|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Kiểu dữ liệu Numeric | |  |  | | --- | --- | | INT | Một số nguyên với kích cỡ thông thường, có thể là signed hoặc unsigned. Nếu có dấu, thì dãy giá trị có thể là từ -2147483648 tới 2147483647, nếu không dấu thì dãy giá trị là từ 0 tới 4294967295. Bạn có thể xác định một độ rộng lên tới 11 chữ số. | | TINYINT | Một số nguyên với kích cỡ rất nhỏ, có thể là signed hoặc unsigned. Nếu có dấu, thì dãy giá trị có thể là từ -128 tới 127, nếu không dấu thì dãy giá trị là từ 0 tới 255. Bạn có thể xác định một độ rộng lên tới 4 chữ số. | | MEDIUMINT | Một số nguyên với kích cỡ trung bình, có thể là signed hoặc unsigned. Nếu có dấu, thì dãy giá trị có thể là từ -8388608 tới 8388607, nếu không dấu thì dãy giá trị là từ 0 tới 16777215. Bạn có thể xác định một độ rộng lên tới 9 chữ số. | | BIGINT | Một số nguyên với kích cỡ lớn, có thể là signed hoặc unsigned. Nếu có dấu, thì dãy giá trị có thể là từ -9223372036854775808 tới 9223372036854775807, nếu không dấu thì dãy giá trị là từ 0 tới 18446744073709551615. Bạn có thể xác định một độ rộng lên tới 20 chữ số. | | FLOAT(M, D) | Một số thực dấu chấm động không dấu. Bạn có thể định nghĩa độ dài hiển thị (M) và số vị trí sau dấy phảy (D). Điều này là không bắt buộc và sẽ có mặc định là 10,2: với 2 là số vị trí sau dấu phảy và 10 là số chữ số (bao gồm các phần thập phân). Phần thập phân có thể lên tới 24 vị trí sau dấu phảy đối với một số FLOAT. | | DOUBLE(M, D) | Một số thực dấu chấm động không dấu. Bạn có thể định nghĩa độ dài hiển thị (M) và số vị trí sau dấy phảy (D). Điều này là không bắt buộc và sẽ có mặc định là 16,4: với 4 là số vị trí sau dấu phảy và 16 là số chữ số (bao gồm các phần thập phân). Phần thập phân có thể lên tới 53 vị trí sau dấu phảy đối với một số DOUBLE. REAL là đồng nghĩa với DOUBLE. | | DECIMAL(M, D) | Một kiểu khác của dấu chấm động không dấu. Mỗi chữ số thập phân chiếm 1 byte. Việc định nghĩa độ dài hiển thị (M) và số vị trí sau dấy phảy (D) là bắt buộc. NUMERIC là một từ đồng nghĩa cho DECIMAL. | |
| **Kiểu dữ liệu Date và Time trong MySQL** | |  |  | | --- | --- | | DATE | Một date trong định dạng YYYY-MM-DD, giữa 1000-01-01 và 9999-12-31. Ví dụ, ngày 25 tháng 12 năm 2015 sẽ được lưu ở dạng 2015-12-25. | | DATETIME | Một tổ hợp Date và Time trong định dạng YYYY-MM-DD HH:MM:SS, giữa 1000-01-01 00:00:00 và 9999-12-31 23:59:59. Ví dụ, 3:30 chiều ngày 25 tháng 12, năm 2015 sẽ được lưu ở dạng 2015-12-25 15:30:00. | | TIMESTAMP | Một Timestamp từ giữa nửa đêm ngày 1/1/1970 và 2037. Trông khá giống với định dạng DATETIME trước, khác biệt ở chỗ không có dấu gạch nối giữa các số. Ví dụ, 3:30 chiều ngày 25 tháng 12, năm 2015 sẽ được lưu dưới dạng 20151225153000 ( YYYYMMDDHHMMSS ). | | TIME | Lưu time trong định dạng HH:MM:SS. | | YEAR(M) | Lưu 1 năm trong định dạng 2 chữ số hoặc 4 chữ số. Nếu độ dài được xác định là 2 (ví dụ: YEAR(2)), YEAR có thể từ 1970 tới 2069 (70 tới 69). Nếu độ dài được xác định là 4, YEAR có thể từ 1901 tới 2155. Độ dài mặc định là 4. | |
| **Kiểu dữ liệu chuỗi trong MySQL** | |  |  | | --- | --- | | CHAR(M) | Một chuỗi có độ dài cố định có độ dài từ 1 tới 255 ký tự (ví dụ CHAR(5)). Nếu giá trị thật của một trường kiểu Char không bằng với độ dài khai báo thì phần thiếu bên phải của nó sẽ được thêm bằng các kí tự trắng một cách tự động. Định nghĩa độ dài là không bắt buộc, giá trị mặc định là 1. | | VARCHAR(M) | Dữ liệu kiểu chuỗi có độ dài thay đổi, có độ dài từ 1 đến 255 kí tự (ví dụ Varchar(24)). Bạn phải định nghĩa độ dài khi tạo một trường VARCHAR.  **VARCHAR**: Là kiểu dữ liệu chuỗi biến đổi. Lưu trữ dữ liệu theo bộ mã ký tự của hệ thống hoặc bộ mã được xác định.  **VARCHAR**: Là kiểu dữ liệu chuỗi biến đổi. Lưu trữ dữ liệu theo bộ mã ký tự của hệ thống hoặc bộ mã được xác định. | | DATE |  | | BLOB hoặc TEXT | Trường kiểu này có độ dài tối đa 65535 kí tự. BLOBs là viết tắt của "Binary Large Objects", và được sử dụng để lưu trữ một lượng lớn dữ liệu nhị phân như các bức ảnh hoặc các loại tập tin khác. Với TEXT, trường cũng lưu trữ được một lượng lớn dữ liệu. Điểm khác nhau giữa chúng là: khi sắp xếp và so sánh dữ liệu đã lưu trữ thì với BLOBs là phân biệt kiểu chữ, còn với TEXT là không phân biệt kiểu chữ. Bạn không phải xác định độ dài với BLOBs hoặc TEXT. | | TINYBLOB hoặc TINYTEXT | Một cột BLOB hoặc TEXT với độ dài tối đa là 255 ký tự. Bạn không cần xác định độ dài với TINYBLOB hoặc TINYTEXT. | | MEDIUMBLOB hoặc MEDIUMTEXT | Một cột BLOB hoặc TEXT với độ dài tối đa là 16777215 ký tự. Bạn không cần xác định độ dài với MEDIUMBLOB hoặc MEDIUMTEXT. | | LONGBLOB hoặc LONGTEXT | Một cột BLOB hoặc TEXT với độ dài tối đa là 4294967295 ký tự. Bạn không cần xác định độ dài với LONGBLOB hoặc LONGTEXT. | | 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). Ví dụ, nếu ta muốn một trường nào đó chỉ nhận một trong các giá trị "A" hoặc "B" hoặc "C" thì ta phải định nghĩa kiểu ENUM cho nó như sau: ENUM ('A', 'B', 'C'). Và chỉ có các giá trị này (hoặc NULL) có thể xuất hiện trong trường đó | |
| **Mối quan hệ 1 - n** | Đối với mối quan hệ 1 - nhiều chúng ta sẽ bổ sung khóa chính của bảng 1 làm khóa ngoại của bảng nhiều.  Ví dụ thực thể PhongBan và thực thể NhanVien có mối quan hệ 1 - nhiều (1 phòng ban có nhiều nhân viên) thì khi đó bảng NhanVien sẽ có thêm một khóa ngoại là MaPHG (đây là khóa chính Bảng phòng ban)    Từ hình trên ta sẽ thu được 2 bảng:  NHANVIEN(**MaNV**, HoNV, TenNV, GT, NS, Luong, DiaChi, MaPHG)  PHONGBAN(**MaPHG**, TenPHG) |
| **Mối quan hệ 1 - 1** | Đối với mối quan hệ 1 - 1 chúng ta sẽ bổ sung khóa chính của bên này làm khóa ngoại của bảng bên kia  Ví dụ thực thể NhanVien có mối quan hệ 1 - 1 với thực thể MayTinh (Mỗi nhân viên chỉ có 1 máy tính) thì chúng ta sẽ bổ sung thêm MaNV làm khóa ngoại của bảng MayTinh    Từ hình trên ta sẽ thu được kết quả là 2 bảng:  NHANVIEN(**MaNV**, HoNV, TenNV, GT, NS, Luong, DiaChi)  MAYTINH(**SoMay**, TenMay, MaNV) |
| **Mối quan hệ nhiều - nhiều** | Đối với mối quan n - m để chuyển đổi từ ERD sang mô hình dữ liệu quan hệ sẽ phải làm các bước như sau:  Bước 1: Tạo ra bảng trung gian được sinh ra từ mối quan hệ n - m với chứa khóa chính của 2 bảng chuyển hóa thành khóa ngoại của bảng trung gian cùng với các thuộc tính của mối quan hệ  Bước 2: Xác định khóa trung gian như sau:  Nếu cặp khóa ngoại đủ điều kiện để đảm bảo thành khóa chính thì sẽ chọn cặp khóa ngoại thành khóa chính  Nếu cặp khóa ngoại không đủ điều kiện để đảm bảo tính duy nhất và not null thì phải tạo thêm một trường mới là mã của bảng trung gian làm khóa chính  Ví dụ thực thể NhanVien có mối quan hệ n - m với thực thể DuAn (Một nhân viên có thể làm nhiều dự án và một dự án có thể do nhiều nhân viên thực hiện) thì chúng ta sẽ làm như sau:    Từ hình trên ta thu được các bảng như sau:  NHANVIEN(**MaNV**, HoNV, TenNV, GT, NS, Luong, DiaChi)  DEAN(**MaDA**, TenDA, DDiem\_DA)  PHANCONG(MaNV, MaDA, ThoiGian) |
| **Thuộc tính đa trị** | Đối với các thuộc tính đa trị của một thực thể chúng ta sẽ chuyển đổi thành một quan hệ mới (bảng mới) có khóa chính bao gồm 2 thuộc tính là thuộc tính đa trị và thuộc tính khóa của thực thể đồng thời loại bỏ thuộc tính đó khỏi thực thể cũ.  Ví dụ: Thực thể NhanVien có thuộc tính đa trị là bằng cấp khi đó chúng ta chuyển đổi sang mô hình dữ liệu quan hệ sẽ sinh thêm một bảng có tên là NV\_BangCap với khóa chính là MaNV và BangCap và đồng thời xóa bỏ trường BangCap khỏi bảng NhanVien    Từ hình trên chúng ta thu được 2 bảng:  NHANVIEN(MaNV, HoNV, TenNV, GT, NS, Luong, DiaChi)  NV\_BANGCAP(MaNV, BangCap) |
| **Khoá chính ( primary key )** | - Khóa chính (Primary Key, hay ràng buộc khóa chính) được sử dụng để định danh duy nhất mỗi bản ghi (dòng) trong bảng của cơ sở dữ liệu.  - Ngoài ra, nó còn dùng để thiết lập quan hệ 1-n (hay ràng buộc tham chiếu) giữa hai bảng trong cơ sở dữ liệu.  - Dữ liệu (value) của field khóa chính phải có tính duy nhất, và không chứa các giá trị Null.  - Mỗi bảng chỉ có một khóa chính  - Khóa chính có thể tạo ra dựa trên một cột hoặc nhiều cột (được gọi là khoá chính tổ hợp) của table.    Hoặc |
| **Khoá ngoại ( Foreign key )** | - Khoá ngoại (foreign key - viết ngắn gọn là FK) là cơ chế để tạo liên kết giữa các bảng trong cùng một cơ sở dữ liệu. Chẳng hạn, trong CSDL quản lý bán hàng, bảng customer và bảng order có thể có khoá ngoại để liên kết với nhau.  - Khoá ngoại được đặt trên một cột của bảng này và tham chiếu đến khoá chính của một bảng khác. Chẳng hạn, khoá ngoại được đặt trên cột customer\_id của bảng order và tham chiếu đến cột id của bảng customer.  - Kiểu dữ liệu của khoá ngoại và khoá chính mà nó tham chiếu phải giống nhau. Thông thường thì các bảng đều sử dụng kiểu INTEGER cho khoá chính. |
| **Constraint ( ràng buộc )** | | Loại ràng buộc | Ý nghĩa | | --- | --- | | NOT NULL | Sử dụng để đảm bảo dữ liệu của cột không được nhận giá trị NULL | | DEFAULT | Gán giá trị mặc định trong trường hợp dữ liệu của cột không được nhập vào hay không được xác định. | | UNIQUE | Sử dụng để đảm bảo dữ liệu của cột là duy nhất, không trùng lặp giá trị trên cùng 1 cột. | | PRIMARY KEY (Khóa chính) | Dùng để thiết lập khóa chính trên bảng, xác định giá trị trên tập các cột làm khóa chính phải là duy nhất, không được trùng lặp. Việc khai báo ràng buộc khóa chính yêu cầu các cột phải NOT NULL. | | FOREIGN KEY (Khóa ngoại) | Dùng để thiết lập khóa ngoại trên bảng, tham chiếu đến bảng khác thông qua giá trị của cột được liên kết. Giá trị của cột được liên kết phải là duy nhất trong bảng kia. | | CHECK | Bảo đảm tất cả giá trị trong cột thỏa mãn điều kiện nào đó. Đây là hình thức sử dụng phổ biến để kiểm tra tính hợp lệ của dữ liệu (validate data) | |
| **Alter ( Thay đổi )** | Câu lệnh ALTER trong SQL được sử dụng để thay đổi cấu trúc của bảng hoặc đối tượng khác trong cơ sở dữ liệu. Với câu lệnh ALTER, bạn có thể thêm, sửa đổi hoặc xóa các cột, ràng buộc hoặc đối tượng khác của bảng. Các tác vụ thường được thực hiện bởi câu lệnh ALTER bao gồm thêm hoặc xóa cột, thêm hoặc xóa ràng buộc, thay đổi kiểu dữ liệu của cột và nhiều tác vụ khác nữa tùy thuộc vào loại đối tượng muốn thay đổi.  - Thêm cột: ALTER TABLE table\_name ADD COLUMN column\_name data\_type;  - Sửa đổi kiểu dữ liệu cột: ALTER TABLE table\_name MODIFY COLUMN column\_name data\_type;  - Xóa cột: ALTER TABLE table\_name DROP COLUMN column\_name;  - Thay đổi tên bảng: ALTER TABLE old\_table\_name RENAME TO new\_table\_name;  - Thay đổi tên cột: ALTER TABLE table\_name RENAME COLUMN old\_column\_name TO new\_column\_name;  - Thêm ràng buộc khóa ngoại: ALTER TABLE child\_table ADD CONSTRAINT foreign\_key\_constraint\_name FOREIGN KEY (child\_column) REFERENCES parent\_table(parent\_column);  - Xóa ràng buộc khóa ngoại: ALTER TABLE child\_table DROP FOREIGN KEY foreign\_key\_constraint\_name; |
| **Từ khoá** | - Add : thêm  - Column: cột trong bảng  - drop: xoá  - modify: sửa  - rename: sửa tên  - add constraint : thêm ràng buộc  - drop constraint: xoá ràng buộc |
| **Insert into ( chèn vào )** | Câu lệnh INSERT INTO trong SQL được sử dụng để thêm các hàng dữ liệu mới vào một bảng trong cơ sở dữ liệu.  Cú pháp  Có hai cú pháp cơ bản của câu lệnh INSERT INTO thể hiện dưới đây.  INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)  VALUES (value1, value2, value3,...valueN);  Ở đây, column1, column2, column3, ... columnN là tên của các cột trong bảng muốn chèn dữ liệu.  Chúng ta có thể không cần phải chỉ rõ tên cột trong truy vấn SQL nếu đang thêm các giá trị cho tất cả các cột của bảng. Nhưng hãy chắc chắn thứ tự của các giá trị theo thứ tự như các cột trong bảng. |
| **Select ( chọn )** | Câu lệnh SELECT trong SQL được sử dụng để lấy dữ liệu từ một bảng cơ sở dữ liệu trả về dữ liệu này dưới dạng một bảng kết quả. Các bảng kết quả này được gọi là tập kết quả (result-sets).  Cú pháp  Cú pháp cơ bản của câu lệnh SELECT trong SQL như sau:  SELECT column1, column2, columnN FROM table\_name;  Ở đây, column1, column2... là các trường của một bảng có các giá trị bạn muốn lấy ra.  Nếu muốn lấy tất cả các cột có trong bảng sử dụng câu lệnh sau:  SELECT \* FROM table\_name; |
| **Where ( ở đâu )** | Mệnh đề WHERE trong SQL được sử dụng để chỉ định một điều kiện trong khi lấy dữ liệu từ một bảng hoặc bằng cách join với nhiều bảng. Nếu điều kiện thỏa mãn, thì nó trả về một giá trị cụ thể từ bảng.  Cú pháp  Cú pháp cơ bản của câu lệnh SELECT với mệnh đề WHERE:  SELECT column1, column2, columnN  FROM table\_name  WHERE [condition] |
| **Order by ( đặt bởi )** | Mệnh đề ORDER BY trong SQL được sử dụng để sắp xếp dữ liệu theo thứ tự tăng dần hoặc giảm dần, dựa trên một hoặc nhiều cột. ASC được sử dụng để sắp xếp tăng dần và DESC được sử dụng để sắp xếp giảm dần.  Cú pháp  Cú pháp cơ bản của mệnh đề ORDER BY như sau:  SELECT column-list  FROM table\_name  [WHERE condition]  [ORDER BY column1, column2, .. columnN] [ASC | DESC];    \* nếu dùng order by sắp xếp theo tên theo thứ tự giảm dần mà nếu có tên trùng nhau thì nó sẽ sắp xếp theo salary. |
| **Group by** | Trong MySQL, GROUP BY là một mệnh đề được sử dụng trong câu lệnh SELECT để nhóm các bản ghi có cùng giá trị trong một hoặc nhiều cột. Khi sử dụng GROUP BY, các bản ghi sẽ được nhóm lại theo giá trị của cột được chỉ định, và kết quả trả về sẽ là một bản ghi cho mỗi nhóm, chứ không phải là tất cả các bản ghi trong bảng.  Cú pháp của GROUP BY như sau:    Trong đó, column1, column2, ..., columnN là các cột được chọn để hiển thị, và aggregate\_function là một hàm tổng hợp như COUNT, SUM, AVG, MAX hoặc MIN, được sử dụng để tính toán giá trị của các cột.  Ví dụ, để tính tổng lương của mỗi phòng ban trong bảng employees, ta có thể sử dụng câu lệnh sau:    Câu lệnh này sẽ nhóm các bản ghi theo cột department và tính tổng lương của mỗi phòng ban. Kết quả trả về sẽ là một bản ghi cho mỗi phòng ban, chứ không phải là tất cả các bản ghi trong bảng. |
| **Having** | HAVING là mệnh đề trong ngôn ngữ truy vấn SQL (Structured Query Language) được sử dụng sau mệnh đề GROUP BY để lọc các bản ghi được nhóm theo các tiêu chí đã xác định.  Một số điểm cần lưu ý về mệnh đề HAVING:  Nó được sử dụng để lọc các bản ghi được nhóm trong câu lệnh SELECT.  HAVING sử dụng các toán tử so sánh (như =, <, >, <=, >=, <>), các toán tử logic (AND, OR) và các hàm tổng hợp (như COUNT, SUM, AVG, MAX, MIN, GROUP\_CONCAT).  HAVING có thể được sử dụng mà không cần GROUP BY, tuy nhiên khi sử dụng cùng GROUP BY, HAVING sẽ được thực hiện sau GROUP BY.  HAVING sẽ lọc các bản ghi đã được nhóm với điều kiện được chỉ định. Các bản ghi không thỏa điều kiện sẽ bị loại bỏ.  Ví dụ: Giả sử có bảng students với các trường name, age và score, ta muốn lọc ra danh sách sinh viên có điểm trung bình lớn hơn 8, ta có thể sử dụng câu lệnh sau:    Lưu ý rằng ở đây ta sử dụng mệnh đề GROUP BY để nhóm các bản ghi theo tên sinh viên, sau đó sử dụng HAVING để lọc ra các sinh viên có điểm trung bình lớn hơn 8. |
| **Toán tử AND** | Toán tử AND là một toán tử logic kết hợp hai hay nhiều biểu thức Boolean và trả về true khi và chỉ khi cả hai biểu thức được đánh giá là true. Toán tử AND trả về false nếu một trong hai biểu thức kết hợp được đánh giá là sai.    Trong cú pháp trên:  WHERE, AND là các từ khóa  boolean\_expression\_1, boolean\_expression\_2 là các biểu thức trả về kiểu boolean. |
| **Dấu ( \* ) trong select** | Dấu sao (\*) trong SELECT được sử dụng để lấy tất cả các trường của bảng "Class" trong kết quả. Lấy tất cả các cột trong cái table. |
| **Toán tử OR** | Toán tử OR kết hợp hai hay nhiều biểu thức dạng boolean, chỉ cần một trong các điều kiện kết hợp là đúng thì kết quả là true. Vậy để sự kết hợp giữa các biểu thức trả về là sai thì tất cả các biểu thức kết hợp đều phải trả về sai.    Trong cú pháp trên:  WHERE, OR là các từ khóa  boolean\_expression\_1 và boolean\_expression\_2 là các biểu thức boolean có thể trả về true, false, hoặc NULL. |
| **Câu lệnh Join** | Trong MySQL, JOIN được sử dụng để kết hợp dữ liệu từ hai hoặc nhiều bảng trong cơ sở dữ liệu dựa trên các điều kiện liên quan đến các cột của các bảng đó. Câu lệnh JOIN thường đi kèm với một số loại khác nhau như INNER JOIN, LEFT JOIN, RIGHT JOIN và FULL OUTER JOIN.  Các loại join phổ biến trong MySQL bao gồm:   1. Inner join: kết hợp các dòng từ hai bảng với điều kiện phù hợp giữa các cột trong bảng. 2. Left join: kết hợp tất cả các dòng từ bảng bên trái và các dòng phù hợp từ bảng bên phải. 3. Right join: kết hợp tất cả các dòng từ bảng bên phải và các dòng phù hợp từ bảng bên trái. 4. Full outer join: kết hợp tất cả các dòng từ cả hai bảng, bao gồm cả các dòng không phù hợp với điều kiện kết nối. 5. Self join: kết hợp dữ liệu từ một bảng với chính nó, tạo ra một bảng ảo được gọi là bảng tự kết nối (self-join table). 6. Cross join: kết hợp mỗi dòng trong bảng bên trái với tất cả các dòng trong bảng bên phải. |
| **Excluding join** | Excluding join không phải là một loại join chính thức trong MySQL, mà thực chất là một thuật toán được sử dụng để tìm các bản ghi không có trong hai bảng. Các bản ghi này được tìm ra bằng cách sử dụng CÁC JOIN và chỉ lấy các bản ghi trong bảng bên trái mà không có kết hợp trong bảng bên phải. |
| **UNION** | các kết quả của các câu lệnh SELECT được kết hợp với nhau để tạo ra một kết quả duy nhất. Kết quả cuối cùng sẽ chứa tất cả các bản ghi từ các câu lệnh SELECT riêng biệt, với các bản ghi trùng lặp được loại bỏ. |
| **UNION ALL** |  |