Bổ sung về Ràng buộc trên Bảng

- Có thể chỉ định ràng buộc khi bảng được tạo bằng **CREATE TABLE** hoặc sau khi bảng được tạo bằng **ALTER TABLE**
- Nếu có bất kỳ vi phạm nào giữa ràng buộc và hành động dữ liệu, hành động sẽ bị hủy bỏ.
- **CHECK - Đảm bảo rằng các giá trị trong một cột thỏa mãn một điều kiện cụ thể**
- **DEFAULT - Đặt giá trị mặc định cho một cột nếu không có giá trị nào được chỉ định**
- **CREATE INDEX - Được sử dụng để tạo và lấy dữ liệu từ cơ sở dữ liệu rất nhanh chóng**

Constraints: Primary keys, Foreign keys, Unique

Primary keys

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);

ALTER TABLE Persons
DROP PRIMARY KEY;
```

Foreign keys

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID);

-- Đặt tên cho FOREIGN KEY constraint
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

```
ALTER TABLE Persons
ADD UNIQUE (ID);

ALTER TABLE Persons
ADD CONSTRAINT UC_Email UNIQUE (Email);
```

Unique

Constraints: Default, Check

DEFAULT

```
CREATE TABLE Persons (
    ...
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

Hoặc

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

CHECK

Thao tác dữ liệu (cont.)

INSERT INTO SELECT

- INSERT INTO SELECT sao chép dữ liệu từ một bảng và chèn vào một bảng khác
- Yêu cầu kiểu dữ liệu trong bảng nguồn và bảng đích **phải khớp nhau.**

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

- Ví dụ

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

UPDATE

- UPDATE được sử dụng để **sửa đổi các bản ghi hiện có** trong một bảng.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- Ví dụ

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;
```

```
update instructor
set salary = salary + 50000
where id > 10
```

Những vấn đề hay gặp khi thay đổi dữ liệu

- Vi phạm khóa chính (PRIMARY KEY)

```
-- Lỗi: Duplicate entry '1' for key 'PRIMARY'
INSERT INTO instructor (id, full_name, dept_name, salary)
VALUES (1, 'Nguyễn Văn X', 'Khoa CNTT', 25000000);
```

- Vi phạm ràng buộc UNIQUE

```
-- Lỗi: Duplicate entry 'IT1001' for key 'code'
INSERT INTO course (code, name, credit)
VALUES ('IT1001', 'Lập trình nâng cao', 4);
```

Những vấn đề hay gặp khi thay đổi dữ liệu

- Vi phạm khóa ngoại (FOREIGN KEY)

```
-- Lỗi: Cannot add or update a child row: a foreign key constraint fails
INSERT INTO teaches (instructor_id, course_id, term)
VALUES (100, 1, 'Học kỳ 1 2024-2025'); -- instructor_id=100 không tồn tại
```

- Vi phạm ràng buộc NOT NULL, CHECK, …

```
-- Lỗi: Check constraint 'chk_salary_positive' is violated
-- (nếu có ràng buộc kiểm tra lương > 0)
INSERT INTO instructor (full_name, dept_name, salary)
VALUES ('Nguyễn Văn X', 'Khoa CNTT', -25000000);
```

DELETE

- DELETE được sử dụng để xóa các bản ghi hiện có trong một bảng

```
DELETE FROM table_name WHERE condition;
```

- Ví dụ

```
-- Xóa hết dữ liệu của 1 bảng
=> Cẩn thận
DELETE FROM table_name;
```

```
-- Giới hạn số lượng hàng bị xóa (LIMIT)
DELETE FROM course WHERE credit < 3
LIMIT 5;
```

Những vấn đề hay gặp khi Delete

- Lỗi vi phạm ràng buộc khóa ngoại (Foreign Key Constraint)

```
DELETE FROM instructor WHERE id = 1;
Error 1451: Cannot delete or update a parent row: a foreign key constraint fails
```

⇒ Lỗi này xảy ra vì bảng **teaches** có khóa ngoại tham chiếu đến **instructor.id**.

- Mặc định, MySQL sử dụng RESTRICT hoặc NO ACTION cho ràng buộc khóa ngoại, nghĩa là:
  - **Không** cho xóa dữ liệu ở bảng cha nếu có dữ liệu tham chiếu ở bảng con
  - Phải xóa dữ liệu ở bảng con trước, sau đó mới xóa ở bảng cha

Các chiến lược này sẽ được trình bày cụ thể ở phần sau

NULL values

NULL values

- Một số thuộc tính của bản ghi có thể nhận giá trị **NULL**
- **null** biểu thị một giá trị không xác định hoặc giá trị không tồn tại.
- Các phép toán liên quan đến **NULL** sẽ trả về **NULL**

```
5 + null  returns null
```

- **IS NULL** và **IS NOT NULL** được sử dụng để kiểm tra giá trị NULL

```
select name
from instructor
where salary is null
```

Aggregation

Các hàm này hoạt động trên tập hợp nhiều giá trị của một cột và trả về **một giá trị**

avg: giá trị trung bình

min: giá trị tối thiểu

max: giá trị tối đa

sum: tổng các giá trị

count: số lượng giá trị

Aggregation: MIN() và MAX()

- Hàm này MIN() trả về giá trị nhỏ nhất của cột được chọn.
- Hàm này MAX() trả về giá trị lớn nhất của cột được chọn.

```
SELECT
MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT
MAX(column_name)
FROM table_name
WHERE condition;
```

- Hàm này MAX()trả về giá trị lớn nhất của cột được chọn.

```
select max(salary) as max_salary
from instructor;
```

#	max_salary
1	33050000.00

Aggregation: COUNT()

- COUNT()trả về số hàng [illegible]h.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

```
select count (*)
from course;
```

#	term
1	Học kỳ 1 2024-2025
2	Học kỳ 1 2024-2025
3	Học kỳ 2 2024-2025
4	Học kỳ 1 2024-2025
5	Học kỳ 2 2024-2025
6	Học kỳ 1 2024-2025
7	Học kỳ 2 2024-2025
8	Học kỳ 1 2024-2025
9	Học kỳ 2 2024-2025
10	Học kỳ 1 2024-2025

- Đếm số lượng Khóa học (course)
- Đếm số lượng kỳ học khác nhau đã được đăng ký dạy học

```
select count(distinct term) as total_term
from teaches;
⇒ Output: 2
```

Aggregation: AVG(), SUM()

- AVG() trả về giá trị trung bình của một cột số

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

- SUM() trả về tổng của một cột số.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Toán tử LIKE

- **LIKE** được sử dụng trong **WHERE** để tìm kiếm một mẫu cụ thể trong một cột.

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Các Pattern và Wildcard hay dùng với LIKE (1)

- **% (Dấu phần trăm)**
  - Đại diện cho không, một hoặc nhiều ký tự bất kỳ.

```
-- Tìm tên bắt đầu bằng 'A'
SELECT * FROM Customers WHERE CompanyName LIKE 'A%';

-- Tìm tên kết thúc bằng 'er'
SELECT * FROM Customers WHERE CompanyName LIKE '%er';

-- Tìm tên chứa chuỗi 'mar'
SELECT * FROM Customers WHERE CompanyName LIKE '%mar%';
```

Các Pattern và Wildcard hay dùng với LIKE (2)

- **_ (Dấu gạch dưới)**
  - Đại diện cho chính xác một ký tự bất kỳ.

```sql
-- Tìm tên có ký tự thứ hai là 'a'
SELECT * FROM Customers WHERE CompanyName LIKE '_a%';

-- Tìm tên có đúng 5 ký tự
SELECT * FROM Customers WHERE CompanyName LIKE '_____';

-- Tìm tên có ký tự thứ 3 là 'r' và ký tự thứ 5 là 'n'
SELECT * FROM Customers WHERE CompanyName LIKE '__r_n%';
```

Các Pattern và Wildcard hay dùng với LIKE (3)

- **[ ] (Dấu ngoặc vuông) - MySQL không hỗ trợ trực tiếp**
  - Đại diện cho **một** ký tự nằm trong **tập hợp** chỉ định.

```
-- Tìm tên bắt đầu bằng 'A', 'B', hoặc 'C'
SELECT * FROM Customers WHERE CompanyName LIKE '[ABC]%';

-- Tìm tên bắt đầu bằng chữ cái từ 'A' đến 'F'
SELECT * FROM Customers WHERE CompanyName LIKE '[A-F]%';

-- Tìm tên có ký tự thứ 2 là số
SELECT * FROM Customers WHERE CompanyName LIKE '_[0-9]%';

-- Tìm tên có ký tự cuối là số chẵn
SELECT * FROM Customers WHERE CompanyName LIKE '%[02468]';
```

Các Pattern và Wildcard hay dùng với LIKE (4)

- **[^] (Dấu mũ trong ngoặc vuông) - MySQL không hỗ trợ trực tiếp**
  - Đại diện cho một ký tự **KHÔNG** nằm trong tập hợp chỉ định.

```sql
-- Tìm tên không bắt đầu bằng 'A', 'B', hoặc 'C'
SELECT * FROM Customers WHERE CompanyName LIKE '[^ABC]%';

-- Tìm tên không kết thúc bằng số
SELECT * FROM Customers WHERE CompanyName LIKE '%[^0-9]';
```

Các Pattern và Wildcard hay dùng với LIKE (5)

- **Kết hợp nhiều wildcard**

```sql
-- Tìm tên có 3 ký tự đầu tiên là 'Ang'
SELECT * FROM Customers WHERE CompanyName LIKE 'Ang%';

-- Tìm tên có 3 ký tự bất kỳ, sau đó là 'son'
SELECT * FROM Customers WHERE CompanyName LIKE '___son%';

-- Tìm tên có chữ 'A' ở vị trí thứ 2 và 'e' ở vị trí thứ 4
SELECT * FROM Customers WHERE CompanyName LIKE '_A_e%';
```

Các Pattern REGEXP hay sử dụng trong SQL (1)

- REGEXP (Regular Expression) mạnh mẽ hơn LIKE, cho phép tìm kiếm và đối sánh chuỗi phức tạp.
- Biểu thức chính quy thường chậm hơn so với LIKE với pattern đơn giản

```sql
-- Sử dụng REGEXP/RLIKE để tìm kiếm pattern
SELECT * FROM Customers WHERE CustomerName REGEXP 'pattern';

-- Sử dụng NOT REGEXP để tìm kiếm không khớp pattern
SELECT * FROM Customers WHERE CustomerName NOT REGEXP 'pattern';
```

Các Pattern REGEXP hay sử dụng trong SQL (2)

- Các pattern hay sử dụng

```sql
-- Tìm CustomerName có chứa từ 'John'
SELECT * FROM Customers WHERE CustomerName REGEXP 'John';

-- Tìm CustomerName chứa 'John' hoặc 'Mary'
SELECT * FROM Customers WHERE CustomerName REGEXP 'John|Mary';

-- Tìm CustomerName bắt đầu bằng 'A'
SELECT * FROM Customers WHERE CustomerName REGEXP '^A';

-- Tìm CustomerName kết thúc bằng 'Ltd'
SELECT * FROM Customers WHERE CustomerName REGEXP 'Ltd$';
```

Các Pattern REGEXP hay sử dụng trong SQL (3)

- Các pattern hay sử dụng trong thực tế

```sql
-- Tìm email hợp lệ (định dạng đơn giản)
SELECT * FROM Customers WHERE Email REGEXP '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$';

-- Tìm ngày dạng YYYY-MM-DD
SELECT * FROM Orders WHERE OrderDate REGEXP '^[0-9]{4}-[0-9]{2}-[0-9]{2}$';
```

Toán tử IN

- **IN** cho phép chỉ định nhiều giá trị trong một mệnh đề **WHERE**
- **IN** là cách viết tắt của nhiều điều kiện **OR**.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

```
-- tất cả khách hàng đến từ cùng quốc gia với nhà cung cấp
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

Toán tử BETWEEN

- **BETWEEN** chọn các giá trị trong phạm vi nhất định. Các giá trị có thể là số, văn bản hoặc ngày tháng.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN start_value AND end_value;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Toán tử BETWEEN

- **BETWEEN** chọn các giá trị trong phạm vi nhất định. Các giá trị có thể là số, văn bản hoặc ngày tháng.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN start_value AND end_value;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Toán tử BETWEEN

- **BETWEEN** chọn các giá trị trong phạm vi nhất định. Các giá trị có thể là số, văn bản hoặc ngày tháng.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN start_value AND end_value;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```