CHƯƠNG 2

TRUY VẤN SQL NÂNG CAO

Giảng viên: Nguyễn Anh Thư

Khoa: Khoa học dữ liệu

NỘI DUNG BÀI HỌC

2.1 SQL JOINs

- 2.1.1 Kết nối bảng theo Tích Descartes
- 2.1.2 Kết nối bảng theo Joins

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

- 2.2.1 SUM, MAX, MIN, AVG, COUNT
- 2.2.2 GROUP BY
- 2.2.3 HAVING
- 2.2.4 SQL CASE

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

- 2.3.1 ROW NUMBER
- 2.2.2 RANK, DENSE RANK
- 2.2.3 NTILE
- 2.2.4 AGGREGATE WINDOW FUNCTIONS

NỘI DUNG BÀI HỌC

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

- 2.4.1 Thiết lập giá trị trường DATETIME
- 2.4.2 EXTRACT và DATE_PART
- 2.4.3 DATE_ADD và DATE_SUB
- 2.4.4 DATEDIFF và TIMESTAMPDIFF

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEX VÀ CONSTRAINT

- 2.5.1 INDEX và EXPLAIN
- 2.5.2 CONSTRAINT

2.6 UNION

NỘI DUNG BÀI HỌC

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

2.7.1 Lưu trữ bằng Table và View

2.7.2 Bổ sung Timestamp

2.7.3 Thêm và cập nhật dữ liệu

MỤC TIÊU BÀI HỌC

Sau khi học xong bài này, cần nắm được các vấn đề sau:

- Cách sử dụng các loại JOIN để kết hợp các bảng.
- Hiểu rõ cách sử dụng các hàm gộp và các kỹ thuật nhóm dữ liệu (như GROUP BY) để tính toán và tổng hợp các giá trị; phép lọc với HAVING.
- Biết rẽ nhánh các trường hợp sử dụng CASE.
- Nắm bắt các window functions để tính toán mà không làm mất đi các dòng dữ liệu, và cách sử dụng subqueries để tạo các truy vấn phức tạp hơn.

MỤC TIÊU BÀI HỌC

Sau khi học xong bài này, cần nắm được các vấn đề sau:

- Một số hàm xử lý giá trị thời gian.
- Tối ưu hóa các câu lệnh truy vấn bằng cách sử dụng Index và Constraint; các câu lệnh khác như UNION và Self-join.
- Lưu trữ các bộ dữ liệu bằng Table và View.

2.1 SQL JOINs

- Các truy vấn đối với một bảng duy nhất chắc chắn không hiếm, nhưng hầu hết các truy vấn sẽ yêu cầu hai, ba hoặc thậm chí nhiều bảng.

- Mệnh đề FROM có thể đề cập đến nhiều hơn một nguồn dữ liệu, khi đó hệ thống phải kết hợp tất cả các nguồn dữ liệu với nhau lại thành một bảng duy nhất. Để làm được điều đó có hai cách trong SQL:
  - Tích Descartes
  - Joins

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO TÍCH DESCARTES

- Khi các bảng được liệt kê trong mệnh đề FROM, hệ thống sẽ tự động sinh ra tích Descartes của tất cả các bảng được để cập đến và/hoặc được biểu thị bởi các truy vấn con.

```
SELECT att_1, att_2
FROM Table_1, Table_2;
```

- **Cách thức hoạt động**: kết hợp theo từng cặp, mỗi hàng từ bảng `Table_1` ghép nối với tất cả các hàng của bảng `Table_2`

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO TÍCH DESCARTES

VD1.2: bảng ***students(student_id, name, class)*** và ***courses(id, course_name)***. Bảng kết hợp theo phép tích Descartes của hai bảng trên được xác định như sau:

students		
student_id	**name**	**class**
1	Van A	May Tinh
2	Van B	Kinh Te

courses	
id	**course_name**
12	Giai tich
34	Thong ke

```
SELECT *
FROM students,
     courses;
```

CP (students, courses)				
student_id	**name**	**class**	**course_id**	**course_name**
1	Van A	May Tinh	12	Giai tich
1	Van A	May Tinh	34	Thong ke
2	Van B	Kinh Te	12	Giai tich
2	Van B	Kinh Te	34	Thong ke

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO JOIN

- JOINs là cách phổ biến để kết hợp các bảng với nhau, trong đó ta cần xác định một điều kiện liên quan đến các thuộc tính của cả hai bảng.

- Có 2 cách sử dụng JOIN trong SQL:

```
SELECT {atts}
FROM T, R
WHERE att1 = att2;
```

Hoặc

```
SELECT {atts}
FROM T JOIN R ON att1 = att2;
[WHERE …]
```

Trong đó: **att1** là một khóa chính (hoặc khóa ngoại) và **att2** là một khóa ngoại (hoặc khóa chính)

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO JOIN

VD2: bảng ***students(student_id, name, class, course_id)*** và ***courses(id, course_name)***. Sử dụng JOIN để kết nối hai bảng.

students			
student_id	**name**	**class**	**course_id**
1	Van A	May Tinh	12
2	Van B	Kinh Te	34
3	Van C	Toan Tin	34

courses	
id	**course_name**
12	Giai tich
34	Thong ke

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO JOIN

VD2: bảng ***students(student_id, name, class, course_id)*** và ***courses(id, course_name)***. Sử dụng JOIN để kết nối hai bảng.

- Cách 1:

```
SELECT student_id, name, class, course_id, course_name
FROM students, course
WHERE students.course_id = course.id;
```

- Cách 2:

```
SELECT student_id, name, class, course_id, course_name
FROM students JOIN course ON students.course_id = course.id;
```

2.1 SQL JOINs

2.1.1 KẾT NỐI BẢNG THEO JOIN

VD2: bảng ***students(student_id, name, class, course_id)*** và ***courses(id, course_name)***. Sử dụng JOIN để kết nối hai bảng.

→ Kết quả của hai cách?

result				
student_id	**name**	**class**	**course_id**	**course_name**
1	Van A	May Tinh	12	Giai tich
2	Van B	Kinh Te	34	Thong ke
3	Van C	Toan Tin	34	Thong ke

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

HÀM (FUNCTIONS)

- Là một trong những tính năng hữu ích nhất của bất kỳ hệ thống cơ sở dữ liệu nào.
- Có 2 kiểu hàm trong SQL:
  - **Các hàm tiêu chuẩn - Standard functions**: là hàm áp dụng cho giá trị của một số thuộc tính trong một hàng và trả về kết quả duy nhất, gồm các hàm xử lý chuỗi, xử lý số (bao gồm xử lý toán học) và xử lý ngày. Một số hàm toán học phổ biến: $m^n = POW(m,n), \sqrt{m} = SQRT(m), \log_b^m = LOG(m,b)$ …
  - **Hàm gộp - Aggregate functions**: là hàm có đầu vào là một tập các giá trị (thay vì là giá trị duy nhất) và trả về kết quả duy nhất. *Chỉ dùng trong mệnh đề SELECT (hoặc HAVING), không dùng trong WHERE.*

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.1 SUM, MAX, MIN, AVG, COUNT (NHẮC LẠI)

Hàm gộp	Chức năng
`SUM([ALL|DISTINCT] col_name)`	Tính tổng các giá trị của cột **`col_name`**.
`AGV([ALL|DISTINCT] col_name)`	Tính trung bình cộng các giá trị của cột **`col_name`**.
`COUNT([ALL|DISTINCT] col_name)`	Đếm số lượng các dòng giá trị có trong cột **`col_name`**.
`COUNT(*)`	Đếm số lượng tất cả các dòng giá trị (bao gồm NULL).
`MAX(col_name)`	Tính GTLN của cột **`col_name`**.
`MIN(col_name)`	Tính GTNN của cột **`col_name`**.

- Theo mặc định các hàm gộp tính toán trên toàn bộ DL, khi cần loại bỏ các giá trị trùng nhau ta sử dụng từ khóa **`DISTINCT`**

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.2 GROUP BY

- Chia dữ liệu thành các nhóm, trong đó các hàng có cùng giá trị trong một cột (được xét) sẽ nằm cùng một nhóm, sau đó các yêu cầu sẽ được thực hiện trong mỗi nhóm.
- Kết quả trả về là một bảng trong đó mỗi hàng chứa một giá trị kết quả cho mỗi nhóm.
- Cú pháp:

```
GROUP BY att1, att2,…;
```

VD3: tính tổng số SV và điểm TB của mỗi lớp khi biết bảng students(student_id, name, class, score):

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.2 GROUP BY

VD3: tính tổng số SV và điểm TB của mỗi lớp khi biết bảng students(student_id, name, class, score):

```
SELECT class, COUNT(student_id) AS so_hoc_sinh, AVG(score) AS
diem_trung_binh
FROM students
GROUP BY class;
```

result		
class	**so_hoc_sinh**	**diem_trung_binh**
May Tinh	2	7.4
Kinh Te	2	7.35
Toan Tin	2	7.35

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.3 HAVING

- Sử dụng khi cần lọc các giá trị sau khi sử dụng hàm gộp; lọc các nhóm dựa trên các giá trị gộp.
- Một cách khác để sử dụng HAVING là khi ta muốn sử dụng điều kiện lên các hàm gộp (vì điều kiện trong WHERE không áp dụng cho hàm gộp).
- Cú pháp:

```
HAVING conditions;
```

VD4: tính tổng số SV của những lớp có điểm TB trên 8.5 khi biết bảng students như sau:

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.3 HAVING

VD4: tính tổng số SV của những lớp có điểm TB trên 8.5 khi biết bảng students:

```
SELECT class, COUNT(student_id) AS so_sv, AVG(score) AS diem_trung_binh
FROM students
GROUP BY class
HAVING AVG(score) > 8.5;
```

result		
class	**so_sv**	**diem_trung_binh**

(Bảng trống vì không có bản ghi nào thỏa mãn)

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.4 SQL CASE

- SQL rẽ nhánh các trường hợp thông qua từ khóa CASE.
- Cú pháp:

```
CASE
   WHEN condition_1 THEN result_1
   WHEN condition_2 THEN result_2
   ELSE result
END
```

- Biểu thức đi qua **condition_i** và trả về một giá trị **result_i** khi thỏa mãn điều kiện; dừng đọc các điều kiện sau (nếu có) .
- Trường hợp không có điều kiện nào được thỏa mãn, trả về NULL.

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.4 SQL CASE

- SQL CASE đều có thể đưa vào SELECT, INSERT, UPDATE và DELETE hay các hàm gộp.

VD5: Hãy phân loại thi đua cho từng lớp và chỉ hiển thị những lớp đạt loại "Trung bình" khi biết bảng students như **VD4**:

- Nếu điểm TB của lớp ≥ 9.0 thì là "Xuất sắc".
- Nếu điểm TB từ 8.0 đến < 9.0 thì là "Khá".
- Nếu điểm TB < 8.0 thì là "Trung bình".

```
SELECT class, AVG(score) AS diem_trung_binh,
    CASE
        WHEN AVG(score) >= 9.0 THEN "Xuất sắc"
        WHEN AVG(score) >= 8.0 AND AVG(score)
            < 9.0 THEN "Khá"
        ELSE "Trung bình"
    END AS category
FROM student
GROUP BY class
HAVING category = "Trung bình";
```

2.2 HÀM GỘP - AGGREGATE FUNCTIONS

2.2.4 SQL CASE

VD5: Hãy phân loại thi đua cho từng lớp như sau và chỉ hiển thị những lớp đạt loại "Trung bình" khi biết bảng students như **VD4**:

- Nếu điểm TB của lớp ≥ 9.0 thì là "Xuất sắc".
- Nếu điểm TB từ 8.0 đến < 9.0 thì là "Khá".
- Nếu điểm TB < 8.0 thì là "Trung bình".

result		
class	**diem_trung_binh**	**category**
May Tinh	7.4	Trung bình
Kinh Te	7.35	Trung bình
Toan Tin	7.35	Trung bình

```
SELECT class, AVG(score) AS diem_trung_binh,
    CASE
        WHEN AVG(score) >= 9.0 THEN "Xuất sắc"
        WHEN AVG(score) >= 8.0 AND AVG(score)
            < 9.0 THEN "Khá"
        ELSE "Trung bình"
    END AS category
FROM student
GROUP BY class
HAVING category = "Trung bình";
```

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

- **VẤN ĐỀ:** Nếu vừa muốn tính toán tổng quan, vừa muốn giữ nguyên chi tiết của từng dòng thì làm thế nào?
  - Hàm tiêu chuẩn: giữ nguyên được chi tiết từng dòng nhưng không tính toán tổng quan được.
  - Hàm gộp: tính toán tổng quan được nhưng kết quả trả về chỉ là một giá trị cho từng nhóm.

- ***HÀM WINDOW**:* là sự kết hợp của hàm tiêu chuẩn và hàm gộp thông qua khái niệm "cửa sổ DL" để thực hiện các phép toán phức tạp trên các dòng mà vẫn giữ nguyên các dòng trong tập DL gốc.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

SO SÁNH:

	Hàm tiêu chuẩn	Hàm gộp	Hàm Window
Cách thức hoạt động	Từng dòng độc lập	Toàn bộ bảng hoặc theo nhóm	Cửa sổ dữ liệu
Giá trị trả về	Mỗi dòng một giá trị	Mỗi nhóm một giá trị	Mỗi dòng một giá trị
Giữ nguyên dòng DL	Có	Không	Có
Sử dụng GROUP BY	Không	Có hoặc không	Không (sử dụng PARTITION BY)
Ví dụ	ABS(), ROUND(), LENGTH()	SUM(), COUNT(), AVG()	ROW_NUMBER(), SUM() OVER

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

a. ROW NUMBER

- Được sử dụng để gán số thứ tự cho từng dòng trong tập DL → xếp hạng các hàng theo giá trị được chỉ định theo nhóm (nếu có).
- Hoạt động trên một cửa sổ DL xác định và đánh số các dòng theo thứ tự mà ta chỉ định.
- Cú pháp:

```
ROW NUMBER() OVER (PARTITION BY col_name_1 ORDER BY col_name_2)
```

Trong đó:

- `ROW NUMBER()` đánh số thứ tự cho mỗi dòng trong cửa sổ dữ liệu, bắt đầu từ 1.
- `PARTITION BY` chia DL thành các phần và đánh số thứ tự riêng biệt cho từng nhóm. (TÙY CHỌN)
- `ORDER BY` tiêu chí sắp xếp khi đánh số thứ tự. (BẮT BUỘC)

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

a. ROW NUMBER

VD6: cho bảng **students(student_id, name, class, score)** như **VD4**

Sử dụng hàm gộp	Sử dụng hàm ROW NUMBER
SELECT class, MAX(score) FROM students GROUP BY class;	SELECT student_id, name, class, score, ROW NUMBER() OVER (PARTITION BY class ORDER BY score DESC) AS ranked FROM students;

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

a. ROW NUMBER

VD6: (tiếp) Nhận xét

Kết quả sử dụng hàm gộp

class	MAX(score)
May Tinh	8.1
Kinh Te	7.5
Toan Tin	7.9

Kết quả sử dụng hàm ROW NUMBER

name	class	score	ranked
Van B	Kinh te	7.5	1
Duc A	Kinh te	7.2	2
Dao C	May tinh	8.1	1

Chỉ liệu kê được số điểm cao nhất của từng lớp	**Liệt kê được số điểm cao nhất của từng lớp và SV nào trong lớp đạt được điểm đó (theo rank)**

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

b. RANK VÀ DENSE RANK

- Có cùng mục đích và cú pháp với ROW_NUMBER:

```
RANK() OVER (PARTITION BY col_name_1 ORDER BY col_name_2)

DENSERANK() OVER (PARTITION BY col_name_1 ORDER BY col_name_2)
```

- **RANK** và **DENSE RANK** cung cấp kết quả trả về theo các cách khác nhau
  - ROW_NUMBER: cho phép hiển thị một giá trị rank với một bản ghi duy nhất.
  - RANK: cho phép hiển thị tất cả các bản ghi có cùng rank.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

b. RANK VÀ DENSE RANK

VD7: cho bảng **students(student_id, name, class, score)** như **VD4,** người ta bổ sung thêm dữ liệu của một bạn sinh viên có thông tin như sau (32, ‘Manh T', 'Kinh te’, 6.8). Hãy sử dụng RANK/ DENSE RANK để xác định thứ hạng của sinh viên theo từng lớp dựa trên điểm số:

students			
student_id	**name**	**class**	**score**
…	…	…	…
32	Manh T	Kinh Te	6.8

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

b. RANK VÀ DENSE RANK

VD7: từ bảng **students(student_id, name, class, score),** sử dụng RANK/ DENSE RANK:

Sử dụng hàm RANK	Sử dụng hàm DENSE RANK
SELECT name, class, score, RANK() OVER (PARTITION BY class ORDER BY score DESC) AS ranked FROM students;	SELECT student_id, name, class, score, DENSE RANK() OVER (PARTITION BY class ORDER BY score DESC) AS ranked FROM students;

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

b. RANK VÀ DENSE RANK

VD7: (tiếp) so sánh giữa sử dụng *ROW NUMBER* và *RANK/ DENSE RANK*

Kết quả sử dụng hàm ROW NUMBER				Kết quả sử dụng hàm RANK			
name	**class**	**score**	**ranked**	**name**	**class**	**score**	**ranked**
Van B	Kinh te	7.5	1	Van B	Kinh te	7.5	1
Duc A	Kinh te	7.2	2	Duc A	Kinh te	7.2	2
Thi D	Kinh te	7.2	3	Thi D	Kinh te	7.2	2
Manh T	Kinh te	6.8	4	Manh T	Kinh te	6.8	4

- **Nhận thấy**: rank = 3 không còn xuất hiện vì rank = 2 thứ hai đã chiếm chỗ của rank = 3. Nếu không muốn xảy ra trường hợp này → sử dụng DENSE_RANK.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

c. NTILE

- Được dùng để chia dữ liệu thành các phần có kích thước xấp xỉ bằng nhau, được gọi là *khối (tile)*.
- Cú pháp:

```
NTILE(n) OVER (PARTITION BY col_name ORDER BY col_name)
```

Trong đó:

- `n` số lượng các khối DL có kích thước bằng nhau, nếu số dòng không chia hết *n* thì các nhóm đầu chứa thêm một dòng so với nhóm sau.
- `PARTITION BY` chia DL thành các phần. (TÙY CHỌN)
- `ORDER BY` tiêu chí sắp xếp DL để phân chia các phần, mỗi dòng được gán vào một nhóm. (BẮT BUỘC)

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

c. NTILE

VD8: cho bảng **students(student_id, name, class, score)** như sau, hãy chia các SV thành 5 nhóm dựa trên điểm số (score).

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

c. NTILE

VD8: (tiếp)

```
SELECT student_id, name, score,
      NTILE(5) OVER (ORDER BY score DESC)
      as “group”
FROM students;
```

result				
student_id	**name**	**class**	**score**	**group**
51	Dao C	May Tinh	8.1	1
32	Van C	Toan Tin	7.9	1
12	Van B	Kinh Te	7.5	2
14	Duc A	Kinh Te	7.2	3
16	Van T	Toan Tin	6.8	4
21	Van A	May Tinh	6.7	5

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. AGGREGATE WINDOW FUNCTIONS – HÀM GỘP WINDOW

- Khi gặp các phép tính phức tạp hơn trên DL, đặc biệt là khi cần tính toán và so sánh các giá trị trong các nhóm dữ liệu thì ta cần kết hợp các hàm gộp (aggregate functions) với các hàm window (window functions) thông qua từ khóa OVER: *Aggregate Window Functions = Aggregate functions + OVER*
- VD: SUM() OVER, AVG() OVER, … → cho phép thực hiện các phép tính trên một cửa sổ DL mà không làm mất cấu trúc DL.
- Cú pháp:

```
AVG(col_name) OVER (PARTITION BY col_name)
```

Trong đó:

- `PARTITION BY` chia DL thành các phần. (TÙY CHỌN)

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. AGGREGATE WINDOW FUNCTIONS – HÀM GỘP WINDOW

VD9: cho bảng **students(student_id, name, class, score)** như sau, hãy cho biết điểm của từng bạn so với điểm TB của lớp mà bạn đó đang theo học.

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. AGGREGATE WINDOW FUNCTIONS – HÀM GỘP WINDOW

VD9: (tiếp)

```
SELECT student_id, name, score,
      AVG(score) OVER(PARTITION BY class)
      as avg_class_score
FROM students;
```

result				
student_id	**name**	**class**	**score**	**avg_class_score**
21	Van A	May Tinh	6.7	7.4
51	Dao C	May Tinh	8.1	7.4
12	Van B	Kinh Te	7.5	7.35
14	Duc A	Kinh Te	7.2	7.35
32	Van C	Toan Tin	7.9	7.35
16	Van T	Toan Tin	6.8	7.35

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. LAG VÀ LEAD

- Cho phép truy xuất giá trị của một cột từ các bản ghi liền kề trước hoặc liền kề sau bản ghi hiện tại, dựa trên thứ tự được xác định bởi **ORDER BY**.
  - **LAG**: lấy giá trị của bản ghi trước đó.
  - **LEAD**: lấy giá trị của bản ghi sau đó.

- Sử dụng để so sánh các bản ghi liên tiếp (vd so sánh điểm tăng giảm giữa hai sinh viên liền kề); xác định xu hướng (vd Xem sản phẩm bán chạy hơn hoặc kém hơn so với ngày trước đó).

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. LAG VÀ LEAD

- Cú pháp chung:

```
LAG(col_n, offset, default_value) OVER (PARTITION BY col_n ORDER BY col_n)
LEAD(col_n, offset, default_value) OVER (PARTITION BY col_n ORDER BY col_n)
```

Trong đó:

- `offset` là khoảng cách so với bản ghi hiện tại (mặc định là 1).
- `default_value` là giá trị trả về nếu không có bản ghi phù hợp (mặc định là NULL).
- `PARTITION BY` chia DL thành các phần. (TÙY CHỌN)
- `ORDER BY` xác định thứ tự.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. LAG VÀ LEAD

VD10: cho bảng **students(student_id, name, class, score)** như sau**,** hãy so sánh điểm của hai bạn liền kề trong cùng một lớp.

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.1 WINDOW FUNCTIONS

d. LAG VÀ LEAD

VD10: (tiếp)

```
SELECT student_id, name, score,
      LAG(score) OVER(PARTITION BY class)
      as prev_score
FROM students;
```

result				
student_id	**name**	**class**	**score**	**prev_score**
21	Van A	May Tinh	6.7	NULL
51	Dao C	May Tinh	8.1	6.7
14	Duc A	Kinh Te	7.2	NULL
12	Van B	Kinh Te	7.5	7.2
16	Van T	Toan Tin	6.8	NULL
32	Van C	Toan Tin	7.9	6.8

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

- Là một câu lệnh SELECT được lồng vào bên trong một câu lệnh SELECT, INSERT, UPDATE, DELETE hoặc một truy vấn con khác.
- Sử dụng khi DL đang truy vấn cần sử dụng đến kết quả của một truy vấn khác.
- Cú pháp:

```
(
SELECT [ALL | DISTINCT] col1, col2,…
FROM   table1, table2, …
[WHERE  conditions]
[GROUP BY col_1, col_2, …]
[HAVING conditions]
)
```

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

MỘT SỐ QUY TẮC

- Truy vấn con phải nằm trong cặp dấu “( )”, thường được sử dụng trong mệnh đề WHERE hoặc HAVING của một truy vấn khác.
- Hầu hết một truy vấn con thường có duy nhất một cột kết quả.
- Mệnh đề COMPUTE và ORDER BY **không** nằm trong truy vấn con.
- Các cột trong truy vấn con có thể là các cột của bảng trong truy vấn ngoài.
- Nếu truy vấn con trả về đúng một giá trị thì nó có thể được coi là một thành phần của một biểu thức.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

PHÉP SO SÁNH VỚI KẾT QUẢ CỦA TRUY VẤN CON

- Kết quả của truy vấn con được sử dụng để so sánh số học với một biểu thức trong truy vấn cha.

```
WHERE att phep_toan_so_hoc [ANY|ALL] (subquery)
```

Trong đó:

- `phep_toan_so_hoc` Là các toán tử so sánh
- `ANY|ALL` Bất kỳ một hoặc tất cả các phần tử trong truy vấn con thỏa mãn điều kiện (OPTIONAL).

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

PHÉP SO SÁNH VỚI KẾT QUẢ CỦA TRUY VẤN CON

VD11: với bảng **students(student_id, name, class, score)** như sau**,** hãy tìm các SV có điểm cao hơn điểm trung bình

students			
student_id	**name**	**class**	**score**
21	Van A	May Tinh	6.7
12	Van B	Kinh Te	7.5
32	Van C	Toan Tin	7.9
14	Duc A	Kinh Te	7.2
51	Dao C	May Tinh	8.1
16	Van T	Toan Tin	6.8

2.3.2 SUBQUERY - TRUY VẤN CON

PHÉP SO SÁNH VỚI KẾT QUẢ CỦA TRUY VẤN CON

VD11: (tiếp)

```
SELECT student_id, name, score
FROM students
WHERE score > (
   SELECT AVG(score)
   FROM students
);
```

students

student_id	name	score
51	Dao C	8.1
12	Van B	7.5
32	Van C	7.9

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

TRUY VẤN CON VỚI IN/NOT IN

- Kiểm tra giá trị của biểu thức có/không xuất hiện trong tập các giá trị của truy vấn con.

```
WHERE bieu_thuc [NOT] IN (subquery)
```

VD12: với bảng **students** như trên, hiển thị những bạn không có điểm trùng với điểm của SV lớp “Kinh Te”.

```
SELECT student_id, name, class, score
FROM students
WHERE score NOT IN (
    SELECT score
    FROM students
    WHERE class = “Kinh Te”
);
```

result			
student_id	**name**	**class**	**score**
16	Van T	Toan Tin	6.8
21	Van A	May Tinh	6.7
32	Van C	Toan Tin	7.9
51	Dao C	May Tinh	8.1

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

TRUY VẤN CON VỚI EXISTS

- Kiểm tra truy vấn con có trả về dòng nào hay không.

```
WHERE [NOT] EXISTS (subquery)
```

- Kết quả trả về là giá trị TRUE / FALSE nếu điều kiện thỏa mãn / không thảo mãn.

- ĐẶC BIỆT: danh sách trả về trong truy vấn con có thể có nhiều hơn hai cột.

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

TRUY VẤN CON VỚI EXISTS

VD13: với yêu cầu như **VD12** nhưng sử dụng (NOT) EXISTS .

```
SELECT student_id, name, class, score
FROM students s1
WHERE NOT EXISTS (
    SELECT 1
    FROM students s2
    WHERE s2.class = “Kinh Te”
            AND s1.score = s2.score
);
```

result			
student_id	**name**	**class**	**score**
16	Van T	Toan Tin	6.8
21	Van A	May Tinh	6.7
32	Van C	Toan Tin	7.9
51	Dao C	May Tinh	8.1

2.3 WINDOW FUNCTIONS VÀ SUBQUERIES

2.3.2 SUBQUERY - TRUY VẤN CON

TRUY VẤN CON VỚI MỆNH ĐỀ HAVING

- Kết quả của truy vấn con được sử dụng để tạo điều kiện đối với các hàm gộp.

```
HAVING bieu_thuc (subquery)
```

VD14: với bảng **students** như trên, hiển thị các lớp có điểm trung bình lớn hơn điểm trung bình chung.

```
SELECT student_id, name, class, score
FROM students
GROUP BY class
HAVING AVG(score) > (
   SELECT AVG(score) FROM students
);
```

result			
student_id	**name**	**class**	**score**
16	Van T	Toan Tin	6.8
21	Van A	May Tinh	6.7
32	Van C	Toan Tin	7.9
51	Dao C	May Tinh	8.1

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

- Bên cạnh chuỗi và số, thông tin về ngày (date) và/hoặc thời gian (time) cũng là dữ liệu thường được sử dụng. Loại dữ liệu này được gọi là dữ liệu thời gian (temporal data).
- Một số ví dụ về dữ liệu thời gian trong cơ sở dữ liệu bao gồm:
  - Ngày trong tương lai mà một sự kiện cụ thể dự kiến sẽ xảy ra (ngày giao đơn hàng của khách hàng,…).
  - Ngày mà đơn hàng của khách hàng đã được giao.
  - Ngày và giờ mà một người dùng đã chỉnh sửa một hàng cụ thể trong một bảng.
  - Ngày sinh của một nhân viên.
  - Thời gian đã trôi qua cần thiết để hoàn thành một đơn hàng.

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

CÁC KIỂU DỮ LIỆU THỜI GIAN			
KIỂU	**ĐỊNH DẠNG MẶC ĐỊNH**	**GIÁ TRỊ CHO PHÉP**	**MỤC ĐÍCH SỬ DỤNG**
date	YYYY-MM-DD	[1000-01-01; 9999-12-31]	Lưu ngày tháng, không cần chi tiết về thời gian.
datetime	YYYY-MM-DD HH:MI:SS	[1000-01-01 00:00:00.000000; 9999-12-31 23:59:59.999999]	Lưu thông tin ngày và thời gian cụ thể.
timestamp	YYYY-MM-DD HH:MI:SS	[1970-01-01 00:00:00.000000; 2038-01-18 22:14:07.999999]	Lưu ngày giờ, tự động cập nhật khi thêm hoặc sửa hàng (thời gian cuối cùng sửa hàng).
year	YYYY	[1901; 2155]	Lưu năm.
time	HHH:MI:SS	[−838:59:59.000000; 838:59:59.000000]	Lưu thông tin về khoảng thời gian (thời gian trôi qua).

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

Thành phần định dạng	Ý nghĩa	Giá trị giới hạn
YYYY	Year (năm)	1000 đến 9999
MM	Month (tháng)	01 (January) đến 12 (December)
DD	Day (ngày)	01 đến 31
HH	Hour (giờ)	00 đến 23
HHH	Hours (giờ đã trôi qua)	−838 đến 838
MI	Minute (phút)	00 đến 59
SS	Second (giây)	00 đến 59

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

ĐỊNH DẠNG MẶC ĐỊNH
YYYY-MM-DD
YYYY-MM-DD HH:MI:SS
YYYY-MM-DD HH:MI:SS
YYYY
HHH:MI:SS

- Mô tả định dạng của dữ liệu khi được hiển thị hoặc truy xuất từ CSDL.
- Các kiểu dữ liệu thời gian hỗ trợ phần giây sẽ có số giây chính xác đến 6 chữ số thập phân (microseconds); chỉ định độ chính xác bằng cách cung cấp một giá trị từ 0 - 6.
- **Ví dụ**: định nghĩa cột DATETIME(2) thì giá trị thời gian trong cột này có thể bao gồm 2 chữ số thập phân cho giây:

  '2020-03-23 15:30:45.12'

  (giờ phút giây kèm theo phần trăm của giây).

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

DỮ LIỆU THỜI GIAN CÓ THỂ ĐƯỢC TẠO THEO 3 CÁCH:

- **Sao chép từ cột có sẵn**: Đây là cách đơn giản nhất, chỉ cần sao chép giá trị từ cột chứa dữ liệu thời gian sang cột mới.
- **Sử dụng hàm tích hợp sẵn**: SQL cung cấp nhiều hàm tích hợp sẵn để tạo dữ liệu thời gian (VD hàm NOW() để lấy thời gian hiện tại).
- **Xây dựng chuỗi đại diện cho dữ liệu thời gian**: Phương pháp này yêu cầu hiểu rõ về các thành phần định dạng ngày tháng (tham khảo hai bảng trên). Sau đó, ta có thể ghép các chuỗi con đại diện cho năm, tháng, ngày, giờ, phút, giây để tạo thành chuỗi dữ liệu thời gian hoàn chỉnh.

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

- Để xây dựng một chuỗi mà server có thể diễn giải là ngày/ ngày giờ/ giờ, ta cần đặt các thành phần khác nhau lại với nhau theo thứ tự nhất định (như 2 bảng trên).
- VD để điền vào cột DATETIME có giá trị là *1:30 chiều ngày 01 tháng 01 năm 2025* thì ta cần xác định chuỗi như sau: **`'2025-01-01 13:30:00'`**
- Vấn đề:
  - Nếu dữ liệu (hoặc cột) không được định nghĩa kiểu DATETIME (VD kiểu chuỗi, số) mà ta cần so sánh nó với một dữ liệu khác ở dạng thời gian thì làm thế nào?
  - SQL thường sử dụng định dạng mặc định `YYYY-MM-DD HH:MM:SS` cho kiểu DATETIME mà dữ liệu không theo định dạng này thì sửa sao?

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

- Giải quyết: Yêu cầu server chuyển đổi dữ liệu sang kiểu thời gian thông qua các hàm chuyển đổi.
  - ***CAST()*** chuyển đổi giá trị từ một kiểu dữ liệu này sang kiểu dữ liệu khác nhưng cần đúng định dạng mặc định của kiểu dữ liệu mong muốn.

```
SELECT CAST(str AS DATE/DATETIME/…) AS converted_datetime;
```

  - ***STR_TO_DATE()*** chuyển đổi chuỗi thành kiểu dữ liệu thời gian theo thành phần được sắp xếp trong chuỗi.

```
SELECT STR_T0_DATE(str, format) AS converted_datetime;
```

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

CÁC THÀNH PHẦN ĐỊNH DẠNG TRONG FORMAT THƯỜNG ĐƯỢC SỬ DỤNG		
Định dạng	**Ý nghĩa**	**Cụ thể**
%d	Ngày dạng số	01 → 31
%j	Ngày trong năm	001 → 366
%W	Tên ngày trong tuần	Sunday → Saturday
%M	Tên tháng	January → December
%m	Tháng dạng số	01 → 12
%Y	Năm	Số có 4 chữ số (VD: 2025)
%y	Năm	Số có 2 chữ số (VD: 25)

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

CÁC THÀNH PHẦN ĐỊNH DẠNG TRONG FORMAT THƯỜNG ĐƯỢC SỬ DỤNG		
Định dạng	**Ý nghĩa**	**Cụ thể**
%H	Giờ dạng 24h	00 → 23
%h	Giờ dạng 12h	01 → 12
%i	Phút	00 → 59
%s	Giây	00 → 59
%p	Xác định sáng/tối	A.M/P.M

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

VD:

1)
```
SELECT CAST('2025-01-01 14:30:00' AS DATETIME) AS converted_datetime;
```

2)
```
SELECT CAST('2025-01-01' AS DATETIME) AS converted_datetime;
```

3)
```
SELECT CAST('01-01-2025' AS DATE) AS converted_datetime;
```

NULL

2025-01-01 00:00:00

2025-01-01 14:30:00

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

BIỂU DIỄN DẠNG CHUỖI CỦA DỮ LIỆU THỜI GIAN:

VD:

1)
```
SELECT STR_T0_DATE('2025-01-01', '%Y-%m-%d') AS converted_datetime;
```

2)
```
SELECT STR_T0_DATE('2025-01-01', '%Y-%M-%d') AS converted_datetime;
```

3)
```
SELECT STR_T0_DATE('2025-01-01 13:00', '%Y-%m-%d %H:%i’)
AS converted_datetime;
```

NULL

2025-01-01

2025-01-01 13:00:00

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.1 THIẾT LẬP GIÁ TRỊ TRƯỜNG DATETIME

MỘT SỐ HÀM KHÁC TẠO DỮ LIỆU KIỂU THỜI GIAN:

- CURRENT_DATE() Lấy ngày hiện tại theo định dạng DATE.
- CURRENT_TIME() Lấy thời gian hiện tại theo định dạng TIME.
- CURRENT_TIMESTAMP() = NOW() Lấy thời gian hiện tại theo định dạng TIMESTAMP.

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.2 EXTRACT VÀ DATE_PART

- Mục đích: chỉ lấy một phần của giá trị được lưu trữ của dữ liệu thời gian.
- VD từ dữ liệu dạng DATE, trích xuất ngày, tháng và năm thành các trường dữ liệu khác nhau để phù hợp với các mục đích khác nhau như so sánh hàng năm, căn chỉnh và trực quan hóa tổng số hàng ngày từ các năm khác nhau theo tháng và ngày,…
- Các hệ quản trị CSDL khác nhau có thể sử dụng hàm và cú pháp khác nhau để trích xuất thông tin nhưng ý tưởng chunng là như nhau: *MYSQL* sử dụng hàm EXTRACT, *Redshift* sử dụng DATE_PART, trong khi *Oracle* và *SQL Server* sử dụng DATEPART.

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.2 EXTRACT VÀ DATE_PART

- Hàm EXTRACT() trong MySQL cho phép trích xuất các phần cụ thể từ một giá trị ngày giờ, chẳng hạn như ngày, tháng, năm, giờ, phút và giây. Cú pháp:

```
EXTRACT(part FROM date)
```

`part` thành phần thời gian mà ta muốn trích xuất (DAY, MONTH, YEAR, HOUR, MINUTE).

`date` giá trị thời gian nguồn trích xuất.

- Ngoài EXTRACT(), MySQL cũng cung cấp các hàm DATE() và TIME() để trích xuất phần ngày (date) và phần giờ (time).

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.2 EXTRACT VÀ DATE_PART

VD14: Giả sử ta có dữ liệu dạng DATE, hãy trích xuất ngày, tháng và năm thành các trường dữ liệu khác nhau.

```
SELECT    a_datetime,
          EXTRACT(DAY FROM a_datetime) AS a_day,
          EXTRACT(MONTH FROM a_datetime) AS a_month,
          EXTRACT(YEAR FROM a_datetime) AS a_year
FROM exam_table.datetime_demo;
```

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

- Một điểm mạnh khi lưu trữ chuỗi ngày dưới dạng giá trị datetime đó là có thể thực hiện các phép tính ngày, đây là điều không thể làm được khi chúng được lưu trữ dưới dạng số, dấu chấm câu và chữ cái trong một trường dạng chuỗi.
- DATE_ADD và DATE_SUB là một trong những hàm dựng sẵn lấy một ngày là đối số và trả lại một ngày khác.

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

DATE_ADD trả về một giá trị kiểu DATE/DATETIME bằng cách cộng thêm một khoảng giá trị vào ngày vào ngày được chỉ định. Cú pháp:

```
DATE_ADD(date, INTERVAL expr unit)
```

Trong đó, `date` là giá trị ngày giờ ban đầu, từ khóa `INTERVAL` bắt buộc, `expr` số nguyên đại diện cho số lượng đơn vị thời gian cần cộng, `unit` là đơn vị thời gian (DAY, MONTH, YEAR, HOUR, MINUTE, SECOND).

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

MỘT SỐ LOẠI UNIT PHỔ BIẾN

UNIT	Ý nghĩa
Second	Số giây
Minute	Số phút
Hour	Số giờ
Day	Số ngày
Month	Số tháng
Year	Số năm
Minute_second	Số phút và số giây, phân tách bởi ":"
Hour_second	Số giờ, số phút và số giây, phân tách bởi ":"
Year_month	Số năm và số tháng, phân tách bởi "-"

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

VD16: hãy cộng thêm 5 ngày vào thời gian hiện tại:

```
SELECT    CURRENT_DATE() AS cur_date,
          DATE_ADD(cur_date, INTERVAL 5 DAY);
```

	current_date()	date_add(current_date(), INTERVAL 5 DAY)
▶	2025-01-16	2025-01-21

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

DATE_SUB trả về một giá trị kiểu DATE/DATETIME bằng cách cộng thêm một khoảng giá trị vào ngày vào ngày được chỉ định. Cú pháp:

```
DATE_SUB(date, INTERVAL expr unit)
```

VD17.1: hãy lùi 5 ngày tính từ thời điểm hiện tại:

```
SELECT    CURRENT_DATE() AS cur_date,
          DATE_SUB(cur_date, INTERVAL 5 DAY);
```

	current_date()	date_sub(current_date(), INTERVAL 5 DAY)
▶	2025-01-16	2025-01-11

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.3 DATE_ADD VÀ DATE_SUB

DATE_SUB

✓ Thay vì chuyển sang DATE_SUB, ta có thể sử dụng hàm DATE_ADD nhưng trong expr ta thêm dấu trừ (-) vào đằng trước.

VD17.2: hãy lùi 5 ngày tính từ thời điểm hiện tại sử dụng DATE_ADD:

```
SELECT    CURRENT_DATE() AS cur_date,
          DATE_ADD(cur_date, INTERVAL -5 DAY);
```

	current_date()	date_add(current_date(), INTERVAL -5 DAY)
▶	2025-01-16	2025-01-11

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.4 DATEDIFF VÀ TIMESTAMPDIFF

DATEDIFF là hàm tính sự chênh lệch về số ngày giữa hai dữ liệu thời gian. Hàm DATEDIFF nhận hai tham số là giá trị DATE/DATETIME và trả về hiệu giữa chúng (date1 - date2) tính bằng ngày. Cú pháp:

```
DATEDIFF(date1, date2)
```

VD18: Tính từ thời điểm hiện tại tới thời điểm 5 tháng sau thì có bao nhiêu ngày?

```
SELECT    DATEDIFF(next_date, CURRENT_DATE()) AS days_difference,
          CURRENT_DATE() AS today_date
FROM (SELECT DATE_ADD(current_date(), INTERVAL 5 MONTH) AS next_date) AS subquery;
```

today_date	days_difference
2025-01-16	151

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.4 DATEDIFF VÀ TIMESTAMPDIFF

TIMESTAMPDIFF tương tự DATEDIFF nhưng cho phép ta chỉ định đơn vị thời gian cho kết quả như giờ/phút/giây thay cho ngày. Cú pháp:

```
TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)
```

Trong đó:

`unit` là đơn vị thời gian trả về (ví dụ: HOUR, MINUTE, SECOND,…);

`datetime_expr1, datetime_expr2` là hai biểu thức ngày giờ cần tính hiệu số.

Kết quả trả về sẽ được tính bởi (`datetime_expr2 - datetime_expr1`).

2.4 XỬ LÝ DỮ LIỆU THỜI GIAN

2.4.4 DATEDIFF VÀ TIMESTAMPDIFF

TIMESTAMPDIFF tương tự DATEDIFF nhưng cho phép ta chỉ định đơn vị thời gian cho kết quả như giờ/phút/giây thay cho ngày. Cú pháp:

```
TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)
```

VD19: Tính từ thời điểm hiện tại tới thời điểm 5 tháng sau thì có bao nhiêu ngày?

```
SELECT today_date, next_date,
            timestampdiff(DAY, today_date, next_date) AS days_difference,
FROM (CURRENT_DATE() AS today_date,
      SELECT DATE_ADD(current_date(), INTERVAL 5 MONTH) AS next_date) AS subquery;
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.1 INDEX – CHỈ MỤC

CHỈ MỤC giống như mục lục trong sách, giúp ta nhanh chóng tìm kiếm thông tin cần thiết. Khi không có chỉ mục, máy chủ cơ sở dữ liệu phải quét toàn bộ bảng để tìm kiếm dữ liệu → thời gian truy vấn lâu, đặc biệt với bảng có số lượng lớn bản ghi. Chỉ mục cho phép máy chủ cơ sở dữ liệu nhanh chóng xác định vị trí dữ liệu mà không cần quét toàn bộ bảng.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.1 INDEX – CHỈ MỤC

➢ THÊM CHỈ MỤC MỚI CHO CỘT:

- ***Khi khởi tạo bảng:***

```
CREATE TABLE table_name (
   {… statements …}
   INDEX idx_name (col_1, col_2,...)
);
```

- ***Thêm chỉ mục vào bảng đã có:***

Cách 1

```
ALTER TABLE table_name ADD INDEX idx_name
(col_1, col_2,…);
```

Cách 2

```
CREATE INDEX idx_name ON table_name
(col_1, col_2,…);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.1 INDEX – CHỈ MỤC

➢ XÓA CHỈ MỤC:

Cách 1 (MYSQL)

```
ALTER TABLE table_name ( DROP INDEX idx_name );
```

Cách 2 (ORACLE)

```
DROP INDEX table_name.idx_name ;
```

Cách 3 (SQL SERVER)

```
DROP INDEX idx_name ON table_name;
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.1 INDEX – CHỈ MỤC

➢ CHỈ MỤC DUY NHẤT – UNIQUE INDEX:

Chỉ mục duy nhất đóng vai trò như cơ chế không cho phép trùng lặp giá trị trong (các) cột được lập chỉ mục. Khi một hàng được chèn hoặc cột chỉ mục bị sửa đổi, máy chủ CSDL sẽ kiểm tra chỉ mục duy nhất xem giá trị đã tồn tại chưa.

```
ALTER TABLE table_name ADD UNIQUE idx_name col_name;
```

Note: Khi xây dựng các chỉ mục nhiều cột, ta nên suy nghĩ cẩn thận cột nào sẽ liệt kê trước, cột nào sẽ liệt kê sau để giúp làm cho chỉ mục hữu ích nhất có thể.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

Là một quy tắc được áp dụng lên một hoặc nhiều cột của một bảng để đảm bảo tính toàn vẹn và nhất quán của dữ liệu; thường được tạo cùng với bảng qua câu lệnh CREATE TABLE và ALTER TABLE (MySQL).

- **Ràng buộc khóa chính**: Xác định một hoặc nhiều cột đảm bảo tính duy nhất của mỗi hàng trong bảng; ngăn chặn việc chèn các hàng trùng lặp. VD: cột student_id trong bảng students có thể được định nghĩa là khóa chính để đảm bảo mỗi SV có ID duy nhất.
- **Ràng buộc khóa ngoại**: Giới hạn giá trị của một hoặc nhiều cột trong bảng con chỉ chứa các giá trị được tìm thấy trong cột khóa chính của bảng cha. Điều này đảm bảo mối quan hệ giữa các bảng được duy trì. VD: cột course_id trong bảng students có thể được định nghĩa là khóa ngoại tham chiếu đến cột id trong bảng course, đảm bảo rằng mỗi SV tham gia khóa học hợp lệ tồn tại trong bảng course.
- **Ràng buộc duy nhất**: Giới hạn một hoặc nhiều cột chứa các giá trị duy nhất trong bảng. Khác với khóa chính, ràng buộc duy nhất cho phép giá trị NULL.
- **Ràng buộc kiểm tra**: Giới hạn các giá trị cho phép của một cột. VD trong bảng students có thể được ràng buộc kiểm tra để đảm bảo giá trị score luôn nhỏ hơn hoặc bằng 10.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

a. TẠO RÀNG BUỘC KHÓA CHÍNH KHI TẠO BẢNG:

```
CREATE TABLE table_name (
    colname_1 datatype,
    colname_2 datatype
        ...
    CONSTRAINT constraint_name PRIMARY KEY (primary_key_colname)
);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

a. TẠO RÀNG BUỘC KHÓA CHÍNH KHI TẠO BẢNG:

VD20:

```
CREATE TABLE customer (
    customer_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    CONSTRAINT pk_customer PRIMARY KEY (customer_id)
);
```

→ `pk_customer` là tên ràng buộc khóa chính, và `customer_id` là cột được chọn làm khóa chính.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

b. THÊM RÀNG BUỘC KHÓA CHÍNH SAU KHI TẠO BẢNG:

```
ALTER TABLE table_name

ADD CONSTRAINT constraint_name PRIMARY KEY (primary_key_colname);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

b. THÊM RÀNG BUỘC KHÓA CHÍNH SAU KHI TẠO BẢNG:

VD21:

```
ALTER TABLE customer

ADD CONSTRAINT pk_customer PRIMARY KEY (customer_id);
```

→ `pk_customer` là tên ràng buộc khóa chính, và `customer_id` là cột được chọn làm khóa chính.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

c. TẠO RÀNG BUỘC KHÓA NGOẠI KHI TẠO BẢNG:

```
CREATE TABLE table_name (
    colname_1 datatype,
    colname_2 datatype,
        ...
    CONSTRAINT constraint_name FOREIGN KEY (foreign_key_colname)
        REFERENCES referenced_table (primary_key_colname)
        [ON DELETE <action>] [ON UPDATE <action>]
);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

c. TẠO RÀNG BUỘC KHÓA NGOẠI KHI TẠO BẢNG:

VD22:

```
CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    CONSTRAINT fk_orders_customer FOREIGN KEY (customer_id)
     REFERENCES customer (customer_id) ON DELETE CASCADE
);
```

→ `fk_orders_customer` là tên ràng buộc khóa ngoại.; `customer_id` trong bảng `orders` là cột khóa ngoại. `customer` là bảng cha, và `customer_id` trong bảng `customer` là cột khóa chính được tham chiếu.

`ON DELETE CASCADE` chỉ định rằng khi một khách hàng bị xóa khỏi bảng customer, các đơn hàng liên quan đến khách hàng đó cũng sẽ bị xóa khỏi bảng orders.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

d. THÊM RÀNG BUỘC KHÓA NGOẠI SAU KHI TẠO BẢNG:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name FOREIGN KEY (foreign_key_colname)
REFERENCES referenced_table (primary_key_colname)
[ON DELETE <action>] [ON UPDATE <action>];
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

d. THÊM RÀNG BUỘC KHÓA NGOẠI SAU KHI TẠO BẢNG:

VD23:

```
ALTER TABLE orders

ADD CONSTRAINT fk_orders_customer FOREIGN KEY (customer_id) REFERENCES
customer (customer_id) ON DELETE CASCADE;
```

→ `fk_orders_customer` là tên ràng buộc khóa ngoại.; `customer_id` trong bảng `orders` là cột khóa ngoại.

`customer` là bảng cha, và `customer_id` trong bảng `customer` là cột khóa chính được tham chiếu.

`ON DELETE CASCADE` chỉ định rằng khi một khách hàng bị xóa khỏi bảng customer, các đơn hàng liên quan đến khách hàng đó cũng sẽ bị xóa khỏi bảng orders.

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

e. TẠO RÀNG BUỘC DUY NHẤT KHI TẠO BẢNG:

```
CREATE TABLE table_name (
    colname_1 datatype,
    colname_2 datatype
        ...
    CONSTRAINT constraint_name UNIQUE (colname)
);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

e. TẠO RÀNG BUỘC DUY NHẤT KHI TẠO BẢNG:

VD24:

```
CREATE TABLE email_addresses (

    email_address VARCHAR(255),

    CONSTRAINT un_email_address UNIQUE (email_address)

);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

f. THÊM RÀNG BUỘC DUY NHẤT SAU KHI TẠO BẢNG:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (colname);
```

VD25:

```
ALTER TABLE email_addresses
ADD CONSTRAINT un_email_address UNIQUE (email_address);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

f. TẠO RÀNG BUỘC KIỂM TRA KHI TẠO BẢNG:

```
CREATE TABLE table_name (
    colname_1 datatype,
    colname_2 datatype
        ...
    CONSTRAINT constraint_name CHECK (colname)
);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

f. TẠO RÀNG BUỘC KIỂM TRA KHI TẠO BẢNG:

VD26:

```
CREATE TABLE products (

    product_id INT,

    price DECIMAL(10,2),

    CONSTRAINT chk_product_price CHECK (price > 0)

);
```

2.5 TỐI ƯU HÓA TRUY VẤN VỚI INDEXES VÀ CONSTRAINTS

2.5.2 CONSTRAINT – RÀNG BUỘC

g. THÊM RÀNG BUỘC KIỂM TRA SAU KHI TẠO BẢNG:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name CHECK (condition);
```

VD27:

```
ALTER TABLE products

ADD CONSTRAINT chk_product_price CHECK (price > 0);
```

NOTE: <constraint_name> là tên tùy chọn cho ràng buộc. Nếu không cung cấp tên, hệ quản lý CSDL sẽ tự động tạo tên.

2.6 UNION

Truy vấn UNION được sử dụng để kết hợp kết quả của hai hoặc nhiều câu truy vấn SELECT riêng biệt, bất kỳ thành một kết quả duy nhất. Các yêu cầu cho câu truy vấn UNION bao gồm:

- Số lượng cột phải giống nhau trong mỗi câu truy vấn SELECT.
- Kiểu dữ liệu của các cột tương ứng phải tương thích.
- Thứ tự của các cột phải giống nhau trong mỗi câu truy vấn SELECT.

Có 2 loại UNION:

- UNION: loại bỏ các hàng trùng lặp trong kết quả cuối cùng.
- UNION ALL: giữ tất cả các hàng, bao gồm cả hàng trùng lặp.

2.6 MỘT SỐ CẤU TRÚC TRUY VẤN NÂNG CAO KHÁC

Cú pháp:

```
SELECT <col_1>, <col_2>, ... FROM <table_1>
UNION [ALL]
SELECT <col_1>, <col_2>, ... FROM <table_2>;
```

2.6 MỘT SỐ CẤU TRÚC TRUY VẤN NÂNG CAO KHÁC

VD28: có hai bảng dữ liệu: customers_2024 và customers_2025; muốn kết hợp dữ liệu từ cả hai bảng để tạo một danh sách duy nhất và loại bỏ các bản ghi trùng lặp như sau:

customers_2024		
ustomer_id	**customer_name**	**city**
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	Chicago

customers_2025		
customer_id	**customer_name**	**city**
3	Charlie	Chicago
4	David	Houston
5	Eve	Miami

```
SELECT customer_id, customer_name, city
FROM customers_2024
UNION
SELECT customer_id, customer_name, city
FROM customers_2025;
```

2.6 UNION

VD29: có hai bảng dữ liệu: customers_2024 và customers_2025; muốn kết hợp dữ liệu từ cả hai bảng để tạo một danh sách duy nhất và loại bỏ các bản ghi trùng lặp như sau:

result		
customer_id	customer_name	city
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	Chicago
4	David	Houston
5	Eve	Miami

➔ UNION tự động loại bỏ các dòng trùng lặp (như bản ghi của Charlie từ cả hai bảng).Kết quả trả về danh sách kết hợp từ cả hai bảng.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng TABLE

Khi lưu trữ dữ liệu được truy vấn dưới dạng bảng, kết quả sẽ được "chụp nhanh" (snapshot) tại thời điểm truy vấn được chạy mà không thay đổi DL gốc. Khi đó DL trả về cucar một truy vấn có thể được lưu như một bảng mới hoặc các hàng mới được thêm vào một bảng hiện có, tùy thuộc vào câu lệnh SQL.

➔ Phù hợp khi cần lưu trữ kết quả truy vấn cố định để sử dụng sau này, ví dụ như trong phân tích lịch sử hoặc huấn luyện mô hình học máy hay tách biệt dữ liệu cho các mục đích khác nhau, tránh ảnh hưởng bởi thay đổi dữ liệu gốc.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng TABLE

Cú pháp tạo bảng từ kết quả truy vấn như sau:

```
CREATE TABLE schema_name.new_table_name AS
(
    [your query here]
);
```

Tên bảng trong câu lệnh `CREATE TABLE` phải là tên mới và duy nhất trong schema. Nếu ta chạy cùng một câu lệnh trên hai lần liên tiếp thì sẽ gặp lỗi bảng đã tồn tại.

Bảng mới sẽ chứa **dữ liệu tĩnh** (dữ liệu tại thời điểm truy vấn được thực thi). Nếu bảng gốc sau này thay đổi, bảng mới sẽ không tự động cập nhật → Nếu muốn bảng tự động cập nhật khi dữ liệu thay đổi, dùng View thay vì Table.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng TABLE

VD30: có một bảng tên là orders lưu trữ thông tin về các đơn hàng, với cấu trúc và dữ liệu như sau, hãy tạo một bảng mới tên là high_value_orders để lưu trữ các đơn hàng có giá trị lớn hơn 300:

Orders			
order_id	**customer_id**	**order_date**	**total_amount**
1	101	1/10/2024	250
2	102	2/15/2024	450
3	101	3/20/2024	150
4	103	4/5/2024	300

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng TABLE

VD31: có một bảng tên là orders lưu trữ thông tin về các đơn hàng, với cấu trúc và dữ liệu như sau, hãy tạo một bảng mới tên là high_value_orders để lưu trữ các đơn hàng có giá trị lớn hơn 300:

```
CREATE TABLE high_value_orders AS
(
    SELECT order_id, customer_id, order_date, total_amount
    FROM orders
    WHERE total_amount > 300
);
```

high_value_orders			
order_id	**customer_id**	**order_date**	**total_amount**
2	102	2/15/2024	450
4	103	4/5/2024	300

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng TABLE

Khi cần xóa bảng của kết quả câu truy vấn, ta có:

```
DROP TABLE [schema_name].[new_table_name];
```

➔ Cần chú ý khi thao tác DROP TABLE vì bảng đã xóa sẽ không khôi phục lại được → cần cài đặt tần suất sao lưu DL.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng VIEW

- VIEW không lưu trữ DL mà chỉ lưu câu lệnh truy vấn SQL. Khi tham chiếu đến VIEW, câu lệnh SQL được thực thi theo yêu cầu và bộ DL mới sẽ được tạo ra dựa trên trạng thái hiện tại của bảng được tham chiếu. Vì VIEW cung cấp DL dạng "động" nên nó luôn phản ánh được DL mới nhất từ các bảng.

- VIEW thường được sử dụng để đơn giản hóa các truy vấn phức tạp bằng cách đặt tên và tái sử dụng chúng; hay cần ẩn giấu chi tiết triển khai của truy vấn, cung cấp cho người dùng cái nhìn đơn giản hơn về DL.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng VIEW

Cú pháp tạo VIEW:

```
CREATE VIEW schema_name.new_table_name AS
(
    [your query here]
);
```

Cú pháp xóa VIEW:

```
DROP VIEW schema_name.new_table_name;
```

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng VIEW

VD32: Giả sử ta có bảng orders lưu trữ thông tin đơn hàng với cấu trúc và dữ liệu sau, hãy muốn tạo một VIEW có tên high_value_orders, chỉ hiển thị các đơn hàng có giá trị tổng lớn hơn 300.

```
CREATE VIEW high_value_orders AS
SELECT order_id, customer_id,
       order_date, total_amount
FROM orders
WHERE total_amount > 300;
```

orders

order_id	customer_id	order_date	total_amount
1	101	1/10/2024	250
2	102	2/15/2024	450
3	101	3/20/2024	150
4	103	4/5/2024	300

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng VIEW

VD33: Giả sử ta có bảng orders lưu trữ thông tin đơn hàng với cấu trúc và dữ liệu sau, hãy muốn tạo một VIEW có tên high_value_orders, chỉ hiển thị các đơn hàng có giá trị tổng lớn hơn 300.

```
SELECT * FROM high_value_orders;
```

order_id	customer_id	order_date	total_amount
2	102	2/15/2024	450
4	103	4/5/2024	300

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

a. LƯU DỮ BỘ DL SQL DƯỚI DẠNG TABLE VÀ VIEW

Lưu trữ DL dạng VIEW

- Tính động của VIEW:
  - VIEW không lưu trữ dữ liệu. Khi truy vấn VIEW, SQL sẽ thực thi câu lệnh SELECT bên trong VIEW mỗi lần ta sử dụng nó.
  - Nếu dữ liệu trong bảng gốc orders thay đổi, dữ liệu trong VIEW cũng tự động thay đổi.

- Cập nhật VIEW:
  - Nếu cần thay đổi nội dung VIEW, ta có thể sử dụng lệnh **CREATE OR REPLACE VIEW** hoặc xóa và tạo lại VIEW.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

b. BỔ SUNG MỘT CỘT TIMESTAMP

Khi tạo hoặc sửa một bảng CSDL, ta thường muốn giữ lại một bản ghi về thời điểm mà từng dòng trong bảng được tạo hoặc sửa đổi. Ta có thể thực hiện việc này bằng cách **thêm cột mốc thời gian** vào câu lệnh **CREATE TABLE** hoặc **UPDATE**. Trong SQL, hàm trả về ngày và giờ hiện tại là **CURENT_TIMESTAMP**. Cột mốc thời gian được tạo bởi máy chủ CSDL nên thời gian do hàm trả về có thể khác với thời gian hiện tại trên máy của ta.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

c. THÊM HÀNG VÀ CẬP NHẬT DL VÀO BẢNG CSDL

Nếu muốn thay đổi DL trong một bảng CSDL, ta sử dụng câu lệnh INSERT để thêm hàng mới hoặc UPDATE để chỉnh sửa bản ghi trong bảng DL. Trong phần này ta sẽ thảo luận về cách chèn kết quả của một truy vấn vào một bảng khác, **INSERT INTO SELECT**.

```
INSERT INTO [schema_name].[table_name] ([col_1, …])
[your SELECT query here]
```

CHÚ Ý: các cột trong cả hai truy vấn phải theo cùng một thứ tự. Các trường tương ứng có thể không có tên giống nhau, nhưng hệ thống sẽ cố gắng để chèn các giá trị trả về từ câu lệnh SELECT theo thứ tự cột được liệt kê trong ngoặc đơn.

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

c. THÊM HÀNG VÀ CẬP NHẬT DL VÀO BẢNG CSDL

Khi gặp lỗi trong việc chèn thêm một hàng và muốn xóa nó, ta có cú pháp như sau:

```
DELETE FROM schema_name.table_name
WHERE [set of conditions that uniquely identifies the row]
```

Đôi khi muốn cập nhật một giá trị trong một hàng thay vì chèn một hàng mới thì có thể sử dụng câu lệnh:

```
UPDATE schema_name.table_name
SET col_name = new value
WHERE [set of conditions that uniquely identifies the rows you want to change]
```

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

c. THÊM HÀNG VÀ CẬP NHẬT DL VÀO BẢNG CSDL

- **VD34:** có bảng **employees** (lưu trữ thông tin nhân viên hiện tại) và **it_employees** (bảng trống dùng để lưu trữ thông tin nhân viên thuộc phòng IT). Hãy chèn thông tin của tất cả nhân viên thuộc phòng IT từ bảng employees vào bảng it_employees.

employees			
employee_id	**name**	**department**	**salary**
1	John Doe	HR	5000
2	Jane Smith	IT	7000
3	Mike Lee	IT	6000
4	Anna Taylor	Finance	8000

it_employees		
employee_id	**name**	**salary**

2.7 LƯU TRỮ VÀ SỬA ĐỔI DỮ LIỆU

c. THÊM HÀNG VÀ CẬP NHẬT DL VÀO BẢNG CSDL

- **VD34:** có bảng **employees** (lưu trữ thông tin nhân viên hiện tại) và **it_employees** (bảng trống dùng để lưu trữ thông tin nhân viên thuộc phòng IT). Hãy chèn thông tin của tất cả nhân viên thuộc phòng IT từ bảng employees vào bảng it_employees.

```
INSERT INTO it_employees (employee_id, name, salary)
SELECT employee_id, name, salary
FROM employees
WHERE department = 'IT';
```

employee_id	name	salary
2	Jane Smith	7000
3	Mike Lee	6000

CÂU HỎI TRẮC NGHIỆM

Câu 1: SQL JOIN nào trả về tất cả các bản ghi từ cả hai bảng, ngay cả khi không có kết quả khớp?

A. INNER JOIN
B. LEFT JOIN
C. RIGHT JOIN
D. FULL OUTER JOIN

Câu 2: Câu lệnh nào dùng để lọc dữ liệu sau khi nhóm dữ liệu bằng GROUP BY?

A. WHERE
B. HAVING
C. GROUP BY
D. LIMIT

CÂU HỎI TRẮC NGHIỆM

Câu 3: Window function nào được sử dụng để gán số thứ tự duy nhất cho từng hàng trong mỗi nhóm?

A. RANK()
B. ROW_NUMBER()
C. NTILE()
D. DENSE_RANK()

Câu 4: Kết quả của LAG() trong SQL là gì?

A. Giá trị của hang kế trước trong tập dữ liệu
B. Giá trị của hàng kế sau trong tập dữ liệu
C. Số thứ tự của hàng hiện tại
D. Tổng của tất cả các hàng trước đó

CÂU HỎI TRẮC NGHIỆM

Câu 5: Lệnh SQL nào được sử dụng để cộng thêm 10 ngày vào cột order_date?

A. DATE_ADD(order_date, INTERVAL 10 DAYS)

B. DATE_ADD(order_date, 10 DAYS)

C. ADD_DATE(order_date, 10)

D. DATE_ADD(order_date, INTERVAL 10 DAY)

Câu 6: Loại Constraint nào đảm bảo giá trị trong một cột là duy nhất?

A. NOT NULL

B. PRIMARY KEY

C. UNIQUE

D. FOREIGN KEY

CÂU HỎI TRẮC NGHIỆM

Câu 7: Khi nào nên sử dụng VIEW thay vì TABLE?

A. Khi cần lưu trữ dữ liệu dài hạn.
B. Khi cần lưu câu truy vấn và không cần lưu dữ liệu thực tế.
C. Khi dữ liệu không thay đổi.
D. Khi cần tạo một bảng vật lý mới.

Câu 8: Lệnh nào thêm một cột created_at với giá trị thời gian hiện tại vào bảng orders?

A. ALTER TABLE orders ADD created_at DATETIME DEFAULT CURRENT_DATE;

B. ALTER TABLE orders ADD created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP;

C. ADD COLUMN orders.created_at CURRENT_DATE;

D. ALTER orders ADD created_at TIMESTAMP;

TỔNG KẾT

- SQL JOIN cho phép kết hợp dữ liệu từ nhiều bảng dựa trên mối quan hệ giữa chúng, bao gồm: INNER JOIN, LEFT JOIN, RIGHT JOIN. Cần chú ý đến việc lọc dữ liệu sau khi JOIN để tránh kết quả không mong muốn.
- Hàm Aggregate thực hiện tính toán trên tập hợp các hàng dữ liệu và trả về một giá trị duy nhất như GROUP BY, SUM, COUNT, MIN, MAX,… HAVING dùng để lọc các nhóm sau khi đã được nhóm bởi GROUP BY; SQL CASE tạo ra các biểu thức điều kiện để phân loại / nhóm DL dựa trên các điều kiện cụ thể.
- Window Function thực hiện tính toán trên một tập hợp các hàng dữ liệu, nhưng vẫn giữ nguyên số lượng hàng ban đầu; Subqueries - truy vấn lồng nhau, được sử dụng để lọc, tính toán hoặc tạo ra dữ liệu tạm thời.
- Các hàm giúp làm việc với DL thời gian: EXTRACT VÀ DATE_PART, DATE_ADD và DATE_SUB, DATEDIFF, TIMESTAMPDIFF, …

TỔNG KẾT

- Việc tối ưu hóa truy vấn với INDEX và CONSTRAINTS: INDEX giúp tăng tốc độ truy vấn bằng cách tạo ra cấu trúc DL đặc biệt, CONSTRAINTS áp đặt các ràng buộc về DL trong bảng, đảm bảo tính toàn vẹn và nhất quán của DL.
- Cách thức lưu trữ và sửa đổi DL bằng VIEW (dành cho DL tạm thời) và TABLE (DL cố định).