SQL Tuning Tips

By,
Srikar Reddy Gondesi,
Junior SQL Server DBA,
Database Team,
Miracle Software Systems. Inc,
Email: sgondesi@miraclesoft.com

Database Tuning

- Database tuning can be an incredibly difficult task, particularly when working with large-scale data where even the most minor change can have a dramatic (positive or negative) impact on performance.

- When working with large-scale data, even the most minor change can have a dramatic impact on performance.

- 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 tune sql queries based on the requirement.

Tuning By Developers

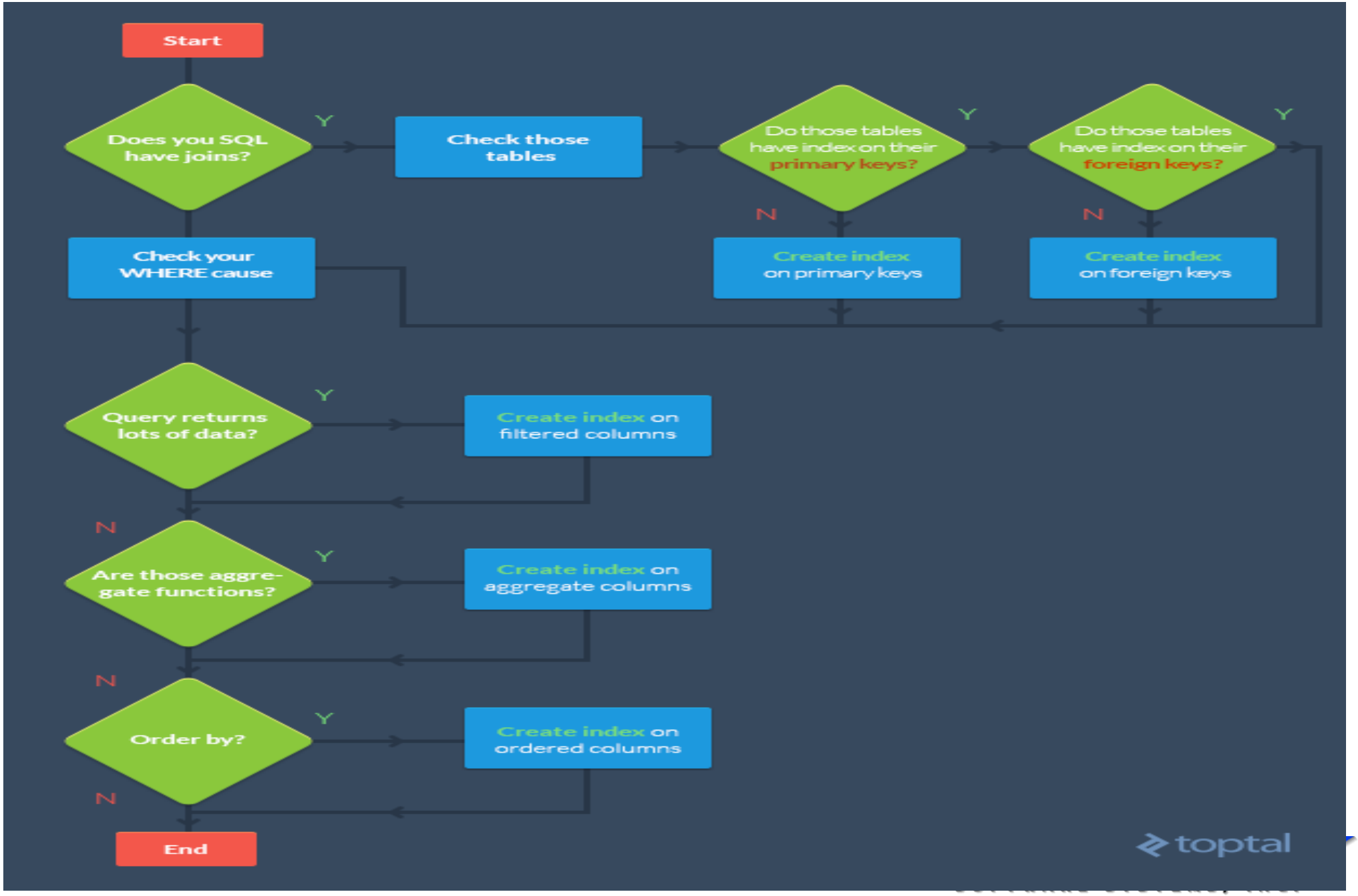
- In mid-sized and large companies, most database tuning will be handled by a Database Administrator (DBA).
- But believe me, there are plenty of developers out there who have to perform DBA-like tasks.
- Further, in many of the companies I've seen that do have DBAs, they often struggle to work well with developers—the positions simply require different modes of problem solving, which can lead to disagreement among coworkers.

Indexes ?

- If you're a complete newcomer to databases, you should know that indexing is an effective way to tune your database that is often neglected during development.

- In basic terms, an index is a data structure that improves the speed of data retrieval operations on a database table by providing rapid random lookups and efficient access of ordered records. This means that once you've created an index, you can select or sort your rows faster than before.

When And Where To Create An Index?



Select Only Required Columns

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

SELECT id, first_name, last_name, age, subject FROM student_details;

- Instead of the below,

SELECT * FROM student_details;

Think Twice To Use A Having Clause

- 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.

```
SELECT subject, count(subject) FROM student_details
WHERE subject != 'Science' AND subject != 'Maths'
GROUP BY subject;
```

- Instead of the below,

```
SELECT subject, count(subject) FROM student_details
GROUP BY subject
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';
```

Keep Sub-Queries To A Less Number

- Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.

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

- Instead of the below,

```
SELECT name FROM employee
WHERE salary = (SELECT MAX(salary) FROM
    employee_details)
AND age = (SELECT MAX(age) FROM employee_details)
AND emp_dept = 'Electronics';
```

Case With IN & EXISTS

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

```
Select * from product p
where EXISTS (select * from order_items o
where o.product_id = p.product_id)
```

- Instead of the below,

```
Select * from product p
where product_id IN (select product_id from order_items
```

Using UNION ALL Is Sugested

- Try to use UNION ALL in place of UNION.

```
SELECT id, first_name FROM student_details_class10
UNION ALL
SELECT id, first_name FROM sports_team;
```

- Instead of the below,

```
SELECT id, first_name, subject FROM
   student_details_class10
UNION
SELECT id, first_name FROM sports_team;
```

Unique Clustered Index

- When you create a new table always create a unique clustered index belong to it, possibly it is a numeric type.
  - Primary Key.
  - Clustered and Non-Clustered Indexes.

JOIN's Vs Sub-Queries

- Use SQL JOIN instead of subqueries. As a programmer, subqueries are something that you can be tempted to use and abuse. Subqueries, as show below, can be very useful:

```
SELECT a.id, (SELECT MAX(created)
FROM posts WHERE author_id = a.id)
AS latest_post FROM authors a
```

- Although subqueries are useful, they often can be replaced by a join, which is definitely faster to execute.

```
SELECT a.id, MAX(p.created) AS latest_post
FROM authors a INNER JOIN posts p
ON (a.id = p.author_id) GROUP BY a.id
```

If Possible Try To Avoid Operators

- The following example use the OR statement to get the result:

SELECT * FROM a, b WHERE a.p = b.q OR a.x = b.y;

- The UNION statement allows you to combine the result sets of 2 or more select queries. The following example will return the same result that the above query gets, but it will be faster

SELECT * FROM a, b WHERE a.p = b.q

UNION

SELECT * FROM a, b WHERE a.x = b.y;

Database statistics

- The most important resource to any SQL optimizer is the statistics collected for different tables within the catalog.
- Statistics is the information about indexes and their distribution with respect to each other.
- Optimizer uses this information to decide the least expensive path that satisfies a query.
- Outdated or missing statistics information will cause the optimizer to take a less optimized path hence increasing the overall response time.

Specify optimizer hints in SELECT

- Although in most cases the query optimizer will pick the appropriate index for a particular table based on statistics, sometimes it is better to specify the index name in your SELECT query. For example, consider the following:

```
SELECT *FROM customer
WITH ( Index(IdxPhone))
WHERE city = 'New York City' AND phone = '212-555-1212'
```

Use EXPLAIN

- Most databases return the execution plan for any SELECT statement that is created by the optimizer. This plan is very useful in fine tuning SQL queries. The following table lists SQL syntax for different databases.

Oracle: **EXPLAIN PLAN FOR >Your query<**

DB2: **EXPLAIN PLAN SET queryno = xxx for >Your query<**

MS SQL Server: **Set SHOWPLAN_ALL ON >Your query<**

Informix: **SET EXPLAIN**

Sybase ASE: **Set SHOWPLAN_ALL ON >Your query<**

Avoid foreign key constraints

- Foreign keys constraints ensure data integrity at the cost of performance.
- Therefore, if performance is your primary goal you can push the data integrity rules to your application layer.

- Every major RDBMS has a set of tables known as system tables. These tables contain meta data information about user databases. Although there are relationships among these tables, there is no foreign key relationship. This is because the client, in this case the database itself, enforces these rules.

Drop indexes before loading data

- Consider dropping the indexes on a table before loading a large batch of data. This makes the insert statement run faster.
- Once the inserts are completed, you can recreate the index again.
- If you are inserting thousands of rows in an online system, use a temporary table to load data.
- Ensure that this temporary table does not have any index. Since moving data from one table to another is much faster than loading from an external source, you can now drop indexes on your primary table, move data from temporary to final table, and finally recreate the indexes.

General Standards

To write queries which provide efficient performance follow the general SQL standard rules:

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

Thank You

MIRACLE MSS
SOFTWARE SYSTEMS, INC.