**SQL Tuning or SQL Optimization**

Sql Statements are used to retrieve data from the database. We can get same results by writing different sql queries. But use of the best query is important when performance is considered. So you need to sql query tuning based on the requirement. Here is the list of queries which we use reqularly and how these sql queries can be optimized for better performance.

**SQL Tuning/SQL Optimization Techniques:**

**1)** The sql query becomes faster if you use the actual columns names in SELECT statement instead of than '\*'.

**For Example:** Write the query as

SELECT id, first\_name, last\_name, age, subject FROM student\_details;

Instead of:

SELECT \* FROM student\_details;

**2)**HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.   
**For Example:**Write the query as

SELECT subject, count(subject)   
FROM student\_details   
WHERE subject != 'Science'   
AND subject != 'Maths'   
GROUP BY subject;

Instead of:

SELECT subject, count(subject)   
FROM student\_details   
GROUP BY subject   
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';

**3)**Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.   
**For Example:**Write the query as

SELECT name   
FROM employee   
WHERE (salary, age ) = (SELECT MAX (salary), MAX (age)   
FROM employee\_details)   
AND dept = 'Electronics';

Instead of:

SELECT name   
FROM employee  
WHERE salary = (SELECT MAX(salary) FROM employee\_details)   
AND age = (SELECT MAX(age) FROM employee\_details)   
AND emp\_dept = 'Electronics';

**4)**Use operator EXISTS, IN and table joins appropriately in your query.   
**a)**Usually IN has the slowest performance.   
**b)**IN is efficient when most of the filter criteria is in the sub-query.   
**c)**EXISTS is efficient when most of the filter criteria is in the main query.

**For Example:**Write the query as

Select \* from product p   
where EXISTS (select \* from order\_items o   
where o.product\_id = p.product\_id)

Instead of:

Select \* from product p   
where product\_id IN   
(select product\_id from order\_items

**5)**Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.   
**For Example:**Write the query as

SELECT d.dept\_id, d.dept   
FROM dept d   
WHERE EXISTS ( SELECT 'X' FROM employee e WHERE e.dept = d.dept);

Instead of:

SELECT DISTINCT d.dept\_id, d.dept   
FROM dept d,employee e   
WHERE e.dept = e.dept;

**6)**Try to use UNION ALL in place of UNION.   
**For Example:**Write the query as

SELECT id, first\_name   
FROM student\_details\_class10   
UNION ALL   
SELECT id, first\_name   
FROM sports\_team;

Instead of:

SELECT id, first\_name, subject   
FROM student\_details\_class10   
UNION   
SELECT id, first\_name   
FROM sports\_team;

**7)**Be careful while using conditions in WHERE clause.   
**For Example:**Write the query as

SELECT id, first\_name, age FROM student\_details WHERE age > 10;

Instead of:

SELECT id, first\_name, age FROM student\_details WHERE age != 10;

Write the query as

SELECT id, first\_name, age   
FROM student\_details   
WHERE first\_name LIKE 'Chan%';

Instead of:

SELECT id, first\_name, age   
FROM student\_details   
WHERE SUBSTR(first\_name,1,3) = 'Cha';

Write the query as

SELECT id, first\_name, age   
FROM student\_details   
WHERE first\_name LIKE NVL ( :name, '%');

Instead of:

SELECT id, first\_name, age   
FROM student\_details   
WHERE first\_name = NVL ( :name, first\_name);

Write the query as

SELECT product\_id, product\_name   
FROM product   
WHERE unit\_price BETWEEN MAX(unit\_price) and MIN(unit\_price)

Instead of:

SELECT product\_id, product\_name   
FROM product   
WHERE unit\_price >= MAX(unit\_price)   
and unit\_price <= MIN(unit\_price)

Write the query as

SELECT id, name, salary   
FROM employee   
WHERE dept = 'Electronics'   
AND location = 'Bangalore';

Instead of:

SELECT id, name, salary   
FROM employee   
WHERE dept || location= 'ElectronicsBangalore';

Use non-column expression on one side of the query because it will be processed earlier.

Write the query as

SELECT id, name, salary   
FROM employee   
WHERE salary < 25000;

Instead of:

SELECT id, name, salary   
FROM employee   
WHERE salary + 10000 < 35000;

Write the query as

SELECT id, first\_name, age   
FROM student\_details   
WHERE age > 10;

Instead of:

SELECT id, first\_name, age   
FROM student\_details   
WHERE age NOT = 10;

**8)**Use DECODE to avoid the scanning of same rows or joining the same table repetitively. DECODE can also be made used in place of GROUP BY or ORDER BY clause.   
**For Example:**Write the query as

SELECT id FROM employee   
WHERE name LIKE 'Ramesh%'   
and location = 'Bangalore';

Instead of:

SELECT DECODE(location,'Bangalore',id,NULL) id FROM employee   
WHERE name LIKE 'Ramesh%';

**9)**To store large binary objects, first place them in the file system and add the file path in the database.

**10)**To write queries which provide efficient performance follow the general SQL standard rules.

**a)**Use single case for all SQL verbs  
**b)**Begin all SQL verbs on a new line  
**c)**Separate all words with a single space   
**d)**Right or left aligning verbs within the initial SQL verb

**Why optimize a query?**Many of us are feel the query that we wrote is final and once it works and provides the result then that query is the best.  
  
Actually the Query optimization is not only the technique to make your query to fetch details or to execute many  CRUD operations but also they are the main scope to make the end user continue with your application.  
  
It’s the same situation when we access some websites, and fetching of data pulls most of your time, then you may prefer to use other websites which get you all results instantly.  
  
Optimization is not only to create impact technically, but also in a business point of view has a very large impact.  
  
**How to Optimize a Query?**Hope now you feel that optimizing a query is better and also there are a few things that you have to consider before writing a query or before deploying the same.

1. **Avoid using “\*” in select query**We have the habit of always writing a query using “\*” in select operations  
     
   **Example:** *SELECT \* FROM Students [X]*  
     
   Where the select will fetch all the columns of the table, whenever there is a need for the column then use it, don’t fetch all the columns of the table. Always use column specific names.  
     
   **Example:** *SELECT ID, Name FROM Students [√]*
2. **Avoid using Copy Paste of Code randomly**We have the habit of copying the code from existing code or from online forums whenever required, while copying please using only the code that exactly applies to the logic, reusing the similar code or copying unwanted lines which may not create impact to your query but may take time for execution, so during code re usability always use only code that exactly required to fetch results for you.
3. **Avoid using functions in Where Clause**When we have to execute any query there will be a conditional check using the where clause for filtering of data.  
     
   **Example:** *SELECT Name, DOB FROM Students WHERE DOB < getdate() [X]*  
     
   When we use any of the system defined or user defined functions in the where clause, then each time; i.e., each of the rows in the table will be checked against the function results which in turn execute the function each time. Obviously this will take more time and we all know the parameterless function will always return the same values.  
     
   Better assign the function value to a variable and then use it in a where clause.  
     
   **Example:**
   1. **DECLARE** @MyDate **as** **date**
   2. **SET** @MyDate = GetDate()
   4. **SELECT** **Name**, DOB **FROM** Students **WHERE** DOB < @MyDate [√]
4. **Avoid using Joins between two types of columns**As all of you know the Type Conversion during the data manipulation is another place where a large amount of time will be consumed, when we are trying to join two types of columns then the other column has to be converted, the values of the lower column has to be converted and this will take some time for converting for each of the row values.  
     
   **Example:**
   1. **SELECT** **Name** **FROM** Student, Parents **WHERE**
   3. Student.SchoolFees = Parents.KidsFees [X]
   4. Student.SchoolFees - **Int** Value
   5. Parents.KidsFees - **Float** Value
5. **Avoid using COUNT(\*) from tables for getting row count**We usually use the COUNT(\*) to get the number of rows available in a table.  
     
   **Example:** *SELECT COUNT(\*) FROM Student [X]*  
     
   This count(\*) makes the full scan of the table, whenever the code is executed.  
     
   **Example:**
   1. **SELECT** **rows** **FROM** Student
   3. **WHERE** id = OBJECT\_ID('dbo.Orders') AND indid < 2 [√]

**Note:**This query will not do full scan of table rows, where if you need to check for any value for that condition available then this query may be useful, not always provide the exact count of rows.

1. **Avoid using DISTINCT when Join Tables**When we use the Distinct keyword it looks as if the query will be automatically optimized by the internal SQL engine, that is true when you use the Distinct for filtering operation when we use the distinct during the table join operations which has one to many relations then it’s not advisable.   
     
   **Example:**
   1. **SELECT** **DISTINCT** s.Dept, d.Dept **FROM** Student s, Department d
   3. **Where** s.Dept = d.Dept [X]

When we use the “Exists” query for fetching some operations during table join then it's better comparing it to the Distinct compares.   
  
**Example:**

* 1. **SELECT** d.Dept **FROM** Department d **Where**
  2. Exists ( **SELECT** ‘Y’ **FROM** Student s **WHERE** s.Dept = d.Dept ) [√]

1. **Avoid Using Temp Tables**Using the temporary table is always an easy functionality for accessing the data values, but we should be careful about the right scenario to use the temp tables.When more than one table are joined and some conditional operations are compared and executed it’s not suggested to store the values in temp table and access the large quantum of values.  
     
   It’s always suggested to use the “View”, where the View has more benefits that the table operations are executed once during the query can be fetched during execution and can be used optimised.  
     
   **Note:** The temporary table may occupy the internal memory and that make the process execution slowness.
2. **Avoid Using Triggers**Trigger usages are the expensive process in the SQL, so try to avoid firing and executing the triggers. Don’t use the triggers with any constraints and also avoid using the same triggers for multiple CRUD operations.
3. **Avoid Dead locks during query executions**The deadlock handling is not an easy job for the larger transactional data volumes, when we handle quite large data processing with multiple processes then we should be clear on dead lock occurrence scenarios and try to avoid it.  
     
   When you write any queries, views, functions and store procedures always follow the same order of access of the tables.  
     
   Always try to break the operations in transactions as small blocks for easy understanding in troubleshooting. Larger transaction blocks may create deadlocks or dependency and consume more operational and execution times.
4. **Avoid Using Locks during read process**Lock of the tables may affect the other process to get impact; there is an alternative query to use “WITH (NOLOCK)” which is equivalent to “READ UNCOMMITED” at transaction isolation.  
     
   This is also called a dirty read but this operation of using “With (NoLock)” can prevent the deadlock occurrences by multiple reads.  
     
   **Example:**
   1. **SELECT** **Name**, ID, Address **FROM** Student **WITH**(NOLOCK) [√]

Hope this will help you to make your SQL optimized and helps to reduce the execution time to some extents.