**Subquery**

In SQL Server, a subquery (also known as an inner query or nested query) is a query nested within another SQL statement, such as SELECT, INSERT, UPDATE, or DELETE. Subqueries are enclosed within parentheses and can be used in various parts of a SQL statement where expressions are allowed.

Subqueries can be categorized into two types:

Single-row subquery: A subquery that returns only one row of results. Single-row subqueries are typically used in expressions where a single value is expected, such as in the SELECT list or WHERE clause.

Example:

SELECT FirstName, LastName

FROM Customers

WHERE CustomerID = (SELECT MAX(CustomerID) FROM Customers);

Multi-row subquery: A subquery that returns multiple rows of results. Multi-row subqueries are often used in expressions where a set of values is expected, such as in the IN or EXISTS predicates.

Example:

SELECT FirstName, LastName

FROM Customers

WHERE CustomerID IN (SELECT CustomerID FROM Orders);

Subqueries can also be correlated, meaning they reference columns from the outer query. Correlated subqueries are evaluated once for each row processed by the outer query.

Example of a correlated subquery:

SELECT FirstName, LastName

FROM Customers c

WHERE Age > (SELECT AVG(Age) FROM Customers WHERE Country = c.Country);

In this example, the subquery calculates the average age of customers in the same country as the current customer being processed by the outer query.

Important rules for Subqueries:

* You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
* A subquery is a query within another query. The outer query is called as main/outer query and inner query is called as subquery.
* The subquery generally executes first when the subquery doesn’t have any correlation with the main query; and when there is a correlation, the parser takes the decision on which query to execute on precedence and uses the output of the subquery accordingly.
* Subqueries are on the right side of the comparison operator.
* A subquery consists of all the clauses an ordinary SELECT clause can contain: GROUP BY, WHERE, HAVING, DISTINCT, TOP, etc. However, an ORDER BY clause is only used when a TOP clause is specified.

Syntax: There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

SELECT column\_name

FROM table\_name

WHERE column\_name expression operator

(SELECT COLUMN\_NAME from TABLE\_NAME WHERE ... );

Examples of subqueries for: INSERT, UPDATE, or DELETE:

* INSERT with Subquery: we can use a subquery in the VALUES clause of an INSERT statement to insert data from another table or the result of another query.

INSERT INTO Employees (FirstName, LastName, DepartmentID)

SELECT FirstName, LastName, DepartmentID

FROM TempEmployees WHERE Age > 30;

In this example, data is inserted into the Employees table from the TempEmployees table for employees older than 30.

Note: When using the “INSERT INTO ... SELECT” syntax in SQL, the “VALUES” keyword is not used. Instead, you directly specify the columns into which you want to insert data in the target table, followed by the “SELECT” statement that provides the data.

* UPDATE with Subquery: we can use a subquery in the SET clause of an UPDATE statement to update records based on the result of another query.

UPDATE Orders

SET Status = 'Shipped'

WHERE OrderID IN (SELECT OrderID FROM OrderDetails WHERE Quantity > 10);

In this example, the status of orders with a quantity greater than 10 in the OrderDetails table is updated to 'Shipped'.

* DELETE with Subquery: we can use a subquery in the WHERE clause of a DELETE statement to delete records based on the result of another query.

DELETE FROM Customers

WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Orders);

In this example, customers who do not have any orders (as identified by the subquery) are deleted from the Customers table.

**IN, ANY, and ALL**

IN, ANY, and ALL are comparison operators in SQL that are used to compare a value to a set of values returned by a subquery or a list of values. They are often used in WHERE clauses to filter rows based on certain conditions. Here's an explanation of each:

IN Operator: The IN operator checks if a value matches any value in a list or a subquery result.

Example:

SELECT \* FROM Products

WHERE CategoryID IN (1, 2, 3);

This query selects all products whose CategoryID is either 1, 2, or 3.

The IN operator is useful when you have a finite list of values to compare against.

ANY Operator: The ANY operator compares a value to each value in a set of values returned by a subquery and returns true if the comparison is true for any of the values.

Example:

SELECT \* FROM Products

WHERE Price > ANY (SELECT Price FROM SpecialOffers);

This query selects all products whose price is greater than any price listed in the SpecialOffers table.

The ANY operator is typically used with comparison operators such as =, >, <, etc.

ALL Operator: The ALL operator compares a value to each value in a set of values returned by a subquery and returns true if the comparison is true for all of the values.

Example:

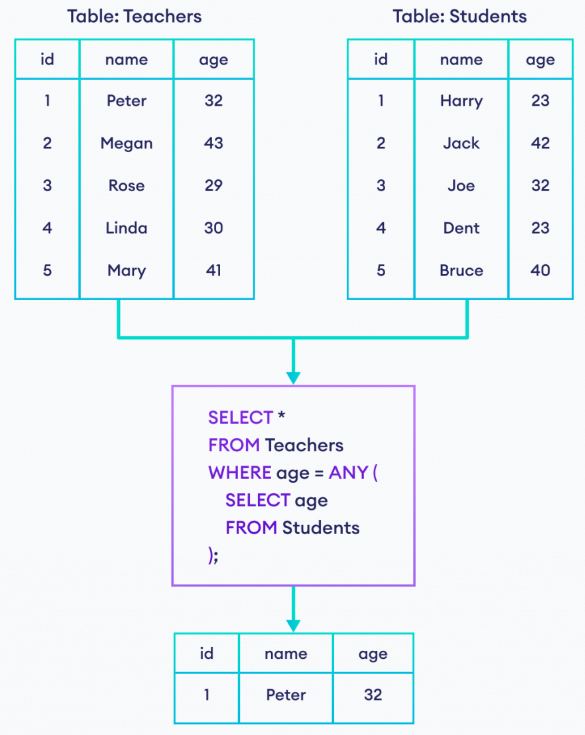
SELECT \* FROM Products

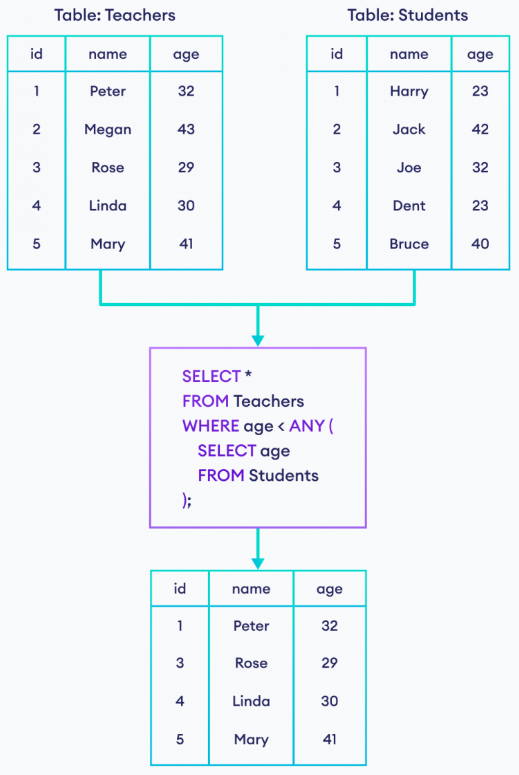
WHERE Price > ALL (SELECT Price FROM Products WHERE CategoryID = 1);

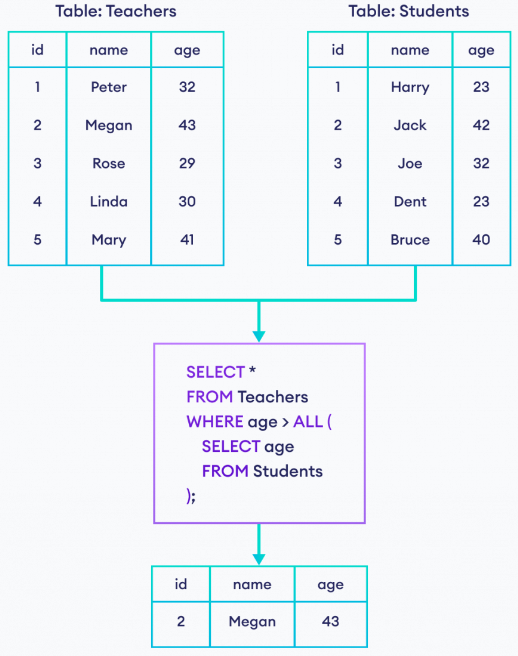
This query selects all products whose price is greater than the maximum price among products in CategoryID 1.

The ALL operator is typically used with comparison operators such as =, >, <, etc.

More examples to understand ‘any’ and ‘all’:







Exercise:

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(50),

CategoryID INT,

Price DECIMAL(10, 2)

);

INSERT INTO Products (ProductID, ProductName, CategoryID, Price)

VALUES

(1, 'Laptop', 1, 1000.00),

(2, 'Smartphone', 1, 700.00),

(3, 'Tablet', 1, 400.00),

(4, 'Headphones', 2, 150.00),

(5, 'Mouse', 2, 20.00),

(6, 'Keyboard', 2, 30.00),

(7, 'T-shirt', 3, 25.00),

(8, 'Jeans', 3, 50.00),

(9, 'Shoes', 3, 80.00),

(10, 'Sunglasses', 4, 60.00);

1. Select all products with a price greater than $100.

SELECT \* FROM Products WHERE Price > 100.00;

1. Select products with ProductID 2, 4, or 6.

SELECT \* FROM Products WHERE ProductID IN (2, 4, 6);

1. Select products with a price greater than the price of the 'Laptop'.

SELECT \* FROM Products WHERE Price > (SELECT Price FROM Products WHERE ProductName = 'Laptop');

1. Select products with a price greater than any price in the 'Electronics' category.

SELECT \* FROM Products WHERE Price > ANY (SELECT Price FROM Products WHERE CategoryID = 1);

1. Select products with a price less than all prices in the 'Clothing' category.

SELECT \* FROM Products WHERE Price < ALL (SELECT Price FROM Products WHERE CategoryID = 3);

1. Select products with a price equal to the maximum price among all products.

SELECT \* FROM Products WHERE Price = ALL (SELECT MAX(Price) FROM Products);

1. Select products with a price not equal to the price of any product in the 'Accessories' category.

SELECT \* FROM Products WHERE Price <> ANY (SELECT Price FROM Products WHERE CategoryID = 4);

1. Select products with a price greater than the average price of all products.

SELECT \* FROM Products WHERE Price > ALL (SELECT AVG(Price) FROM Products);

1. Select products with a price greater than the price of all products in the 'Electronics' category.

SELECT \* FROM Products WHERE Price > ALL (SELECT Price FROM Products WHERE CategoryID = 1);

**EXISTS and NOT EXISTS**

EXISTS and NOT EXISTS are logical operators used in conjunction with subqueries to test for the existence or non-existence of rows in a subquery result set. These operators return a Boolean value (TRUE or FALSE) based on whether the specified condition is met.

EXISTS Operator returns TRUE if the subquery returns one or more rows; otherwise, it returns FALSE.

Syntax:

SELECT columns FROM table WHERE EXISTS (subquery);

Example:

SELECT FirstName, LastName FROM Customers

WHERE EXISTS (SELECT \* FROM Orders WHERE Orders.CustomerID = Customers.CustomerID);

This query retrieves the first and last names of customers who have placed at least one order.

NOT EXISTS Operator returns TRUE if the subquery returns no rows; otherwise, it returns FALSE.

Syntax:

SELECT columns FROM table WHERE NOT EXISTS (subquery);

Example:

SELECT FirstName, LastName FROM Customers

WHERE NOT EXISTS (SELECT \* FROM Orders WHERE Orders.CustomerID = Customers.CustomerID);

This query retrieves the first and last names of customers who have not placed any orders.

* EXISTS and NOT EXISTS are typically used with correlated subqueries, where the outer query references columns from the inner subquery.
* These operators are commonly used in WHERE clauses to filter rows based on the existence or non-existence of related rows in another table.
* EXISTS and NOT EXISTS are often more efficient than using COUNT(\*) or other aggregate functions when you only need to check for the existence of rows.

Exercise: tables named -> Students, Courses, and Enrollments:

-- Create Students table

CREATE TABLE Students ( StudentID INT PRIMARY KEY, StudentName VARCHAR(50) );

-- Insert data into Students table

INSERT INTO Students (StudentID, StudentName) VALUES

(1, 'John'),

(2, 'Emma'),

(3, 'Michael'),

(4, 'Sophia'),

(5, 'William'),

(6, 'Olivia'),

(7, 'James'),

(8, 'Isabella'),

(9, 'Alexander'),

(10, 'Ava');

-- Create Courses table

CREATE TABLE Courses ( CourseID INT PRIMARY KEY, CourseName VARCHAR(50) );

-- Insert data into Courses table

INSERT INTO Courses (CourseID, CourseName) VALUES

(101, 'Mathematics'),

(102, 'English'),

(103, 'Science'),

(104, 'History'),

(105, 'Geography'),

(106, 'Computer Science'),

(107, 'Art'),

(108, 'Music'),

(109, 'Physical Education'),

(110, 'Foreign Language');

-- Create Enrollments table

CREATE TABLE Enrollments (EnrolmentID INT PRIMARY KEY, StudentID INT, CourseID INT,

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID));

-- Insert data into Courses table

INSERT INTO Enrollments (EnrolmentID, StudentID, CourseID) VALUES

(1, 1, 101),

(2, 1, 102),

(3, 2, 103),

(4, 2, 104),

(5, 3, 105),

(6, 3, 106),

(7, 4, 107),

(8, 5, 108),

(9, 6, 109),

(10, 7, 110);

1. Select students who are enrolled in at least one course.

SELECT StudentID, StudentName FROM Students WHERE EXISTS (SELECT \* FROM Enrollments WHERE Enrollments.StudentID = Students.StudentID);

1. Select students who are not enrolled in any course.
2. Select courses with at least one student enrolled.
3. Select courses with no students enrolled.
4. Select students who are enrolled in the 'Mathematics' course.
5. Select students who are not enrolled in the 'English' course.
6. Select courses with more than three students enrolled.
7. Select courses with no more than two students enrolled.
8. Select students who are enrolled in exactly three courses.
9. Select students who are not enrolled in more than one course.