông tin.

**mẫu**: BIểu thức kí tự có chứa đối chiếu mẫu. Các mẫu có thể chọn từ bảng dưới đây.

## Cú pháp lệnh LIKE trong SQL

Cú pháp cơ bản của lệnh LIKE như sau:

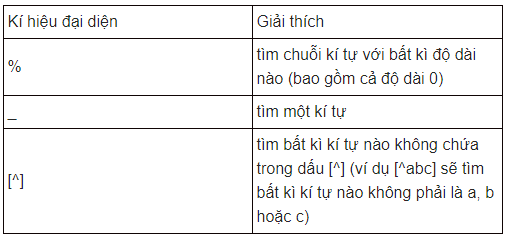
SELECT cot1, cot2,.... cotn

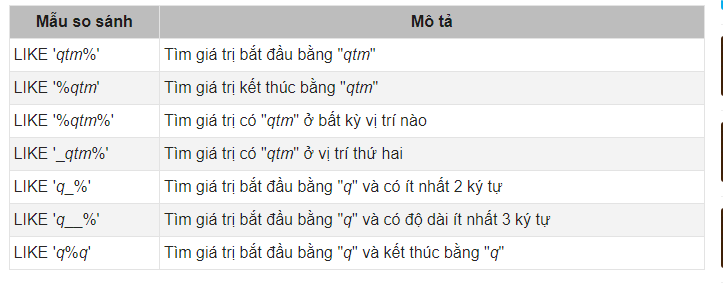
FROM ten\_bang

WHERE ten\_cot LIKE [mẫu so sánh];

**Mẹo**: Bạn cũng có thể kết hợp bất kỳ số lượng điều kiện nào bằng cách sử dụng toán tử AND hoặc OR.

Dưới đây là một số ví dụ hiển thị các toán tử LIKE khác nhau, bằng cách kết hợp các ký tự đại diện '%' và '\_':





## Ví dụ về LIKE trong SQL

**SQL Like với nhiều giá trị**

Bạn có thể dùng toán tử LIKE với nhiều mẫu chuỗi để chọn hàng bằng cách dùng toán tử OR. Ví dụ:

SELECT \*

FROM Customers

WHERE last\_name LIKE 'R%t' OR last\_name LIKE '%e';

Lệnh SQL này chọn khách hàng có last\_name bắt đầu bằng chữ R và kết thúc bằng t, hoặc khách hàng có last\_name kết thúc bằng e.

**Toán tử SQL NOT LIKE**

Bạn cũng có thể đảo ngược hoạt động của toán tử LIKE và bỏ qua nhóm kết quả khớp với mẫu chuỗi được đưa ra bằng cách dùng toán tử NOT. Ví dụ:

SELECT \*

FROM Customers

WHERE country NOT LIKE 'USA';

Tại đây, lệnh SQL chọn toàn bộ khách hàng, ngoại trừ ở Mỹ.

# **Hàm CONCAT trong SQL Server**

## Mô tả

**Hàm CONCAT** trong SQL Server được sử dụng để nối hai hay nhiều chuỗi thành một chuỗi lớn duy nhất.

## Cú pháp

Để chạy hàm CONCAT trong SQL Server, ta sử dụng cú pháp như sau:

CONCAT(chuoi1, chuoi2,... chuoi\_n)

**Tham số**:

* chuoi1, chuoi2,... chuoi\_n: các chuỗi muốn nối với nhau.

**Lưu ý**:

* Input hàm CONCAT có độ dài tối đa là 255 chuỗi, tối thiểu là 2 chuỗi. Nếu chỉ truyền 1 chuỗi đầu vào thì hàm CONCAT sẽ phát sinh lỗi.
* Nếu truyền các chuỗi không có ký tự, CONCAT sẽ tự động chuyển đổi các giá trị đó thành các chuỗi trước khi nối.
* Hàm CONCAT chuyển đổi NULL thành một chuỗi rỗng ở kiểu dữ liệu VARCHAR(1).
* Hàm CONCAT chỉ có thể được sử dụng trong các phiên bản sau của SQL Server: SQL Server 2017, SQL Server 2016, SQL Server 2014, SQL Server 2012.

## Ví dụ

Hãy xem và khám phá một số ví dụ về hàm CONCAT trong SQL Server.

SELECT CONCAT('Quantrimang', '.com');Result: 'Quantrimang.com'SELECT CONCAT('Quan', 'tri', 'mang', '.com');Result: 'Quantrimang.com'SELECT CONCAT('Quan ', 'Tri ', 'Mang ');Result: 'Quan Tri Mang'

Khi kết hợp các chuỗi với nhau, bạn có thể thêm các ký tự khoảng trắng để phân tách các giá trị được nối của mình để kết quả dễ đọc hơn.

SELECT CONCAT('Orange', ' ', 'Peach', ' ', 'Apple');Result: 'Orange Peach Apple'

Trong ví dụ này, Quantrimang đã sử dụng tham số thứ 2 và thứ 4 trong hàm CONCAT để thêm ký tự khoảng trắng giữa các giá trị Orange, Peach và Apple. Điều này sẽ giúp các chuỗi không bị gắn liền lại với nhau.

# **Unix\_date JAVA**

Trong SQL, thuật ngữ "Unix date" thường được sử dụng để đề cập đến định dạng thời gian Unix timestamp. Thời gian Unix là một cách biểu diễn thời gian dưới dạng số giây tính từ thời điểm 00:00:00 UTC ngày 1 tháng 1 năm 1970. Đây là một cách phổ biến để lưu trữ và truyền thời gian trong các hệ thống máy tính và cơ sở dữ liệu.

**UNIX\_TIMESTAMP(), UNIX\_TIMESTAMP(date)**

Hàm UNIX\_TIMESTAMP() trong SQL có 2 dạng. Nếu được gọi với không có đối số, hàm này trả về số giây theo Unix timestamp (kể từ '1970-01-01 00:00:00' UTC) dưới dạng số nguyên không dấu (unsigned integer). Nếu UNIX\_TIMESTAMP() được gọi với đối số date, nó trả về giá trị của đối số là giây (kể từ '1970-01-01 00:00:00' UTC). Đối số date có thể là một chuỗi DATE, một chuỗi DATETIME, TIMESTAMP hoặc một số ở định dạng YYMMDD hoặc YYYYMMDD.

Cách converte từ Unixtimestamp sang Datetime:

***-- Chuyển đổi Unix timestamp sang datetime***

**SELECT FROM\_UNIXTIME(1642246200) AS converted\_datetime;**

***-- Chuyển đổi Unix timestamp sang timestamp***

**-- Chuyển đổi Unix timestamp sang timestamp   
SELECT UNIX\_TIMESTAMP(FROM\_UNIXTIME(1642246200)) AS converted\_timestamp;**

**----------------------------------------------------------------------------------------------------------------**

**Locking**

Khóa bảng (Table Locking):

**Lock** là một flag được liên kết với một table, **MySQL** cho phép một client session có thể nhận được một table lock một cách rõ ràng nhằm mục đích ngăn không cho các session khác truy cập thao tác dữ liệu vào cùng một bảng trong 1 thời gian cụ thể.

Một client session chỉ có thể lấy hoặc giải phóng các table lock cho chính nó. Tương tự thì một client session không thể giải phóng table lock cho các client session khác.

### Câu lệnh LOCK TABLES

Dưới đây là câu lệnh để thực hiện khóa một bảng :

LOCK TABLES table\_name [READ | WRITE]

Trong cú pháp này, ta chỉ định tên của bảng muốn khóa sau từ khóa LOCK TABLES. Ngoài ra, chúng ta còn có thể chỉ định loại khóa là READ hoặc WRITE

MySQL cho phép ta khóa nhiều bảng bằng cách chỉ định danh sách các bảng được phân tách bằng dấu phẩy với cú pháp như sau:

LOCK TABLES table\_name1 [READ | WRITE],

table\_name2 [READ | WRITE],

... ;

### UNLOCK TABLES Câu lệnh MySQL

Để mở khóa cho một bảng, bạn sử dụng câu lệnh sau:

UNLOCK TABLES;

### READ Locks

**READ Lock** có các tính năng sau:

* **READ Lock** cho một bảng có thể được sử dụng lại bởi nhiều session đồng thời. Ngoài ra, các session khác có thể đọc dữ liệu từ bảng mà không cần lấy khóa.
* Session hiện tại thực hiện READ Lock chỉ có thể đọc dữ liệu từ bảng nhưng không thể ghi, các session khác không thể ghi dữ liệu vào bảng cho đến khi READ Lock được giải phóng. Các hoạt động ghi từ một session sẽ được đưa vào trạng thái chờ cho đến khi READ Lock được giải phóng.
* Nếu session kết thúc bình thường hoặc bất thường, MySQL sẽ giải phóng tất cả các khóa một cách ngầm định. Tính năng này cũng có liên quan đến WRITE Lock.

### Write Locks

**WRITE lock** có các tính năng sau:

* Chỉ có session hiện tại đang locking mới có thể đọc và ghi dữ liệu từ bảng
* Các session khác không thể đọc và ghi dữ liệu vào bảng cho đến khi WRITE lock được giải phóng.

**TRANSACTION TRONG SQL**

**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).

Trong SQL, có các lệnh sau được 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.
* Các lệnh điều khiển Transaction chỉ được sử dụng với các lệnh thao tác dữ liệu như INSERT, UPDATE và DELETE. Tuy nhiên chúng không thể được sử dụng trong lệnh CREATE TABLE hoặc DROP TABLE vì các hoạt động này được tự động xác định trong cơ sở dữ liệu.

# Stored Procerdure

# Các thiết kế bảng

# Cài đặt

# JDBC

# Hibernate