

## Airline Management System – Performance Tuning

This document covers some of the experiments run as part of the project related to performance tuning.

- **INDEXING:**

What is does:

Indexes help the query optimizer to speed up the execution of a query.

Procedure:

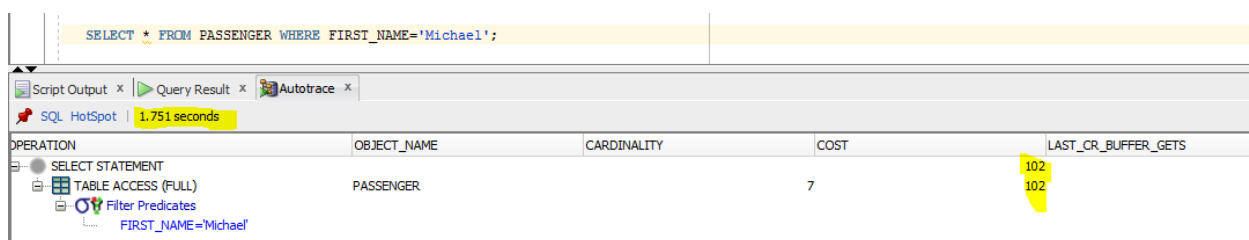
```
CREATE INDEX IDX_PASSENGER ON PASSENGER (FIRST_NAME) ONLINE;
```

Results:

Below are the results for below query:

```
SELECT * FROM PASSENGERS WHERE FIRST_NAME='Michael';
```

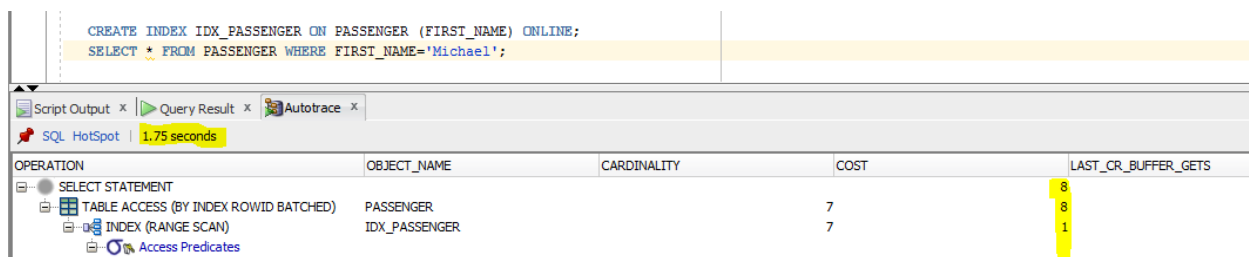
**Before:**



The screenshot shows the SQL Developer interface with the Autotrace window open. The query being executed is `SELECT * FROM PASSENGER WHERE FIRST_NAME='Michael';`. The execution time is 1.751 seconds. The Autotrace table shows the following details:

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT				102
TABLE ACCESS (FULL)	PASSENGER	7	7	102
Filter Predicates				
FIRST_NAME='Michael'				

**After:**



The screenshot shows the SQL Developer interface with the Autotrace window open. The query being executed is `SELECT * FROM PASSENGER WHERE FIRST_NAME='Michael';`. The execution time is 1.75 seconds. The Autotrace table shows the following details:

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT				8
TABLE ACCESS (BY INDEX ROWID BATCHED)	PASSENGER	7	7	8
INDEX (RANGE SCAN)	IDX_PASSENGER	7	7	1
Access Predicates				

**Inference:**

We can infer from the screenshots that the execution time has been reduced and the cost was reduced by almost one-tenth of its previous value after the usage of index.

- **Materialized Views:**

What is does:

Materialized views are generally used to pre-compute and store aggregated data. This increases data availability by providing local access to the target data and when combined with mass deployment and data sub setting (both of which also reduce network loads), greatly enhance the performance and reliability of the database.

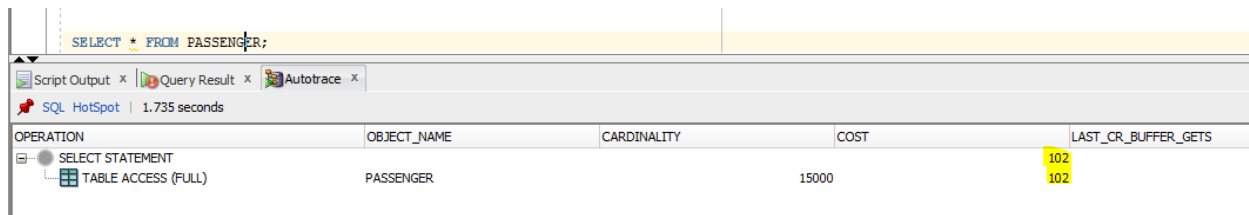
Procedure:

**CREATE** MATERIALIZED **VIEW** MVF **AS**  
**SELECT \* FROM** PASSENGER;

Results:

Below are the results:

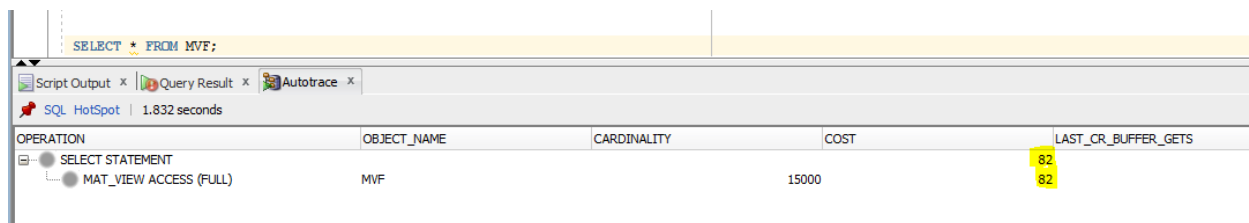
**Before:**



The screenshot shows the SQL Developer interface with the query 'SELECT \* FROM PASSENGER;' entered. The Autotrace window displays the execution plan for this query. The execution time is 1.735 seconds. The execution plan table is as follows:

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT				102
TABLE ACCESS (FULL)	PASSENGER		15000	102

**After:**



The screenshot shows the SQL Developer interface with the query 'SELECT \* FROM MVF;' entered. The Autotrace window displays the execution plan for this query. The execution time is 1.832 seconds. The execution plan table is as follows:

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT				82
MAT_VIEW ACCESS (FULL)	MVF		15000	82

**Inference:**

Since the data is already pre-computed cost of execution is brought down considerably when implementing Materialized Views.

- **Optimizer Hints:**

What is does:

Hints provide a mechanism to instruct the optimizer to choose a certain query execution plan based on the specific criteria. The hint `FIRST_ROWS(n)` (where `n` is any positive integer) or `FIRST_ROWS` instruct Oracle to optimize an individual SQL statement for fast response. `FIRST_ROWS(n)` affords greater precision, because it instructs Oracle to choose the plan that returns the first `n` rows most efficiently.

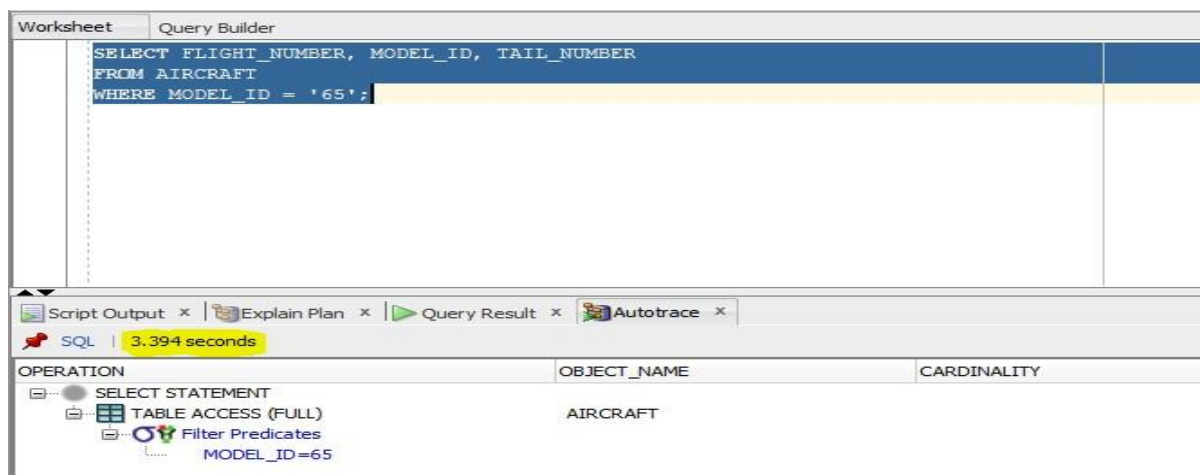
Procedure:

```
SELECT /*+FIRST_ROWS(10)*/FLIGHT_NUMBER, MODEL_ID, TAIL_NUMBER FROM AIRCRAFT
WHERE MODEL_ID='65';
```

Results:

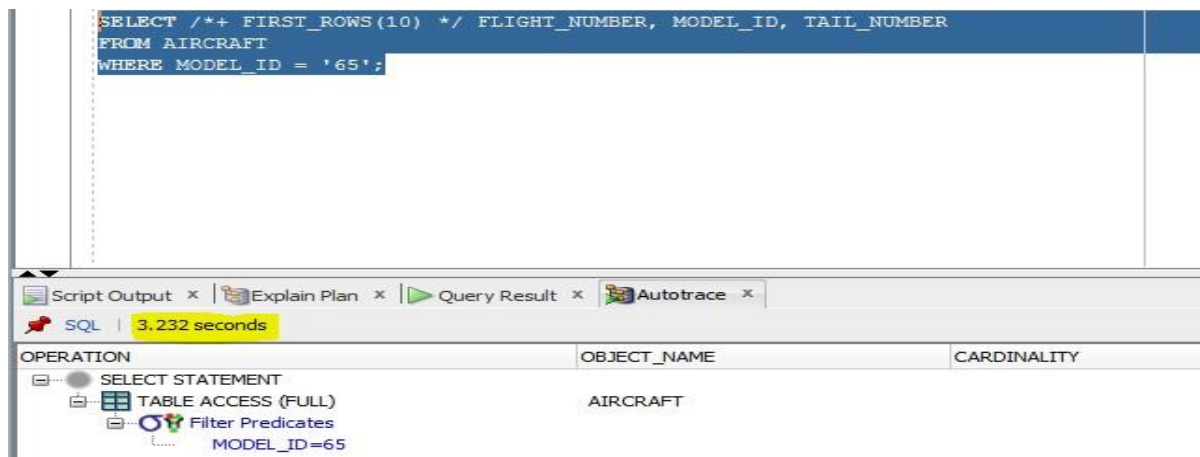
Below are the results:

**Before:**



OPERATION	OBJECT_NAME	CARDINALITY
SELECT STATEMENT		
TABLE ACCESS (FULL)	AIRCRAFT	
Filter Predicates		
MODEL_ID=65		

**After:**



OPERATION	OBJECT_NAME	CARDINALITY
SELECT STATEMENT		
TABLE ACCESS (FULL)	AIRCRAFT	
Filter Predicates		
MODEL_ID=65		

**Inference:**

In the above example, there are various models of aircrafts. However, the user wants the first 10 aircrafts of `MODEL_ID '65'` to be displayed as quickly as possible. The optimizer uses the query optimization approach to optimize this statement for best response time.

- **Clustering:**

What is does:

A cluster is a schema object that contains data from one or more tables, all of which have one or more columns in common.

When using clusters, a query can retrieve all the related data with usually no more than 2 logical reads -- one to get the cluster key and another to retrieve a data block that contains only the relevant data.

Procedure:

```
CREATE CLUSTERCLSTR_LEG (LEG_IDVARCHAR(20)) SIZE 600 TABLESPACE ANIMALS STORAGE (INITIAL 200K NEXT 300K MINEXTENTS 2 PCTINCREASE 10);
```

```
CREATE TABLE LEGS (LEG_IDVARCHAR(20) PRIMARYKEY, .etc) CLUSTERCLSTR_LEG (LEG_ID);
```

```
CREATE TABLE ITINERARY_LEGS (LEG_IDVARCHAR(20) PRIMARYKEY, ..etc) CLUSTERCLSTR_LEG (LEG_ID);
```

```
CREATE INDEXCLSTR_LEG_INDEX ON CLUSTERCLSTR_LEG TABLE SPACE ANIMALS STORAGE (INITIAL 50K NEXT 50K MINEXTENTS 2 MAXEXTENTS 10 PCTINCREASE10);
```

Results:

Since, this concept struck us after we had created all our tables and the related constraints, we were unable to implement and verify this in our database and we couldn't alter the design as it would affect other tables and relations.

**Inference:**

As per our understanding of the concept, Clustering helps in reducing the amount of storage space required and also can improve data compression, and hence achieve better results when it comes to the table scan costs.

- **SQL Tuning:**

The queries can be optimized for better performance by following certain techniques as below:

- Various performance improvements can be made so that SELECT statements run faster and consume less memory cache.
  - ✓ When ORDER BY columns are the prefix of GROUP BY columns, and all columns are sorted in either ascending or in descending order, the sorting step for the query result is eliminated.
- The operator such as EXISTS and IN should be used appropriately in queries.
  - ✓ In case of EXISTS clause, the main query is evaluated first and then if the rows exists, they are joined with the results of sub-query. So, EXISTS is efficient when most of the filter criteria is in the main query.
  - ✓ In the case of IN clause, the sub-query is evaluated first and the results are joined with the main table. So, IN is efficient when most of the filter criteria are in the sub-query.
- Efficient usage of JOINS.
  - ✓ The smaller tables should be joined first to reduce temporary table sizes.
  - ✓ The most restrictive SELECT and JOIN operations are applied first.
- Conversion of IN sub-query to a join.

For example:

```
SELECT LEG_ID
FROM ITINERARY_LEGS
WHERE LEG_ID IN
      (SELECT LEG_ID
       FROM LEGS);
```

Can be written AS:

```
SELECT LEG_ID
FROM ITINERARY_LEGS,
      LEGS
WHERE ITINERARY_LEGS.LEG_ID=LEGS.LEG_ID;
```

- Usage of sub-queries should be avoided wherever possible. Lesser the number of sub-queries, greater will be the efficiency.
- The performance of joins on multiple tables can be improved by indexing the columns used for joins.
- Including column names in the SELECT statement instead of "\*" helps in faster retrieval of data.
- When using JOINS with tables having one-to-many relationship, it's advisable to use EXISTS instead of DISTINCT, since DISTINCT is a relatively costlier operator