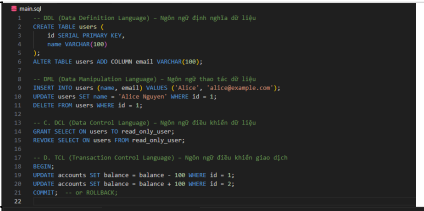
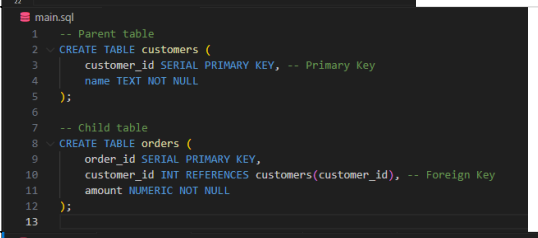
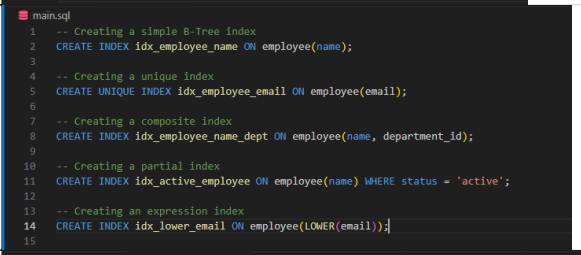
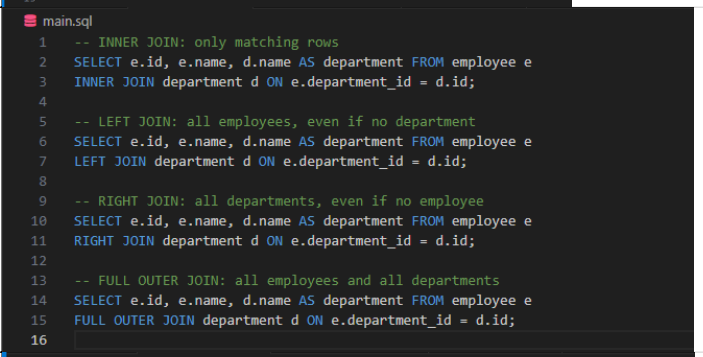
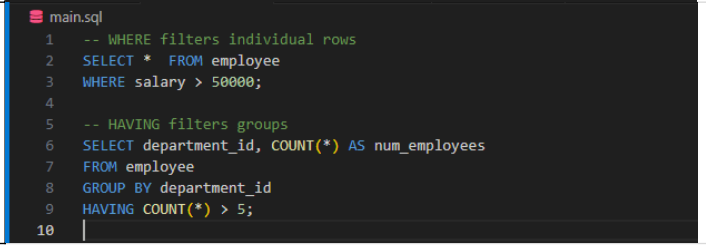
Basic	SQL là gì? Phân biệt DDL, DML, DCL và TCL. SQL (Structured Query Language) là ngôn ngữ chuẩn dùng để làm việc với các hệ quản trị cơ sở dữ liệu quan hệ. Bạn dùng SQL để lưu trữ, truy vấn, cập nhật và quản lý dữ liệu. Hầu hết các hệ quản trị CSDL quan hệ như PostgreSQL, MySQL, SQL Server, Oracle đều sử dụng SQL như ngôn ngữ chính. DDL dùng để định nghĩa hoặc thay đổi cấu trúc của cơ sở dữ liệu, như tạo bảng hoặc sửa bảng. DML dùng để thao tác với dữ liệu bên trong bảng (thêm, sửa, xóa, truy vấn). DCL dùng để kiểm soát quyền truy cập dữ liệu – ai được đọc, sửa hoặc quản lý dữ liệu. TCL quản lý giao dịch—nhóm các thao tác SQL cần thực hiện cùng nhau. Nếu một bước lỗi, ta có thể rollback để an toàn dữ liệu.	
	Primary key và Foreign key khác nhau như thế nào?" Primary Key là một cột (hoặc nhóm cột) dùng để xác định duy nhất mỗi hàng trong một bảng. Nó đảm bảo: 1. Duy nhất – không có hai hàng nào trùng giá trị khóa chính. 2. Không rỗng (NOT NULL) – khóa chính không được chứa giá trị NULL. Foreign Key là một cột (hoặc nhóm cột) trong một bảng dùng để tham chiếu đến Primary Key của bảng khác. Mục đích của nó là đảm bảo tính toàn vẹn tham chiếu – nghĩa là dữ liệu luôn hợp lệ; bạn không thể chèn giá trị không tồn tại ở bảng cha vào bảng con. Tóm lại: - Primary Key dùng để định danh một bản ghi trong chính bảng đó. - Foreign Key dùng để liên kết bản ghi với bảng khác.	
	Index là gì? Có những loại index nào? Index là một cấu trúc trong cơ sở dữ liệu giúp tăng tốc độ truy xuất dữ liệu trên bảng, nhưng sẽ tốn thêm không gian lưu trữ và làm các thao tác ghi (insert, update, delete) chậm hơn một chút. Các loại index phổ biến gồm: - Primary Key Index: Tự động tạo khi bạn định nghĩa khóa chính, đảm bảo tính duy nhất. - Unique Index: Đảm bảo không có hai hàng có giá trị trùng nhau trên cột được lập chỉ mục. - B-Tree Index: Loại phổ biến nhất, tối ưu cho truy vấn bằng giá trị chính xác hoặc phạm vi. - Hash Index: Hiệu quả cho tìm kiếm chính xác, nhưng không dùng cho truy vấn theo phạm vi. - Composite Index: Index trên nhiều cột, tối ưu cho các truy vấn lọc theo nhiều cột. - Partial Index: Chỉ lập chỉ mục một phần các hàng thỏa điều kiện. - Expression Index: Lập chỉ mục dựa trên biểu thức tính toán thay vì cột đơn thuần.	
	Khác biệt giữa INNER JOIN, LEFT JOIN, RIGHT JOIN, và FULL OUTER JOIN. Trong SQL, join dùng để kết hợp dữ liệu từ hai hay nhiều bảng dựa trên cột liên quan. - INNER JOIN chỉ lấy những hàng có dữ liệu khớp ở cả hai bảng. - LEFT JOIN lấy tất cả hàng từ bảng bên trái, và các hàng khớp từ bảng bên phải; nếu không khớp, bên phải sẽ là NULL. - RIGHT JOIN ngược lại: lấy tất cả hàng từ bảng bên phải, và các hàng khớp từ bảng bên trái. - FULL OUTER JOIN lấy tất cả hàng từ cả hai bảng, nếu không khớp ở bên nào sẽ là NULL.	
	Khác biệt giữa WHERE và HAVING. Trong SQL, WHERE dùng để lọc các hàng trước khi thực hiện nhóm dữ liệu, còn HAVING dùng để lọc kết quả nhóm sau khi dùng GROUP BY. Nói cách khác, WHERE lọc theo từng hàng, HAVING lọc theo các giá trị tổng hợp như COUNT, SUM, AVG…	

	Khác biệt giữa UNION và UNION ALL. Trong SQL, UNION kết hợp kết quả của hai hay nhiều truy vấn và loại bỏ các hàng trùng nhau, còn UNION ALL kết hợp kết quả bao gồm tất cả các hàng trùng nhau. Vì vậy, UNION đảm bảo tính duy nhất nhưng có thể chậm hơn do phải kiểm tra trùng lặp, trong khi UNION ALL nhanh hơn vì chỉ ghép tất cả hàng.	main.sql -- Using UNION (duplicates removed) SELECT name FROM employee_2023 UNION SELECT name FROM employee_2024; -- Using UNION ALL (duplicates kept) SELECT name FROM employee_2023 UNION ALL SELECT name FROM employee_2024;
	Khác biệt giữa TRUNCATE, DELETE và DROP. Trong SQL, DELETE, TRUNCATE, và DROP đều dùng để xóa dữ liệu nhưng cách hoạt động khác nhau. - DELETE xóa các hàng cụ thể dựa trên điều kiện và có thể undo nếu nằm trong transaction. - TRUNCATE xóa tất cả dữ liệu trong bảng nhanh chóng, reset các bộ đếm ID, nhưng thường không thể undo. - DROP xóa toàn bộ bảng hoặc cơ sở dữ liệu, bao gồm dữ liệu, index, và constraint, vĩnh viễn.	main.sql -- DELETE: remove specific rows DELETE FROM employee WHERE department_id = 5; -- TRUNCATE: remove all rows quickly TRUNCATE TABLE employee; -- DROP: remove the entire table DROP TABLE employee;
	Khác biệt giữa CHAR và VARCHAR. Trong SQL, CHAR và VARCHAR đều lưu trữ chuỗi văn bản nhưng khác nhau về cách lưu. CHAR(n) là chuỗi cố định độ dài, luôn chiếm n ký tự; nếu dữ liệu ngắn hơn, nó sẽ thêm khoảng trắng để đủ độ dài. VARCHAR(n) là chuỗi biến đổi độ dài, chỉ lưu đúng số ký tự bạn nhập, tiết kiệm bộ nhớ hơn. Dùng CHAR khi độ dài dữ liệu cố định, VARCHAR khi độ dài thay đổi.	main.sql -- CHAR: fixed length of 10 CREATE TABLE employee_char (name CHAR(10)); -- VARCHAR: variable length up to 50 CREATE TABLE employee_varchar (name VARCHAR(50)); -- Example inserts INSERT INTO employee_char (name) VALUES ('Tom'); -- stored as 'Tom ' INSERT INTO employee_varchar (name) VALUES ('Tom'); -- stored as 'Tom'
	SQL Injection là gì? Cách phòng tránh. SQL Injection là lỗ hổng bảo mật xảy ra khi kẻ tấn công có thể thay đổi câu lệnh SQL bằng cách chèn dữ liệu độc hại, có thể đọc, sửa hoặc xóa dữ liệu trong cơ sở dữ liệu. Nguyên nhân thường là do dữ liệu từ người dùng được ghép thẳng vào SQL mà không kiểm tra hay xử lý an toàn. Cách phòng tránh: - Sử dụng truy vấn có tham số / prepared statements: không ghép trực tiếp dữ liệu từ người dùng vào câu lệnh SQL. - Kiểm tra và làm sạch dữ liệu nhập vào: kiểm tra kiểu dữ liệu, độ dài, và các ký tự được phép. - Sử dụng các framework ORM: thường tự động xử lý việc escape dữ liệu và dùng tham số an toàn. - Giới hạn quyền trên cơ sở dữ liệu: chỉ cấp quyền tối thiểu cần thiết cho tài khoản ứng dụng.	main.sql -- Unsafe: vulnerable to SQL Injection -- user_input = '1 OR 1=1' SELECT * FROM employee WHERE id = 'user_input'; -- Safe: using parameterized query PREPARE get_employee(int) AS SELECT * FROM employee WHERE id = $1; EXECUTE get_employee(1);
	Viết truy vấn để tìm giá trị lớn nhất/nhỏ nhất trong một cột. Trong SQL, bạn có thể dùng hàm MAX() để tìm giá trị lớn nhất trong cột, và MIN() để tìm giá trị nhỏ nhất. Đây là các hàm tổng hợp (aggregate functions) quét toàn bộ cột và trả về một giá trị duy nhất.	main.sql -- Find the maximum salary SELECT MAX(salary) AS max_salary FROM employee; -- Find the minimum salary SELECT MIN(salary) AS min_salary FROM employee;
	Viết truy vấn để đếm số bản ghi theo nhóm. Trong SQL, để đếm số bản ghi theo nhóm, bạn dùng hàm tổng hợp COUNT() cùng với GROUP BY. Cách này giúp bạn biết có bao nhiêu hàng thuộc mỗi nhóm hoặc mỗi giá trị của một cột.	main.sql -- Count departments with more than 5 employees SELECT department_id, COUNT(*) AS num_employees FROM employee GROUP BY department_id HAVING COUNT(*) > 5;

	Làm thế nào để lấy 5 bản ghi mới nhất từ một bảng? Trong SQL, để lấy các bản ghi mới nhất, bạn thường cần một cột để xác định thứ tự, ví dụ như ngày, timestamp, hoặc ID tự tăng. Bạn dùng ORDER BY để sắp xếp giảm dần và LIMIT để giới hạn số hàng trả về.	main.sql -- Get 5 most recent employees by created_date SELECT * FROM employee ORDER BY created_date DESC LIMIT 5; -- Get 5 most recent employees by auto-increment ID SELECT * FROM employee ORDER BY id DESC LIMIT 5;
	Viết truy vấn để lọc dữ liệu theo nhiều điều kiện. Trong SQL, để lọc dữ liệu theo nhiều điều kiện, bạn dùng WHERE kết hợp với AND, OR, và dấu ngoặc () để ghép các điều kiện theo logic. Cách này giúp chọn chính xác các hàng thỏa các tiêu chí phức tạp.	main.sql -- Filter employees in department 1 with salary > 50000 SELECT * FROM employee WHERE department_id = 1 AND salary > 50000; -- Filter employees in department 1 or department 2 with salary > 50000 SELECT * FROM employee WHERE (department_id = 1 OR department_id = 2) AND salary > 50000;
	Làm thế nào để kết hợp dữ liệu từ nhiều bảng? Trong SQL, để kết hợp dữ liệu từ nhiều bảng, bạn dùng các JOIN. Các loại JOIN phổ biến là INNER JOIN, LEFT JOIN, RIGHT JOIN, và FULL OUTER JOIN. JOIN giúp kết nối các hàng từ bảng này với bảng khác dựa trên một cột liên quan, thường là mối quan hệ khóa chính và khóa ngoại.	main.sql -- Combine employee data with department data using INNER JOIN SELECT e.id, e.name, d.name AS department FROM employee e INNER JOIN department d ON e.department_id = d.id; -- LEFT JOIN example: get all employees, including those without a department SELECT e.id, e.name, d.name AS department FROM employee e LEFT JOIN department d ON e.department_id = d.id;
	Viết truy vấn để tìm bản ghi xuất hiện nhiều hơn một lần. Trong SQL, để tìm các bản ghi xuất hiện nhiều hơn một lần, bạn dùng GROUP BY trên cột muốn kiểm tra và HAVING COUNT (*) > 1 để lọc các nhóm có bản ghi trùng lặp.	main.sql -- Find duplicate employee names SELECT name, COUNT(*) AS count_name FROM employee GROUP BY name HAVING COUNT(*) > 1; -- Find duplicate emails SELECT email, COUNT(*) AS count_email FROM employee GROUP BY email HAVING COUNT(*) > 1;
	Viết truy vấn để cập nhật dữ liệu trong một bảng dựa trên dữ liệu của bảng khác. Trong SQL, để cập nhật dữ liệu trong một bảng dựa trên dữ liệu của bảng khác, bạn dùng câu lệnh UPDATE kết hợp với JOIN (hoặc subquery) để nối các hàng giữa hai bảng. Cách này cho phép thay đổi giá trị trong bảng này theo giá trị tương ứng ở bảng khác.	main.sql -- Update employee department names based on department table UPDATE employee e SET department_name = d.name FROM department d WHERE e.department_id = d.id; -- Another way using subquery UPDATE employee SET department_name = (SELECT name FROM department WHERE department.id = employee.department_id);

	Khác biệt giữa NULL và chuỗi rỗng là gì?
	
	Trong SQL, NULL biểu thị không có giá trị — nghĩa là giá trị chưa biết hoặc bị thiếu. Chuỗi rỗng ('') là một giá trị xác định nhưng không có ký tự nào. NULL và chuỗi rỗng không giống nhau, và khi so sánh với NULL, bạn phải dùng IS NULL hoặc IS NOT NULL.
		main.sql
		-- NULL example
		SELECT * FROM employee
		WHERE middle_name IS NULL; -- selects employees with unknown middle name
		
		-- Empty string example
		SELECT * FROM employee
		WHERE middle_name = ''; -- selects employees whose middle name is explicitly empty
		
		-- Optional: checking differences / Kiểm tra sự khác biệt
		SELECT
		    COUNT(*) AS null_count,
		    COUNT(CASE WHEN middle_name = '' THEN 1 END) AS empty_string_count
		FROM employee;

	Làm thế nào để sắp xếp kết quả truy vấn theo thứ tự tăng dần và giảm dần? Trong SQL, để sắp xếp kết quả truy vấn, bạn dùng ORDER BY. Mặc định, ORDER BY sắp xếp theo thứ tự tăng dần (ASC), và bạn có thể dùng DESC để sắp xếp giảm dần. Bạn cũng có thể sắp xếp theo nhiều cột, mỗi cột có thứ tự riêng.	main.sql -- Sort employees by salary ascending SELECT * FROM employee ORDER BY salary ASC; -- Sort employees by salary descending SELECT * FROM employee ORDER BY salary DESC; -- Sort by department ascending and then salary descending SELECT * FROM employee ORDER BY department_id ASC, salary DESC;
	Các hàm tổng hợp (aggregate functions) trong SQL là gì? Cho ví dụ. Trong SQL, hàm tổng hợp (aggregate functions) là các hàm thực hiện các phép tính trên tập hợp giá trị và trả về một giá trị tóm tắt duy nhất. Chúng thường được dùng cùng GROUP BY để tổng hợp dữ liệu theo nhóm. Các hàm tổng hợp phổ biến gồm COUNT(), SUM(), AVG(), MAX(), MIN().	main.sql -- Count the total number of employees SELECT COUNT(*) AS total_employees FROM employee; -- Sum of all salaries SELECT SUM(salary) AS total_salary FROM employee; -- Average salary SELECT AVG(salary) AS average_salary FROM employee; -- Maximum and minimum salary SELECT MAX(salary) AS max_salary, MIN(salary) AS min_salary FROM employee; -- Count employees in each department SELECT department_id, COUNT(*) AS num_employees FROM employee GROUP BY department_id;
	Khác biệt giữa toán tử BETWEEN và IN là gì? Trong SQL, BETWEEN dùng để lọc các giá trị nằm trong một khoảng, bao gồm cả giá trị biên. IN dùng để lọc các giá trị khớp với bất kỳ giá trị nào trong danh sách. BETWEEN phù hợp cho khoảng liên tục, còn IN phù hợp cho tập hợp các giá trị rời rạc.	main.sql -- Using BETWEEN: select employees with salary between 40000 and 60000 SELECT * FROM employee WHERE salary BETWEEN 40000 AND 60000; -- Using IN: select employees in departments 1, 2, or 3 SELECT * FROM employee WHERE department_id IN (1, 2, 3); -- Optional: combine with NOT / Kết hợp với NOT -- NOT BETWEEN SELECT * FROM employee WHERE salary NOT BETWEEN 40000 AND 60000; -- NOT IN SELECT * FROM employee WHERE department_id NOT IN (1, 2, 3);
	Làm thế nào để tìm độ dài của một chuỗi hoặc số ký tự trong một cột? Trong SQL, để tìm độ dài của một chuỗi hoặc số ký tự trong một cột, bạn dùng hàm LENGTH() trong PostgreSQL. Hàm này trả về tổng số ký tự, bao gồm cả khoảng trắng.	main.sql -- Find the length of employee names SELECT name, LENGTH(name) AS name_length FROM employee; -- Example: find employees with name longer than 10 characters SELECT * FROM employee WHERE LENGTH(name) > 10; -- Optional: using for numeric values / Dùng cho số: -- Convert number to text to find length SELECT id, LENGTH(CAST(id AS TEXT)) AS id_length FROM employee;

	Khác biệt giữa DISTINCT và không dùng DISTINCT trong câu lệnh SELECT là gì? Trong SQL, DISTINCT trong câu lệnh SELECT dùng để trả về các giá trị duy nhất, loại bỏ các bản ghi trùng lặp trong kết quả. Nếu không dùng DISTINCT, truy vấn sẽ trả về tất cả các hàng khớp, bao gồm cả các bản ghi trùng lặp. DISTINCT hữu ích khi bạn muốn tránh các giá trị bị lặp trong kết quả.	main.sql -- Without DISTINCT: may return duplicate department IDs SELECT department_id FROM employee; -- With DISTINCT: only unique department IDs SELECT DISTINCT department_id FROM employee; -- Example with multiple columns SELECT DISTINCT department_id, job_title FROM employee;
	Làm thế nào để đổi tên cột hoặc bảng trong kết quả truy vấn? Trong SQL, để đổi tên cột hoặc bảng trong kết quả truy vấn, bạn dùng alias với từ khóa AS. Alias không thay đổi tên cột hay bảng trong cơ sở dữ liệu, chỉ đổi tên trong kết quả truy vấn để dễ đọc hơn.	main.sql -- Rename a column in the result SELECT name AS employee_name, salary AS employee_salary FROM employee; -- Rename a table in the query SELECT e.name, e.salary FROM employee AS e; -- Using both column and table aliases SELECT e.name AS employee_name, e.salary AS employee_salary FROM employee AS e;
	Làm thế nào để giới hạn số hàng trả về trong một truy vấn? Trong SQL, để giới hạn số hàng trả về trong một truy vấn, bạn dùng LIMIT. Cách này giúp chỉ lấy một số hàng nhất định thay vì toàn bộ kết quả. Trong PostgreSQL, bạn có thể kết hợp với OFFSET để bỏ qua một số hàng đầu tiên.	main.sql -- Get only 5 rows SELECT * FROM employee LIMIT 5; -- Get 5 rows starting from the 6th row SELECT * FROM employee LIMIT 5 OFFSET 5;
Intermediate	Khác biệt giữa PRIMARY KEY và UNIQUE. Trong SQL, cả PRIMARY KEY và UNIQUE đều đảm bảo tính duy nhất cho một cột hoặc tập cột. Điểm khác nhau chính là: PRIMARY KEY dùng để xác định duy nhất mỗi hàng trong bảng và không cho phép NULL, và mỗi bảng chỉ có một khóa chính. UNIQUE cũng đảm bảo duy nhất nhưng có thể chứa một giá trị NULL (trong hầu hết cơ sở dữ liệu) và mỗi bảng có thể có nhiều ràng buộc UNIQUE.	main.sql -- PRIMARY KEY CREATE TABLE employee (id SERIAL PRIMARY KEY, name VARCHAR(50)); -- UNIQUE constraint CREATE TABLE employee_email (id SERIAL PRIMARY KEY, email VARCHAR(100) UNIQUE); -- Optional: multiple UNIQUE constraints CREATE TABLE example (col1 INT UNIQUE, col2 INT UNIQUE);
	Khác biệt giữa CHECK và DEFAULT constraint. Trong SQL, CHECK và DEFAULT có mục đích khác nhau. CHECK đảm bảo một quy tắc cho giá trị được chèn vào cột, giúp dữ liệu hợp lệ. DEFAULT tự động gán một giá trị mặc định cho cột khi không có giá trị nào được cung cấp khi chèn dữ liệu.	main.sql -- CHECK constraint: salary must be positive CREATE TABLE employee (id SERIAL PRIMARY KEY, name VARCHAR(50), salary NUMERIC CHECK (salary > 0)); -- DEFAULT constraint: department defaults to 'General' CREATE TABLE employee_default (id SERIAL PRIMARY KEY, name VARCHAR(50), department VARCHAR(50) DEFAULT 'General'); -- Example insert INSERT INTO employee_default (name) VALUES ('Tom'); -- department will automatically be 'General'

	Cascade delete là gì? Khi nào nên dùng? Trong SQL, cascade delete là một tùy chọn của khóa ngoại cho phép tự động xóa các hàng con khi hàng cha tương ứng bị xóa. Điều này đảm bảo tính toàn vẹn dữ liệu mà không để lại các bản ghi "mồ côi" trong bảng liên quan. Nên dùng cascade delete khi bạn muốn các bản ghi liên quan tự động bị xóa cùng với bản ghi cha, ví dụ xóa một đơn hàng và tất cả các chi tiết đơn hàng của nó. Cần thận trọng khi sử dụng vì có thể xóa nhầm nhiều dữ liệu nếu không cẩn thận.	main.sql -- Parent table: department CREATE TABLE department (id SERIAL PRIMARY KEY, name VARCHAR(50)); -- Child table: employee with foreign key and cascade delete CREATE TABLE employee (id SERIAL PRIMARY KEY, name VARCHAR(50), department_id INT REFERENCES department(id) ON DELETE CASCADE); -- Deleting a department automatically deletes all employees in that department DELETE FROM department WHERE id = 1;
	Khác biệt giữa ON DELETE CASCADE và ON DELETE SET NULL. Trong SQL, cả ON DELETE CASCADE và ON DELETE SET NULL đều là hành động khóa ngoại được thực hiện khi một hàng cha bị xóa. ON DELETE CASCADE tự động xóa tất cả các hàng con tham chiếu đến hàng cha, đảm bảo không còn bản ghi "mồ côi". ON DELETE SET NULL không xóa hàng con mà thay vào đó đặt giá trị cột khóa ngoại trong bảng con thành NULL, giữ lại hàng con nhưng mất tham chiếu đến hàng cha đã xóa.	main.sql -- Parent table: department CREATE TABLE department (id SERIAL PRIMARY KEY, name VARCHAR(50)); -- Child table with ON DELETE CASCADE CREATE TABLE employee_cascade (id SERIAL PRIMARY KEY, name VARCHAR(50), department_id INT REFERENCES department(id) ON DELETE CASCADE); -- Child table with ON DELETE SET NULL CREATE TABLE employee_setnull (id SERIAL PRIMARY KEY, name VARCHAR(50), department_id INT REFERENCES department(id) ON DELETE SET NULL); -- Delete department with id = 1 DELETE FROM department WHERE id = 1; -- In employee_cascade: all employees with department_id = 1 are deleted -- In employee_setnull: all employees with department_id = 1 have department_id set to NULL
	Làm thế nào để tối ưu một truy vấn chậm? Trong SQL, để tối ưu một truy vấn chậm, bạn có thể áp dụng nhiều cách. Đầu tiên, phân tích và thêm index cho các cột trong WHERE, JOIN, ORDER BY hoặc GROUP BY để tăng tốc tìm kiếm. Thứ hai, **tránh SELECT *** và chỉ lấy những cột cần thiết. Thứ ba, viết lại truy vấn phức tạp bằng JOIN, subquery hoặc CTE hiệu quả hơn. Thứ tư, kiểm tra execution plan bằng EXPLAIN để tìm điểm nghẽn. Cuối cùng, xem xét caching, phân vùng (partitioning), hoặc denormalization nếu phù hợp. Một số lưu ý nhanh: - Đảm bảo các cột dùng để lọc (WHERE) và join có index. - Tránh sử dụng hàm trên các cột đã có index trong WHERE (ví dụ LENGTH(name) > 5 có thể làm index không được dùng). - Giới hạn lượng dữ liệu quét bằng cách sử dụng các điều kiện WHERE phù hợp.	main.sql -- Check how the query runs EXPLAIN ANALYZE SELECT e.id, e.name, d.name AS department FROM employee e JOIN department d ON e.department_id = d.id WHERE e.salary > 50000;
	Khi nào nên dùng Index? Khi nào không nên? Trong SQL, bạn nên dùng index khi có các truy vấn thường xuyên lọc, sắp xếp, hoặc join trên một cột, đặc biệt với các bảng lớn. Index giúp truy vấn SELECT nhanh hơn nhiều vì cơ sở dữ liệu không phải quét toàn bộ bảng. Bạn nên tránh dùng index khi: bảng nhỏ (full scan đủ nhanh), cột thường xuyên thay đổi (INSERT/UPDATE/DELETE sẽ chậm hơn), hoặc có quá nhiều index gây tốn bộ nhớ và chi phí bảo trì.	main.sql -- Create an index on the salary column CREATE INDEX idx_employee_salary ON employee(salary); -- Use index when filtering by salary SELECT * FROM employee WHERE salary > 50000; -- Optional: Avoid indexing small or frequently updated columns -- Not recommended for a column updated every few frequently CREATE INDEX idx_temp_data ON employee(temp_value);
	Khác biệt giữa clustered và non-clustered index. Trong SQL, clustered index quyết định thứ tự vật lý của dữ liệu trong bảng. Mỗi bảng chỉ có một clustered index, và các hàng được lưu theo thứ tự của index này. Non-clustered index không thay đổi thứ tự vật lý của bảng; nó tạo ra một cấu trúc riêng trỏ tới dữ liệu thực, và mỗi bảng có thể có nhiều non-clustered index. Clustered index tốt cho các truy vấn theo khoảng (range queries), còn non-clustered index tốt cho tra cứu theo cột cụ thể. Điểm chính: - Clustered index = thứ tự vật lý của các hàng trong bảng. - Non-clustered index = cấu trúc riêng trỏ tới các hàng dữ liệu. - Mỗi bảng → chỉ 1 clustered index, nhưng có thể có nhiều non-clustered index.	main.sql -- PostgreSQL automatically creates a clustered index for PRIMARY KEY CREATE TABLE employee (id SERIAL PRIMARY KEY, name VARCHAR(50), salary NUMERIC); -- Create a non-clustered index (PostgreSQL uses B-tree by default) CREATE INDEX idx_employee_salary ON employee(salary); -- Optional: cluster table manually on a column (PostgreSQL) CLUSTER employee USING idx_employee_salary;

	Explain Plan là gì? Dùng để làm gì? Trong SQL, Explain Plan là công cụ hiển thị cách cơ sở dữ liệu thực thi một truy vấn, bao gồm các bước, thứ tự thực hiện và các index được sử dụng. Nó được dùng chủ yếu để phân tích và tối ưu truy vấn bằng cách xác định các điểm nghẽn hoặc thao tác không hiệu quả. Lưu ý nhanh: - Giúp biết index nào được sử dụng - Giúp ước lượng chi phí truy vấn - Giúp phát hiện quét toàn bộ bảng	main.sql -- Check the execution plan of a query EXPLAIN SELECT e.id, e.name, d.name AS department FROM employee e JOIN department d ON e.department_id = d.id WHERE e.salary > 50000; -- Check the actual execution with timing EXPLAIN ANALYZE SELECT e.id, e.name, d.name AS department FROM employee e JOIN department d ON e.department_id = d.id WHERE e.salary > 50000;
	Khác biệt giữa transaction và lock. Trong SQL, transaction là một đơn vị công việc logic gồm một hoặc nhiều câu lệnh SQL được thực hiện như một khối duy nhất. Transaction tuân theo các thuộc tính ACID (Atomicity – Tính nguyên tử, Consistency – Tính nhất quán, Isolation – Tách biệt, Durability – Tính bền vững) để đảm bảo toàn vẹn dữ liệu. Lock là cơ chế mà cơ sở dữ liệu dùng để kiểm soát truy cập đồng thời vào dữ liệu, tránh các xung đột như đọc dữ liệu bẩn (dirty read) hoặc mất cập nhật (lost update). Trong khi transaction xác định công việc thực hiện nguyên tử, lock là công cụ mà cơ sở dữ liệu sử dụng để quản lý đồng thời trong transaction. Lưu ý nhanh: - Transaction = đơn vị công việc logic - Lock = cơ chế kiểm soát truy cập đồng thời - Transaction sử dụng lock nội bộ để bảo vệ dữ liệu	main.sql -- Start a transaction BEGIN; -- Update a row UPDATE employee SET salary = salary + 5000 WHERE id = 1; -- Commit or rollback COMMIT; -- apply changes -- ROLLBACK; -- undo changes -- Example of a lock on a table LOCK TABLE employee IN ACCESS EXCLUSIVE MODE;
	Giải thích sự khác biệt giữa INNER JOIN và CROSS JOIN. INNER JOIN trả về các hàng từ hai bảng khi có sự trùng khớp theo điều kiện chỉ định, thường dựa trên khóa chính và khóa ngoại. Chỉ các hàng thỏa mãn điều kiện mới xuất hiện trong kết quả. CROSS JOIN, ngược lại, trả về tích Descartes của hai bảng, nghĩa là mỗi hàng từ bảng thứ nhất được kết hợp với mỗi hàng từ bảng thứ hai, bất kể điều kiện nào. CROSS JOIN có thể tạo ra rất nhiều hàng nếu cả hai bảng lớn. Lưu ý nhanh: - INNER JOIN = ghép các hàng theo điều kiện - CROSS JOIN = tất cả các kết hợp giữa các hàng - Dùng CROSS JOIN cẩn thận vì có thể tạo ra tập kết quả rất lớn	main.sql -- INNER JOIN example SELECT e.name, d.name AS department FROM employee e INNER JOIN department d ON e.department_id = d.id; -- CROSS JOIN example SELECT e.name, d.name AS department FROM employee e CROSS JOIN department d;

	Subquery là gì và khi nào bạn sẽ sử dụng nó?
	
	Subquery là một truy vấn lồng bên trong truy vấn khác, thường nằm trong câu lệnh SELECT, INSERT, UPDATE hoặc DELETE. Subquery được dùng khi bạn cần lấy một giá trị hoặc tập giá trị từ bảng này để sử dụng trong truy vấn khác. Chúng giúp chia các truy vấn phức tạp thành các phần dễ quản lý và có thể dùng để lọc, tính tổng hợp, hoặc kiểm tra sự tồn tại.
	
	Lưu ý nhanh:
	Subquery = truy vấn lồng trong truy vấn khác
	Có thể trả về giá trị đơn, danh sách, hoặc bảng
	Dùng để lọc, tổng hợp, hoặc kiểm tra điều kiện
		main.sql
		-- Subquery in WHERE clause
		SELECT name, salary
		FROM employee
		WHERE department_id = (
		  SELECT id
		  FROM department
		  WHERE name = 'Sales'
		);
		
		-- Subquery in SELECT clause
		SELECT name, (SELECT name FROM department d WHERE d.id = e.department_id) AS department_name
		FROM employee e;

	Correlated subquery là gì? Cho ví dụ.
	
	Correlated subquery là một subquery phụ thuộc vào một cột từ truy vấn bên ngoài. Nó được thực thi một lần cho mỗi hàng của truy vấn bên ngoài, khác với subquery thông thường chạy độc lập. Correlated subquery hữu ích khi bạn cần so sánh hoặc tính toán từng hàng dựa trên dữ liệu của truy vấn ngoài.
	
	Lưu ý nhanh:
	Correlated subquery tham chiếu cột của truy vấn ngoài
	Thực thi một lần cho mỗi hàng của truy vấn ngoài
	Dùng để so sánh hoặc tính toán điều kiện theo từng hàng
		main.sql
		-- Find employees whose salary is greater than the average salary of their department
		SELECT e1.name, e1.salary, e1.department_id
		FROM employee e1
		WHERE e1.salary > (
		  SELECT AVG(e2.salary)
		  FROM employee e2
		  WHERE e2.department_id = e1.department_id
		);

	Giải thích sự khác biệt giữa EXISTS và IN. Trong SQL, cả EXISTS và IN đều dùng để lọc hàng dựa trên một truy vấn khác, nhưng cách hoạt động khác nhau. EXISTS kiểm tra xem subquery có trả về bất kỳ hàng nào hay không và trả về TRUE hoặc FALSE cho mỗi hàng của truy vấn ngoài; thường hiệu quả hơn với dữ liệu lớn. IN kiểm tra xem giá trị của một cột có khớp với bất kỳ giá trị nào trong danh sách hoặc subquery hay không. EXISTS thường nhanh hơn khi subquery trả về nhiều hàng, còn IN dễ đọc hơn với danh sách nhỏ. Lưu ý nhanh: EXISTS = TRUE/FALSE dựa trên kết quả subquery IN = so khớp giá trị trong danh sách hoặc subquery EXISTS thường nhanh hơn với subquery lớn	main.sql -- Using IN SELECT name FROM employee WHERE department_id IN (SELECT id FROM department WHERE name LIKE 'S%'); -- Using EXISTS SELECT name FROM employee e WHERE EXISTS (SELECT 1 FROM department d WHERE d.id = e.department_id AND d.name LIKE 'S%');
	Làm thế nào để đảm bảo tính toàn vẹn dữ liệu bằng các ràng buộc khác ngoài PRIMARY KEY và FOREIGN KEY? Trong SQL, ngoài PRIMARY KEY và FOREIGN KEY, bạn có thể đảm bảo tính toàn vẹn dữ liệu bằng các ràng buộc khác như UNIQUE, CHECK, và DEFAULT. UNIQUE đảm bảo một cột hoặc tập cột có giá trị duy nhất. CHECK áp dụng quy tắc cho giá trị cột, ví dụ giới hạn lương phải lớn hơn 0. DEFAULT tự động gán giá trị mặc định cho cột nếu không có giá trị nào được cung cấp. Những ràng buộc này giúp dữ liệu trong cơ sở dữ liệu hợp lệ, nhất quán và dự đoán được.	main.sql CREATE TABLE employee (id SERIAL PRIMARY KEY, name VARCHAR(50) UNIQUE, -- Ràng buộc UNIQUE salary NUMERIC CHECK (salary > 0), -- Ràng buộc CHECK department VARCHAR(50) DEFAULT 'General' -- Ràng buộc DEFAULT);
	Khác biệt giữa CASE và IF trong SQL là gì? Trong SQL, CASE và IF dùng cho logic điều kiện, nhưng khác nhau. CASE là chuẩn SQL và có thể dùng trong SELECT, UPDATE, ORDER BY và các câu lệnh khác; nó có thể đánh giá nhiều điều kiện và trả về kết quả khác nhau dựa trên chúng. IF thường là câu lệnh thủ tục được dùng trong các ngôn ngữ lập trình cơ sở dữ liệu như PL/pgSQL, T-SQL hoặc trong stored procedure, không dùng trực tiếp trong SELECT chuẩn.	main.sql -- Using CASE in a SELECT query SELECT name, salary, CASE WHEN salary > 50000 THEN 'High' WHEN salary BETWEEN 30000 AND 50000 THEN 'Medium' ELSE 'Low' END AS salary_level FROM employee;
	Composite key là gì? Nó khác gì so với khóa một cột? Composite key là một khóa chính gồm hai hoặc nhiều cột trong một bảng, nơi sự kết hợp giá trị của các cột này xác định duy nhất mỗi hàng. Khóa một cột (hoặc simple primary key) chỉ sử dụng một cột để xác định duy nhất các hàng. Composite key được dùng khi không có cột đơn nào đảm bảo tính duy nhất, nhưng sự kết hợp của các cột có thể đảm bảo.	main.sql -- Ví dụ composite key CREATE TABLE order_item (order_id INT, product_id INT, quantity INT, PRIMARY KEY (order_id, product_id) -- khóa composite); -- Ví dụ khóa một cột CREATE TABLE employee (id SERIAL PRIMARY KEY, -- khóa một cột name VARCHAR(50));
	Transactions trong SQL là gì và các thuộc tính ACID gồm những gì? Trong SQL, transaction (giao dịch) là một chuỗi các thao tác được xử lý như một đơn vị công việc duy nhất, nghĩa là hoặc tất cả thao tác đều thành công, hoặc toàn bộ bị hủy và dữ liệu quay về trạng thái ban đầu. Transaction giúp đảm bảo tính nhất quán của dữ liệu, đặc biệt khi có lỗi hoặc sự cố hệ thống. Tính đáng tin cậy của transaction được mô tả bằng các thuộc tính ACID: - Atomicity (Tính nguyên tử): tất cả thao tác phải thành công; nếu một thao tác thất bại thì toàn bộ giao dịch bị hủy. - Consistency (Tính nhất quán): dữ liệu phải chuyển từ trạng thái hợp lệ này sang trạng thái hợp lệ khác. - Isolation (Tính cô lập): các transaction không được ảnh hưởng lẫn nhau; trạng thái trung gian không được nhìn thấy. - Durability (Tính bền vững): khi giao dịch đã được commit, dữ liệu sẽ tồn tại vững chắc kể cả khi hệ thống gặp sự cố.	main.sql BEGIN; UPDATE accounts SET balance = balance - 100 WHERE id = 1; UPDATE accounts SET balance = balance + 100 WHERE id = 2; COMMIT; -- or ROLLBACK;
	Làm thế nào để ngăn chặn deadlock trong cơ sở dữ liệu? Để ngăn chặn deadlock trong cơ sở dữ liệu, bạn cần tuân thủ các nguyên tắc tốt về transaction và locking. Chiến lược quan trọng nhất là giữ cho transaction ngắn gọn và nhất quán, giúp giảm thời gian giữ khóa. Luôn truy cập các bảng và các dòng theo một thứ tự giống nhau trong toàn bộ ứng dụng để tránh tình trạng chờ vòng tròn. Sử dụng index phù hợp để giảm các lần quét toàn bảng có thể dẫn đến khóa lớn. Hạn chế khóa không cần thiết bằng cách chỉ truy vấn đúng các dòng cần thiết và dùng mức isolation thấp hơn khi phù hợp để giảm mức độ khóa. Cuối cùng, theo dõi các truy vấn và sử dụng công cụ như EXPLAIN để phát hiện các thao tác chậm có thể gây cạnh tranh khóa.	main.sql -- Truy cập bảng theo thứ tự nhất quán BEGIN; UPDATE orders SET status = 'paid' WHERE id = 10; UPDATE order_items SET shipped = true WHERE order_id = 10; COMMIT;

Advanced	Khác biệt giữa ROW_NUMBER(), RANK(), và DENSE_RANK().
	
	ROW_NUMBER(), RANK() và DENSE_RANK() là các hàm window dùng để đánh số thứ tự cho các dòng, nhưng chúng khác nhau khi có giá trị trùng. ROW_NUMBER() luôn gán số thứ tự duy nhất cho từng dòng, kể cả khi giá trị bằng nhau. RANK() gán cùng thứ hạng cho các dòng trùng giá trị nhưng nhảy số sau nhóm trùng. DENSE_RANK() cũng gán thứ hạng giống nhau cho các dòng trùng nhưng không nhảy số, thứ tự liền mạch.
	
	Lưu ý nhanh:
	- ROW_NUMBER(): không xét trùng, luôn 1,2,3,…
	- RANK(): trùng → cùng hạng, có nhảy số
	- DENSE_RANK(): trùng → cùng hạng, không nhảy số
		main.sql
		`SELECT name, score,`
		`ROW_NUMBER() OVER (ORDER BY score DESC) AS row_num,`
		`RANK() OVER (ORDER BY score DESC) AS rnk,`
		`DENSE_RANK() OVER (ORDER BY score DESC) AS dense_rnk`
		`FROM students;`

	Khác biệt giữa INNER JOIN và EXISTS.
	
	Sự khác biệt giữa INNER JOIN và EXISTS nằm ở cách chúng lọc dữ liệu và hiệu suất hoạt động. INNER JOIN kết hợp các dòng từ hai bảng dựa trên giá trị khớp nhau và trả về dữ liệu của cả hai bảng; nó phù hợp khi bạn cần lấy dữ liệu từ cả hai phía. EXISTS chỉ kiểm tra xem subquery có trả về ít nhất một dòng hay không; nó trả về TRUE hoặc FALSE cho từng dòng của truy vấn ngoài và không trả về cột từ subquery. EXISTS thường hiệu quả hơn khi bạn chỉ cần kiểm tra sự tồn tại của dữ liệu liên quan, đặc biệt vì subquery có thể dừng ngay khi tìm thấy dòng đầu tiên.
	
	Lưu ý nhanh:
	- INNER JOIN trả về dữ liệu kết hợp từ hai bảng.
	- EXISTS chỉ kiểm tra sự tồn tại, không cần trả về cột.
	- EXISTS có thể nhanh hơn khi subquery chứa nhiều dòng.
		main.sql
		`-- Ví dụ INNER JOIN`
		`SELECT e.name, d.name AS department`
		`FROM employee e`
		`INNER JOIN department d ON e.department_id = d.id;`
		
		`-- Ví dụ EXISTS`
		`SELECT e.name`
		`FROM employee e`
		`WHERE EXISTS (`
		`SELECT 1`
		`FROM department d`
		`WHERE d.id = e.department_id`
		`);`

	Viết truy vấn phân trang dữ liệu.
	
	Cách phổ biến để phân trang dữ liệu trong SQL là sử dụng LIMIT và OFFSET. LIMIT quy định số dòng muốn lấy, còn OFFSET cho biết phải bỏ qua bao nhiêu dòng trước khi bắt đầu trả dữ liệu. Điều này rất hữu ích khi bạn muốn hiển thị dữ liệu theo từng trang trong ứng dụng web. Khi phân trang kiểu page-number, bạn thường tính offset theo công thức: (số_trang - 1) * kích_thước_trang.
		main.sql
		`SELECT *`
		`FROM employees`
		`ORDER BY id`
		`LIMIT 10 OFFSET 20;`
		`-- Truy vấn này sẽ trả về trang số 3 nếu mỗi trang có 10 dòng, vì nó bỏ qua 20 dòng đầu và lấy 10 dòng tiếp theo.`

	Làm thế nào để xử lý NULL trong SQL?
	
	Trong SQL, NULL đại diện cho dữ liệu thiếu hoặc không xác định. Xử lý NULL đúng cách rất quan trọng để tránh lỗi hoặc kết quả sai. Một số cách xử lý:
	- IS NULL / IS NOT NULL để lọc các dòng có hoặc không có giá trị NULL.
	- COALESCE() để thay thế NULL bằng giá trị mặc định.
	- NULLIF() để trả về NULL nếu hai biểu thức bằng nhau.
	- Các hàm tổng hợp như COUNT(column) bỏ qua NULL, nhưng COUNT(*) tính tất cả các dòng.
	Xử lý NULL cẩn thận giúp tính toán và lọc dữ liệu chính xác.
		main.sql
		`-- Lọc các dòng có salary là NULL`
		`SELECT * FROM employees WHERE salary IS NULL;`
		
		`-- Thay thế NULL bằng giá trị mặc định`
		`SELECT name, COALESCE(salary, 0) AS salary FROM employees;`
		
		`-- Trả về NULL nếu hai giá trị bằng nhau`
		`SELECT name, NULLIF(salary, 0) AS salary_or_null FROM employees;`

	Khác biệt giữa STORED PROCEDURE và FUNCTION.
	
	Trong SQL, cả stored procedure và function đều là các khối mã có thể tái sử dụng và lưu trong cơ sở dữ liệu, nhưng có những điểm khác nhau quan trọng. Stored procedure có thể thực hiện các thao tác như INSERT, UPDATE, DELETE hoặc logic phức tạp và không nhất thiết phải trả về giá trị. Nó được gọi bằng lệnh CALL. Function bắt buộc trả về một giá trị và có thể dùng trong các biểu thức SQL, như trong SELECT hoặc WHERE. Function thường dùng cho tính toán hoặc chuyển đổi dữ liệu, còn procedure dùng để thực hiện logic nghiệp vụ hoặc các workflow.
		main.sql
		`-- Ví dụ function`
		`CREATE FUNCTION get_employee_salary(emp_id INT)`
		`RETURNS NUMERIC AS $$`
		`BEGIN`
		`RETURN (SELECT salary FROM employee WHERE id = emp_id);`
		`END;`
		`$$ LANGUAGE plpgsql;`
		
		`-- Ví dụ stored procedure`
		`CREATE PROCEDURE raise_salary(emp_id INT, amount NUMERIC)`
		`LANGUAGE plpgsql`
		`AS $$`
		`BEGIN`
		`UPDATE employee`
		`SET salary = salary + amount`
		`WHERE id = emp_id;`
		`END;`
		`$$;`

	Khác biệt giữa VIEW và MATERIALIZED VIEW. VIEW là một bảng ảo trong SQL được định nghĩa bởi một truy vấn. Nó không lưu trữ dữ liệu vật lý, nên mỗi lần bạn truy vấn, cơ sở dữ liệu sẽ thực hiện lại truy vấn gốc. MATERIALIZED VIEW, ngược lại, lưu kết quả truy vấn vật lý. Điều này cải thiện hiệu suất cho các truy vấn phức tạp vì dữ liệu đã được tính trước, nhưng có thể trở nên cũ nếu bảng gốc thay đổi, và cần làm mới thủ công. Lưu ý nhanh: - VIEW = bảng ảo, không lưu dữ liệu vật lý - MATERIALIZED VIEW = lưu kết quả truy vấn vật lý - Dùng materialized view để tăng hiệu suất cho các truy vấn nặng	main.sql -- Tạo view thông thường CREATE VIEW employee_view AS SELECT id, name, salary FROM employee WHERE salary > 50000; -- Tạo materialized view CREATE MATERIALIZED VIEW employee_mat_view AS SELECT id, name, salary FROM employee WHERE salary > 50000; -- Làm mới materialized view REFRESH MATERIALIZED VIEW employee_mat_view;
	Làm thế nào để tìm các sản phẩm chưa có trong đơn hàng? Để tìm các sản phẩm chưa có trong bất kỳ đơn hàng nào, bạn có thể dùng LEFT JOIN kết hợp kiểm tra NULL hoặc subquery với NOT EXISTS. Cách LEFT JOIN ghép bảng products với order_items và lọc các dòng mà order không tồn tại. Cách NOT EXISTS kiểm tra sản phẩm nào không có order tương ứng. Cả hai đều được, nhưng NOT EXISTS thường dễ đọc hơn trong trường hợp này. Lưu ý nhanh: - LEFT JOIN + NULL = tìm các dòng không khớp - NOT EXISTS = kiểm tra sự vắng mặt của các dòng liên quan - Cả hai đều trả về kết quả giống nhau; chọn cách nào tùy theo readability và query planner	main.sql -- Dùng LEFT JOIN SELECT p.id, p.name FROM products p LEFT JOIN order_items oi ON p.id = oi.product_id WHERE oi.product_id IS NULL; -- Dùng NOT EXISTS SELECT p.id, p.name FROM products p WHERE NOT EXISTS (SELECT 1 FROM order_items oi WHERE oi.product_id = p.id);
	Làm thế nào để xóa tất cả các bản ghi trùng nhau nhưng giữ lại một bản ghi duy nhất? Để xóa tất cả bản ghi trùng nhưng giữ lại một bản ghi duy nhất, bạn có thể dùng CTE (Common Table Expression) kết hợp hàm ROW_NUMBER(). Ý tưởng là đánh số thứ tự cho các bản ghi trùng trong mỗi nhóm giá trị giống nhau, sau đó xóa các dòng có số thứ tự > 1. Cách này đảm bảo giữ lại một dòng và loại bỏ tất cả bản trùng. Giải thích: - PARTITION BY name, email nhóm các bản ghi trùng theo các cột này. - ROW_NUMBER() gán số thứ tự 1 cho bản ghi đầu tiên trong mỗi nhóm. - DELETE xóa các dòng có rn > 1, giữ lại dòng đầu tiên.	main.sql WITH duplicates AS (SELECT id, ROW_NUMBER() OVER (PARTITION BY name, email ORDER BY id) AS rn FROM users) DELETE FROM users WHERE id IN (SELECT id FROM duplicates WHERE rn > 1);
	Viết truy vấn thống kê doanh số theo tháng, năm. Để tính thống kê doanh số theo tháng và năm, bạn có thể dùng hàm EXTRACT() trong PostgreSQL để lấy năm và tháng từ cột ngày, sau đó nhóm (GROUP BY) theo các giá trị này. Các hàm tổng hợp như SUM() tính tổng doanh số, COUNT() đếm số đơn, AVG() tính doanh số trung bình. Cách này hữu ích cho báo cáo theo tháng hoặc năm. Lưu ý nhanh: EXTRACT(YEAR FROM ...) và EXTRACT(MONTH FROM ...) lấy năm và tháng. Các hàm tổng hợp tóm tắt dữ liệu. GROUP BY cho phép tổng hợp theo tháng và năm.	main.sql SELECT EXTRACT(YEAR FROM order_date) AS year, EXTRACT(MONTH FROM order_date) AS month, COUNT(*) AS total_orders, SUM(total_amount) AS total_sales, AVG(total_amount) AS avg_sales FROM orders GROUP BY year, month ORDER BY year, month;
	Viết truy vấn để tính tổng, trung bình, max, min cho từng nhóm. Để tính tổng, trung bình, giá trị lớn nhất và nhỏ nhất cho từng nhóm, bạn sử dụng các hàm tổng hợp (SUM(), AVG(), MAX(), MIN()) cùng với GROUP BY. Điều này giúp tóm tắt dữ liệu theo từng nhóm, ví dụ theo phòng ban, danh mục, hoặc bất kỳ cột nhóm nào.	main.sql SELECT department_id, SUM(salary) AS total_salary, AVG(salary) AS avg_salary, MAX(salary) AS max_salary, MIN(salary) AS min_salary FROM employee GROUP BY department_id ORDER BY department_id;
	Làm thế nào để gộp nhiều cột thành một cột duy nhất? Để gộp nhiều cột thành một cột duy nhất trong SQL, bạn có thể dùng hàm nối chuỗi. Trong PostgreSQL, có thể dùng toán tử || hoặc hàm CONCAT(). Việc này hữu ích khi bạn muốn nối ví dụ tên và họ thành cột full name. Bạn cũng có thể thêm ký tự phân cách như khoảng trắng hoặc dấu phẩy. Lưu ý nhanh: - CONCAT() = nối nhiều cột hoặc giá trị - || = toán tử nối chuỗi trong PostgreSQL - Thêm ký tự phân cách (dấu cách, dấu phẩy) nếu cần	main.sql -- Dùng hàm CONCAT SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM employees; -- Dùng toán tử || SELECT first_name || ' ' || last_name AS full_name FROM employees;

	Giải thích sự khác biệt giữa temporary tables, table variables và CTEs (Common Table Expressions). Trong SQL, temporary tables, table variables, và CTEs (Common Table Expressions) đều dùng để lưu trữ kết quả trung gian, nhưng khác nhau về phạm vi, thời gian tồn tại và cách sử dụng. - Temporary tables (CREATE TEMP TABLE) được lưu trong cơ sở dữ liệu như bảng thường nhưng sẽ tự động xóa khi kết thúc phiên hoặc tùy chọn khi kết thúc transaction. Chúng có thể có index, constraint và thống kê, và có thể tham chiếu nhiều lần trong các truy vấn. Phù hợp cho dữ liệu lớn hoặc nhiều truy vấn liên tiếp. - Table variables (ví dụ T-SQL DECLARE @table TABLE) là các biến lưu cấu trúc bảng trong bộ nhớ. Chúng có phạm vi giới hạn (thường trong batch hoặc procedure) và thường không có thống kê hay index ngoài primary key. Phù hợp cho dữ liệu nhỏ và lưu trữ tạm nhẹ. - CTEs (Common Table Expressions) là kết quả tạm thời định nghĩa bằng WITH. Chúng chỉ tồn tại trong một truy vấn duy nhất và không lưu trữ vật lý. Thường dùng để tăng readability, viết truy vấn đệ quy, hoặc đơn giản hóa truy vấn phức tạp.	main.sql -- Temporary table CREATE TEMP TABLE temp_orders AS SELECT * FROM orders WHERE order_date >= '2025-01-01'; -- CTE WITH recent_orders AS (SELECT * FROM orders WHERE order_date >= '2025-01-01') SELECT customer_id, COUNT(*) FROM recent_orders GROUP BY customer_id;
	Làm thế nào để triển khai truy vấn đệ quy (recursive queries) trong SQL? Trong SQL, bạn triển khai truy vấn đệ quy bằng CTE (Common Table Expression) với từ khóa WITH RECURSIVE. Truy vấn đệ quy hữu ích cho dữ liệu dạng phân cấp, ví dụ sơ đồ tổ chức, bill-of-materials, hoặc cấu trúc cây. Một CTE đệ quy gồm hai phần: 1. Anchor member: truy vấn cơ sở cung cấp các dòng bắt đầu. 2. Recursive member: tham chiếu chính CTE để lặp lại lấy các dòng con hoặc liên quan. Quá trình đệ quy dừng khi không còn dòng mới nào được trả về.	main.sql -- Giả sử bảng employees có các cột id và manager_id: WITH RECURSIVE employee_hierarchy AS (-- Anchor member: chọn quản lý cấp cao nhất SELECT id, name, manager_id, 1 AS level FROM employees WHERE manager_id IS NULL UNION ALL -- Recursive member: chọn nhân viên thuộc quản lý ở cấp trước SELECT e.id, e.name, e.manager_id, eh.level + 1 FROM employees e INNER JOIN employee_hierarchy eh ON e.manager_id = eh.id) SELECT * FROM employee_hierarchy ORDER BY level, manager_id;
	What is windowing in SQL, and how is it different from GROUP BY? Windowing in SQL refers to window functions, which perform calculations across a set of rows related to the current row, without collapsing them into a single result per group. Unlike GROUP BY, which aggregates rows into a single output per group, window functions preserve the original number of rows while providing aggregates, rankings, or moving averages over a "window" of rows. Common window functions include ROW_NUMBER(), RANK(), SUM() OVER(...), AVG() OVER(...), and LEAD()/LAG(). The OVER() clause defines the window, optionally partitioned and ordered. Quick notes: GROUP BY collapses rows into one per group Window functions operate without reducing rows Useful for rankings, running totals, moving averages, etc.	main.sql -- Tính tổng lương theo phòng ban nhưng giữ từng dòng riêng SELECT id, name, department_id, salary, SUM(salary) OVER (PARTITION BY department_id) AS total_department_salary, ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn FROM employees;
	Giải thích sự khác biệt giữa RANK() và NTILE(). Cả RANK() và NTILE() đều là các hàm cửa sổ trong SQL dùng để đánh số cho các dòng, nhưng cách hoạt động khác nhau. - RANK() gán thứ hạng cho từng dòng theo thứ tự một cột. Các dòng có giá trị bằng nhau sẽ cùng thứ hạng, và có thể có khoảng nhảy trong thứ hạng sau các dòng trùng. - NTILE(n) chia các dòng đã sắp xếp thành n nhóm xấp xỉ bằng nhau và gán số nhóm (bucket) cho mỗi dòng. Nó không quan tâm đến các giá trị trùng mà chỉ đảm bảo phân phối các dòng vào số nhóm được chỉ định. Lưu ý nhanh: RANK = xếp hạng, có thể có khoảng nhảy do giá trị trùng NTILE = chia dòng thành các nhóm bằng nhau Dùng RANK để xếp vị trí, NTILE để chia percentile/quartile	main.sql -- Ví dụ RANK SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rnk FROM employees; -- Ví dụ NTILE (chia thành 4 phần) SELECT name, salary, NTILE(4) OVER (ORDER BY salary DESC) AS quartile FROM employees;

Làm thế nào để pivot các hàng thành cột trong SQL?

Để pivot các hàng thành cột trong SQL, bạn biến giá trị của hàng thành tiêu đề cột. Trong PostgreSQL, có thể dùng crosstab() từ extension tablefunc hoặc mô phỏng bằng CASE kết hợp GROUP BY. Pivot hữu ích cho báo cáo hoặc tóm tắt dữ liệu, ví dụ chuyển doanh số theo tháng thành cột riêng cho từng tháng.

main.sql

```sql
-- Ví dụ PostgreSQL dùng CASE
SELECT
    product_id,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 1 THEN total_amount ELSE 0 END) AS jan_sales,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 2 THEN total_amount ELSE 0 END) AS feb_sales,
    SUM(CASE WHEN EXTRACT(MONTH FROM order_date) = 3 THEN total_amount ELSE 0 END) AS mar_sales
FROM orders
GROUP BY product_id
ORDER BY product_id;

-- Ví dụ PostgreSQL dùng crosstab()
-- Kích hoạt extension tablefunc
CREATE EXTENSION IF NOT EXISTS tablefunc;
SELECT *
FROM crosstab(
    'SELECT product_id, EXTRACT(MONTH FROM order_date) AS month, SUM(total_amount)
     FROM orders
     GROUP BY product_id, month
     ORDER BY product_id, month'
) AS ct (product_id INT, jan NUMERIC, feb NUMERIC, mar NUMERIC);

```

Làm thế nào để unpivot các cột thành hàng trong SQL?

Để unpivot các cột thành hàng trong SQL, bạn biến nhiều cột thành một cột duy nhất với nhiều dòng. Trong PostgreSQL, có thể dùng phương pháp UNION ALL hoặc jsonb/lateral. Unpivot hữu ích khi bạn muốn chuẩn hóa dữ liệu để phân tích, ví dụ chuyển các cột doanh số theo tháng thành cột month và sales.

main.sql

```sql
-- Ví dụ PostgreSQL dùng UNION ALL
SELECT product_id, 'jan' AS month, jan_sales AS sales FROM product_sales
UNION ALL
SELECT product_id, 'feb' AS month, feb_sales AS sales FROM product_sales
UNION ALL
SELECT product_id, 'mar' AS month, mar_sales AS sales FROM product_sales;

-- Ví dụ PostgreSQL dùng VALUES + LATERAL
SELECT ps.product_id, v.month, v.sales
FROM product_sales ps,
LATERAL (
    VALUES
        ('jan', jan_sales),
        ('feb', feb_sales),
        ('mar', mar_sales)
) AS v(month, sales);

```

Giải thích sự khác biệt giữa INNER JOIN với nhiều bảng và sử dụng nhiều subquery.

Khi lấy dữ liệu từ nhiều bảng, bạn có thể dùng INNER JOIN nhiều bảng hoặc nhiều subquery, nhưng có sự khác nhau quan trọng.
- INNER JOIN với nhiều bảng kết hợp các dòng từ tất cả bảng dựa trên khóa phù hợp trong một truy vấn duy nhất. Nó thường hiệu quả hơn, dễ đọc hơn và cho phép trình tối ưu truy vấn thực hiện tối ưu hóa joins. Kết quả là một bộ dữ liệu duy nhất với các cột từ tất cả các bảng được join.
- Nhiều subquery liên quan đến việc chạy các truy vấn riêng biệt, thường trong WHERE hoặc SELECT. Điều này có thể ít hiệu quả hơn, đặc biệt nếu subquery chạy nhiều lần cho từng dòng, và khó đọc hơn. Subquery hữu ích khi bạn cần tính toán hoặc lọc dữ liệu độc lập trước khi kết hợp với truy vấn chính.

main.sql

```sql
-- INNER JOIN nhiều bảng
SELECT o.id AS order_id, c.name AS customer_name, p.name AS product_name
FROM orders o
INNER JOIN customers c ON o.customer_id = c.id
INNER JOIN order_items oi ON o.id = oi.order_id
INNER JOIN products p ON oi.product_id = p.id;

-- Dùng subquery
SELECT o.id AS order_id,
       (SELECT name FROM customers WHERE id = o.customer_id) AS customer_name,
       (SELECT name FROM products WHERE id = oi.product_id) AS product_name
FROM orders o
INNER JOIN order_items oi ON o.id = oi.order_id;

```

Làm thế nào để xử lý dữ liệu theo cấu trúc cây hoặc phân cấp (hierarchical/tree-structured data) trong SQL?

Dữ liệu theo cấu trúc cây hoặc phân cấp trong SQL, như sơ đồ tổ chức hoặc danh mục, có thể xử lý bằng truy vấn đệ quy với CTE (Common Table Expression). Ý tưởng là bắt đầu với root node (các mục cấp cao nhất) làm anchor, sau đó đệ quy lấy các node con cho đến khi lấy hết tất cả các cấp. Bạn cũng có thể dùng quan hệ parent_id/child_id và hàm window để tính mức hoặc đường dẫn.

main.sql

```sql
-- Giả sử bảng categories có id và parent_id:
WITH RECURSIVE category_tree AS (
    -- Anchor: danh mục cấp cao nhất
    SELECT id, name, parent_id, 1 AS level, name AS path
    FROM categories
    WHERE parent_id IS NULL

    UNION ALL

    -- Recursive: các mục con
    SELECT c.id, c.name, c.parent_id, ct.level + 1, ct.path || ' > ' || c.name
    FROM categories c
    INNER JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT *
FROM category_tree
ORDER BY path;

```

	Làm thế nào để triển khai soft delete (xóa logic) trong một bảng? Soft delete (xóa logic) nghĩa là bạn đánh dấu một dòng là đã xóa thay vì xóa vật lý khỏi bảng. Thường thực hiện bằng cách thêm cột như is_deleted hoặc deleted_at. Các truy vấn sau đó sẽ lọc các dòng đã xóa bằng cách kiểm tra cột này. Soft delete hữu ích để lưu lịch sử, khôi phục dữ liệu, hoặc phục vụ audit. Cách thay thế: dùng cột timestamp deleted_at thay cho boolean, ghi lại thời điểm xóa. Lưu ý nhanh: Soft delete = đánh dấu dòng, không xóa vật lý Lọc truy vấn để loại bỏ các dòng đã xóa Cho phép khôi phục và audit dữ liệu	main.sql -- Thêm cột đánh dấu dòng bị xóa ALTER TABLE employees ADD COLUMN is_deleted BOOLEAN DEFAULT FALSE; -- Soft delete một dòng UPDATE employees SET is_deleted = TRUE WHERE id = 123; -- Truy vấn các dòng còn hiệu lực (chưa xóa) SELECT * FROM employees WHERE is_deleted = FALSE;
	Làm thế nào để xác định và tối ưu các truy vấn có lượng I/O cao? Các truy vấn có lượng I/O cao có thể làm chậm cơ sở dữ liệu vì chúng đọc hoặc ghi nhiều dữ liệu từ đĩa. Để xác định, bạn có thể dùng execution plan, thống kê của cơ sở dữ liệu, hoặc công cụ giám sát. Dấu hiệu phổ biến bao gồm sequential scan trên bảng lớn, join lớn, hoặc scan toàn bộ bảng thay vì sử dụng index. Để tối ưu các truy vấn I/O cao: 1. Dùng index trên các cột thường xuất hiện trong WHERE, JOIN, hoặc ORDER BY. 2. Giới hạn dữ liệu quét bằng điều kiện lọc thích hợp. 3. Tránh SELECT * không cần thiết; chỉ lấy cột cần thiết. 4. Tách các truy vấn phức tạp thành các bước nhỏ hoặc tổng hợp dữ liệu trước nếu có thể. 5. Phân tích query plan (EXPLAIN) để thấy các thao tác nào gây I/O cao. Lưu ý nhanh: Kiểm tra sequential scan trên các bảng lớn So sánh estimated rows và actual rows trong EXPLAIN Thêm index hoặc viết lại truy vấn có thể giảm I/O	main.sql -- Phân tích kế hoạch truy vấn EXPLAIN ANALYZE SELECT * FROM orders o JOIN customers c ON o.customer_id = c.id WHERE o.order_date >= '2025-01-01';
	Giải thích sự khác biệt giữa NOLOCK (hoặc READ UNCOMMITTED) và mức isolation mặc định. NOLOCK (hoặc READ UNCOMMITTED) là mức isolation cho phép truy vấn đọc dữ liệu mà không cần shared lock. Điều này có nghĩa là truy vấn có thể đọc dữ liệu chưa được commit ("dirty") từ các transaction khác. Nó giúp tăng hiệu năng và giảm khóa, nhưng có thể dẫn đến dữ liệu không chính xác hoặc không nhất quán. Mức isolation mặc định trong hầu hết cơ sở dữ liệu (ví dụ READ COMMITTED trong PostgreSQL, SQL Server) đảm bảo rằng truy vấn chỉ đọc dữ liệu đã được commit, ngăn chặn dirty reads. Mức này an toàn hơn nhưng có thể gây khóa hoặc blocking khi nhiều transaction chạy đồng thời. Lưu ý nhanh: NOLOCK = nhanh, không chặn, có thể đọc dữ liệu chưa commit Mặc định = an toàn, chỉ đọc dữ liệu đã commit, có thể gây blocking Dùng NOLOCK cẩn thận; tránh cho dữ liệu quan trọng như tài chính hoặc audit	main.sql -- Dùng NOLOCK / READ UNCOMMITTED SELECT * FROM orders WITH (NOLOCK) WHERE order_date >= '2025-01-01'; -- Mức READ COMMITTED mặc định (không cần hint) SELECT * FROM orders WHERE order_date >= '2025-01-01';
	Làm thế nào để merge dữ liệu từ hai bảng (UPSERT) trong SQL? UPSERT trong SQL là kết hợp giữa INSERT và UPDATE: chèn dòng mới nếu chưa tồn tại, hoặc cập nhật dòng hiện có nếu đã tồn tại. Trong PostgreSQL, điều này thực hiện bằng INSERT ... ON CONFLICT, chỉ định constraint hoặc primary key để phát hiện xung đột. UPSERT hữu ích khi đồng bộ hoặc merge dữ liệu giữa các bảng. Lưu ý nhanh: UPSERT = chèn nếu mới, cập nhật nếu đã tồn tại ON CONFLICT (key) = phát hiện key trùng EXCLUDED = dòng sẽ được chèn nếu không xung đột	main.sql -- Giả sử bạn có hai bảng: source_table và target_table với primary key id. INSERT INTO target_table (id, name, value) SELECT id, name, value FROM source_table ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name, value = EXCLUDED.value;

	Giải thích cách indexing ảnh hưởng đến hiệu năng của các thao tác INSERT, UPDATE và DELETE. Index giúp cải thiện hiệu năng SELECT bằng cách cho phép cơ sở dữ liệu nhanh chóng tìm các dòng, nhưng nó cũng ảnh hưởng đến các thao tác INSERT, UPDATE và DELETE: - INSERT: Khi chèn một dòng mới, tất cả các index liên quan cũng phải được cập nhật. Nhiều index = tốn tài nguyên hơn → insert chậm hơn. - UPDATE: Cập nhật các cột có index cũng phải cập nhật index. Nếu nhiều index tồn tại trên các cột được cập nhật, thao tác sẽ chậm hơn. - DELETE: Xóa một dòng yêu cầu xóa tất cả các mục trong các index liên quan đến dòng đó, làm tăng overhead. Tóm lại, index cải thiện hiệu năng đọc nhưng làm tăng khối lượng công việc cho các thao tác ghi. Cần cân nhắc giữa tốc độ truy vấn và hiệu năng ghi. Lưu ý nhanh: Nhiều index = đọc nhanh, ghi chậm Chỉ index các cột thường được truy vấn Cân nhắc index có điều kiện hoặc partial index nếu phù hợp	