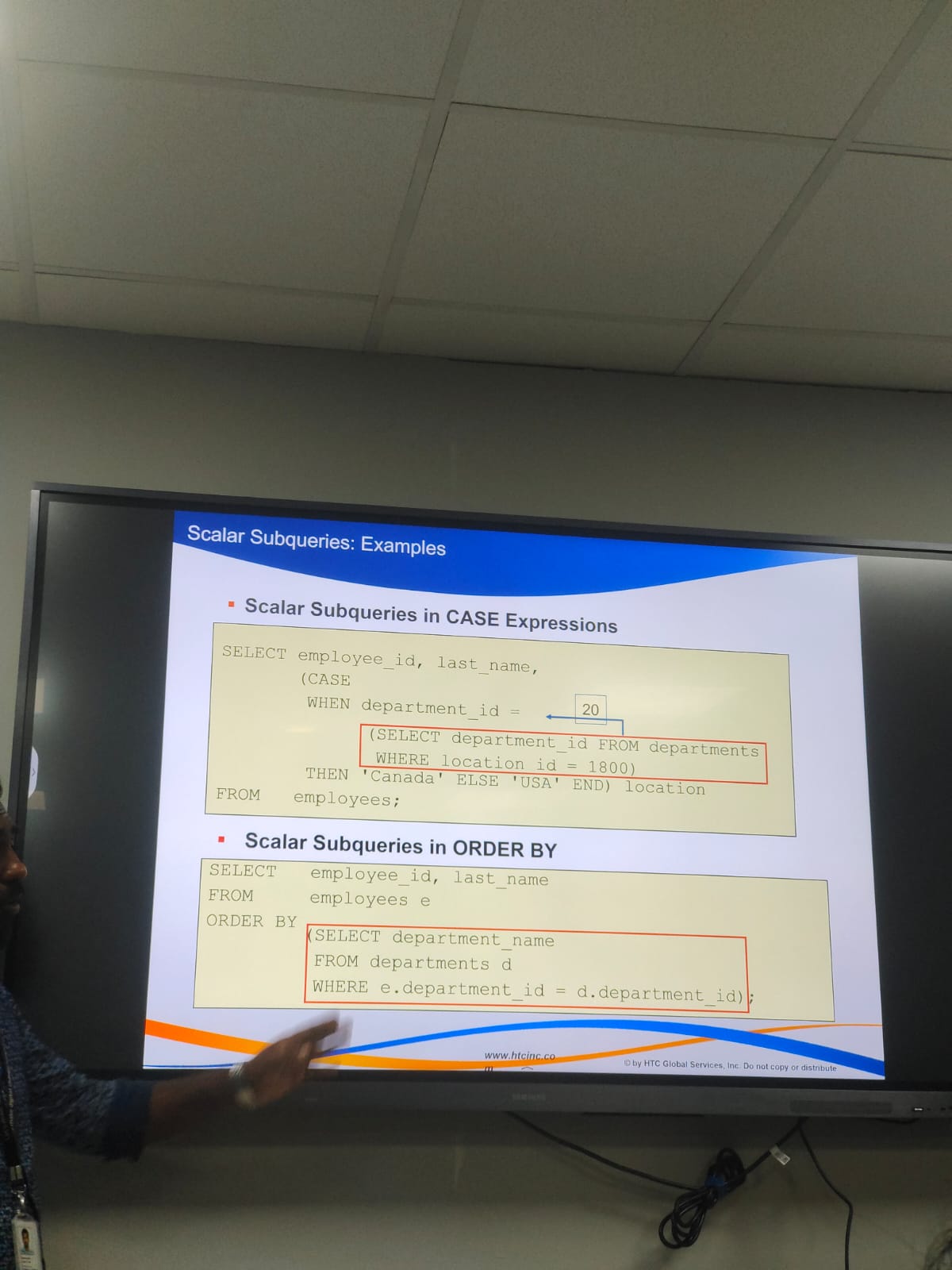
SUBQUERIES



A screen with a white board

AI-generated content may be incorrect.

A screen with text on it

AI-generated content may be incorrect.

A screen with a white board and text on it

AI-generated content may be incorrect.

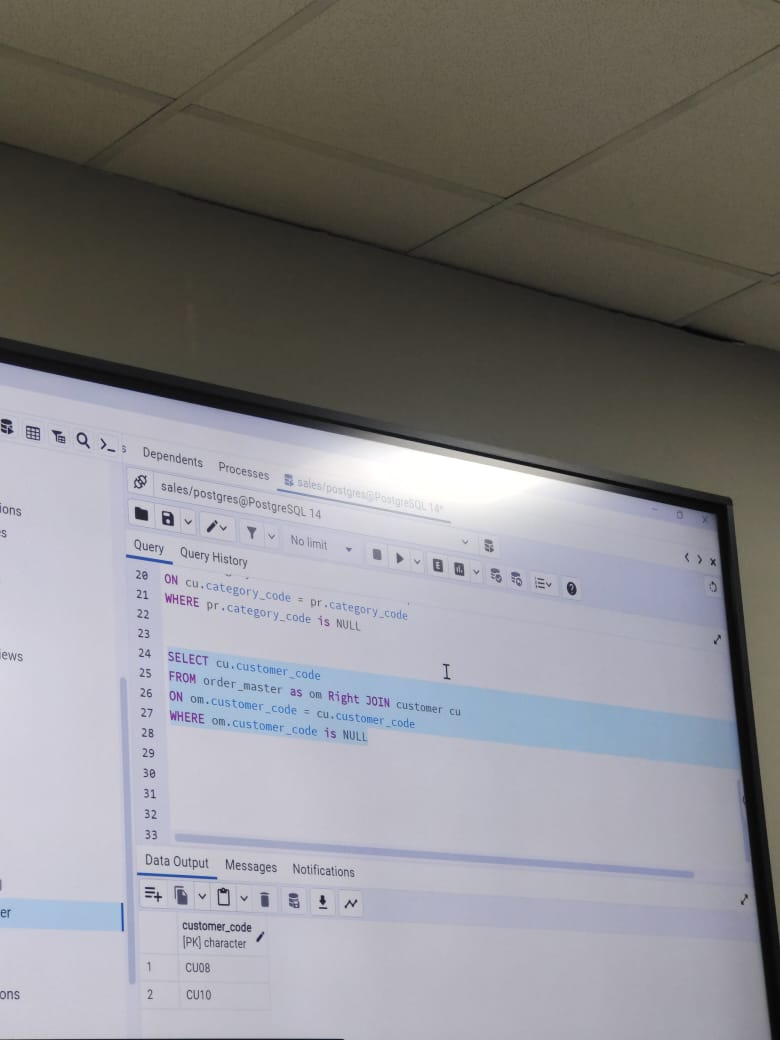
A computer screen with a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

RIGHT JOIN



EXIST operator

A screenshot of a computer

AI-generated content may be incorrect.

NOT EXISTS Operator

A screenshot of a computer

AI-generated content may be incorrect.

UNIONALL, RIGHT join, IS NULL

A computer screen with text on it

AI-generated content may be incorrect.

RANK, JOIN

A computer screen with text on it

AI-generated content may be incorrect.

NOT IN, LEFT JOIN

A screen with text on it

AI-generated content may be incorrect.

RIGHT JOIN

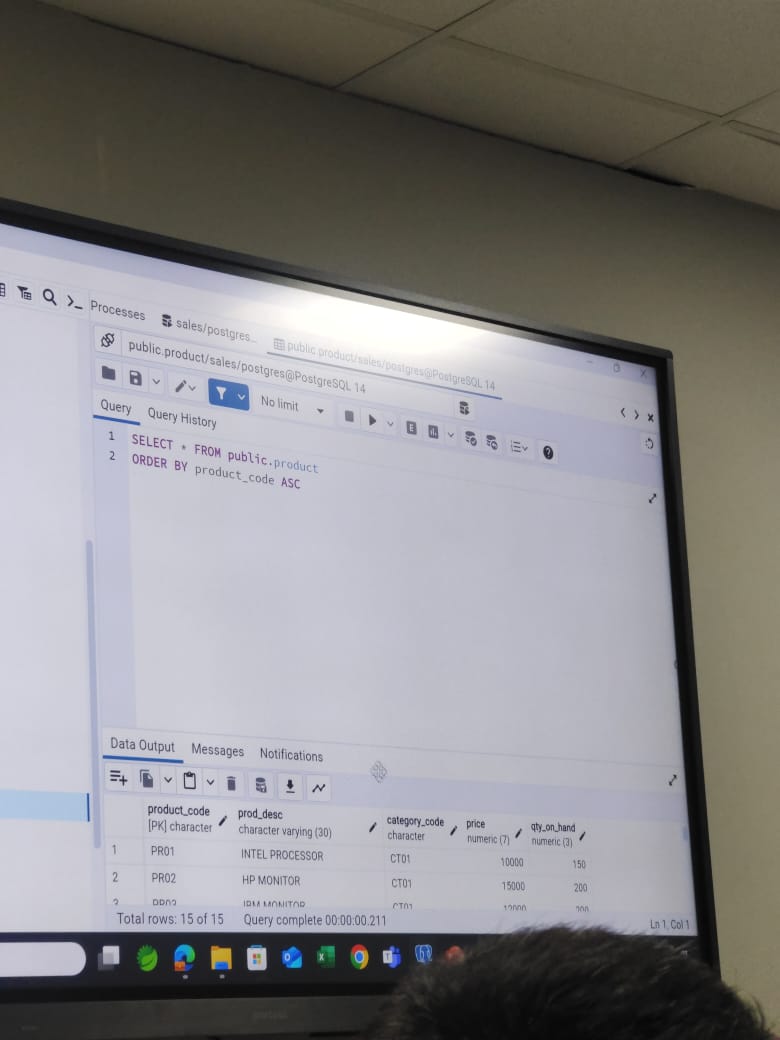
A screen with text on it

AI-generated content may be incorrect.

EXCEPT

A computer screen with a white text on it

AI-generated content may be incorrect.



CASE STATEMENT

A screen with text on it

AI-generated content may be incorrect.

CUBE

A computer screen with a screen on

AI-generated content may be incorrect.

CROSSSTAMP

A computer screen with a white screen

AI-generated content may be incorrect.

Here's a detailed explanation of the topics mentioned in your SQL notes from the image:

**✅ SQL Concepts Explained**

**1. Subquery**

A subquery is a query within another query. It's enclosed in parentheses and used in SELECT, FROM, or WHERE clauses.

SELECT name FROM employees WHERE department\_id IN (SELECT id FROM departments WHERE location = 'NY');

**2. Joins**

Used to combine rows from two or more tables based on related columns:

* **INNER JOIN**: Returns matching rows.
* **LEFT JOIN**: All from the left, matching from the right.
* **RIGHT JOIN**: All from the right, matching from the left.
* **FULL OUTER JOIN**: All rows from both sides.

**3. CASE Statement**

Used for conditional logic in SQL queries.

SELECT name,

CASE

WHEN salary > 50000 THEN 'High'

ELSE 'Low'

END AS Salary\_Category

FROM employees;

**4. PIVOT / UNPIVOT**

* **PIVOT**: Rotates data from rows to columns.
* **UNPIVOT**: Converts columns back to rows.

Example:

-- Pivot to convert 'subject' rows into columns

SELECT \* FROM

(SELECT student, subject, marks FROM scores)

PIVOT (

SUM(marks) FOR subject IN ([Math], [Science], [English])

) AS pivot\_table;

**5. All Possibilities**

Likely refers to using SQL constructs (joins, subqueries, unions) in all possible combinations for comprehensive querying.

**6. ROLLUP**

Used to create subtotals in grouped data.

SELECT department, role, SUM(salary)

FROM employees

GROUP BY ROLLUP(department, role);

**7. CUBE**

Provides subtotals for all combinations of grouping columns.

SELECT department, role, SUM(salary)

FROM employees

GROUP BY CUBE(department, role);

**8. Correlated Subquery**

A subquery that references a column from the outer query.

SELECT name

FROM employees e

WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);

**9. NOT IN**

Filters rows where a value is not present in a list or subquery result.

SELECT name FROM employees WHERE department\_id NOT IN (SELECT id FROM closed\_departments);

**10. IS NULL**

Checks if a value is NULL.

SELECT name FROM employees WHERE manager\_id IS NULL;

**11. Aggregate Functions (MIN, MAX, SUM, etc.)**

* **MIN()**: Minimum value
* **MAX()**: Maximum value
* **SUM()**: Total sum
* **AVG()**: Average
* **COUNT()**: Number of rows

**12. WHERE Clause**

Filters rows before aggregation or selection.

SELECT \* FROM employees WHERE department\_id = 5;

**13. Stored Procedure Function**

A precompiled collection of one or more SQL statements stored in the database.

CREATE PROCEDURE GetEmployeeCount

AS

BEGIN

SELECT COUNT(\*) FROM employees;

END;

**14. RANK Function**

Used for ranking rows based on some condition.

SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rank FROM employees;

Here's an extended explanation of the **theory section** from your SQL notes with more in-depth insights and real-world relevance.

**Theory Section (SQL)**

**1. What is Indexing?**

Indexing is a **database optimization technique** used to **speed up data retrieval operations** on a table. It works like an index in a book, allowing the database engine to find data without scanning every row.

* **Types of Indexes**:
  + **Clustered Index**: Alters the way records are stored (1 per table).
  + **Non-Clustered Index**: A separate structure pointing to the actual data.
* **Advantages**:
  + Faster SELECT queries
  + Enhances performance for WHERE, JOIN, ORDER BY
* **Disadvantages**:
  + Slows down INSERT, UPDATE, DELETE
  + Consumes extra storage
* **Example**:
* CREATE INDEX idx\_emp\_name ON employees(name);

**2. Query Execution**

The process by which the SQL engine interprets and runs a query. It includes:

* **Parsing**: SQL syntax is checked.
* **Optimization**: Query is analyzed to choose the most efficient execution plan.
* **Execution**: Query is run based on the plan.
* **Execution Plan**:
  + Can be viewed using EXPLAIN or EXPLAIN PLAN
  + Shows the order of operations, index usage, table access methods
* **Optimization Tips**:
  + Use indexes
  + Avoid unnecessary columns
  + Filter early (push down predicates)

**3. Use of Functions**

SQL supports **built-in functions** that manipulate data or compute values.

* **Scalar Functions**: Return single value per row
  + UPPER(), LOWER(), LEN(), ROUND()
* **Aggregate Functions**: Work on groups of rows
  + SUM(), COUNT(), MAX(), AVG()
* **Date Functions**:
  + GETDATE(), DATEADD(), DATEDIFF()
* **Conversion Functions**:
  + CAST(), CONVERT()
* **Example**:
* SELECT UPPER(name), ROUND(salary, 2) FROM employees;

**4. Stored Procedures**

A stored procedure is a **precompiled group of one or more SQL statements** that you can call to perform a task.

* **Benefits**:
  + Code reusability
  + Better performance (compiled once)
  + Improved security
* **Syntax**:
* CREATE PROCEDURE GetDeptSalary
* @DeptId INT
* AS
* BEGIN
* SELECT SUM(salary) FROM employees WHERE department\_id = @DeptId;
* END;
* **Calling a Procedure**:
* EXEC GetDeptSalary @DeptId = 101;

**5. Query Tuning Techniques**

Refers to techniques used to optimize SQL queries for better performance.

* **Tips**:
  + Use proper indexes
  + Avoid SELECT \*
  + Use EXISTS instead of IN for subqueries
  + Avoid unnecessary joins
  + Filter using indexed columns
  + Use LIMIT to avoid fetching excessive rows
* **Tools**:
  + SQL Server Profiler
  + MySQL EXPLAIN
  + Oracle AUTOTRACE

**6. Generate Report**

Generating reports involves creating structured result sets, often aggregated or filtered, to analyze business performance.

* **Common Techniques**:
  + **GROUP BY** for aggregation
  + **ROLLUP/CUBE** for subtotals
  + **PIVOT** for cross-tabular reports
  + **CASE WHEN** for conditional categorization
* **Example**:
* SELECT department,
* COUNT(\*) AS total\_employees,
* SUM(salary) AS total\_salary
* FROM employees
* GROUP BY department
* ORDER BY total\_salary DESC;