Case Study 1: Employee Database - Nested Queries

Scenario:
A company stores employee information in two tables: Employees (EmployeeID, Name, DepartmentID, Salary) and Departments (DepartmentID, DepartmentName). The company wants to find employees earning more than the average salary of their respective departments.

Questions:

1. Which SQL technique is best suited to solve this problem?
   a) Correlated subquery
   b) JOIN operation
   c) UNION operation
   d) Index scan
   Answer: a
   Explanation: Correlated subqueries allow comparing each employee's salary to the average salary of their department.

2. What is the correct SQL snippet to calculate the average salary per department inside a nested query?

   a) SELECT AVG(Salary) FROM Employees

   b) SELECT AVG(Salary) FROM Employees WHERE DepartmentID = E.DepartmentID
   c) SELECT Salary FROM Employees WHERE DepartmentID = E.DepartmentID
   d) SELECT AVG(Salary) FROM Departments
   Answer: b
   Explanation: The subquery must correlate with the outer query on DepartmentID.

3. If this query is executed without indexes on DepartmentID, what is the likely impact?
   a) Slow performance due to full table scans
   b) Faster performance due to no overhead
   c) No impact on performance
   d) Query will fail
   Answer: a
   Explanation: Without indexes, correlated subqueries must scan tables repeatedly, slowing down execution.

4. To optimize this query, what is a recommended approach?
   a) Avoid subqueries and write multiple queries
   b) Remove the WHERE clause

   c) Create indexes on DepartmentID and Salary columns

   d) Use UNION operation instead
   Answer: c
   Explanation: Indexes on filtering and joining columns improve query efficiency.

Case Study 2: Product Reviews - Set Operations

Scenario:
An e-commerce platform stores customer reviews in two tables: VerifiedReviews and GuestReviews. They want to create a list of all unique products reviewed by either verified or guest users, excluding products reviewed by both.

Questions:

1. Which set operation returns products reviewed only by either verified or guest users but not both?
   a) MINUS operation applied on UNION and INTERSECT results
   b) UNION operation
   c) INTERSECT operation
   d) JOIN operation
   Answer: a
   Explanation: The symmetric difference can be achieved by (A UNION B) MINUS (A INTERSECT B).

2. What does the INTERSECT operation return in this scenario?

   a) Products reviewed only by verified users
   b) Products reviewed only by guest users

   c) Products reviewed by both verified and guest users
   d) All products reviewed
   Answer: c
   Explanation: INTERSECT returns common rows between two queries.

3. If the tables have large numbers of reviews, what is a possible performance concern using MINUS?
   a) Requires scanning large datasets twice, potentially slow
   b) Uses indexes automatically, so no concern
   c) MINUS does not support large datasets
   d) MINUS is faster than UNION
   Answer: a
   Explanation: Set operations may involve scanning full tables multiple times without proper indexes.

4. How can query optimization be improved for these set operations?

   b) Avoid set operations and use nested subqueries
   c) Use DISTINCT instead of UNION

   c) Create indexes on product IDs in both tables
   d) Drop indexes before running queries
   Answer: c
   Explanation: Indexes on join or filtering columns help optimize set operations.

Case Study 3: Sales Data - Query Execution Plan

Scenario:
A sales manager runs a query to get total sales per region. The query uses multiple joins and filters on a large dataset. The execution plan shows full table scans and nested loop joins.

Questions:

1. What does a full table scan indicate in the execution plan?
   a) Query uses an index seek
   b) Query is optimized
   c) Query will return no rows
   d) No indexes are used, the entire table is read

   Answer: d
   Explanation: Full table scans read all rows, usually slower for large tables.

2. When is a nested loop join preferred by the query optimizer?
   a) When one table is small and the other is indexed
   b) Always for large tables
   c) Never used by modern optimizers
   d) When there are no joins
   Answer: a
   Explanation: Nested loop joins are efficient when joining a small number of rows to indexed tables.

3. How can the query performance be improved based on the execution plan?
   a) Create indexes on join and filter columns
   b) Remove WHERE clauses
   c) Use CROSS JOINs instead
   d) Increase result set size
   Answer: a
   Explanation: Indexes reduce full table scans and improve join performance.

4. What is the significance of updating statistics for the query optimizer?
   a) Slows down query execution
   b) Helps optimizer choose better plans
   c) Deletes table data
   d) Has no effect
   Answer: b
   Explanation: Accurate statistics help the optimizer estimate costs and select efficient plans.

Case Study 4: Student Records - Correlated Subqueries

Scenario:
A university stores student marks in the Marks table with columns (StudentID, SubjectID, Marks). They want to find students scoring above the average mark in each subject.

Questions:

1. Which SQL construct can help find students scoring above average marks per subject?
   a) Correlated subquery comparing each student's mark with average marks per subject
   b) UNION operation
   c) MINUS operation
   d) JOIN only
   Answer: a
   Explanation: Correlated subqueries allow per-row comparison with aggregated values.

2. What is the main disadvantage of correlated subqueries in large datasets?
   a) Can cause slow query due to repeated execution per row
   b) Always faster than joins
   c) Cannot be used with aggregate functions
   d) No disadvantages
   Answer: a
   Explanation: Correlated subqueries execute the subquery for each row of outer query, leading to performance issues.

3. How can this query be optimized?
   a) Rewrite using JOINs with GROUP BY
   b) Use CROSS JOIN
   c) Remove WHERE clause
   d) Use MINUS operation
   Answer: a
   Explanation: Joins combined with GROUP BY can improve efficiency over correlated subqueries.

4. What indexing strategy helps optimize this query?
   a) Index on StudentID and SubjectID columns
   b) Index on Marks only
   c) No indexing needed
   d) Index on unrelated columns
   Answer: a
   Explanation: Indexes on join and filtering keys improve data retrieval.

Case Study 5: Inventory Management - Set Operations and Indexes

Scenario:
An inventory system tracks available products in Warehouse1 and Warehouse2. Management wants a list of products present in both warehouses and also products unique to each warehouse.

Questions:

1. Which set operation gives products present in both warehouses?

   a) UNION
   b) MINUS
   c) JOIN

d) INTERSECT
Answer: d
Explanation: INTERSECT returns common rows.

2. Which set operation returns products unique to Warehouse1?
a) UNION
b) INTERSECT
c) JOIN
d) MINUS (Warehouse1 MINUS Warehouse2)

Answer: d
Explanation: MINUS returns rows in first query but not in second.

3. To speed up these set operations on large tables, what is recommended?
a) Create indexes on product IDs in both tables
b) Avoid indexes
c) Use correlated subqueries
d) Drop primary keys
Answer: a
Explanation: Indexes improve search and comparison speed.

4. What is the role of execution plans in optimizing these queries?
a) Automatically rewrite queries
b) Delete duplicate rows
c) Show how SQL engine executes queries and identifies bottlenecks
d) Run queries in parallel always
Answer: c
Explanation: Execution plans help developers understand query performance.
