Tuần 3 - Tổng hợp kiến thức Buổi học số 3

Time-Series Team

Ngày 23 tháng 6 năm 2025

Buổi học số 3 (Thứ 5, 19/06/2025) bao gồm 6 nội dung chính:

- ***Phần 1: SQL Join***
- ***Phần 2: Common Table Expression (CTE)***
- ***Phần 3: Subqueries***
- ***Phần 4: Temp Table***
- ***Phần 5: Store Procedures***
- ***Phần 6: Trigger***
- ***Phần 7: Recursive in SQL***

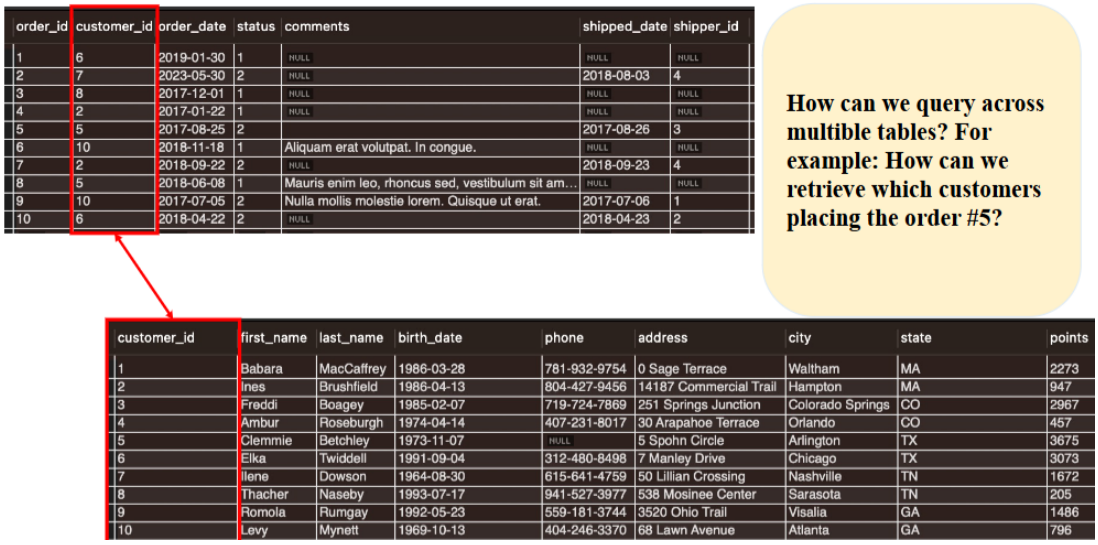
Phần 1: SQL Join

1. INNER JOIN

1.1. Đặt vấn đề

Trong một hệ thống quản lý bán hàng, dữ liệu khách hàng và đơn hàng không nằm trong một bảng duy nhất.

Vấn đề đặt ra: Làm sao để truy vấn dữ liệu từ nhiều bảng? Ví dụ: Khách hàng nào đã đặt đơn hàng #5?



order_id	customer_id	order_date	status	comments	shipped_date	shipper_id
1	6	2019-01-30	1	NULL	NULL	NULL
2	7	2023-05-30	2	NULL	2018-08-03	4
3	8	2017-12-01	1	NULL	NULL	NULL
4	2	2017-01-22	1	NULL	NULL	NULL
5	5	2017-08-25	2		2017-08-26	3
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL
7	2	2018-09-22	2	NULL	2018-09-23	4
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am...	NULL	NULL
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1
10	6	2018-04-22	2	NULL	2018-04-23	2

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
1	Babara	MacCaffrey	1986-03-28	781-932-9754	0 Sage Terrace	Waltham	MA	2273
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
3	Freddi	Boagey	1985-02-07	719-724-7869	251 Springs Junction	Colorado Springs	CO	2967
4	Ambur	Roseburgh	1974-04-14	407-231-8017	30 Arapahoe Terrace	Orlando	CO	457
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073
7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	TN	205
9	Romola	Rumgay	1992-05-23	559-181-3744	3520 Ohio Trail	Visalia	GA	1486
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796

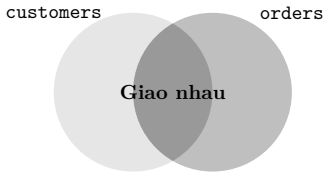
Nếu chỉ dùng bảng `orders` thì chúng ta sẽ không biết được thông tin khách hàng nào đã đặt hàng vì bảng này chỉ có thông tin đơn hàng. Vì vậy, cần phải **Join** với bảng `customers` để giải quyết được vấn đề đặt ra. Đây là lúc ta cần đến **INNER JOIN**.

1.2. INNER JOIN là gì ?

Giả sử:

- Tập A = `orders` (các đơn hàng).
- Tập B = `customers` (khách hàng).

Ta thực hiện phép giao giữa hai tập — chỉ giữ lại các bản ghi có `customer_id` trùng nhau ở cả hai bảng.



(Phần giao nhau giữa hai bảng — chính là kết quả INNER JOIN)

Khái niệm

- `INNER JOIN` là phép kết hợp **giao nhau** ($\cap$) giữa hai bảng.
- Chỉ những dòng thỏa mãn điều kiện `ON` mới được đưa vào kết quả.

—

1.3. Cú pháp

```sql
SELECT table1.column_a, table2.column_b
FROM table1
INNER JOIN table2
ON table1.primary_key = table2.foreign_key;
```

Ghi chú

- Dùng **alias** để viết ngắn gọn: `orders o`, `customers c`, ...
- Điều kiện trong `ON` là **trường chung** giữa hai bảng (thường là khóa chính – khóa ngoại).

—

1.4. Một số ví dụ

Ví dụ 1 — Lấy thông tin khách hàng đã đặt hàng

Từ nội dung kiến thức trên, ta đã có thể thực hiện được truy vấn **Lấy tên và họ của mọi khách hàng đặt hàng:**

```sql
SELECT *
FROM orders
INNER JOIN customers
ON orders.customer_id = customers.customer_id;
```

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id
1	6	2019-01-30	1	NULL	NULL	NULL
2	7	2023-05-30	2	NULL	2018-08-03	4
3	8	2017-12-01	1	NULL	NULL	NULL
4	2	2017-01-22	1	NULL	NULL	NULL
5	5	2017-08-25	2		2017-08-26	3
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL
7	2	2018-09-22	2	NULL	2018-09-23	4
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am...	NULL	NULL
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1
10	6	2018-04-22	2	NULL	2018-04-23	2

customer_id	first_name	last_name	birth_date	phone	address	city	state	points
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073
7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	TN	205
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	MA	947
5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	TX	3073

Quan sát kết quả, có thể thấy các dòng trong bảng `orders` đã được ghép nối tương ứng với thông tin khách hàng từ bảng `customers`, thông qua khóa `customer_id`.
Tuy nhiên, thay vì lấy toàn bộ các cột, ta có thể giới hạn truy vấn chỉ trả về các cột cần thiết như `order_id`, `first_name`, và `last_name`:

```sql
SELECT order_id, first_name, last_name
FROM orders
INNER JOIN customers
ON orders.customer_id = customers.customer_id;
```

Truy vấn này trả về danh sách các đơn hàng và tên khách hàng tương ứng. Như vậy, để trả lời cho câu hỏi ban đầu — *"Khách hàng nào đã đặt đơn hàng #5?"* — ta chỉ cần thêm điều kiện:

```sql
SELECT order_id, first_name, last_name
FROM orders
INNER JOIN customers
ON orders.customer_id = customers.customer_id
WHERE order_id = 5;
```

Sử dụng bí danh (alias) để truy vấn gọn hơn

Khi viết truy vấn JOIN nhiều bảng, việc dùng bí danh (alias) giúp giảm độ dài dòng lệnh và tăng khả năng đọc:

```sql
SELECT order_id, first_name, last_name
FROM orders o
INNER JOIN customers c
ON o.customer_id = c.customer_id;
```

Ví dụ 2 — Tính giá tiền từng sản phẩm

Giả sử:

- Tập C là bảng `order_items` (chi tiết đơn hàng).
- Tập D là bảng `products` (danh mục sản phẩm).

Khi đó, **INNER JOIN** giúp ghép thông tin sản phẩm (tên, đơn giá) với từng mặt hàng trong chi tiết đơn hàng.

```sql
SELECT oi.product_id, p.name, oi.quantity, oi.unit_price,
       quantity * oi.unit_price AS price
FROM order_items oi
JOIN products p
ON oi.product_id = p.product_id;
```

produc...	name	quantity	unit_price	price
1	Foam Dinner Plate	2	9.10	18.20
1	Foam Dinner Plate	4	8.65	34.60
1	Foam Dinner Plate	10	6.01	60.10
2	Pork - Bacon,back Peameal	3	9.89	29.67
2	Pork - Bacon,back Peameal	4	3.28	13.12
3	Lettuce - Romaine, Heart	10	9.12	91.20
3	Lettuce - Romaine, Heart	7	6.99	48.93
3	Lettuce - Romaine, Heart	4	7.46	29.84
3	Lettuce - Romaine, Heart	7	9.17	64.19
4	Brocolinni - Gaylan, Chinese	4	3.74	14.96
4	Brocolinni - Gaylan, Chinese	4	1.66	6.64
5	Sauce - Ranch Dressing	1	3.45	3.45
5	Sauce - Ranch Dressing	2	6.94	13.88
6	Petit Baguette	2	2.94	5.88
6	Petit Baguette	5	7.28	36.40
8	Island Oasis - Raspberry	2	8.59	17.18
9	Longan	9	4.28	38.52
10	Broom - Push	7	6.40	44.80

Kết quả truy vấn cho biết:

- Mỗi dòng là một mặt hàng cụ thể, kèm theo số lượng, đơn giá, và tổng tiền.
- Dữ liệu đầu ra rất hữu ích cho các mục đích như: tính doanh thu, xuất hóa đơn, hoặc kiểm kê hàng hóa.

Điểm mới trong truy vấn này:

- **Chỉ định rõ bảng nguồn của từng cột** bằng cú pháp `alias.column_name`, ví dụ:
  - `oi.product_id` lấy từ bảng `order_items` (alias `oi`).
  - `p.name` lấy từ bảng `products` (alias `p`).

  Việc này giúp:
  - Tránh nhầm lẫn nếu các bảng có tên cột giống nhau (như `product_id`).
  - Làm rõ nguồn gốc dữ liệu khi JOIN nhiều bảng.
  - Giúp truy vấn gọn gàng và dễ đọc hơn.
- **Tạo thêm thuộc tính `price`** bằng phép tính `quantity * oi.unit_price`. Đây là một *thuộc tính tạm* chỉ tồn tại trong kết quả truy vấn — không làm tăng dung lượng lưu trữ, nhưng rất tiện để xử lý logic trong câu lệnh SQL.

1.5. Tổng kết

- INNER JOIN giống phép giao giữa hai tập hợp.
- Chỉ lấy các bản ghi **có giá trị khóa trùng nhau ở cả hai bảng**.
- Thường sử dụng trên các cặp bảng có mối quan hệ khoá chính – khoá ngoại.
- Không lấy bản ghi nào từ một bảng nếu không có bản ghi tương ứng ở bảng kia.

2. SELF JOIN

2.1. Đặt vấn đề

Trong sơ đồ cơ sở dữ liệu `sql_hr`, bảng `employees` chứa thông tin nhân viên. Một trong các cột đáng chú ý là `reports_to`, thể hiện ID của người quản lý trực tiếp.

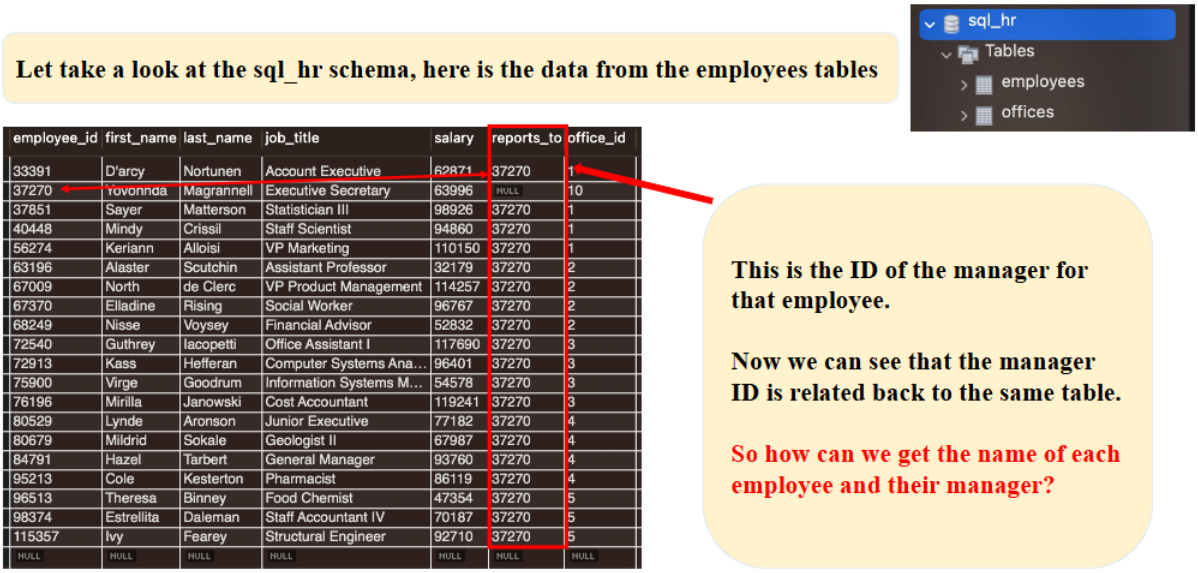
> **Vấn đề đặt ra: Làm thế nào để truy vấn ra cả tên nhân viên và tên người quản lý của họ, khi hai thông tin này cùng nằm trong một bảng?**

Ở đây, mối liên hệ xảy ra **nội tại** trong bảng: cột `reports_to` tham chiếu lại `employee_id` trong cùng một bảng. Đây chính là lúc ta sử dụng đến kỹ thuật **SELF JOIN**.
—

2.2. SELF JOIN là gì?

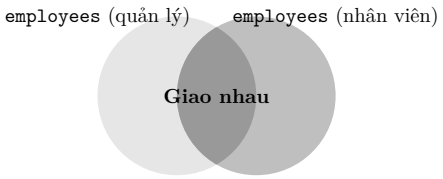
Giả sử:

- Tập A = `employees` (nhân viên).
- Tập B = `employees` (cũng là nhân viên, nhưng được xem như "quản lý").



employee_id	first_name	last_name	job_title	salary	reports_to	office_id
33391	D'arcy	Nortunen	Account Executive	62871	37270	1
37270	Yovonnda	Magrannell	Executive Secretary	63996	NULL	10
37851	Sayer	Matterson	Statistician III	98926	37270	1
40448	Mindy	Crissil	Staff Scientist	94860	37270	1
56274	Keriann	Alloisi	VP Marketing	110150	37270	1
63196	Alaster	Scutchin	Assistant Professor	32179	37270	2
67009	North	de Clerc	VP Product Management	114257	37270	2
67370	Elladine	Rising	Social Worker	96767	37270	2
68249	Nisse	Voysey	Financial Advisor	52832	37270	2
72540	Guthrey	Iacopetti	Office Assistant I	117690	37270	3
72913	Kass	Hefferan	Computer Systems Ana...	96401	37270	3
75900	Virge	Goodrum	Information Systems M...	54578	37270	3
76196	Mirilla	Janowski	Cost Accountant	119241	37270	3
80529	Lynde	Aronson	Junior Executive	77182	37270	4
80679	Mildrid	Sokale	Geologist II	67987	37270	4
84791	Hazel	Tarbert	General Manager	93760	37270	4
95213	Cole	Kesterton	Pharmacist	86119	37270	4
96513	Theresa	Binney	Food Chemist	47354	37270	5
98374	Estrellita	Daleman	Staff Accountant IV	70187	37270	5
115357	Ivy	Fearey	Structural Engineer	92710	37270	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Mặc dù chỉ có một bảng, ta vẫn coi như đang làm việc với hai phiên bản khác nhau của bảng đó. Ta thực hiện phép nối giữa hai tập — chỉ giữ lại các bản ghi sao cho `employee.reports_to = manager.employee_id`.



(Phần giao giữa nhân viên và người quản lý — chính là kết quả SELF JOIN)

Khái niệm

- `SELF JOIN` là phép nối một bảng với chính nó, để tìm mối quan hệ giữa các dòng dữ liệu bên trong cùng một bảng.
- Trong ví dụ này: mỗi nhân viên (`e`) được JOIN với người quản lý tương ứng của họ (`m`).
- Phép nối này chỉ giữ lại các bản ghi mà nhân viên thực sự có người quản lý (tức là `reports_to` không null).

—

2.3. Cú pháp

```sql
SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    m.employee_id AS manager_id,
    m.first_name,
    m.last_name
FROM employees e
```

```
JOIN employees m
    ON e.reports_to = m.employee_id;
```

—

2.4. Ví dụ — Lấy tên nhân viên và tên quản lý

Giả sử ta làm việc với CSDL `sql_hr`. Truy vấn sau sẽ trả về danh sách tất cả nhân viên cùng với thông tin người quản lý trực tiếp:

```
USE sql_hr;

SELECT
    e.employee_id,
    e.first_name,
    e.last_name,
    m.employee_id AS manager_id,
    m.first_name,
    m.last_name
FROM employees e
JOIN employees m
    ON e.reports_to = m.employee_id;
```

Kết quả truy vấn:

employee_id	first_name	last_name	manager_id	first_name	last_name
33391	D'arcy	Nortunen	37270	Yovonnda	Magrannell
37851	Sayer	Matterson	37270	Yovonnda	Magrannell
40448	Mindy	Crissil	37270	Yovonnda	Magrannell
56274	Keriann	Alloisi	37270	Yovonnda	Magrannell
63196	Alaster	Scutchin	37270	Yovonnda	Magrannell
67009	North	de Clerc	37270	Yovonnda	Magrannell
67370	Elladine	Rising	37270	Yovonnda	Magrannell
68249	Nisse	Voysey	37270	Yovonnda	Magrannell
72540	Guthrey	Iacopetti	37270	Yovonnda	Magrannell
72913	Kass	Hefferan	37270	Yovonnda	Magrannell
75900	Virge	Goodrum	37270	Yovonnda	Magrannell
76196	Mirilla	Janowski	37270	Yovonnda	Magrannell
80529	Lynde	Aronson	37270	Yovonnda	Magrannell
80679	Mildrid	Sokale	37270	Yovonnda	Magrannell
84791	Hazel	Tarbert	37270	Yovonnda	Magrannell
95213	Cole	Kesterton	37270	Yovonnda	Magrannell
96513	Theresa	Binney	37270	Yovonnda	Magrannell
98374	Estrellita	Daleman	37270	Yovonnda	Magrannell
115357	Ivy	Fearey	37270	Yovonnda	Magrannell

Ghi chú

- Cần dùng **alias** khác nhau cho bảng gốc và bảng JOIN (ở đây là `e` và `m`).
- SELF JOIN hoạt động như một INNER JOIN thông thường — chỉ trả về những dòng có `reports_to` hợp lệ (có người quản lý).
- Đây là cách hiệu quả để thể hiện mối quan hệ phân cấp (hierarchical relationship) trong cùng một bảng.

2.5. Tổng kết

- SELF JOIN dùng để truy vấn quan hệ giữa các dòng trong cùng một bảng.
- Rất hữu ích trong các bài toán phân cấp, ví dụ: nhân viên – quản lý, sản phẩm – danh mục cha.
- Về mặt kỹ thuật, nó vẫn là JOIN 2 bảng, nhưng cả hai đều là cùng một bảng nguồn.

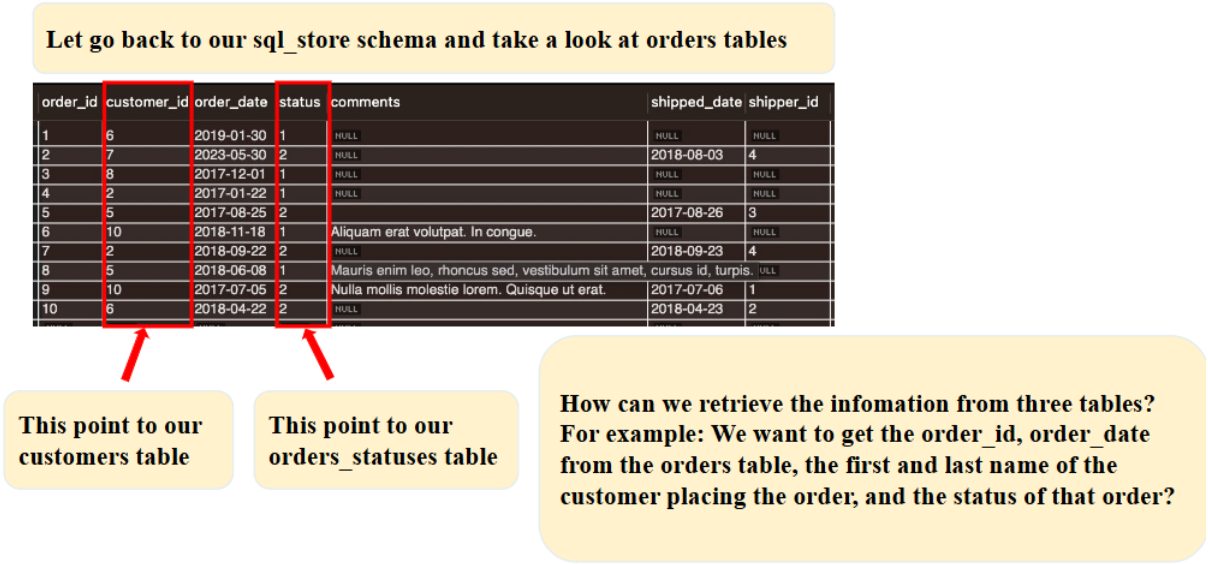
3. JOINNING MUTIPLE TABLES

3.1. Đặt vấn đề

Trong sơ đồ CSDL `sql_store`, bảng `orders` liên kết với nhiều bảng khác:

- `orders.customer_id` → bảng `customers`
- `orders.status` → bảng `order_statuses`

Vấn đề đặt ra: Làm sao để truy vấn thông tin từ cả ba bảng? Ví dụ: Lấy `order_id`, `order_date` từ bảng `orders`; Tên khách hàng từ bảng `customers`; Và trạng thái đơn hàng từ bảng `order_statuses`.



3.2. JOINING MULTIPLE TABLES - JOIN nhiều bảng là gì?

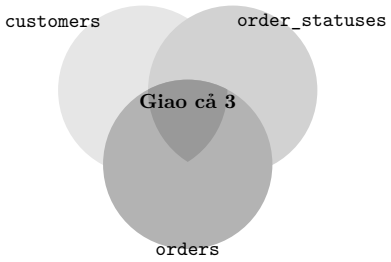
Giả sử:

- Tập A = `orders` (đơn hàng)
- Tập B = `customers` (khách hàng)
- Tập C = `order_statuses` (trạng thái đơn hàng)

Ta muốn truy vấn dữ liệu từ cả ba bảng — nghĩa là chỉ giữ lại những bản ghi mà:

- `orders.customer_id = customers.customer_id`
- `orders.status = order_statuses.order_status_id`

Khi đó, bản ghi xuất hiện trong kết quả phải đồng thời thỏa mãn điều kiện JOIN với cả hai bảng còn lại.



(Chỉ các bản ghi thuộc phần giao giữa cả 3 bảng mới được đưa vào kết quả)

Khái niệm

- `JOINING MULTIPLE TABLES` là việc thực hiện nhiều phép `JOIN` liên tiếp để kết nối từ 3 bảng trở lên.
- Mỗi lần JOIN tương ứng với một điều kiện logic giữa hai bảng.
- Chỉ các bản ghi thỏa mãn toàn bộ điều kiện JOIN mới xuất hiện trong kết quả.

—

3.3. Một số ví dụ

Ví dụ 1 — Truy vấn đơn hàng từ 3 bảng

Làm việc với CSDL `sql_store`. Yêu cầu: Lấy thông tin đơn hàng gồm mã đơn, ngày đặt, tên khách hàng và trạng thái đơn.

```sql
USE sql_store;

SELECT
    o.order_id,
    o.order_date,
    c.last_name,
    c.first_name,
    s.name AS status
FROM orders o
JOIN customers c
    ON o.customer_id = c.customer_id
JOIN order_statuses s
    ON o.status = s.order_status_id;
```

Giải thích:

- `o.customer_id = c.customer_id`: nối mỗi đơn hàng với thông tin khách hàng tương ứng.
- `o.status = s.order_status_id`: lấy tên trạng thái đơn hàng từ bảng trạng thái.

Kết quả truy vấn:

order_id	order_date	last_name	first_name	status
8	2018-06-08	Betchley	Clemmie	Processed
6	2018-11-18	Mynett	Levy	Processed
4	2017-01-22	Brushfield	Ines	Processed
3	2017-12-01	Naseby	Thacher	Processed
1	2019-01-30	Twiddell	Elka	Processed
10	2018-04-22	Twiddell	Elka	Shipped
9	2017-07-05	Mynett	Levy	Shipped
7	2018-09-22	Brushfield	Ines	Shipped
5	2017-08-25	Betchley	Clemmie	Shipped
2	2018-08-02	Dowson	Ilene	Shipped

Ví dụ 2 — Truy vấn thông tin thanh toán từ 4 bảng

Làm việc với CSDL `sql_invoicing`. Yêu cầu: Truy vấn chi tiết thanh toán gồm:

- Mã thanh toán, ngày, số tiền (`payments`)
- Tên khách hàng, số điện thoại (`clients`)
- Số hoá đơn, tổng tiền, số tiền đã trả (`invoices`)
- Tên phương thức thanh toán (`payment_methods`)

```sql
USE sql_invoicing;

SELECT
    p.payment_id, p.date, p.amount,
    c.name, c.phone,
    i.number, i.invoice_total, i.payment_total,
    pm.name AS payment_method
FROM payments p
JOIN clients c
    ON p.client_id = c.client_id
JOIN invoices i
    ON p.invoice_id = i.invoice_id
JOIN payment_methods pm
    ON p.payment_method = pm.payment_method_id;
```

Ghi chú:

- JOIN tuần tự từng bảng, mỗi bảng dùng đúng cặp khóa liên kết.
- Kỹ thuật này giúp tập hợp dữ liệu phân tán thành một kết quả duy nhất, rất hữu ích khi làm báo cáo.

Kết quả truy vấn:

payment_id	date	amount	name	phone	number	invoice_total	payment_total	payment_method
1	2019-02-12	8.18	Topiclounge	971-888-9129	03-898-6735	175.32	8.18	Credit Card
2	2019-01-03	74.55	Vinte	315-252-7305	75-587-6626	157.78	74.55	Credit Card
3	2019-01-11	0.03	Yadel	415-144-6037	20-848-0181	126.15	0.03	Credit Card
4	2019-01-26	87.44	Topiclounge	971-888-9129	41-666-1035	135.01	87.44	Credit Card
5	2019-01-15	80.31	Yadel	415-144-6037	55-105-9605	167.29	80.31	Credit Card
6	2019-01-15	68.10	Yadel	415-144-6037	33-615-4694	126.38	68.10	Credit Card
7	2019-01-08	32.77	Topiclounge	971-888-9129	52-269-9803	180.17	42.77	Credit Card
8	2019-01-08	10.00	Topiclounge	971-888-9129	52-269-9803	180.17	42.77	Cash

3.4. Tổng kết

- Có thể JOIN nhiều bảng bằng cách nối từng bảng một theo thứ tự logic.
- Mỗi lần JOIN cần chỉ rõ điều kiện ON giữa hai bảng.
- Nên sử dụng alias để rút gọn truy vấn và tránh nhầm lẫn.
- JOIN nhiều bảng thường được áp dụng trong các truy vấn báo cáo, thống kê tổng hợp dữ liệu.

4. COMPOUND JOIN CONDITION

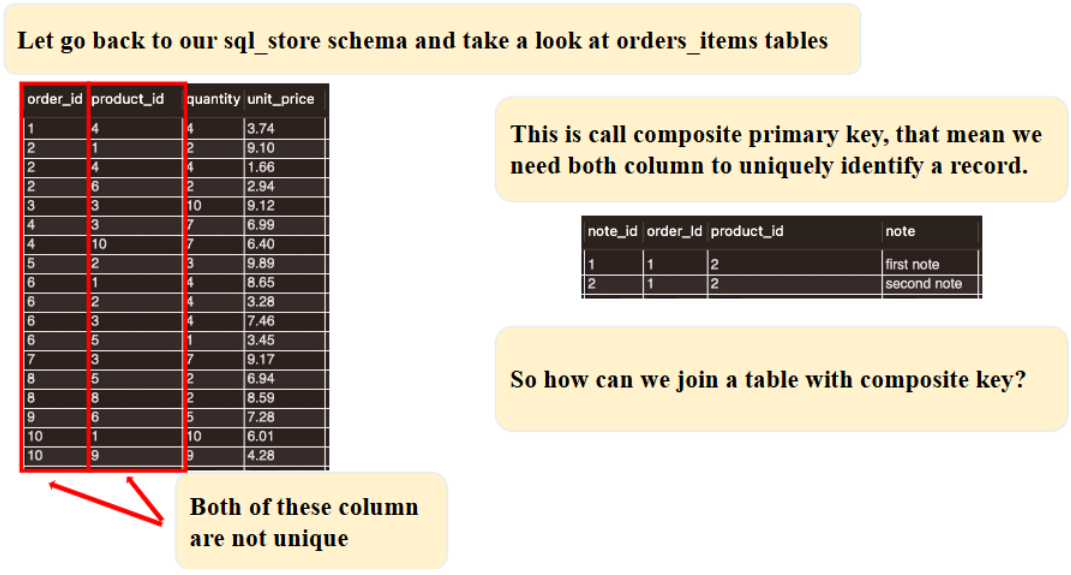
4.1. Đặt vấn đề

Trong sơ đồ CSDL `sql_store`, bảng `order_items` có các cột:

- `order_id`
- `product_id`

Cả hai cột này **không phải khóa chính riêng lẻ**, nhưng kết hợp lại thì tạo thành một khóa chính tổng hợp (composite primary key).

Vấn đề đặt ra: Làm sao để JOIN bảng `order_item_notes` với `order_items`, khi mỗi bảng phải dựa trên cả hai cột `order_id` và `product_id` mới xác định được bản ghi?



4.2. COMPOUND JOIN CONDITION là gì?

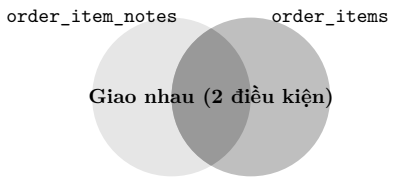
Giả sử:

- Tập A = `order_items` (chi tiết đơn hàng).
- Tập B = `order_item_notes` (ghi chú từng dòng sản phẩm).

Ta thực hiện phép giao giữa hai tập — nhưng thay vì chỉ dựa trên một cột khóa, ta cần kết hợp **hai điều kiện**:

`order_id` **và** `product_id`

Điều này là cần thiết vì một đơn hàng có thể chứa nhiều sản phẩm khác nhau, nên `order_id` chưa đủ để xác định bản ghi duy nhất.



(Phần giao giữa hai bảng khi thoả mãn đồng thời cả `order_id` và `product_id`)

Khái niệm

- `COMPOUND JOIN CONDITION` là phép JOIN có nhiều hơn một điều kiện trong mệnh đề `ON`.
- Thường dùng với các bảng có khóa chính tổng hợp (composite key), ví dụ: `(order_id, product_id)`.
- Tất cả các điều kiện phải đồng thời đúng thì bản ghi mới xuất hiện trong kết quả JOIN.

4.3. Ví dụ

Yêu cầu: Truy vấn toàn bộ thông tin từ bảng `order_items` kết hợp với nội dung ghi chú tương ứng trong bảng `order_item_notes`.

```sql
USE sql_store;

SELECT *
FROM order_items o
JOIN order_item_notes n
    ON o.order_id = n.order_id
    AND o.product_id = n.product_id;
```

Giải thích:

- `o.order_id = n.order_id` → JOIN theo mã đơn hàng.
- `o.product_id = n.product_id` → JOIN theo mã sản phẩm trong đơn.
- Dùng dấu `AND` trong mệnh đề `ON` để kết hợp hai điều kiện thành một **COMPOUND JOIN CONDITION**.

Kết quả truy vấn:

order_id	product_id	quantity	unit_price	note_id	order_id	product_id	note
2	6	2	2.94	1	2	6	first note
5	2	3	9.89	2	5	2	second note

(Hai dòng dữ liệu được ghép đúng theo cả `order_id` và `product_id`)

4.4. Tổng kết

- COMPOUND JOIN CONDITION rất quan trọng khi hai bảng liên kết qua nhiều trường.
- Thường gặp trong các bảng có khóa chính là tổ hợp nhiều cột.
- Mỗi điều kiện JOIN là một phần của logic kết nối — thiếu một là sai quan hệ.

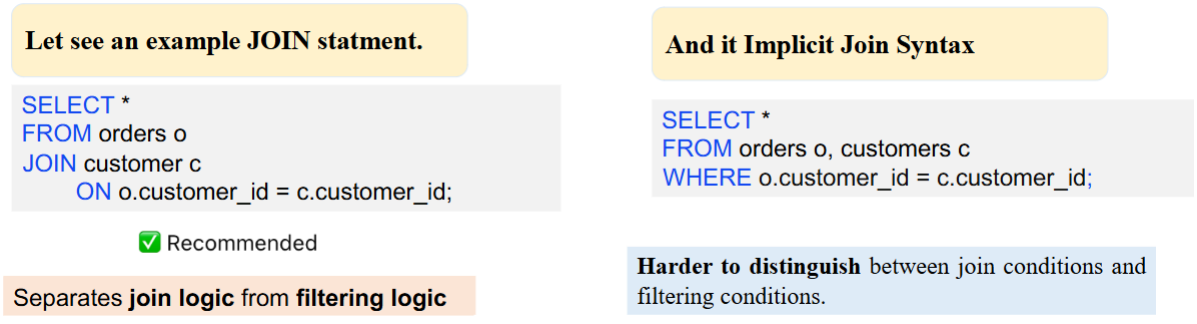
5. IMPLICIT JOIN SYNTAX

5.1. Đặt vấn đề

Trong SQL, có hai cách để thực hiện phép `JOIN`:

- Dùng từ khóa `JOIN ... ON` — gọi là **explicit join** (rõ ràng).
- Dùng từ khóa `FROM ... WHERE` — gọi là **implicit join** (ngầm định).

Vấn đề đặt ra: Hai cú pháp này khác nhau ở điểm nào? Tại sao cú pháp `JOIN ... ON` lại được khuyến nghị?



5.2. IMPLICIT JOIN SYNTAX là gì?

Implicit Join là cú pháp nối bảng trong SQL bằng cách liệt kê nhiều bảng trong mệnh đề `FROM`, rồi viết điều kiện nối ở mệnh đề `WHERE`.
Nó thường được gọi là **cú pháp nối ngầm định** (implicit), vì không dùng từ khóa `JOIN`.

Giả sử ta cần truy vấn dữ liệu từ hai bảng `orders` và `customers`, dựa theo cột `customer_id` chung giữa hai bảng. Có hai cách viết:

Explicit Join Syntax (Khuyến nghị):

```sql
SELECT *
FROM orders o
JOIN customers c
ON o.customer_id = c.customer_id;
```

Khuyến nghị: Cách viết rõ ràng và tách biệt phần logic `JOIN` khỏi điều kiện lọc `WHERE`.

Implicit Join Syntax (Không khuyến nghị):

```sql
SELECT *
FROM orders o, customers c
WHERE o.customer_id = c.customer_id;
```

Khó phân biệt giữa điều kiện `JOIN` và điều kiện lọc trong `WHERE`. Có thể gây lỗi khi truy vấn phức tạp.

5.3. So sánh hai cú pháp

- Cú pháp `JOIN ... ON` giúp:
  - Tách biệt logic liên kết bảng với logic lọc.
  - Tránh nhầm lẫn khi kết hợp với các điều kiện `WHERE`.
  - Đọc code dễ hơn, dễ bảo trì.
- Cú pháp `FROM ... WHERE` chỉ còn dùng trong mã cũ hoặc khi học SQL cơ bản.

5.4. Kết quả minh họa

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	customer_id	order_date	status	comments	shipped_date	shipper_id	customer_id	first_name	last_name	birth_date	phone	address	city	state	points
1	6	2019-01-30	1	NULL	NULL	NULL	6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	IL	3073
2	7	2018-08-02	2	NULL	2018-08-03	4	7	Ilene	Dowson	1964-08-30	615-641-4759	50 Lillian Crossing	Nashville	TN	1672
3	8	2017-12-01	1	NULL	NULL	NULL	8	Thacher	Naseby	1993-07-17	941-527-3977	538 Mosinee Center	Sarasota	FL	205
4	2	2017-01-22	1	NULL	NULL	NULL	2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	VA	947
5	5	2017-08-25	2		2017-08-26	3	5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
6	10	2018-11-18	1	Aliquam erat volutpat. In congue.	NULL	NULL	10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
7	2	2018-09-22	2	NULL	2018-09-23	4	2	Ines	Brushfield	1986-04-13	804-427-9456	14187 Commercial Trail	Hampton	VA	947
8	5	2018-06-08	1	Mauris enim leo, rhoncus sed, vestibulum sit am...	NULL	NULL	5	Clemmie	Betchley	1973-11-07	NULL	5 Spohn Circle	Arlington	TX	3675
9	10	2017-07-05	2	Nulla mollis molestie lorem. Quisque ut erat.	2017-07-06	1	10	Levy	Mynett	1969-10-13	404-246-3370	68 Lawn Avenue	Atlanta	GA	796
10	6	2018-04-22	2	NULL	2018-04-23	2	6	Elka	Twiddell	1991-09-04	312-480-8498	7 Manley Drive	Chicago	IL	3073

(Cả hai cú pháp đều trả về cùng kết quả, nhưng explicit join rõ ràng và dễ bảo trì hơn)

5.5. Tổng kết

- `Implicit Join` sử dụng mệnh đề `WHERE` để nối bảng, nhưng dễ gây nhầm lẫn.
- `Explicit Join` dùng `JOIN ... ON`, rõ ràng và tách biệt logic.
- Trong thực tế và khi làm việc nhóm, luôn ưu tiên dùng `JOIN ... ON`.

6. OUTER JOIN

6.1. Đặt vấn đề

Khi dùng `INNER JOIN`, chỉ các bản ghi thỏa mãn điều kiện mới được giữ lại.

> **Vấn đề đặt ra:** Làm sao để lấy tất cả khách hàng, kể cả người chưa từng đặt hàng? Hoặc lấy tất cả đơn hàng, kể cả khi không biết khách hàng là ai?

Câu trả lời: Ta dùng `OUTER JOIN` — một kỹ thuật mở rộng kết quả JOIN bằng cách giữ lại các bản ghi không khớp.

—

6.2. Khái niệm OUTER JOIN

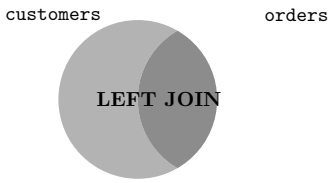
Khái niệm

- `OUTER JOIN` giữ lại các bản ghi **không khớp** ở một phía.
- Gồm 2 dạng chính:
  - `LEFT JOIN`: giữ toàn bộ bản ghi bên trái.
  - `RIGHT JOIN`: giữ toàn bộ bản ghi bên phải.
- Các cột không khớp được điền bằng `NULL`.

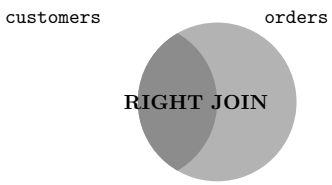
—

6.3. Minh họa trực quan

LEFT JOIN:



RIGHT JOIN:



—

6.4. Ví dụ 1 — LEFT JOIN tất cả khách hàng

Yêu cầu: Lấy toàn bộ khách hàng, kể cả khi họ không đặt đơn hàng nào.

```sql
SELECT
    c.customer_id,
    c.first_name,
    o.order_id
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
ORDER BY c.customer_id;
```

Giải thích:

- Khách hàng không có đơn hàng sẽ được điền `NULL` ở cột `order_id`.
- LEFT JOIN giữ lại tất cả bản ghi của bảng `customers`.

—

6.5. Ví dụ 2 — RIGHT JOIN tất cả đơn hàng

Yêu cầu: Lấy toàn bộ đơn hàng, kể cả khi không biết khách hàng là ai.

```sql
SELECT
    c.customer_id,
    c.first_name,
    o.order_id
FROM customers c
RIGHT JOIN orders o
    ON c.customer_id = o.customer_id
ORDER BY c.customer_id;
```

Giải thích:

- Đơn hàng không có khách hàng vẫn xuất hiện — cột thông tin khách hàng sẽ là `NULL`.
- RIGHT JOIN giữ lại tất cả bản ghi của bảng `orders`.

—

6.6. Ví dụ 3 — LEFT JOIN 3 bảng

Yêu cầu: Truy vấn toàn bộ khách hàng, đơn hàng (nếu có), và tên shipper cho mỗi đơn hàng (nếu có).

```sql
SELECT
    c.customer_id,
    c.first_name,
    o.order_id,
    sh.name AS shipper
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
LEFT JOIN shippers sh
    ON o.shipper_id = sh.shipper_id
ORDER BY c.customer_id;
```

Ghi chú:

- Mỗi lần JOIN vẫn cần chỉ định điều kiện rõ ràng.
- Các bảng có thể không có dữ liệu khớp — các cột đó sẽ là `NULL`.

—

6.7. Tổng kết

- `OUTER JOIN` giúp giữ lại dữ liệu "không khớp" — rất hữu ích trong báo cáo tổng hợp.
- `LEFT JOIN`: giữ toàn bộ bản ghi bên trái.
- `RIGHT JOIN`: giữ toàn bộ bản ghi bên phải.
- Luôn chú ý: các cột không khớp sẽ có giá trị `NULL`.

7. USING CLAUSE

6.1. Đặt vấn đề

Trong SQL, khi sử dụng phép `JOIN`, nếu hai bảng có cùng tên cột để liên kết, ta thường viết điều kiện JOIN như sau:

```
SELECT
    c.customer_id,
    c.first_name,
    o.order_id
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id;
```

Tuy nhiên, cú pháp này có thể rút gọn khi cả hai cột dùng để JOIN có cùng tên.

6.2. USING CLAUSE là gì?

Khái niệm

`USING` là một từ khóa trong SQL cho phép rút gọn cú pháp `JOIN` khi hai bảng có cùng tên cột dùng để liên kết.
Thay vì viết điều kiện đầy đủ trong `ON`, ta có thể dùng:

```
JOIN <table_name> USING (<common_column>)
```

Kết quả vẫn giữ nguyên, nhưng cú pháp gọn gàng và dễ đọc hơn.

Ví dụ 1 — Rút gọn INNER JOIN thông thường
Cách viết đầy đủ:

```
SELECT
    c.customer_id,
    c.first_name,
```

```
    o.order_id
FROM customers c
JOIN orders o
    ON c.customer_id = o.customer_id;
```

Rút gọn với USING:

```
SELECT
    c.customer_id,
    c.first_name,
    o.order_id
FROM customers c
JOIN orders o
    USING (customer_id);
```

Hai cách viết trên hoàn toàn tương đương về kết quả, nhưng `USING` giúp giảm lặp lại tên bảng.

Ví dụ 2 — Dùng trong Compound Join Condition

Khi cần JOIN nhiều cột cùng tên giữa hai bảng, `USING` đặc biệt hữu ích để viết gọn điều kiện JOIN.

Cách viết đầy đủ:

```
SELECT *
FROM order_items o
JOIN order_item_notes n
    ON o.order_id = n.order_id
    AND o.product_id = n.product_id;
```

Rút gọn với USING:

```
SELECT *
FROM order_items o
JOIN order_item_notes n
    USING (order_id, product_id);
```

6.3. Tổng kết

- `USING` cho phép viết JOIN gọn hơn khi các cột liên kết có cùng tên ở cả hai bảng.
- Hữu ích nhất trong các truy vấn có nhiều điều kiện JOIN như `compound key`.
- Tuy nhiên, chỉ dùng được khi tên cột giống nhau tuyệt đối.

8. UNION

8.1. Đặt vấn đề

Giả sử trong bảng `customers`, bạn muốn xếp hạng khách hàng theo mức điểm:

- $<$ 2000 điểm $\rightarrow$ `Bronze`
- 2000–3000 điểm $\rightarrow$ `Silver`
- $>$ 3000 điểm $\rightarrow$ `Gold`

Vấn đề đặt ra: Làm sao để gộp kết quả từ nhiều truy vấn con tương ứng với từng hạng, và hiển thị thành một bảng duy nhất?

Let see the result of this query.

```
SELECT customer_id, first_name, points
FROM customers;
```

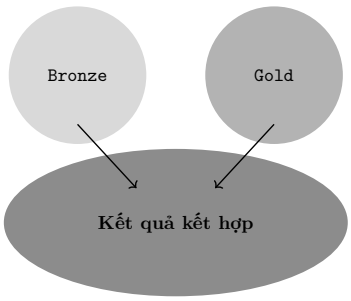
customer_id	first_name	points
1	Babara	2273
2	Ines	947
3	Freddi	2967
4	Ambur	457
5	Clemmie	3675
6	Elka	3073
7	Ilene	1672
8	Thacher	205
9	Romola	1486
10	Levy	796

What if we tank to rank them by tier:
- **<2000 point: Bronze**
- **2000-3000 point: Silver**
- **>3000 point: Gold**

customer_id	first_name	points	tier
4	Ambur	457	Bronze
1	Babara	2273	Silver
5	Clemmie	3675	Gold
6	Elka	3073	Gold
3	Freddi	2967	Silver
7	Ilene	1672	Bronze
2	Ines	947	Bronze
10	Levy	796	Bronze
9	Romola	1486	Bronze
8	Thacher	205	Bronze

8.2. UNION là gì?

`UNION` là một toán tử trong SQL dùng để kết hợp kết quả của hai (hoặc nhiều) truy vấn con lại với nhau — tạo thành một bảng kết quả duy nhất.



Khái niệm

- `UNION` kết hợp nhiều kết quả SELECT lại thành một bảng duy nhất.
- Các truy vấn con phải có cùng số cột và kiểu dữ liệu tương ứng.
- **Mặc định loại bỏ bản ghi trùng lặp** (distinct).

8.3. Ví dụ — Xếp hạng khách hàng

```sql
SELECT customer_id, first_name, points, 'Bronze' AS tier
FROM customers
WHERE points < 2000

UNION

SELECT customer_id, first_name, points, 'Silver' AS tier
FROM customers
WHERE points BETWEEN 2000 AND 3000

UNION

SELECT customer_id, first_name, points, 'Gold' AS tier
FROM customers
WHERE points > 3000;
```

Giải thích:

- Mỗi truy vấn chọn dữ liệu từ nhóm điểm khác nhau và gán nhãn `tier`.
- `UNION` gộp tất cả lại thành một bảng duy nhất gồm 4 cột.
- Nếu một khách hàng bị trùng ở nhiều truy vấn (rất hiếm), kết quả vẫn chỉ hiển thị một dòng.

7.4. UNION ALL là gì?

Phân biệt `UNION` và `UNION ALL`

- `UNION` loại bỏ các dòng trùng lặp (DISTINCT).
- `UNION ALL` giữ lại toàn bộ kết quả, **kể cả trùng lặp**.

Ví dụ:

```sql
SELECT 'Alice' AS name
UNION
SELECT 'Alice' AS name;  -- Only 1 row is returned
```

```sql
SELECT 'Alice' AS name
UNION ALL
SELECT 'Alice' AS name;  -- Returns 2 rows
```

Ghi chú:

- Dùng `UNION` khi muốn **loại bỏ các dòng trùng lặp** trong kết quả. Tuy nhiên, vì cần kiểm tra trùng lặp nên câu truy vấn sẽ **chậm hơn** do phải **sắp xếp (sort)** dữ liệu nội bộ.
- Dùng `UNION ALL` khi muốn **giữ nguyên toàn bộ kết quả**, bao gồm cả các dòng bị trùng. Lúc này, truy vấn sẽ **nhanh hơn** vì **không cần sắp xếp hay kiểm tra trùng lặp**.

7.5. Tổng kết

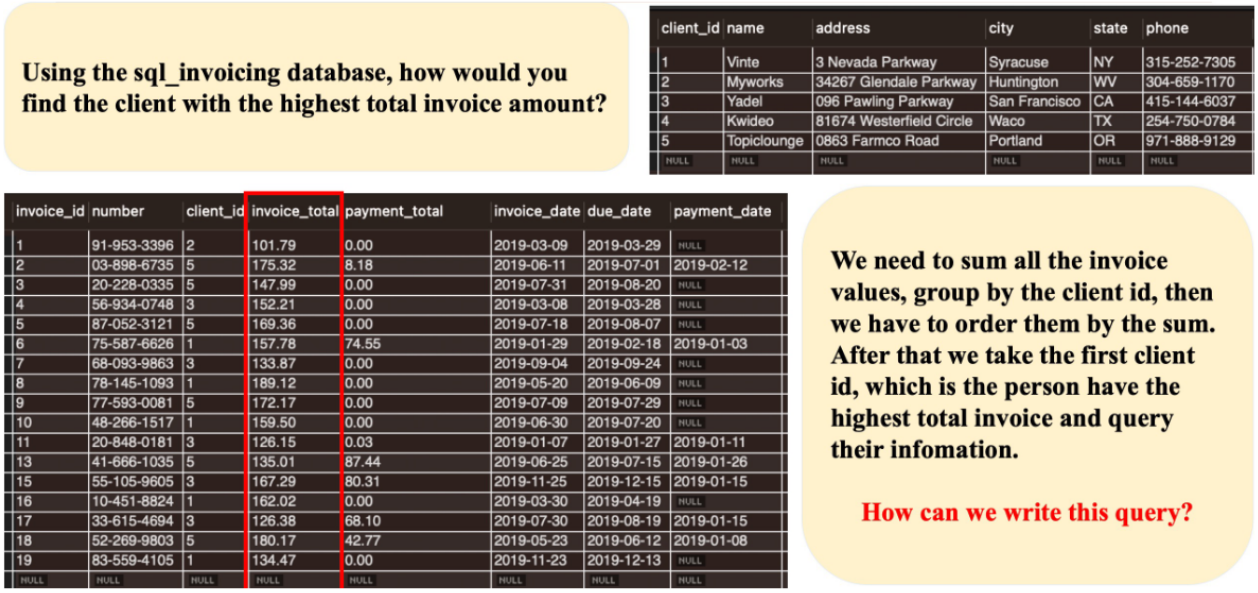
- `UNION` và `UNION ALL` dùng để gộp nhiều truy vấn SELECT lại thành một bảng duy nhất.
- `UNION` loại bỏ trùng, `UNION ALL` thì không.
- Cần đảm bảo các truy vấn con có cùng số cột và kiểu dữ liệu tương ứng.

Phần 2: Common Table Expression (CTE)

1. Đặt vấn đề — Ai là khách hàng có tổng hóa đơn cao nhất?

Bài toán: Sử dụng CSDL `sql_invoicing`, hãy tìm **khách hàng có tổng giá trị hóa đơn lớn nhất**. Sau đó, lấy thông tin khách hàng tương ứng.

Quan sát bảng dữ liệu:



client_id	name	address	city	state	phone
1	Vinte	3 Nevada Parkway	Syracuse	NY	315-252-7305
2	Myworks	34267 Glendale Parkway	Huntington	WV	304-659-1170
3	Yadel	096 Pawling Parkway	San Francisco	CA	415-144-6037
4	Kwideo	81674 Westerfield Circle	Waco	TX	254-750-0784
5	Topiclounge	0863 Farmco Road	Portland	OR	971-888-9129

invoice_id	number	client_id	invoice_total	payment_total	invoice_date	due_date	payment_date
1	91-953-3396	2	101.79	0.00	2019-03-09	2019-03-29	NULL
2	03-898-6735	5	175.32	8.18	2019-06-11	2019-07-01	2019-02-12
3	20-228-0335	5	147.99	0.00	2019-07-31	2019-08-20	NULL
4	56-934-0748	3	152.21	0.00	2019-03-08	2019-03-28	NULL
5	87-052-3121	5	169.36	0.00	2019-07-18	2019-08-07	NULL
6	75-587-6626	1	157.78	74.55	2019-01-29	2019-02-18	2019-01-03
7	68-093-9863	3	133.87	0.00	2019-09-04	2019-09-24	NULL
8	78-145-1093	1	189.12	0.00	2019-05-20	2019-06-09	NULL
9	77-593-0081	5	172.17	0.00	2019-07-09	2019-07-29	NULL
10	48-266-1517	1	159.50	0.00	2019-06-30	2019-07-20	NULL
11	20-848-0181	3	126.15	0.03	2019-01-07	2019-01-27	2019-01-11
13	41-666-1035	5	135.01	87.44	2019-06-25	2019-07-15	2019-01-26
15	55-105-9605	3	167.29	80.31	2019-11-25	2019-12-15	2019-01-15
16	10-451-8824	1	162.02	0.00	2019-03-30	2019-04-19	NULL
17	33-615-4694	3	126.38	68.10	2019-07-30	2019-08-19	2019-01-15
18	52-269-9803	5	180.17	42.77	2019-05-23	2019-06-12	2019-01-08
19	83-559-4105	1	134.47	0.00	2019-11-23	2019-12-13	NULL

Ta nhận thấy cần các bước:

1. Tính tổng hóa đơn theo từng khách hàng.
2. Sắp xếp giảm dần theo tổng hóa đơn.
3. Lấy ra **khách hàng đầu tiên** và truy vấn thông tin tương ứng từ bảng `clients`.

2. CTE là gì?

CTE (Common Table Expression) là một biểu thức tạm, giúp ta định nghĩa một bảng ảo ngay trong truy vấn SQL.
Cú pháp chung:

```
WITH cte_name AS (
    -- Subquery
)
SELECT ...
FROM cte_name;
```

Hiểu đơn giản: CTE giống như một "biến bảng tạm thời", được tạo ra để giúp viết truy vấn chính gọn gàng, dễ hiểu hơn. Nó thường được dùng để:

- Chia nhỏ truy vấn phức tạp.
- Dễ đọc, dễ bảo trì và debug.
- Dùng lại nhiều lần nếu có nhiều CTE kết hợp.

3. Truy vấn cơ bản — Không dùng CTE

Một cách tiếp cận đơn giản: **JOIN** hai bảng `clients` và `invoices`, sau đó `GROUP BY` và `LIMIT`:

```sql
SELECT c.name
FROM clients c JOIN invoices i
USING (client_id)
GROUP BY i.client_id
ORDER BY SUM(i.invoice_total) DESC
LIMIT 1;
```

Kết quả: Lấy ra khách hàng có tổng `invoice_total` cao nhất — ở đây là **Topiclounge**.

name	client_id	invoice_amount
Topiclounge	5	980.02
Vinte	1	802.89
Yadel	3	705.90
Myworks	2	101.79

4. Dùng Common Table Expression (CTE)

CTE là một công cụ mạnh để chia nhỏ truy vấn phức tạp thành nhiều bước rõ ràng hơn.
Ý tưởng: Truy vấn con để lấy `client_id` có tổng hóa đơn cao nhất trước, sau đó dùng truy vấn chính để lấy thông tin chi tiết.

Cách viết với CTE:

```sql
WITH invoice_amount AS (
    SELECT client_id,
           SUM(invoice_total) AS invoice_amount
    FROM invoices
    GROUP BY client_id
    ORDER BY SUM(invoice_total) DESC
    LIMIT 1
)
SELECT c.client_id, c.name, i.invoice_amount
FROM clients c JOIN invoice_amount i
USING (client_id);
```

Ưu điểm của CTE:

- Chia nhỏ truy vấn phức tạp thành các bước dễ hiểu.
- Tăng khả năng tái sử dụng và bảo trì.
- Cấu trúc rõ ràng hơn so với truy vấn lồng nhau.

name	client_id	invoice_amount
Topiclounge	5	980.02
Vinte	1	802.89
Yadel	3	705.90
Myworks	2	101.79

5. Tổng kết

- **CTE** (Common Table Expression) là cách đặt tên tạm cho truy vấn con, giúp chia nhỏ và làm rõ ý đồ truy vấn.
- Trong bài toán tìm khách hàng có tổng hóa đơn cao nhất, CTE giúp bạn biểu diễn logic "chia để trị" cực kỳ rõ ràng.
- Hãy ưu tiên dùng CTE nếu bạn cảm thấy câu truy vấn quá dài, khó hiểu hoặc có thể tái sử dụng nhiều lần.

Phần 3: Subqueries

1. Subquery — Một lựa chọn khác thay cho JOIN và CTE

Giả sử bạn muốn tính tổng hóa đơn hoặc trung bình hóa đơn cho từng khách hàng, nhưng không muốn dùng JOIN hoặc WITH/CTE? Đó là lúc chúng ta sử dụng kỹ thuật **Subquery**.

Đặt vấn đề: Với CSDL `sql_invoicing`, hãy:

- Tính tổng số tiền hoá đơn của từng khách hàng.
- Sắp xếp giảm dần theo tổng hóa đơn.
- Trả về thông tin của khách hàng có tổng hóa đơn cao nhất.

2. Subquery trong SELECT — Dùng SUM

Ý tưởng: Trong phần `SELECT`, chèn một truy vấn con (subquery) để tính tổng hoá đơn tương ứng với từng khách hàng:

```sql
SELECT client_id,
       name,
       (SELECT SUM(invoice_total)
        FROM invoices
        WHERE c.client_id = client_id) AS invoice_amount
FROM clients c
ORDER BY invoice_amount DESC
LIMIT 1;
```

Subquery sẽ được tính trước — với mỗi khách hàng trong bảng `clients`, ta chạy truy vấn phụ để tính tổng `invoice_total` trong bảng `invoices`.

3. Subquery trong SELECT — Dùng AVG

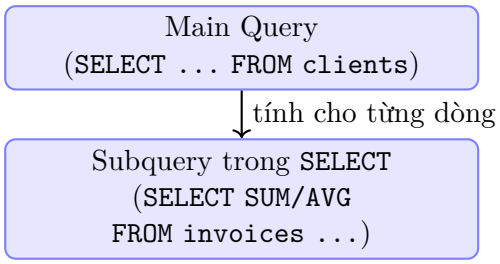
Một biến thể khác: dùng `AVG` thay vì `SUM` để tính trung bình số tiền hóa đơn của từng khách hàng:

```sql
SELECT client_id,
       (SELECT AVG(invoice_total)
        FROM invoices
        WHERE client_id = c.client_id) AS avg_invoice
FROM clients c;
```

client_id	avg_invoice
1	160.578000
2	101.790000
3	141.180000
4	NULL
5	163.336667

Đây là một trong những cách viết SQL ngắn gọn, giúp **tránh việc dùng JOIN hoặc GROUP BY**, nhưng không phải lúc nào cũng tối ưu hiệu suất.

4. Minh họa trực quan Subquery lồng nhiều lớp



5. Lưu ý khi dùng Subquery

- Mỗi lần duyệt 1 dòng trong truy vấn chính, subquery sẽ được thực thi riêng — dễ gây chậm với dữ liệu lớn.
- Subquery hoạt động tốt cho các truy vấn đơn giản, dễ đọc.
- Nếu phải dùng lại subquery nhiều lần → nên cân nhắc dùng `CTE` hoặc `JOIN`.

6. Tổng kết

Subquery là một công cụ mạnh mẽ khi viết SQL — giúp bạn tính toán động ngay trong phần `SELECT`, `FROM`, hoặc `WHERE`.
Nó là cách đơn giản để:

- Viết nhanh các logic tính toán cho từng dòng.
- Tránh JOIN nếu bạn chỉ cần lấy dữ liệu phụ trợ.
- Tăng khả năng đọc hiểu nếu query không quá phức tạp.

Phần 4: Temp Table

1. Đặt vấn đề

Trong CSDL `sql_invoicing`, bạn muốn thực hiện nhiều phép tính tổng và trung bình `invoice_total` cho từng khách hàng. Nhưng nếu tính trực tiếp nhiều lần thì câu truy vấn rất dài và khó đọc.
Giải pháp: Dùng một bảng tạm (Temp Table) để lưu lại kết quả tạm thời rồi truy vấn tiếp trên đó.

2. Temp Table là gì?

Khái niệm

Temp Table là bảng tạm thời được tạo trong phiên làm việc hiện tại. Khi phiên kết thúc hoặc bạn xoá thủ công, bảng này sẽ biến mất.
Cú pháp tạo bảng tạm:

```
CREATE TEMPORARY TABLE ten_bang (
    ...
);
```

Temp Table rất hữu ích khi:

- Truy vấn con quá phức tạp để dùng `CTE` hay `Subquery`.
- Muốn tái sử dụng kết quả truy vấn nhiều lần.
- Truy vết, gỡ lỗi, hoặc tách bước xử lý dữ liệu.

3. Ví dụ — Tạo Temp Table từ bảng invoices

Yêu cầu: Với mỗi khách hàng, tính tổng và trung bình `invoice_total` và lưu vào một bảng tạm.

Bước 1: Tạo bảng tạm `temp_invoice`

```
CREATE TEMPORARY TABLE temp_invoice (
    client_id INT,
    invoice_sum DECIMAL(10,2),
    invoice_avg DECIMAL(10,2)
);
```

Bước 2: Chèn dữ liệu vào bảng tạm

```
INSERT INTO temp_invoice
SELECT client_id,
       SUM(invoice_total) AS invoice_sum,
       AVG(invoice_total) AS invoice_avg
FROM invoices
GROUP BY client_id;
```

Bước 3: Truy vấn bảng tạm

```sql
SELECT *
FROM temp_invoice;
```

client_id	invoice_sum	invoice_avg
1	802.89	160.58
2	101.79	101.79
3	705.90	141.18
5	980.02	163.34

(Bảng tạm chứa kết quả tổng và trung bình hoá đơn theo từng khách hàng)

4. Ứng dụng Temp Table để xử lý tiếp

Dựa vào bảng `temp_invoice`, ta có thể tiếp tục xử lý nhanh chóng các truy vấn như:
Truy vấn khách hàng có tổng chi tiêu cao nhất:

```sql
SELECT client_id, name,
       (SELECT invoice_sum
        FROM temp_invoice
        WHERE c.client_id = client_id) AS invoice_sum
FROM clients c
ORDER BY invoice_sum DESC
LIMIT 1;
```

Truy vấn trung bình hoá đơn của từng khách:

```sql
SELECT client_id, name,
       (SELECT invoice_avg
        FROM temp_invoice
        WHERE c.client_id = client_id) AS invoice_avg
FROM clients c;
```

5. So sánh với CTE và Subquery

Khác biệt chính:

- **CTE/Subquery:** Chỉ tồn tại trong phạm vi 1 câu truy vấn.
- **Temp Table:** Tồn tại trong toàn bộ phiên làm việc hiện tại, có thể truy vấn nhiều lần.

6. Tổng kết

- **Temp Table** là bảng tạm thời, giúp chia nhỏ truy vấn và tái sử dụng kết quả dễ dàng.
- Thích hợp cho bài toán nhiều bước, dữ liệu trung gian cần xử lý riêng.
- Hữu dụng khi debug hoặc tối ưu hiệu năng thay vì lặp lại truy vấn nặng.
- Tuy nhiên, bảng tạm sẽ tự động biến mất khi kết thúc phiên làm việc.

Phần 5: Store Procedures

1. Đặt vấn đề — Lặp lại quá nhiều lần?

Giả sử bạn liên tục phải tạo bảng tạm để tính toán tổng và trung bình `invoice_total` cho từng khách hàng. Mỗi lần thực hiện, bạn đều phải viết lại toàn bộ các bước:

- `DROP TABLE` nếu bảng đã tồn tại
- `CREATE TABLE` định nghĩa cấu trúc bảng
- `INSERT INTO` để tính toán và ghi dữ liệu

Việc lặp đi lặp lại khiến truy vấn rườm rà, dễ lỗi và khó bảo trì.

Giải pháp: Dùng **Stored Procedure** để đóng gói toàn bộ quy trình này vào một "hàm" SQL có thể gọi bất kỳ lúc nào.

—

2. Stored Procedure là gì?

Khái niệm

Stored Procedure là một thủ tục được định nghĩa sẵn trong CSDL, có thể gọi lại nhiều lần. Nó giống như một **hàm có logic xử lý dữ liệu** được lưu trữ trong hệ quản trị SQL.
Cú pháp tổng quát:

```
DELIMITER //
CREATE PROCEDURE ten_thu_tuc()
BEGIN
    -- ộNi dung ửx lý ữd ệliu
END;
//
DELIMITER ;
```

Sau khi tạo, bạn chỉ cần dùng `CALL ten_thu_tuc();` để thực thi.

—

3. Ví dụ — Tạo bảng tạm bằng Stored Procedure

Yêu cầu: Tạo bảng tạm `temp_invoice` chứa tổng và trung bình `invoice_total` theo từng khách hàng.

Bước 1: Định nghĩa Stored Procedure

```
DELIMITER //
CREATE PROCEDURE create_temp_invoice()
BEGIN
    DROP TABLE IF EXISTS temp_invoice;

    CREATE TEMPORARY TABLE temp_invoice (
```

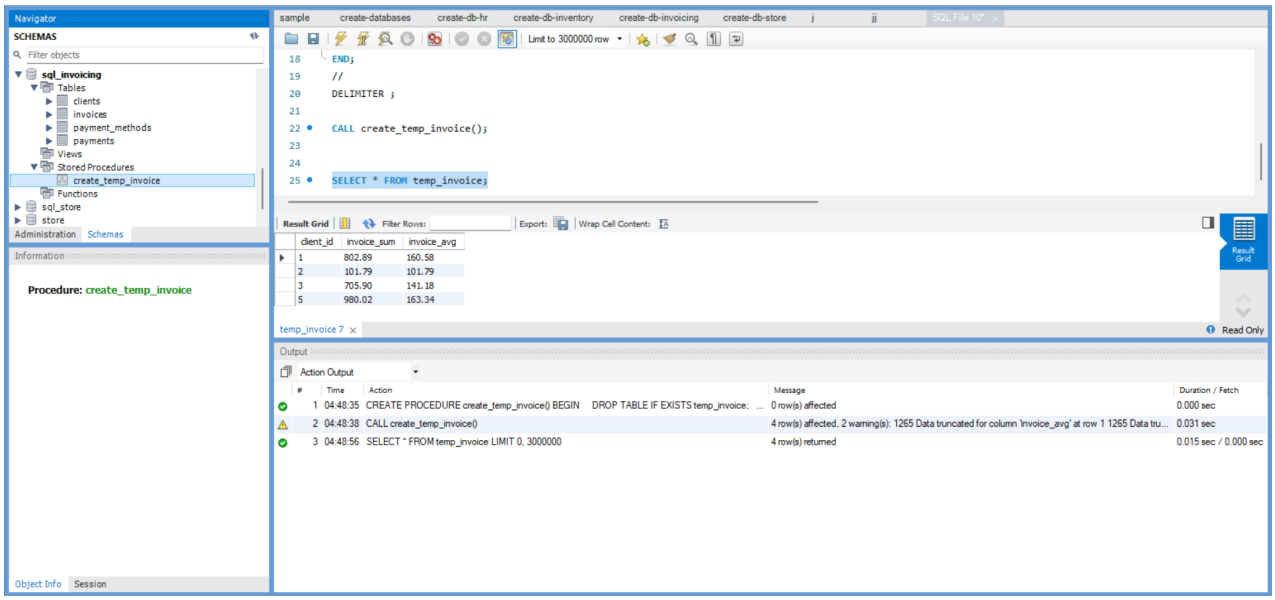
```
        client_id INT,
        invoice_sum DECIMAL(10,2),
        invoice_avg DECIMAL(10,2)
    );

    INSERT INTO temp_invoice
    SELECT client_id,
           SUM(invoice_total) AS invoice_sum,
           AVG(invoice_total) AS invoice_avg
    FROM invoices
    GROUP BY client_id;
END;
//
DELIMITER ;
```

Bước 2: Gọi Stored Procedure

```
CALL create_temp_invoice();
```

Bước 3: Truy vấn dữ liệu bảng tạm

```
SELECT * FROM temp_invoice;
```



(Kết quả của bảng tạm sau khi gọi thủ tục)

4. Vì sao Stored Procedure là bước tiến hóa của Temp Table?

So sánh nhanh:

- **Temp Table:** Phải gõ lại toàn bộ truy vấn mỗi lần sử dụng. Không linh hoạt.
- **Stored Procedure:** Tái sử dụng dễ dàng, chỉ cần gọi tên là đủ. Quản lý logic dễ hơn.

Stored Procedure = Temp Table + Automation + Bảo trì dễ dàng

5. Tổng kết

- **Stored Procedure** giúp đóng gói các truy vấn SQL phức tạp thành thủ tục có thể gọi lại.
- Rất hiệu quả khi cần thao tác nhiều bước như drop, create, insert...
- Là công cụ mạnh để thay thế cách viết thủ công khi dùng Temp Table.

Phần 6: Trigger

1 Định nghĩa SQL Trigger

1 Định nghĩa

Trigger là một khối lệnh được hệ quản trị cơ sở dữ liệu (DBMS) tự động thực thi khi xảy ra một sự kiện xác định như `INSERT`, `UPDATE`, `DELETE`, hoặc khi có thay đổi về cấu trúc hay trạng thái của hệ thống.
Trigger **không cần gọi thủ công**, mà tự động kích hoạt khi điều kiện được chỉ định xảy ra.

2 Phân loại chính

- **A. Trigger thao tác dữ liệu (DML Trigger)**
  Kích hoạt bởi các lệnh: `INSERT`, `UPDATE`, `DELETE`.
  Mục đích: kiểm tra tính hợp lệ, ghi log, đồng bộ dữ liệu.

  Ví dụ:

```
CREATE OR REPLACE TRIGGER log_delete
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO delete_log(emp_id, deleted_at)
  VALUES (:OLD.emp_id, SYSDATE);
END;
```

- **B. Trigger định nghĩa dữ liệu (DDL Trigger)**
  Kích hoạt bởi các lệnh: `CREATE`, `ALTER`, `DROP`.
  Mục đích: ngăn thay đổi cấu trúc không mong muốn, theo dõi chỉnh sửa hệ thống.

  Ví dụ:

```
CREATE OR REPLACE TRIGGER prevent_drop
BEFORE DROP ON DATABASE
BEGIN
  RAISE_APPLICATION_ERROR(-20001, 'ấCm xóa ảbng!');
END;
```

- **C. Trigger sự kiện hệ thống (System/Event Trigger)**
  Kích hoạt bởi các sự kiện hệ thống như:
  - `LOGON`: khi người dùng đăng nhập
  - `LOGOFF`: khi người dùng đăng xuất
  - `STARTUP`: khi hệ thống khởi động
  - `SHUTDOWN`: khi hệ thống tắt
  - `SERVERERROR`: khi có lỗi hệ thống

  Mục đích: ghi log truy cập, kiểm soát vận hành, bảo mật.

  Ví dụ:

```sql
CREATE OR REPLACE TRIGGER log_login
AFTER LOGON ON DATABASE
BEGIN
  INSERT INTO login_audit(username, login_time)
  VALUES (USER, SYSDATE);
END;
```

3 Ghi chú khi sử dụng Trigger

- Tránh lạm dụng hoặc viết trigger chồng chéo, gây xung đột logic hoặc giảm hiệu suất hệ thống.
- Cẩn thận với việc tạo vòng lặp vô hạn nếu trigger kích hoạt chính nó.
- Trigger phù hợp cho tự động hóa hành động đơn giản trong cơ sở dữ liệu; không nên thay thế toàn bộ logic xử lý trong ứng dụng.

4 Lỗi thường gặp khi thiết lập Trigger trong SQL

Mutating Table là lỗi thường gặp trong các hệ quản trị cơ sở dữ liệu như Oracle, xảy ra khi một **trigger dạng row-level** cố gắng truy cập hoặc thay đổi **chính bảng dữ liệu đang bị tác động** bởi hành động DML (INSERT, UPDATE, DELETE).

Nguyên nhân: Khi trigger dạng `FOR EACH ROW` đang xử lý một dòng, bảng đó sẽ tạm thời bị khóa để đảm bảo tính toàn vẹn dữ liệu. Nếu bạn cố thực hiện thao tác SELECT, INSERT, UPDATE hoặc DELETE trên chính bảng đó trong trigger, lỗi **mutating table** sẽ xảy ra.

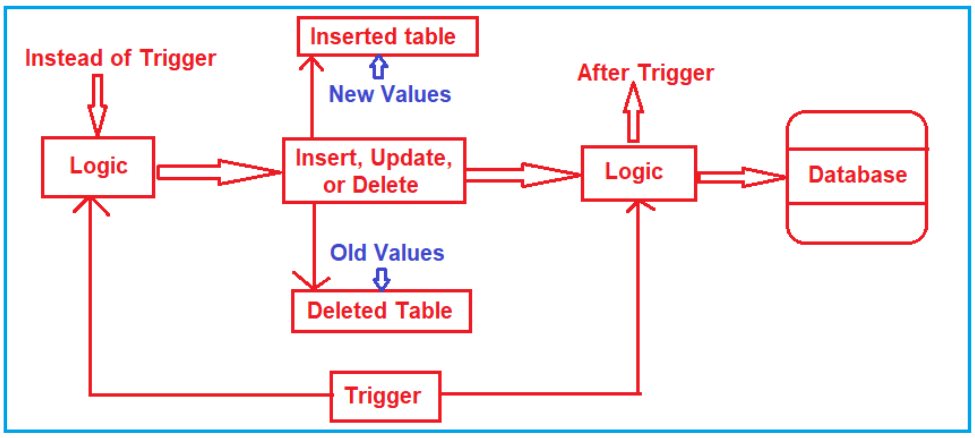
```sql
CREATE OR REPLACE TRIGGER trg_update_salary
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  DECLARE
    v_count NUMBER;
  BEGIN
    SELECT COUNT(*) INTO v_count
    FROM employees
    WHERE department_id = :NEW.department_id; -- <mutating error>
  END;
END;
```

Code Listing 1: Ví dụ gây lỗi Mutating Table

Giải pháp:

- Sử dụng **trigger dạng STATEMENT-level** thay vì row-level.
- Tách xử lý sang package và dùng biến toàn cục (global variable).

2 Table Trigger



Hình 1: Enter Caption

Trigger là một đối tượng trong cơ sở dữ liệu được tự động thực thi khi một sự kiện cụ thể xảy ra trên một bảng hoặc một view. Trong đó, **DML Trigger** (Data Manipulation Language Trigger) là loại trigger liên kết với các thao tác dữ liệu như `INSERT`, `UPDATE`, hoặc `DELETE`. Khi một câu lệnh DML được thực thi trên bảng có trigger, đoạn mã bên trong trigger sẽ được thực thi tự động. Trigger có thể được định nghĩa để phản hồi một hay nhiều loại thao tác DML.

Thời điểm kích hoạt (Trigger Timing)

Trigger có thể được thực thi trước hoặc sau khi sự kiện DML xảy ra, thể hiện qua hai từ khóa `BEFORE` và `AFTER`:

Từ khóa	Thời điểm kích hoạt	Ý nghĩa
`BEFORE`	Trước khi thao tác xảy ra	Kiểm tra/điều kiện trước khi dữ liệu thay đổi
`AFTER`	Sau khi thao tác hoàn tất	Ghi log, cập nhật bảng phụ, v.v.

Phạm vi kích hoạt (Trigger Scope)

Trigger hoạt động theo hai cấp độ:

- **Statement-level** (mặc định): Trigger chỉ kích hoạt một lần cho toàn bộ câu lệnh DML.
- **Row-level** (`FOR EACH ROW`): Trigger được kích hoạt một lần cho mỗi dòng bị ảnh hưởng.

Ví dụ: Nếu một câu lệnh `UPDATE` ảnh hưởng đến 10 dòng:

- Trigger statement-level chỉ chạy 1 lần.
- Trigger row-level sẽ chạy 10 lần.

Ví dụ minh họa:

Ví dụ tạo trigger để ghi lại mỗi lần xóa nhân viên:

```
CREATE OR REPLACE TRIGGER log_emp_delete
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_delete_log(emp_id, deleted_at)
  VALUES (:OLD.emp_id, SYSDATE);
END;
```

Nếu câu lệnh `DELETE` xóa 5 nhân viên, trigger trên sẽ chạy 5 lần, mỗi lần ghi lại thông tin một nhân viên bị xóa.

Cấu trúc tổng quát

```
CREATE [OR REPLACE] TRIGGER <ten_trigger>
{BEFORE | AFTER | INSTEAD OF}
{INSERT | UPDATE | DELETE} [OF <ten_cot>]
ON <ten_bang_or_view>
[COMPOUND TRIGGER]
[FOLLOW <ten_trigger2>]
[ENABLE/ DISABLE]
[REFERENCEING {OLD [AS] old-name [NEW [AS] new-name]
    | NEW [AS] new-name [OLD [AS] old-name]}]
[FOR EACH ROW]
[WHEN (<dieu_kien>)]
DECLARE
    -- Khai bao bien (neu can)
BEGIN
    -- Cac cau lenh thuc thi khi trigger kich hoat
EXCEPTION
    -- Xu ly ngoai le (neu co)
END;
```

Code Listing 2: Cấu trúc tổng quát của Trigger

Giải thích các thành phần

- **CREATE [OR REPLACE] TRIGGER <tên_trigger>**: Câu lệnh tạo trigger mới. Nếu sử dụng `OR REPLACE`, trigger cũ cùng tên sẽ bị thay thế.
- **BEFORE / AFTER / INSTEAD OF**: Xác định thời điểm trigger được thực thi:
  - `BEFORE`: Trigger chạy trước khi sự kiện DML xảy ra.
  - `AFTER`: Trigger chạy sau khi sự kiện DML hoàn tất.
  - `INSTEAD OF`: Dùng cho các view, trigger sẽ thay thế hành động DML không thể thực hiện trực tiếp trên view.
- **INSERT / UPDATE / DELETE [OF <tên_cột>]**: Chỉ định loại thao tác DML mà trigger phản hồi. Với `UPDATE OF <tên_cột>`, trigger chỉ kích hoạt nếu cột đó bị cập nhật.
- **ON <tên_bảng_hoặc_view>**: Bảng hoặc view mà trigger được gắn vào.
- **COMPOUND TRIGGER** (tuỳ chọn): Dùng để khai báo một trigger hỗn hợp gồm nhiều phần thực thi tại các thời điểm khác nhau trong cùng một khối trigger (chỉ có trong Oracle).

- **FOLLOW <tên_trigger2>** (tuỳ chọn): Chỉ định thứ tự thực thi trigger nếu có nhiều trigger cùng loại trên cùng một bảng. Trigger hiện tại sẽ thực thi sau `<tên_trigger2>`.
- **ENABLE / DISABLE** (tuỳ chọn): `ENABLE` để bật trigger ngay sau khi tạo; `DISABLE` để tạo nhưng chưa kích hoạt trigger.
- **REFERENCING**: Cho phép gán tên khác cho các giá trị trước (`OLD`) và sau (`NEW`) của cột dữ liệu, dùng trong row-level trigger. Ví dụ:
  - `OLD.salary`: giá trị lương cũ trước khi cập nhật.
  - `NEW.salary`: giá trị lương mới sau khi cập nhật.
- **FOR EACH ROW** (tuỳ chọn): Nếu có, trigger sẽ chạy một lần cho mỗi dòng bị ảnh hưởng (row-level). Nếu không có, trigger chỉ chạy một lần cho toàn bộ câu lệnh (statement-level).
- **WHEN (<điều_kiện>)** (tuỳ chọn): Điều kiện logic để trigger được kích hoạt. Điều kiện này chỉ áp dụng cho row-level trigger.
- **DECLARE - BEGIN - EXCEPTION - END**: Khối PL/SQL xử lý logic trigger:
  - `DECLARE`: khai báo biến.
  - `BEGIN ... END`: nội dung thực thi chính.
  - `EXCEPTION`: xử lý ngoại lệ nếu có lỗi.

3 Xử lý ngoại lệ trong Trigger (`EXCEPTION`)

Trong trigger, phần `EXCEPTION` dùng để **bắt và xử lý lỗi** xảy ra trong quá trình thực thi trigger. Điều này giúp tránh việc chương trình bị dừng đột ngột và cho phép thực hiện các hành động như ghi log hoặc thông báo lỗi rõ ràng.

Cấu trúc cơ bản

```
BEGIN
    -- Cac cau lenh chinh
EXCEPTION
    WHEN <ten_loi> THEN
        -- Xu ly loi cu the
    WHEN OTHERS THEN
        -- Xu ly moi loi cu the
END;
```

Code Listing 3: Khối EXCEPTION trong trigger

Ý nghĩa

- Giúp chương trình **không bị dừng đột ngột** nếu có lỗi xảy ra.
- Cho phép **ghi lại lỗi** hoặc thực hiện các hành động thay thế.
- Xử lý các tình huống **ngoài dự đoán**, đảm bảo hệ thống hoạt động ổn định.

Ví dụ minh hoạ

```sql
CREATE OR REPLACE TRIGGER check_salary
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    IF :NEW.salary > 10000 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Muc luong vuot qua gioi han!');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        INSERT INTO error_log(msg, log_date)
        VALUES(SQLERRM, SYSDATE);
END;
```

Code Listing 4: Ví dụ về xử lý ngoại lệ trong trigger

Giải thích

- `RAISE_APPLICATION_ERROR`: Tạo ra lỗi tùy chỉnh với mã và thông báo cụ thể.
- `WHEN OTHERS`: Bắt mọi lỗi không được chỉ định cụ thể.
- `SQLERRM`: Trả về thông điệp lỗi đã xảy ra.
- Trong ví dụ, thay vì để lỗi làm dừng hệ thống, trigger sẽ **ghi lỗi vào bảng `error_log`** để theo dõi và xử lý sau.

4 INSTEAD OF Trigger

1 Khái niệm

INSTEAD OF Trigger là một loại trigger đặc biệt trong cơ sở dữ liệu, thường được sử dụng cho **VIEW (khung nhìn)** — đặc biệt là các view không thể cập nhật trực tiếp. Khi người dùng thực hiện thao tác `INSERT`, `UPDATE`, hoặc `DELETE` trên view, thay vì hệ thống xử lý thao tác đó, `INSTEAD OF Trigger` sẽ được kích hoạt và thực hiện phần hành động thay thế đã được lập trình.

2 Trường hợp sử dụng

INSTEAD OF Trigger thường cần thiết khi:

- View có cấu trúc phức tạp: chứa `JOIN`, `GROUP BY`, `DISTINCT`, hoặc `UNION`.
- Không thể cập nhật trực tiếp do phụ thuộc nhiều bảng.
- Cần kiểm soát logic chèn/sửa/xoá dữ liệu một cách tuỳ chỉnh.

3 Cách hoạt động

Khi người dùng gọi lệnh DML (`INSERT`, `UPDATE`, hoặc `DELETE`) trên view:

1. Trigger loại `INSTEAD OF` được kích hoạt.
2. Hành động DML ban đầu bị thay thế hoàn toàn.
3. Phần thân trigger sẽ xử lý logic thay thế.

4 Ví dụ minh hoạ

```
-- Tao view khong the cap nhat truc tiep
CREATE VIEW emp_dept_view AS
SELECT e.emp_id, e.name, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;

-- Tao trigger thay the INSERT
CREATE OR REPLACE TRIGGER trg_emp_dept_insert
INSTEAD OF INSERT ON emp_dept_view
FOR EACH ROW
BEGIN
  INSERT INTO employees (emp_id, name, dept_id)
  VALUES (
    :NEW.emp_id,
    :NEW.name,
    (SELECT dept_id FROM departments
     WHERE dept_name = :NEW.dept_name)
  );
END;
```

Code Listing 5: Tạo INSTEAD OF Trigger cho view

Người dùng có thể thực hiện:

```
INSERT INTO emp_dept_view (emp_id, name, dept_name)
VALUES (101, 'An', 'IT');
```

Khi đó, trigger sẽ thay người dùng chèn dữ liệu vào bảng `employees` với `dept_id` được ánh xạ từ `dept_name`.

5 Lợi ích

Lợi ích	**Mô tả**
Cho phép thao tác DML trên view phức tạp	View có `JOIN`, `GROUP BY` vẫn có thể cập nhật gián tiếp qua trigger.
Tách logic xử lý	Kiểm soát rõ ràng cách dữ liệu được chèn, sửa, xoá vào bảng gốc.
Đảm bảo toàn vẹn dữ liệu	Có thể bổ sung kiểm tra ràng buộc trước khi thực hiện thay đổi dữ liệu thực.

5 Ví dụ việc triển khai quy tắc nghiệp vụ (business rule) bằng trigger

Quy tắc nghiệp vụ:
Không cho phép xóa khách hàng nếu họ còn đơn hàng đang xử lý (chưa giao hàng).

```
CREATE OR REPLACE TRIGGER trg_prevent_customer_delete
BEFORE DELETE ON customers
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM orders
    WHERE customer_id = :OLD.customer_id
      AND status != 'Da giao';

```

```
    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20002,
        'Khong the xoa khach hang vi con don hang chua giao.');
    END IF;
END;
```

Code Listing 6: Trigger kiểm tra trạng thái đơn hàng trước khi xóa khách hàng

Giải thích:

- **BEFORE DELETE ON customers**: Trigger được kích hoạt trước khi thực hiện thao tác xóa trên bảng `customers`.
- **FOR EACH ROW**: Áp dụng cho từng bản ghi bị xóa.
- `v_count`: Biến dùng để đếm số lượng đơn hàng chưa giao của khách hàng.
- `RAISE_APPLICATION_ERROR`: Gây ra lỗi để ngăn không cho thao tác xóa tiếp tục nếu còn đơn hàng chưa giao.

Ý nghĩa thực tiễn:

- Đảm bảo không làm mất dữ liệu khách hàng khi các đơn hàng của họ vẫn đang trong quá trình xử lý.
- Duy trì tính toàn vẹn nghiệp vụ tại tầng cơ sở dữ liệu, tăng độ an toàn cho hệ thống.

Phần 7: Recursive

1. Đặt vấn đề — Khi dữ liệu tự tham chiếu chính nó

Giả sử bạn quản lý dữ liệu nhân sự trong một công ty. Mỗi nhân viên có một người quản lý trực tiếp (trừ CEO), và thông tin này được lưu trong cùng một bảng như sau:

```sql
CREATE TABLE employees (
    employee_id   INT PRIMARY KEY,
    employee_name VARCHAR(100),
    manager_id    INT          -- NULL neu la CEO
);
```

Dữ liệu mẫu:

```sql
INSERT INTO employees (employee_id, employee_name, manager_id) VALUES
(1, 'Alice (CEO)', NULL),
(2, 'Bob', 1),
(3, 'Charlie', NULL),
(4, 'David', 2),
(5, 'Eve', 2),
(6, 'Frank', 3);
```

Làm sao truy vấn ra toàn bộ những nhân viên do CEO quản lý — kể cả trực tiếp và gián tiếp?

Giải pháp: Dùng **CTE đệ quy** để duyệt theo cấu trúc phân cấp của bảng.

2. CTE đệ quy là gì?

Khái niệm

Common Table Expression (CTE) là một khối truy vấn tạm, thường để chia nhỏ truy vấn phức tạp. **CTE đệ quy** là loại CTE tự gọi lại chính nó, giúp duyệt qua dữ liệu phân cấp.
Cấu trúc gồm 2 phần:

- **Anchor Member:** truy vấn gốc (CEO)
- **Recursive Member:** phần truy vấn gọi lại CTE

3. Ví dụ — Tìm cấp dưới của CEO

Cấu trúc bảng:

```sql
CREATE TABLE employees (
  employee_id   INT PRIMARY KEY,
  employee_name VARCHAR(100),
  manager_id    INT
);
```

Truy vấn CTE đệ quy:

```sql
WITH RECURSIVE subordinates AS (
  -- Anchor member
  SELECT employee_id, employee_name, manager_id
  FROM employees
```

```
  WHERE employee_name = 'Alice (CEO)'

  UNION ALL

  -- Recursive member
  SELECT e.employee_id, e.employee_name, e.manager_id
  FROM employees e
  JOIN subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM subordinates;
```

Kết quả: Liệt kê toàn bộ nhân viên do Alice quản lý, kể cả trực tiếp và gián tiếp.

4. Phân tích theo vòng lặp truy vấn

Gọi kết quả mỗi vòng lặp là T_n. Dưới đây là cách CTE truy vấn dữ liệu theo từng vòng:

- **Vòng 1 (Anchor Member):**

$$T_0 = \{(1, \texttt{'Alice (CEO)'}, \text{NULL})\}$$

  Truy vấn khởi tạo với người đứng đầu là Alice.

- **Vòng 2:**

$$T_1 = \{(2, \texttt{'Bob'}, 1)\}$$

  Lấy nhân viên có `manager_id = 1` — tức là cấp dưới trực tiếp của Alice.

- **Vòng 3:**

$$T_2 = \{(4, \texttt{'David'}, 2), (5, \texttt{'Eve'}, 2)\}$$

  Lấy cấp dưới trực tiếp của Bob.

- **Vòng 4:**

$$T_3 = \varnothing$$

  Không còn nhân viên nào quản lý bởi David hoặc Eve. Truy vấn dừng lại.

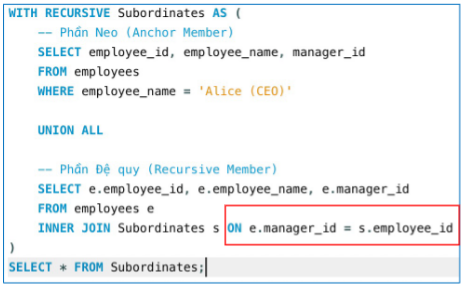
CTE sẽ tự động dừng khi vòng lặp không tạo ra thêm bản ghi mới.
—

5. Kết quả cuối cùng

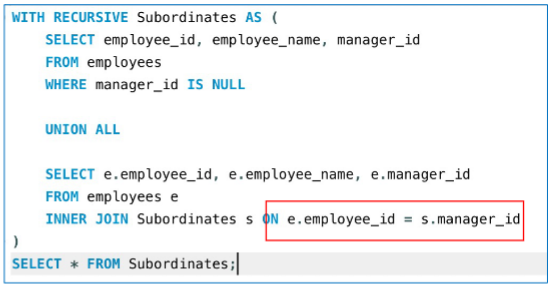
employee_id	employee_name	manager_id
1	Alice (CEO)	NULL
2	Bob	1
4	David	2
5	Eve	2

Toàn bộ cây tổ chức phía dưới Alice (CEO) đã được truy vấn thông qua CTE đệ quy.

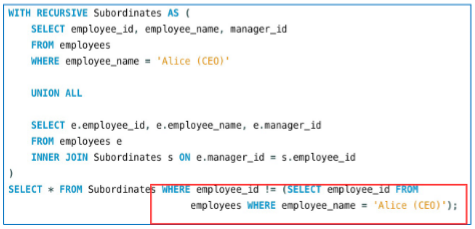
6. Các lỗi sai thường gặp



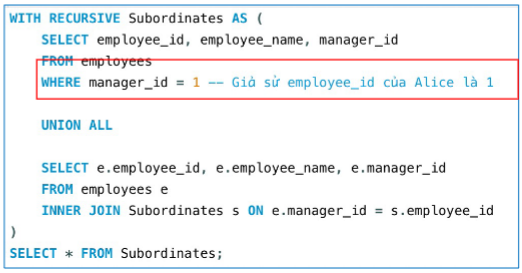
```
WITH RECURSIVE Subordinates AS (
    -- Phần Neo (Anchor Member)
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE employee_name = 'Alice (CEO)'

    UNION ALL

    -- Phần Đệ quy (Recursive Member)
    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```



```
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.employee_id = s.manager_id
)
SELECT * FROM Subordinates;
```



```
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE employee_name = 'Alice (CEO)'

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates WHERE employee_id != (SELECT employee_id FROM
                                employees WHERE employee_name = 'Alice (CEO)');
```



```
WITH RECURSIVE Subordinates AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE manager_id = 1 -- Giả sử employee_id của Alice là 1

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```

Sai JOIN:

```
-- Sai:
ON e.employee_id = s.manager_id
-- Đúng:
ON e.manager_id = s.employee_id
```

Sai Anchor: WHERE manager_id IS NULL (lấy nhiều hơn một người)
Hardcode ID: Dùng employee_id = 1 thay vì lọc theo tên nhân viên

Truy vấn đúng

```
WITH RECURSIVE Subordinates AS (
  SELECT employee_id, employee_name, manager_id
  FROM employees
  WHERE employee_name = 'Alice (CEO)'

  UNION ALL

  SELECT e.employee_id, e.employee_name, e.manager_id
  FROM employees e
  INNER JOIN Subordinates s ON e.manager_id = s.employee_id
)
SELECT * FROM Subordinates;
```

7. Ứng dụng

- Cây thư mục
- Cây menu website
- Câu truy vấn phân tổng cục
- Dòng dãy chu trình trong đồ thị

8. Tổng kết

- CTE đệ quy giúp truy vấn cấu trúc phân cấp linh hoạt
- Cần xác định rõ Anchor Member và Recursive Member
- Dùng JOIN đúng chiều tham chiếu
- Tránh hardcode ID — hãy lọc theo logic linh hoạt