SQL OPTIMIZATION TECHNIQUES

NIC Webinar- Knowledge Sharing Among Peers PAN INDIA

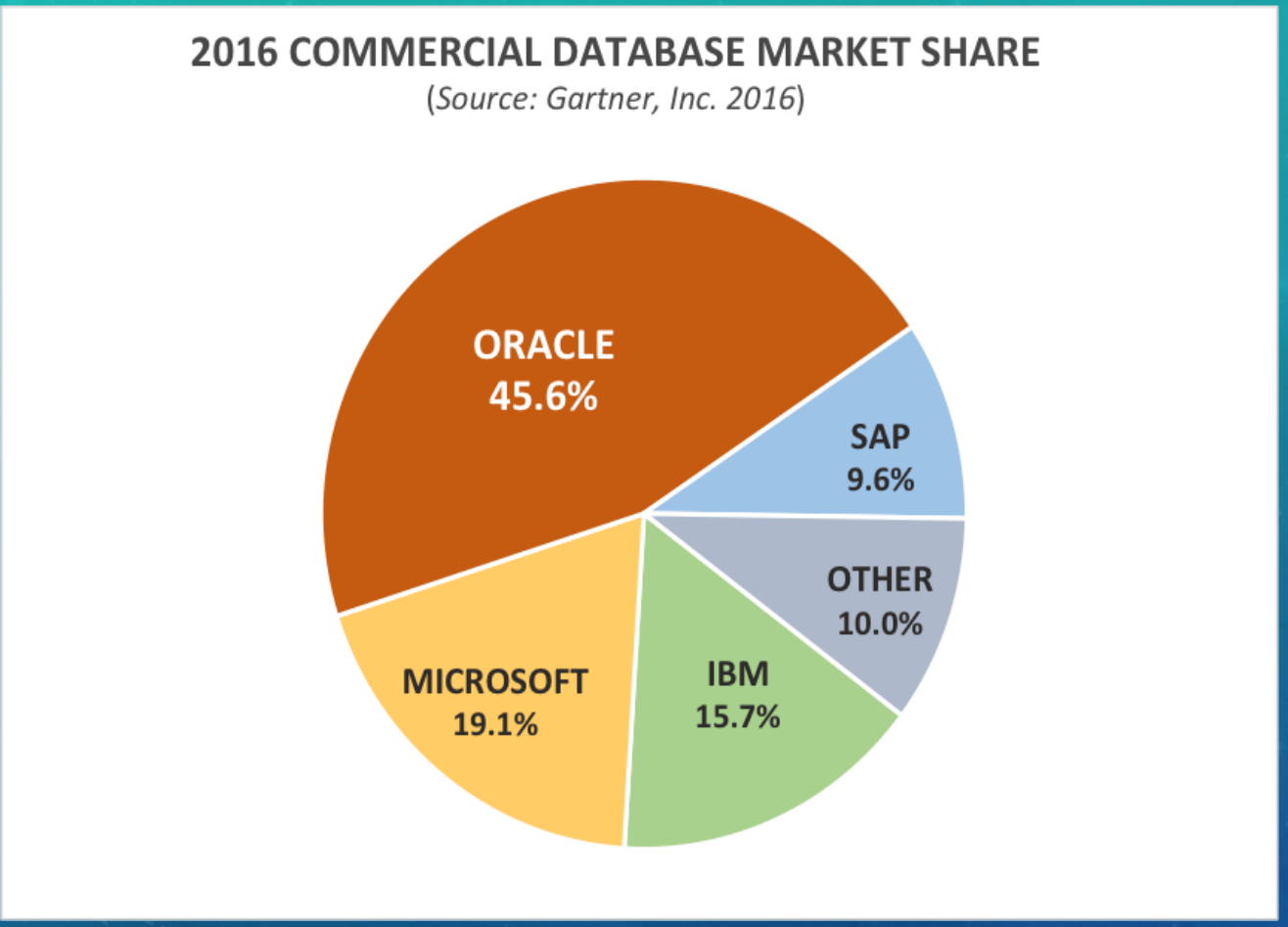
On

14th Nov 2018

By

G. Mayil Muthu Kumaran
Scientist - F

RDBMS and its usage in todays world



RDBMS dominates the industry with over 70% dominance over other available Databases

Database which is one of the major component of an application, plays a major role in application performance

90% of NIC's applications run on RDBMS

Source: Gartner, Inc. 2016

SQL Optimization

- Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans

Purpose

- Processing times of the same query may have large variance, from seconds to hours, depending on the way the query is written.
- The purpose of query optimization, is to find the way to process a given query in optimum time.

Why Query Optimization ?

- It provides the user with faster results, which makes the application perform faster.

- It allows the system to handle more queries in the same time, as an optimized query takes considerably less time than an un-optimized query.

- Query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

Tables for executing test cases

TABLE: MASTER_CARD_TYPE	
Number of Records: 6	
Column Name	**Data Type**
card_type_id	smallint
card_desc_en	character varying(150)
card_desc_ll	character varying(150)
active	integer

TABLE: RATION CARD	
Number of Records: 5,00,000	
Column Name	**Data Type**
ration_card_id	integer
ration_card_no	character varying(12)
card_type_id	integer
application_type	bigint
fps_id	character varying(12)
village_code	character varying(16)
panchayat_code	character varying(13)
tehsil_code	character varying(5)
subdivision_code	character varying
district_code	character varying(3)
state_code	character varying(2)
plc_code	character varying(16)
total_income	integer
income_type_id	smallint
rc_status	character varying(30)
active	smallint

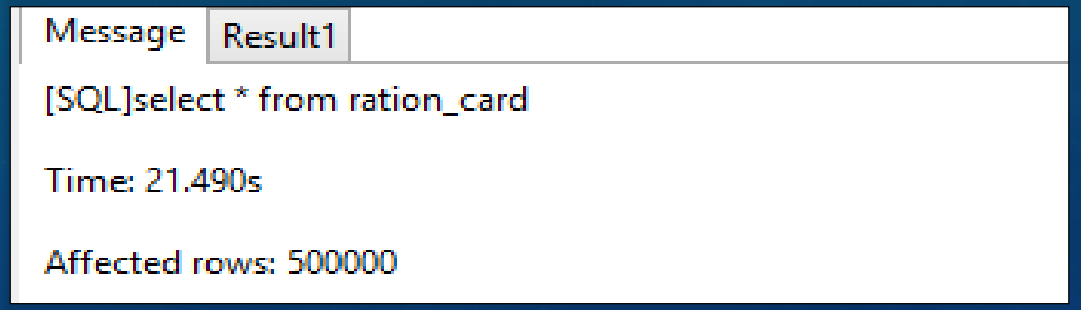
Query Optimization Techniques

Select Statement

1. Avoid unnecessary columns in select clause. Select required columns instead of select *

Query

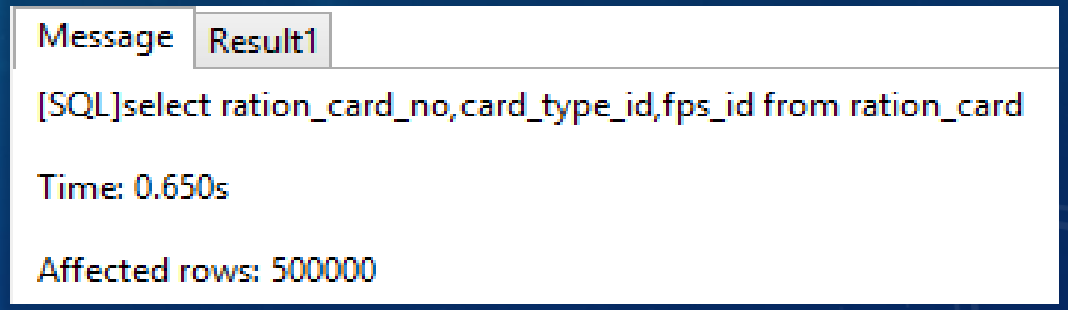
Select ***** *from ration_card*



Time: 21.490s

Optimized Query

Select ***ration_card_no,card_type_id,fps_id*** *from ration_card*



Time: 0.650s

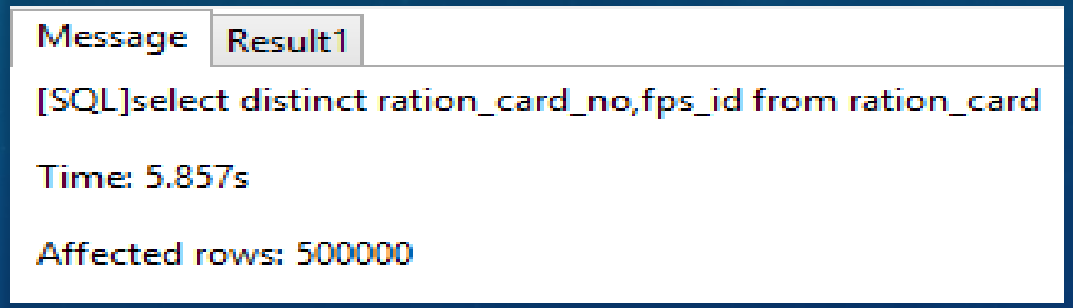
Query Optimization Techniques

Distinct

2. DISTINCT could be avoided if the objective can be achieved otherwise. DISTINCT requires extra sort operation and therefore slowdowns the queries.

Query

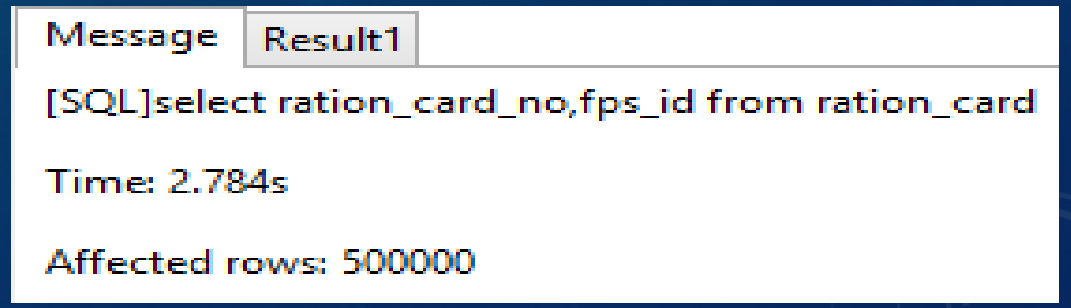
*Select **distinct** ration_card_no,fps_id from ration_card*



Time: 5.857s

Optimized Query

Select ration_card_no,fps_id from ration_card



Time: 2.784s

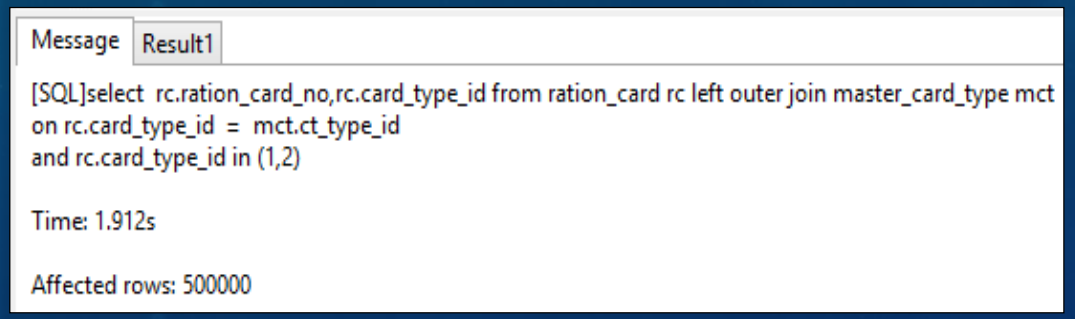
Query Optimization Techniques

Joins

3. Duplicate conditions for constant values whenever possible

Query

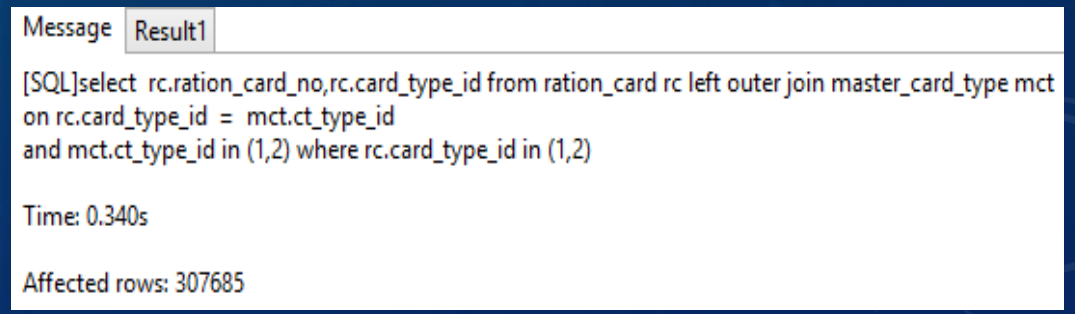
select rc.ration_card_no,rc.card_type_id from ration_card rc ***left outer join*** *master_card_type mct on rc.card_type_id = mct.ct_type_id and* ***rc.card_type_id in (1,2)***



Time: 1.912s

Optimized Query

select rc.ration_card_no,rc.card_type_id from ration_card rc ***left outer join*** *master_card_type mct on rc.card_type_id = mct.ct_type_id and* ***mct.ct_type_id in (1,2) where rc.card_type_id in (1,2)***



Time: 0.340s

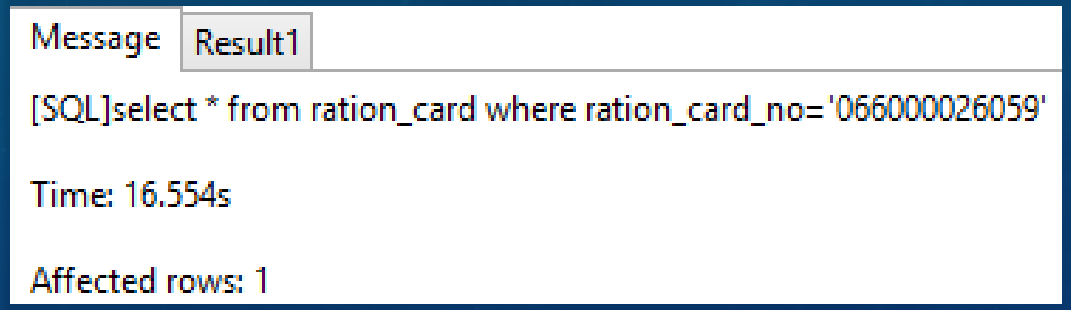
Query Optimization Techniques

Where

4. Using indexed column in where clause improves performance compared to non-indexed column in where clause.

Query

*select * from ration_card where ration_card_no= '066000026059'*

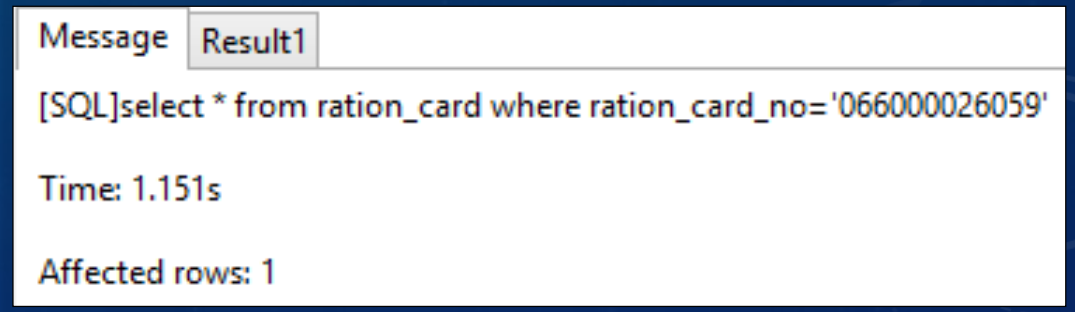


Time: 16.554s

Optimized Query

Create Index: create index rc_no on ration_card(ration_card_no);

*select * from ration_card where ration_card_no= '066000026059'*



Time: 1.151s

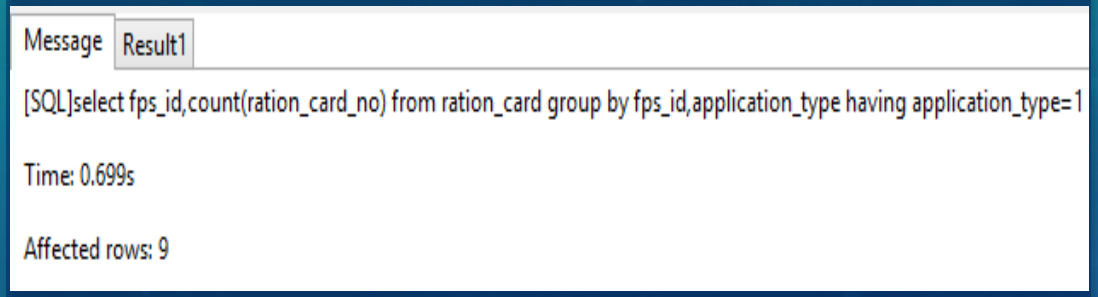
Query Optimization Techniques

Where

5. WHERE clause can be used in place of HAVING clause to define filters, since SQL evaluates the WHERE clause before HAVING clause.

Query

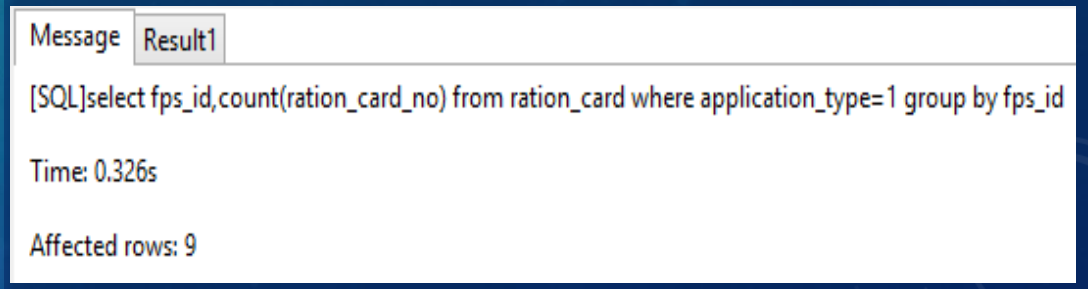
select fps_id,count(ration_card_no) from ration_card group by fps_id,application_type ***having*** *application_type=1*



Time: 0.699s

Optimized Query

select fps_id,count(ration_card_no) from ration_card ***where*** *application_type=1 group by fps_id*



Time: 0.326s

Query Optimization Techniques

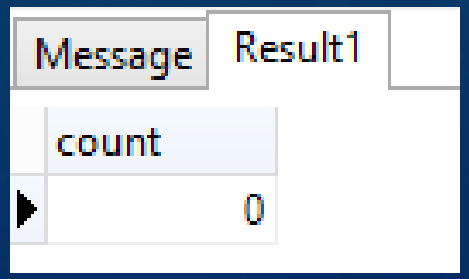
IN and EXISTS

6. The EXISTS clause is much faster than IN clause, when the subquery result is very large. Conversely, the IN clause is faster than EXISTS when the subquery result is very small. ***(Oracle)***

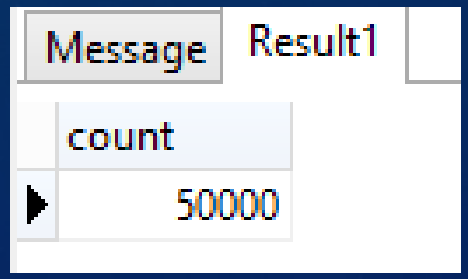
7. Use NOT EXISTS operator instead of NOT IN (NOT IN doesn't consider null values) to obtain accurate result.

The Following Queries gives different outputs

select count(1) from ration_card where card_type_id ***not in*** (select ct_type_id from master_card_type)



*select count(1) from ration_card rc where **not exists** (select 1 from master_card_type mct where rc.card_type_id=mct.ct_type_id)*



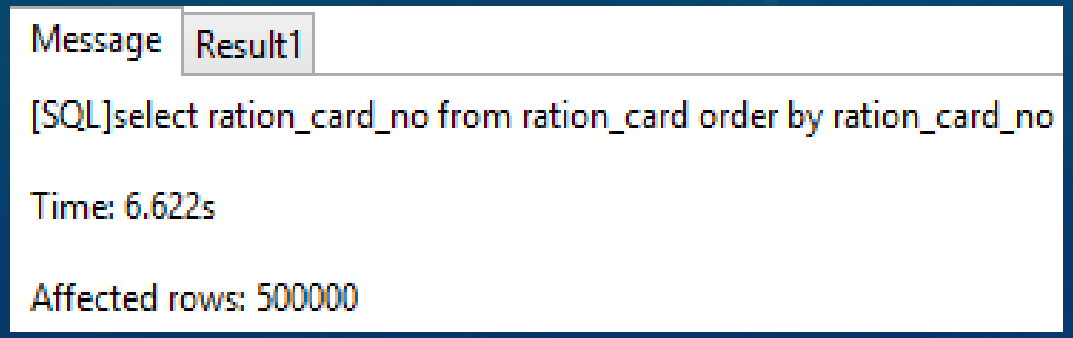
Query Optimization Techniques

ORDER BY and GROUP BY

8. Avoid using ORDER BY on a large data set especially if the response time is important

Query

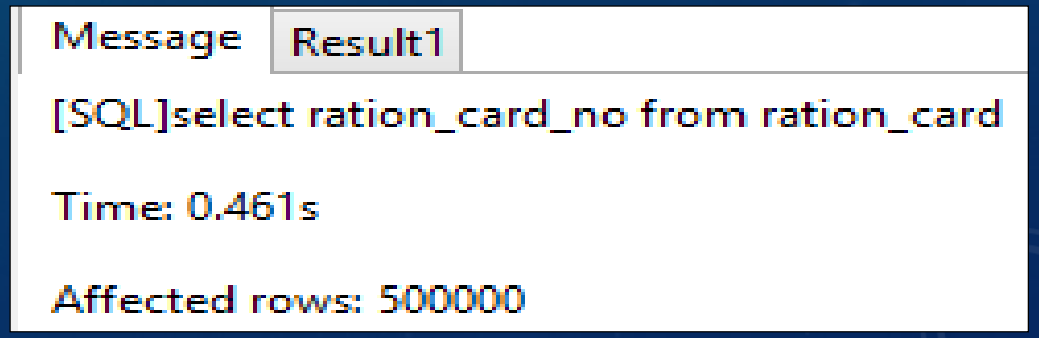
select ration_card_no from ration_card ***order by*** *ration_card_no*



Time: 6.622s

Optimized Query

select ration_card_no from ration_card



Time: 0.461s

Query Optimization Techniques

DELETE vs TRUNCATE

9. To delete all the rows of table permanently, use truncate instead of delete.

- ✓ ***Delete*** command maintains log and it ***can be undone***.
- ✓ ***Truncate*** removes the data by de-allocating the data pages used to store the table data and it ***can not*** be undone.

Query Optimization Techniques

UNION

10. SET operator UNION could be avoided if the objective can be achieved through an UNION ALL.

- ✓ UNION incurs an extra sort operation which can be avoided.

Query Optimization Techniques

DATATYPES

11. ***Char vs. Varchar2***: Prefer char to varchar2 if column width is less than 5.

    ✓ This prevents surplus overhead of adjustments when data is changed.

12. ***Varchar vs. Varchar2***: Varchar is prone to changes in future, so it is advisable to prefer varchar2 over varchar.

13. Do not mix data types

    ✓ Don’t use quotes if a WHERE clause column predicate is numeric. Similarly use quotes for char index columns. This reduces the rate of typecasting.

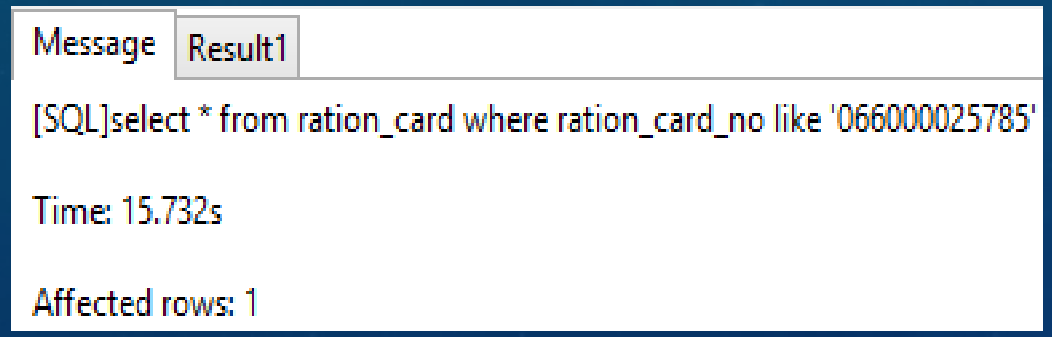
Query Optimization Techniques

MISCELLANEOUS

14. Avoid LIKE predicate: Always replace "like" with an equality, when appropriate.

Query

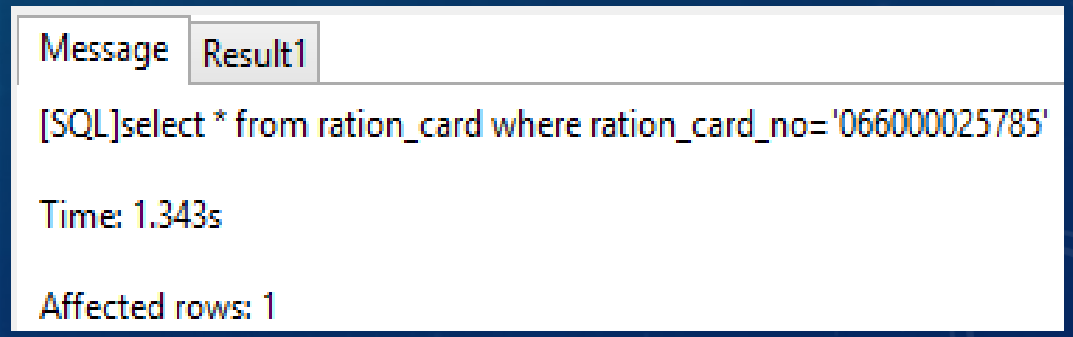
*select * from ration_card where ration_card_no **like** '066000025785'*



Time: 15.732s

Optimized Query

*select * from ration_card where ration_card_no= '066000025785'*



Time: 1.343s

Query Optimization Techniques

MISCELLANEOUS

15. Use Identical Query statements

- ✓ Identical queries are parsed once, whereas non-identical statements are parsed each time on their arrival

Sample

Different data values:
select a from t where c=1;
select a from t where c=2;

Uneven spacing:
select a from t where c=1;
select a from t where c = 1;

Discrepancy in case of letters: *(in case of case sensitive databases)*
select a from t where c=1;
select a FROM t where c=1;

Query Optimization Techniques

MISCELLANEOUS

16. Rewrite complex sub-queries with temporary tables

- ✓ Long and complex queries are hard to understand and optimize. Staging tables can break a complicated SQL statement into several smaller statements, and then store the result of each step in the temporary table.

Sample

select A.ration_card_no from ***(select rc.ration_card_no,rc.card_type_id from ration_card rc join master_card_type mct on rc.card_type_id=mct.ct_type_id)A*** *where A.card_type_id in (1,2) and active=1*

Query Optimization Techniques

MISCELLANEOUS

17. Always use table aliases when referencing columns

- ✓ Alias removes ambiguity when multiple tables are used in the query.
- ✓ The SQL parsing engine looks at the aliases, and uses it to help remove ambiguities in symbol lookups.

Sample

```
Select rc.card_type_id from ration_card rc join
master_card_type mct on
rc.card_type_id=mct.ct_type_id
```

Query Optimization Techniques

MISCELLANEOUS

18. Tracing Query Execution

- ✓ Tracing the query execution provides us time elapsed for execution, cost of CPU, plan hash value and number of bytes accessed and other details which can be used for monitoring the query performance

Sample *(Oracle)*

```
Select x.sid,x.serial#,x.username,x.sql_id,
x.sql_child_number,optimizer_mode,hash_value,address,sq
l_text from v$sqlarea sqlarea,v$session x where
x.sql_hash_value = sqlarea.hash_value and
x.sql_address=sqlarea.address and x.username is not null;
```

Query Optimization Techniques

MISCELLANEOUS

19. Use decode and case

- ✓ Performing complex aggregations with the "decode" or "case" functions can minimize the number of times a table has to be selected.
- ✓ In order to perform more calculations upon same rows in table, prefer CASE to multiple queries.

Sample *(Oracle)*

```
select CASE WHEN count(total_income)>0 THEN
sum(total_income) ELSE 0 END as total_income from
ration_card
```

Query Optimization Techniques

MISCELLANEOUS

20. Use Wildcards at the End of a Phrase only

Query

Consider this query to pull cities beginning with 'char':

Select** city from beneficiaries **where** city like **'%char%'

This query will pull the expected results of charles, charley and charlian. However, it will also pull unexpected results, such as cape charles, carl orchard, and richardson.

Optimized Query

Select** city from beneficiaries **where** city like **'char%'

This query will pull only the expected results of charles, charley and charlian.

Query Optimization Techniques

MISCELLANEOUS

21. If a query is to be run over a large dataset, validate the query by using limit statement to fetch limited records. (Some DBMS systems, uses ‘top’ interchangeably with limit.)

- ✓ The limit statement returns only the number of records specified. Using a **limit** statement prevents taxing the production database with a large query, only to find out the query needs editing or refinement.

Query Optimization Techniques

MISCELLANEOUS

22. Consider using materialized views. These are pre-computed tables comprising aggregated or joined data from fact and possible dimension tables.

- ✓ A materialized view is a view whose result set is stored on disk, much like a base table, but that is computed, much like a view.
- ✓ Materialized views are designed to improve performance in environments where:

  - The database is large
  - Frequent queries result in repetitive aggregation and join operations on large amounts of data
  - Changes to underlying data are relatively infrequent

Query Optimization Techniques

MISCELLANEOUS

23. Use full-table scans when needed

- ✓ Not all OLTP queries are optimal when they use indexes.
- ✓ If the query returns a large percentage of table's rows, a full-table scan may be faster than an index scan.
- ✓ This depends on many factors, including configuration (values for db_file_multiblock_read_count, db_block_size), query parallelism and the number of table/index blocks in the buffer cache.

24. Use Stored Procedure for frequently used data and more complex queries.

Query Optimization Techniques

MISCELLANEOUS

25. Triggers shall not be overused

- ✓ Trigger chaining will drag down the performance.

26. Access rows using Row id/similar identifier depending on the database

- ✓ Accessing rows using rowid is fastest compared to other methods.

27. Avoid using functions such as RTRIM, TO_CHAR, UPPER, TRUNC with indexed columns as It prevents the optimizer from identifying the index.

28. Avoid Using nested Views

- ✓ A view inside another view is an inefficient way of creating views. It is complex and consumes more time.

Thank You !

support-sqg@nic.in
VoIP : 5748