Beginner SQL Tutorial

Learn SQL Programming...

Search

Home

SQL Commands

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

BookMark This Page BOOKMARK

Search