**CTE;**

**Common Table Expression**

A Common Table Expression (CTE) is a temporary result set in SQL that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs improve readability and maintainability of complex queries by breaking them into simpler parts.

WITH DeptAvgSalaries AS (

SELECT deptno, AVG(salary) AS avg\_salary

FROM emp

GROUP BY deptno

)

SELECT e.empno, e.ename, e.salary, d.avg\_salary

FROM emp e

JOIN DeptAvgSalaries d ON e.deptno = d.deptno

WHERE e.salary > d.avg\_salary;

Performance optimization in Oracle involves several strategies to enhance the execution speed of SQL queries and overall system performance. Here are some key techniques:

**1. Indexing**

* **Create Indexes**: Proper indexing can significantly improve query performance by reducing the amount of data that needs to be scanned.
* **Index Maintenance**: Regularly monitor and maintain indexes to ensure they are effective and not causing overhead.

**2. Query Optimization**

* **Analyze Execution Plans**: Use tools like Oracle's SQL Trace and Explain Plan to analyze and optimize the execution plans of your queries.
* **Rewrite Queries**: Simplify and rewrite queries to make them more efficient. Avoid unnecessary subqueries, joins, and functions.

**3. Database Structure Optimization**

* **Partitioning**: Partition large tables to improve query performance and manageability.
* **Normalization and Denormalization**: Balance normalization to reduce redundancy and denormalization to optimize read performance.

**4. PL/SQL Optimization**

* **Subprogram Inlining**: Use the INLINE pragma to allow the compiler to replace subprogram invocations with the subprogram code itself.
* **Bulk Operations**: Use bulk SQL operations (e.g., BULK COLLECT, FORALL) to process large data sets more efficiently.

**5. System Resource Management**

* **Memory and CPU**: Ensure that the database has adequate memory and CPU resources to handle the workload.
* **I/O Optimization**: Optimize disk I/O by using faster storage solutions and optimizing file layouts.

**6. Monitoring and Tuning**

* **Automatic Workload Repository (AWR)**: Use AWR reports to identify performance bottlenecks and tune the system accordingly.
* **Real-Time SQL Monitoring**: Monitor SQL performance in real-time to detect and address issues promptly.

**7. Best Practices**

* **Regular Maintenance**: Perform regular maintenance tasks such as statistics gathering, index rebuilding, and data purging.
* **Code Reviews**: Conduct code reviews to identify inefficient code and apply best practices.

**Employee & Manager, need to identify the highly paid employee under the manager. How we can fetch data.**

WITH RankedEmployees AS (

SELECT e.employee\_id,

e.employee\_name,

e.salary,

e.manager\_id,

ROW\_NUMBER() OVER (PARTITION BY e.manager\_id ORDER BY e.salary DESC) AS rn

FROM employees e

)

SELECT employee\_id, employee\_name, salary, manager\_id

FROM RankedEmployees

WHERE rn = 1;

**Hash Join**

* List 1 (Accounting Department):
  1. Alice (Accounting)
  2. Bob (Accounting)
* List 2 (All Employees):
  1. Alice (Accounting)
  2. Bob (Accounting)
  3. Charlie (Sales)
  4. Dave (HR)

Notebook (Hash Table) Content:

* Page for "Accounting": Alice, Bob

Find Matches:

* Alice (Accounting): Match found in the notebook.
* Bob (Accounting): Match found in the notebook.
* Charlie (Sales): No match in the notebook.
* Dave (HR): No match in the notebook.

SQL Example:

In SQL, this process is automated using the HASH JOIN. Here's the query:

sql

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

Hash Join Execution Plan:

1. Hash Table (Notebook):
   * Oracle creates a hash table for the departments table.
2. Probe Phase:
   * Oracle scans the employees table and uses the hash table to find matches based on department\_id.

Benefits:

* Efficient: Quickly finds matches by using a hash table.
* Scalable: Works well with large datasets.

**Sort Merge Join**

A Sort-Merge Join is a method used by Oracle to join two sets of data (usually tables) based on a join condition. It is particularly effective for large datasets that are already sorted or can be sorted efficiently. Here's how a Sort-Merge Join works in a nutshell:

Steps in a Sort-Merge Join:

1. Sort Phase:
   * Both input tables (or datasets) are sorted on the join key(s). If the data is already sorted, this step can be skipped.
2. Merge Phase:
   * After sorting, the two sorted datasets are merged together. The join key values from each dataset are compared, and matching rows are combined to form the result set.

When to Use Sort-Merge Join:

* Large Datasets: Sort-Merge Joins are efficient for large tables, especially when other join methods (like nested loops joins) are not suitable.
* Sorted Data: If the data is already sorted on the join key, the sorting phase can be skipped, making the join operation faster.
* Equality Joins: Sort-Merge Joins work best for equi-joins (joins based on equality conditions).

Example Scenario:

Suppose you have two tables: employees and departments, and you want to join them based on the department\_id column.

Query:

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

Execution Plan:

You can use the EXPLAIN PLAN statement to see if a Sort-Merge Join is used:

EXPLAIN PLAN FOR

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

Expected Execution Plan Output:

plaintext

Plan hash value: 1234567890

--------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | XX | XXX | XX (X)| 00:00:XX |

| 1 | SORT JOIN | | XX | XXX | XX (X)| 00:00:XX |

| 2 | TABLE ACCESS FULL | EMPLOYEES | XX | XXX | XX (X)| 00:00:XX |

| 3 | TABLE ACCESS FULL | DEPARTMENTS | XX | XXX | XX (X)| 00:00:XX |

--------------------------------------------------------------------------------

Explanation:

* SORT JOIN: Indicates that a Sort-Merge Join is being used. Both tables are sorted on the join key (department\_id).
* TABLE ACCESS FULL: Full table scans are performed on both employees and departments.

Benefits:

* Efficient for Large Tables: Sort-Merge Joins can handle large datasets efficiently.
* Sorted Data: If input data is pre-sorted, the sorting phase is skipped, saving time.

Summary:

Sort-Merge Join is a powerful join method used in Oracle to merge sorted data efficiently. It's particularly useful for large datasets and equality joins. Understanding when and how to use it can significantly enhance query performance.

If you have more questions or need further clarification, feel free to ask!

example

Let's walk through a concrete example of a Sort-Merge Join. Imagine we have two tables: employees and departments, and we want to join them based on the department\_id column.

Tables and Data:

employees Table:

| employee\_id | employee\_name | department\_id |
| --- | --- | --- |
| 1 | Alice | 10 |
| 2 | Bob | 20 |
| 3 | Charlie | 10 |
| 4 | David | 30 |

departments Table:

| department\_id | department\_name |
| --- | --- |
| 10 | Accounting |
| 20 | Sales |
| 30 | HR |
| 40 | Marketing |

SQL Query:

We want to join these tables to get a list of employees along with their department names.

sql

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

Steps for Sort-Merge Join:

1. Sort Phase:
   * Both tables are sorted on the join key department\_id.

Sorted employees Table:

| employee\_id | employee\_name | department\_id |
| --- | --- | --- |
| 1 | Alice | 10 |
| 3 | Charlie | 10 |
| 2 | Bob | 20 |
| 4 | David | 30 |

Sorted departments Table:

| department\_id | department\_name |
| --- | --- |
| 10 | Accounting |
| 20 | Sales |
| 30 | HR |
| 40 | Marketing |

1. Merge Phase:

The sorted tables are merged together based on the department\_id.

Merging Process:

* Compare the first row of both tables: (10, Accounting) and (1, Alice, 10). Since they match on department\_id, they are joined.
* Move to the next row of the employees table: (3, Charlie, 10) and match it with (10, Accounting) in the departments table.
* Continue this process for all rows.

Result:

| employee\_id | employee\_name | department\_name |
| --- | --- | --- |
| 1 | Alice | Accounting |
| 3 | Charlie | Accounting |
| 2 | Bob | Sales |
| 4 | David | HR |

Explain Plan:

The explain plan for this query would look something like this:

plaintext

Plan hash value: 1234567890

--------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 4 | 120 | 5 (20)| 00:00:01 |

| 1 | SORT JOIN | | 4 | 120 | 5 (20)| 00:00:01 |

| 2 | TABLE ACCESS FULL | EMPLOYEES | 4 | 80 | 2 (10)| 00:00:01 |

| 3 | TABLE ACCESS FULL | DEPARTMENTS | 4 | 40 | 2 (10)| 00:00:01 |

--------------------------------------------------------------------------------

Explanation:

* SORT JOIN: Indicates a Sort-Merge Join is used.
* TABLE ACCESS FULL: Both tables are fully scanned.
* ROWS/Bytes/Cost/Time: Estimates for the rows processed, memory used, cost of the operation, and time taken.

**. Hash Join**

* **Description**: Uses a hash table to join two tables based on the equality of the join keys.
* **When to Use**: Effective for joining large tables, especially when the join columns are not indexed.
* **How it Works**:
  + **Build Phase**: Creates a hash table for the smaller table.
  + **Probe Phase**: Scans the larger table and probes the hash table to find matches.

**Example**:

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

**2. Sort-Merge Join**

* **Description**: Joins two tables by first sorting them on the join key and then merging the sorted datasets.
* **When to Use**: Efficient for large datasets that are already sorted or can be sorted quickly.
* **How it Works**:
  + **Sort Phase**: Sorts both tables on the join key.
  + **Merge Phase**: Merges the sorted datasets to find matches.
* **Example**:

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

**3. Nested Loops Join**

* **Description**: Iterates through each row of one table and for each row, finds matching rows in the other table.
* **When to Use**: Suitable for small tables or when the join columns are indexed.
* **How it Works**:
  + For each row in the outer table, the database probes the inner table to find matching rows.
* **Example**:

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

**Comparison of Join Methods:**

| **Join Type** | **Use Case** | **Advantages** | **Disadvantages** |
| --- | --- | --- | --- |
| **Hash Join** | Large tables, no index | Efficient for large tables, no need for sorting | Requires memory for hash table |
| **Sort-Merge** | Large tables, sorted data | Efficient for pre-sorted data | Requires sorting phase, can be slow for unsorted data |
| **Nested Loops** | Small tables, indexed join columns | Simple, efficient for small datasets or indexed columns | Can be slow for large datasets without indexes |

**Explain Plan Analysis:**

To understand which join method Oracle is using for your query, you can use the EXPLAIN PLAN statement.

**Example**:

EXPLAIN PLAN FOR

SELECT e.employee\_id, e.employee\_name, d.department\_name

FROM employees e

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

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);