**Queries Equivalence Type**

1. **Swapping the order of nestedness level (change-nestedness)**

**“Swapping\_Nestedness”**

Example:

**Query:** SELECT customer\_id

FROM Orders

WHERE customer\_id IN (

SELECT customer\_id

FROM Customers

WHERE country = 'USA'

);

**Equivalent Query:** SELECT customer\_id

FROM Customers

WHERE country = 'USA'

AND customer\_id IN (

SELECT customer\_id

FROM Orders

);

1. **Change the join order (change-order)**

**"Join\_Order"**

Example query:

**Query:** SELECT first\_name, last\_name FROM employees, department WHERE department\_id = 10;

**Equivalent Query:** SELECT first\_name, last\_name FROM department, employees WHERE department\_id = 10;

1. **Converting JOIN(Explicit join to implicit join or vice versa)**

**“Explicit\_Implicit\_Joins”**

**Query:**  SELECT first\_name, last\_name FROM employees, department WHERE department\_id = 10;

**Equivalent Query:** SELECT e.first\_name, e.last\_name FROM employees e JOIN departments d ON e.department\_id = d.department\_id WHERE d.department\_id = 10;

1. **Equivalent Query Using a Subquery (changing from join to nestedness)**

**“Join\_Nested”**

**Query:**  SELECT e.EmployeeName, e.EmployeeID

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID

WHERE d.DepartmentName = 'Sales';

**Equivalent Query:** SELECT EmployeeName, EmployeeID

FROM Employees

WHERE DepartmentID IN (

SELECT DepartmentID

FROM Departments

WHERE DepartmentName = 'Sales'

);

1. **(Using Subqueries)**

**"Condition\_Nested"**

**Query:**  SELECT first\_name, last\_name FROM employees, department WHERE department\_id = 10;

**Equivalent Query:** SELECT first\_name, last\_name FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_id = 10);

**7. Equivalent Query Using a join (changing from nestedness to join)**

**“Nested\_Join”**

**Query:**  SELECT first\_name, last\_name FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_id = 10);

**Equivalent Query:** SELECT first\_name, last\_name FROM employees, department WHERE department\_id = 10;

**8. Using Common Table Expressions (CTEs)**

**“CTEs”**

**Query:** SELECT text

FROM DBObjects

WHERE name = 'PhotoTag'

**Equivalent Query:** WITH FilteredObjects AS (

SELECT text

FROM DBObjects

WHERE name = 'PhotoTag'

)

SELECT text

FROM FilteredObjects

**9. Using UNION or INTERSECT ALL (convert and/or to union and intersect)**

**"Condition\_IntersectUnion"**

**Query:** SELECT employee\_id, name, department, salary

FROM employees

WHERE department = 'Sales' OR salary > 50000;

**Equivalent Query:** SELECT employee\_id, name, department, salary

FROM employees

WHERE department = 'Sales'

UNION

SELECT employee\_id, name, department, salary

FROM employees

WHERE salary > 50000;

The UNION operator selects only distinct values by default. If you want to include duplicates, you would use UNION ALL.

**10. Using EXISTS**

**“Condition\_EXISTs”**

**Query:** SELECT DISTINCT d.department\_name

FROM departments d

JOIN employees e ON d.department\_id = e.department\_id;

**Equivalent Query:** SELECT department\_name

FROM departments d

WHERE EXISTS (

SELECT 1

FROM employees e

WHERE e.department\_id = d.department\_id

);

**11. Using CASE Statements**

**"Case\_Statement"**

**Query:** SELECT p.product\_id,

p.price,

p.category,

p.price \* (1 - d.discount\_rate) AS sale\_price

FROM products p

LEFT JOIN discounts d

ON p.category = d.category;

**Equivalent Query:** SELECT product\_id,

price,

category,

CASE

WHEN category = 'Electronics' THEN price \* 0.90 -- 10% discount

WHEN category = 'Clothing' THEN price \* 0.85 -- 15% discount

WHEN category = 'Books' THEN price \* 0.80 -- 20% discount

ELSE price -- No discount

END AS sale\_price

FROM products;

**12. Simplification and Direct Application**

**“Simplification”**

**Query:**  SELECT \* max(salary), min(salary), avg(salary), count(\*)

FROM (

SELECT Name, [Job Title] as job\_title, [2010 Gross Earnings] as salary

FROM [1314howe].[uw\_salaries\_2011.txt]

) x

WHERE job\_title LIKE '%RESEAR%SR'

AND job\_title NOT LIKE '%APL%'

AND salary > 100000

ORDER BY salary desc;

**Equivalent Query:** SELECT MAX(salary) AS max\_salary,

MIN(salary) AS min\_salary,

AVG(salary) AS avg\_salary,

COUNT(\*) AS total\_count

FROM [1314howe].[uw\_salaries\_2011.txt]

WHERE [Job Title] LIKE '%RESEAR%SR'

AND [Job Title] NOT LIKE '%APL%'

AND [2010 Gross Earnings] > 100000;

**13. Reordering the conditions**

**Query:** SELECT \* FROM Orders

WHERE CustomerID = 1234 AND OrderDate > '2020-01-01' AND Status = 'Shipped';

**Equivalent Query:** SELECT \* FROM Orders

WHERE Status = 'Shipped' AND OrderDate > '2020-01-01' AND CustomerID = 1234;

**Creating Not Equivalent Queries**

1. **Use Different Aggregate Functions**

**"Aggregate\_Function"**

**Query:** SELECT department, AVG(salary) FROM employees GROUP BY department;

**Non-equivalent:** SELECT department, SUM(salary) FROM employees GROUP BY department;

1. **Change Join Condition**

**“Change\_Join\_Condition”**

**Query:** SELECT \* FROM orders INNER JOIN customers ON orders.customer\_id = customers.id;

**Non-equivalent:** SELECT \* FROM orders LEFT JOIN customers ON orders.customer\_id = customers.id;

1. **Modify the Grouping Criteria**

**"Group\_by\_Criteria"**

**Query:**  SELECT city, COUNT(\*) FROM employees GROUP BY city;

**Non-equivalent:** SELECT state, city, COUNT(\*) FROM employees GROUP BY state, city;

1. **Introduce Subtle Changes in**

* **Logical conditions**

**"Logical\_Conditions"**

**Query:**  SELECT \* FROM products WHERE price < 20 AND quantity > 100;

**Non-equivalent:** SELECT \* FROM products WHERE price < 20 OR quantity > 100;

* **Changing values**

**"Value\_Change"**

**Query:**  SELECT \* FROM employees WHERE salary > 50000;

**Non-equivalent:** SELECT \* FROM employees WHERE salary > 500000;

* **Operators**

**“Operators”**

**Query:** SELECT \* FROM employees WHERE salary > 50000;

**Non-equivalent:** SELECT \* FROM employees WHERE salary >= 50000;

1. **Change Sorting and Limiting**

**“Change\_Sorting\_Limiting”**

**Query:**  SELECT \* FROM employees ORDER BY hire\_date DESC LIMIT 10;

**Non-equivalent:** SELECT \* FROM employees ORDER BY hire\_date ASC LIMIT 10;

1. **Remove a join/ a condition**
2. **Modify the Order By Criteria**

**“Order\_by\_Column”**

**Query:** SELECT city, COUNT(\*) FROM employees Order BY city;

**Non-equivalent:** SELECT state, city, COUNT(\*) FROM employees Order BY state, city;

1. **Introduce Ambiguous Column References**

**"Adding\_Removing\_Select\_Column"**

**Query:** SELECT EmployeeName, Department, Salary, HireDate, Position, Email

FROM Employees;

**Non Equivalent:** SELECT EmployeeName, Department, Salary, HireDate, Position

FROM Employees;

1. **Change Table Name**

**”Change\_Table”**

**Query:** SELECT EmployeeID, EmployeeName, Department, Salary

FROM Employees;

**Non Equivalent:** SELECT EmployeeID, EmployeeName, Department, Salary

FROM Employee;

**Syntax Error Type:**

We used 6 advanced syntax error types.

1. **Aggr-attribute:**

**Query:** SELECT DepartmentID, ManagerID, AVG(Salary) AS AverageSalary, MAX(HireDate) AS LatestHireDate

FROM Employees

WHERE HireDate > '2000-01-01'

GROUP BY DepartmentID, ManagerID;

**Modified Query:** SELECT DepartmentID, ManagerID, AVG(Salary), MAX(HireDate)

FROM Employees

WHERE HireDate > '2000-01-01';

Here, AVG(Salary) and MAX(HireDate) are aggregate functions, but DepartmentID and ManagerID are not part of any aggregation and there is no GROUP BY clause. This is incorrect because the SQL engine does not know how to group the non-aggregated columns relative to the aggregated ones.

1. **Aggr-having:**

Query: SELECT EmployeeID, COUNT(\*) AS NumProjects

FROM Projects

JOIN ProjectAssignments ON Projects.ProjectID = ProjectAssignments.ProjectID

WHERE Projects.Status = 'Active'

GROUP BY EmployeeID;

**Modified Query:** SELECT EmployeeID, COUNT(\*) AS NumProjects

FROM Projects

JOIN ProjectAssignments ON Projects.ProjectID = ProjectAssignments.ProjectID

GROUP BY EmployeeID

HAVING Projects.Status = 'Active';

Here, The HAVING clause is improperly used to filter on the Projects.Status column, which is neither an aggregated column nor included in the GROUP BY clause. HAVING should be used to filter aggregated results, not to perform row-level filtering that belongs in a WHERE clause.

1. **Type-Mismatch-nested:**

**Query:** SELECT s.fiberid, s.plate, s.mjd, s.run2d, s.class FROM PhotoObj AS p JOIN SpecObj AS s ON s.bestobjid = p.objid**;**

**Modified Query:** SELECT s.fiberid, s.plate, s.mjd, s.run2d, s.class

FROM PhotoObj AS p

JOIN SpecObj AS s ON s.bestobjid = (SELECT objid FROM PhotoObj WHERE ra > 180)

Here, the subquery will return more than one objid. SQL JOIN operations require that a subquery within the JOIN condition must return exactly one row (or a single value) to equate it to another field correctly. When multiple rows are returned, SQL will throw an error, typically something like "Subquery returns more than 1 row."

1. **Type-Mismatch-Condition:**

**Query:** SELECT EmployeeID, Name, Salary \* Rate -- Assuming 'Rate' is a numeric column or a numeric placeholder

FROM Employee;

**Modified Query:** SELECT EmployeeID, Name, Salary \* 'Rate'

FROM Employee;

Here, the query attempts to multiply Salary (presumably a numeric data type like integer or decimal) by a string literal 'Rate'. This is a type mismatch because SQL does not support direct mathematical operations between numbers and non-numeric strings.

1. **Alias-Undefined:**

**Query:** SELECT employeeName, DepartmentID

FROM Employees e

JOIN Departments d ON e.DeptID = d.DepartmentID;

**Modified Query:** SELECT employeeName, DepartmentID

FROM Employees e

JOIN Departments d ON e.DeptID = dept.DepartmentID;

In this query, dept is used as an alias for the Departments table in the JOIN condition, but it was never defined. The correct alias d was defined but not used consistently.

1. **Alias-Ambiguous:**

**Query:** SELECT e.EmployeeID as Employee\_EmployeeID, e.EmployeeName, m.EmployeeID as Manager\_EmployeeID, m.ManagerName

FROM Employees e

JOIN Managers m ON e.ManagerID = m.ManagerID;

**Modified Query:** SELECT EmployeeID, e.EmployeeName, ManagerName

FROM Employees e

JOIN Managers m ON e.ManagerID = m.ManagerID;

Here The column EmployeeID in the SELECT clause does not specify whether it comes from the **Employees** (e) or **Managers** (m) table. If both tables have an EmployeeID column, this would lead to an "Alias Ambiguous" error because the SQL engine cannot determine which table's EmployeeID to use based solely on the information provided.