Oracle Performance Tuning

2005 Macneil Fernandes
Technical Leader

Course Objectives

At the end of the course you will be able to

- List tuning goals
- Create efficient SQL queries
- Identify appropriate indexes
- Use SQL Tuning tools
- Gain hands on experience in tuning SQL queries

Prerequisites for the course

- **Oracle SQL**

Session Plan

Forenoon

- Tuning overview
- Oracle Architecture
- SQL concepts

Afternoon

- SQL Tuning Tools
- Indexing Principles
- Tuning SQL
- Case Study
- Mock Q&A Session
- Test

H/W & S/W Required

Windows / Unix

Oracle database (8i / 9i)

Oracle Client

Topics of Discussion

1. **Tuning overview**
   - Who tunes?
   - Purpose of tuning
   - Tuning Parameters
   - Tuning Phases
2. **Oracle Architecture**
   - Basic Architecture
   - Storage Hierarchy

Topics of Discussion (contd..)

3. SQL concepts

- SQL Processing architecture
- Understanding optimizer
- Optimizer hints
- Data access paths
- Understanding Joins
- Shared SQL
- Partitioning

4. SQL Tuning Tools

- Explain Plan
- Auto trace
- RBO vs CBO
- Exercise

Topics of Discussion (contd..)

5. Indexing principles

- When to use Index
- Selectivity
- Type of Indexes
- Fixing bad index
- Concatenated Index
- Multiple Index
- Exercise

6. Tuning SQL

- General tuning tips
- Using indexes
- Suppressing Indexes
- Working with dates
- Parallel Execution
- Exercise

Topics of Discussion (contd..)

7. **Case Study**
8. **Mock Session**
9. **Summary**
10. **References**
11. **Test**

Session

Tuning Overview

Tuning Overview

Who tunes?

- **Database Designer**

  Develops logical model which is interpreted to a physical database design.

- **Application Developer**

  Builds the actual application code.

- **System Administrator**

  Responsible for resource management and allocation between oracle and other applications sharing the same platform.

- **Database Administrator**

  Monitors and tunes the database.

Tuning Overview (contd..)

Purpose of tuning

The best practice of database tuning is to carefully design the system and application.

Performance tuning is done to:

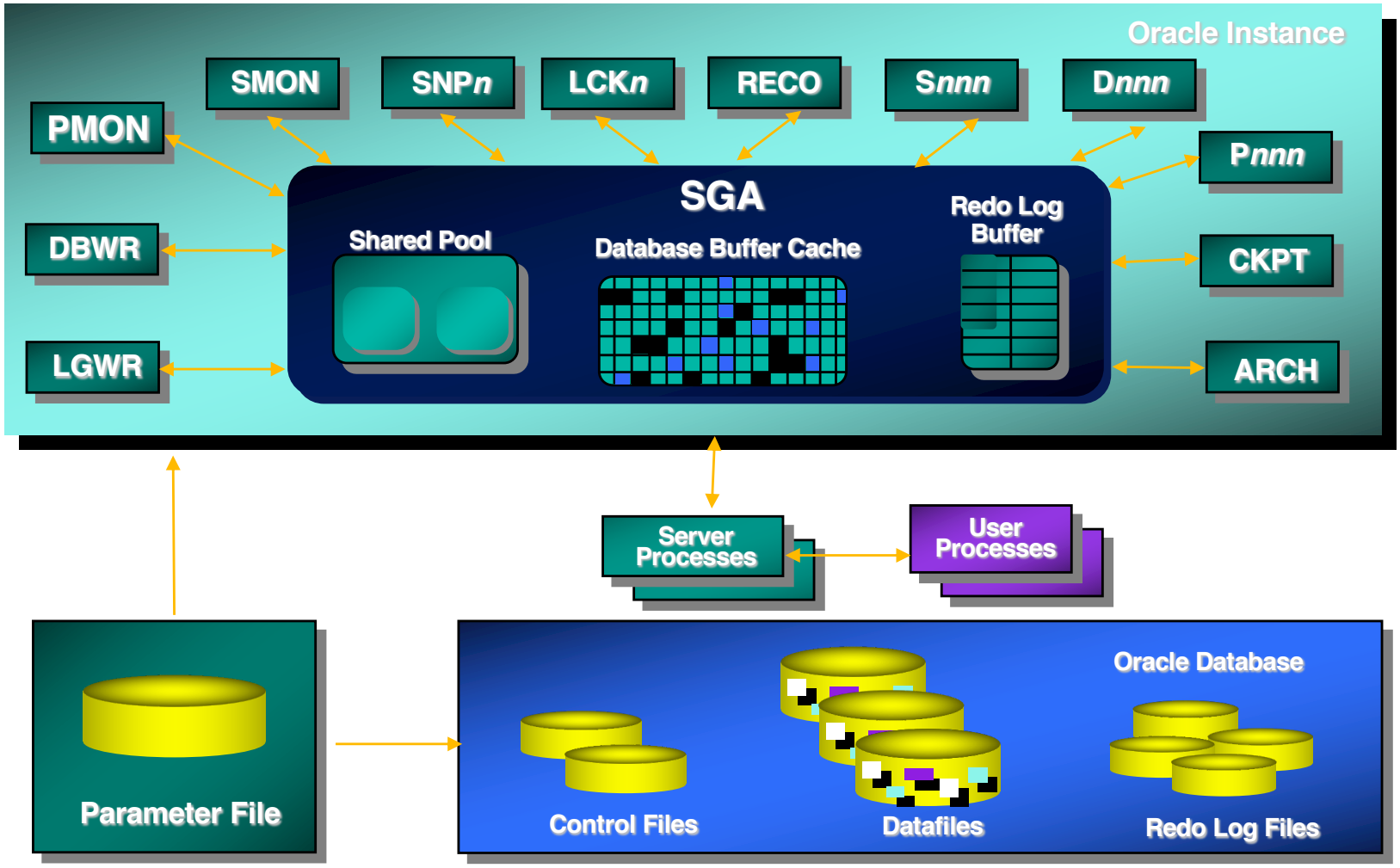
- improve customer service
- optimize hardware usage to save money
- prevent performance crisis
- to meet the performance tuning goals

Tuning Overview (contd..)

Tuning Parameters

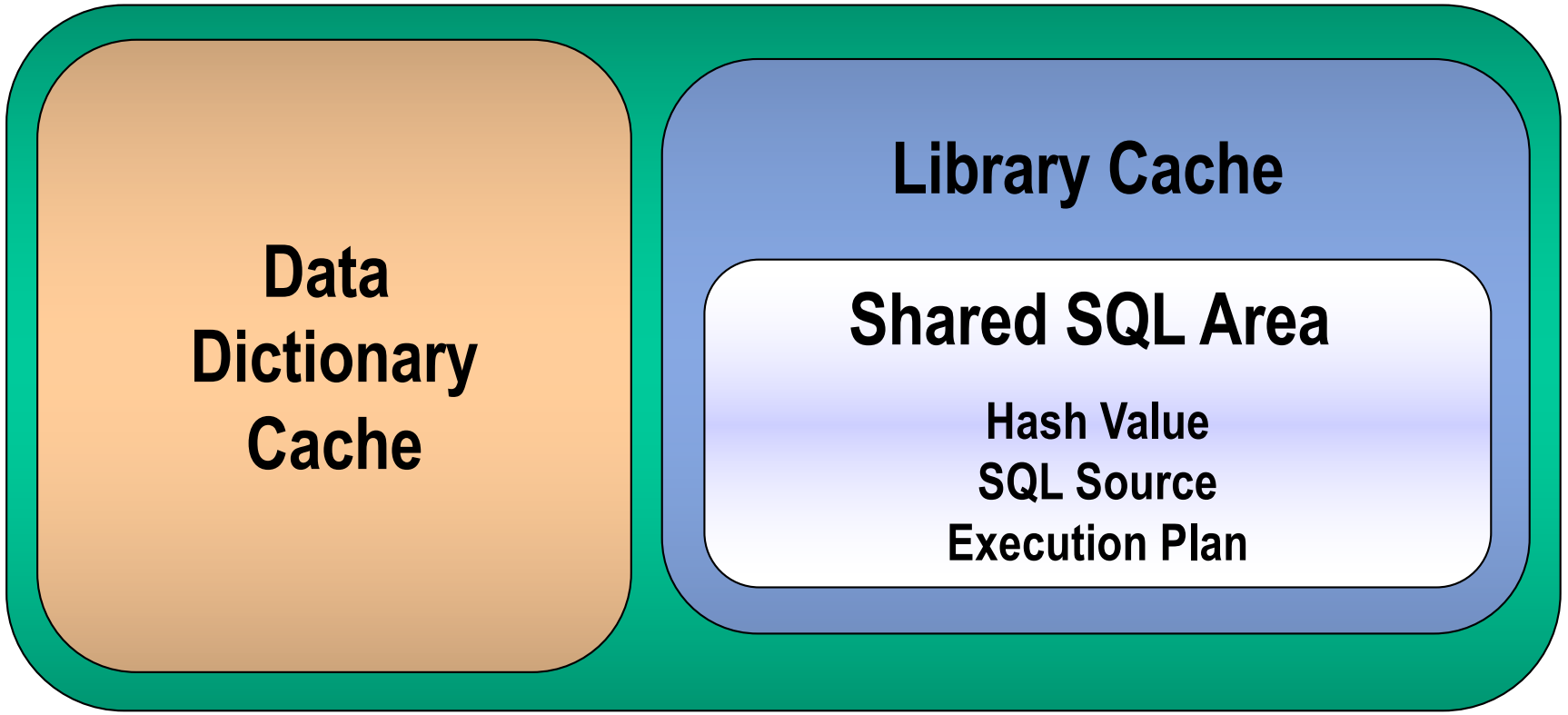
- ➢ Response Time
- ➢ Increase Throughput
- ➢ Scalability v/s Performance
- ➢ Effective usage of resources
- ➢ Reducing or eliminating waits

80% of all performance problems are due to the way SQL statements access the data.Only 20% or less are due to database and system parameters

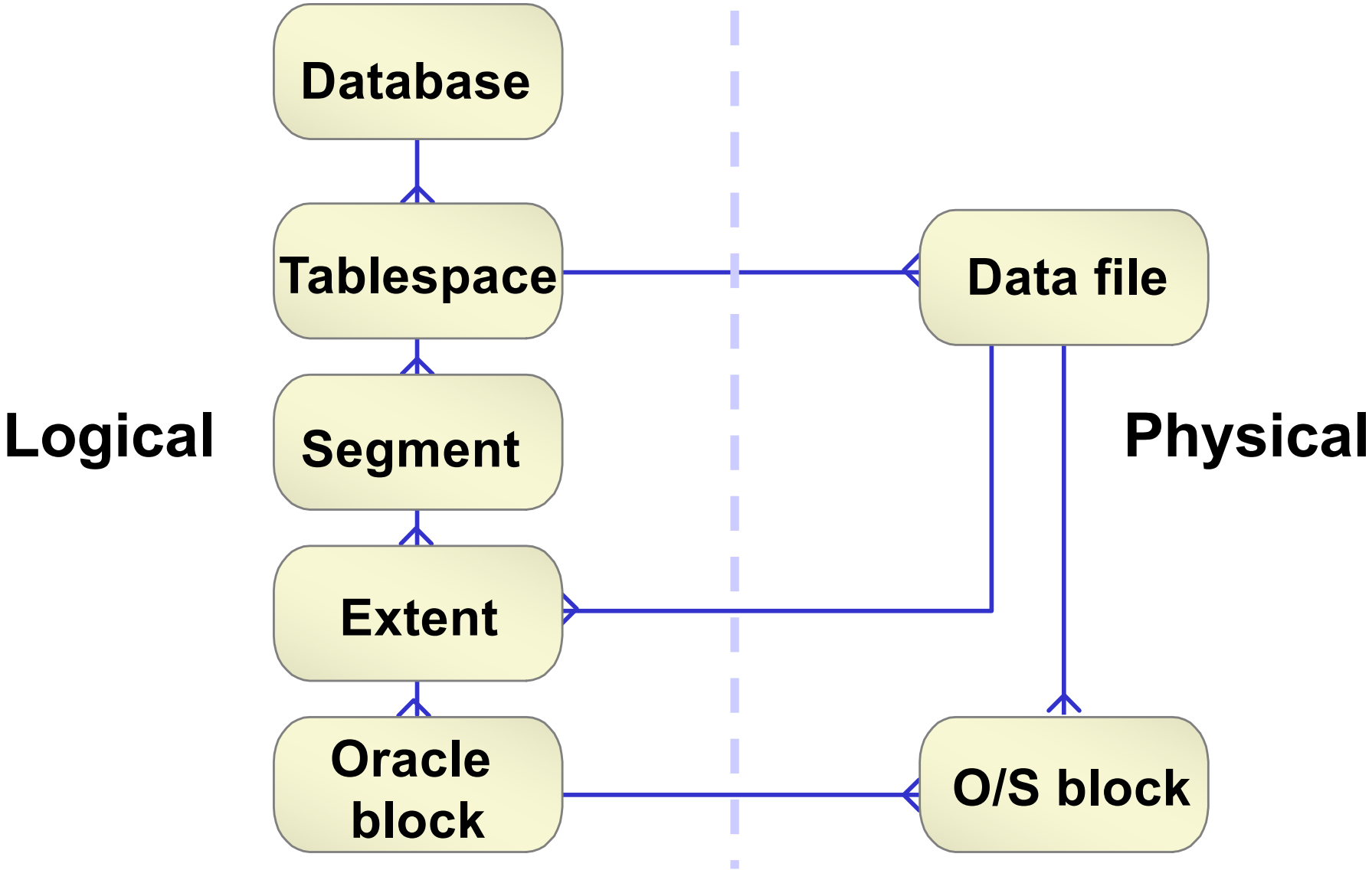

Oracle Instance
SMON
SNPn
LCKn
RECO
Snnn
Dnnn
PMON
Pnnn
SGA
Shared Pool
Database Buffer Cache
Redo Log Buffer
DBWR
CKPT
LGWR
ARCH
Server Processes
User Processes
Parameter File
Oracle Database
Control Files
Datafiles
Redo Log Files

Oracle Database Architecture contd..

Shared Pool



Database Storage Hierarchy



Session

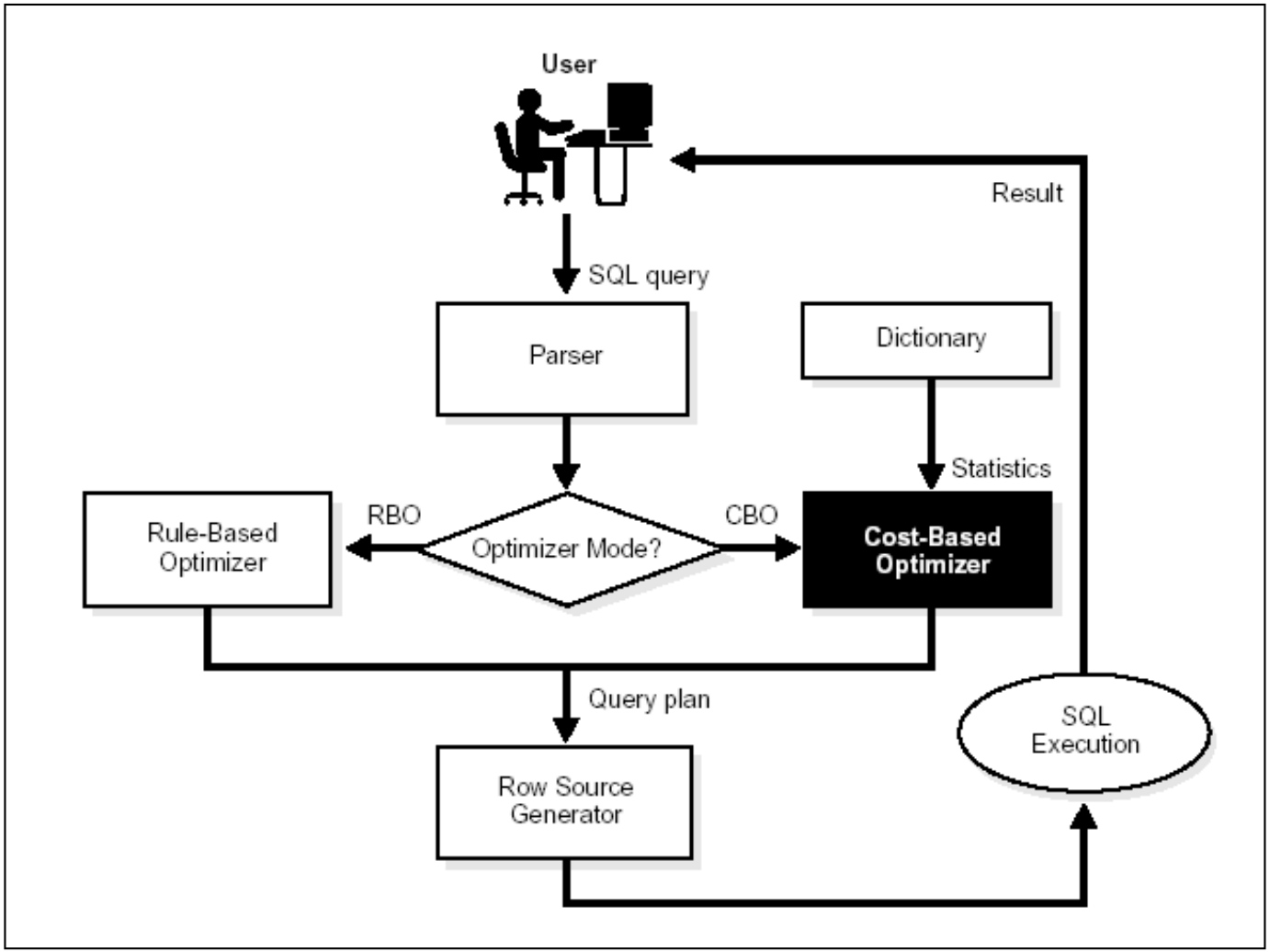
SQL Tuning Concepts

SQL Processing architecture

SQL processing architecture is comprised of the following main components:

- **Parser**
- **Optimizer**
- **Row source Generator**
- **SQL Execution**

SQL Processing Architecture



Optimizer

What is an optimizer?

- The optimizer determines the most efficient way to execute a SQL statement.
- This is an Important step in the processing of any DML.
- Primary Modes of operation

  RULE

  COST
- Optimizer mode can be specified at instance level,session level and at query level.

Optimizer Hints

- Hints provide a mechanism to direct the optimizer to choose a certain query execution plan.
- Hints (except for the RULE hint) invoke the cost-based optimizer (CBO). If you have not gathered statistics, then defaults are used.

  Table statistics can be gathered by giving ANALYZE command or by using DBMS_STATS package.

Example:

```
SELECT /*+  FULL(TRG_EMP) */
FROM   TRG_EMP
WHERE em_id ='1000';
```

Optimizer Hints

Frequently used Hints:

- INDEX
- FULL
- FIRST_ROWS
- RULE
- ORDERED
- LEADING
- USE_NL
- USE_HASH
- PARALLEL
- APPEND

Data Access Paths

Access paths are ways in which data is retrieved from the database. Any row in any table can be located and retrieved by one of the following methods:

- Full table scan
- Rowid scan
- Index unique scan
- Index range scan
- Index skip scan
- Fast full index scan

Data Access Paths (Contd..)

Full Table Scan

- Sequentially reads each row of a table.
- Used whenever there is no where clause on a query or no index on the column being used in where clause .

Table Access by Rowid

- Locating a row by its rowid is the fastest way for Oracle to find a single row.
- To access a table by rowid, Oracle first obtains the rowids of the selected rows, either from the statement's WHERE clause or through an index scan of one or more of the table's indexes.Oracle then locates each selected row in the table based on its rowid.

Example:

```
SELECT    ename, deptno
FROM      EMP
WHERE     rowid='AAAKJXAABAAAKECAAA'
```

Data Access Paths (Contd..)

Index Unique Scan

This scan returns, at most, a single rowid. Oracle performs a unique scan if a statement contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed.

Index Range Scan

An index range scan is a common operation for accessing selective data.

The optimizer uses a range scan when it finds one or more leading columns of an index specified in conditions, such as the following:

- col1 = :b1 , col1 < :b1 , col1 > :b1
- AND combination of the preceding conditions for leading columns in the index
- col1 like '%ASD' Wild-card searches should not be in a leading position. The condition col1 like '%ASD' does not result in a range scan..

Data Access Paths (Contd..)

Index skip scan

- Index skip scan feature enables the optimizer to use a concatenated index even if its leading column is not listed in the WHERE clause.
- Faster than full scans of the index,requiring fewer reads to be performed.

Fast Full Index scan

Fast full index scans are an alternative to a full table scan when the index contains all the columns that are needed for the query, and at least one column in the index key has the NOT NULL constraint.

Join Operations

Nested Loops

- Nested loops joins are ideal when the driving row source is small and the joined columns of the inner row source are uniquely indexed or have a highly selective non-unique index.
- You drive from the outer loop to the inner loop, so the order of tables in the execution plan is important.

Sort Merge Join

- Sort merge joins can be used to join rows from two independent sources.
- Oracle sorts the first row source by its join columns,sorts the second row source by its join columns ,and then merges the sorted row sources together.

Join Operations (contd..)

Hash Join

- Hash joins are used for joining large data sets.
- The optimizer uses the smaller of two tables or data sources to build a hash table on the join key in memory. It then scans the larger table, probing the hash table to find the joined rows.

Join Operations (contd..)

	Nested Loops Join	Sort-Merge Join	Hash Join	Cluster Join
When can be used:	Any join	Equijoins only	Equijoins only	Equijoins on complete cluster key of clustered tables only
Optimizer hint:	use_nl	use_merge	use_hash	None
Resource concerns:	CPU Disk I/O	Temporary segments	Memory	Storage
init.ora parameters:	None	sort_area_size db_file_multiblock_read_count	hash_join_enabled hash_area_size hash_multiblock_io_count	None
Features:	Works with any join Efficient with highly selective indexes and restrictive search criteria Returns first rows faster than sort-merge and hash Requires no sorting	Better than nested loops when index is missing or search criteria is not restrictive Can work with limited memory	Better than nested loops when index is missing or search criteria is not restrictive Can be faster than sort-merge	Reduces I/O for master-detail queries and self-joins based on cluster key Returns first rows the fastest Requires no sorting
Drawbacks:	Very inefficient when no suitable index exists or criteria isn't restrictive	Must perform an extra sort Cannot return first rows quickly	Can require lots of memory Cannot return first rows quickly Requires Oracle 7.3 or later	Clustered data can take more space to store Clusters slow down updates and full table scans

Shared SQL

- ORACLE holds SQL statements in memory after it has parsed them, so the parsing and analysis won't have to be repeated if the same statement is issued again.
- The single shared context area in the shared buffer pool of the System Global Area (SGA) is shared by all the users.
- The larger the area, the more statements that can be retained and the more likely statements actually get shared.At the same time, its not advisable to set it to a very high value.
- To make use of shared sql,following conditions need to be satisfied.
    1. There must be a character-by-character match between the statement.
    2. The objects being referenced in the new statement are exactly the same as those in shared sql.
    3. If bind variables are referenced,they must have the same name in both the queries.

Session

SQL Tuning Tools

SQL Tuning Tools

Overview of Diagnostic tools

- **EXPLAIN PLAN**
- **SQL*Plus AUTOTRACE**

Explain Plan

Explain Plan

- The EXPLAIN PLAN statement allows you to submit a SQL statement to Oracle and have the database prepare the execution plan for the statement without actually executing it.
- The execution plan is made available to you in the form of rows inserted into a special table called a *plan table*. You may query the rows in the plan table using ordinary SELECT statements in order to see the steps of the execution plan for the statement you explained.
- You may keep multiple execution plans in the plan table by assigning each a unique statement_id. Or you may choose to delete the rows from the plan table after you are finished looking at the execution plan.
- The EXPLAIN PLAN statement runs very quickly, even if the statement being explained is a query that might run for hours. This is because the statement is simply parsed and its execution plan saved into the plan table. The actual statement is never executed by EXPLAIN PLAN

Explain Plan (contd..)

Steps for using Explain Plan

1. Create the Plan table by running the script
   ORACLE_HOME/rdbms/admin/utlxplan.sql

2. Delete the records from PLAN_TABLE (if statement id is reused)
   DELETE FROM PLAN_TABLE WHERE statement_id = ‘my_id’

3. Run the EXPLAIN PLAN for the query to be tuned
   explain plan set statement_id = ‘my_id’ for << select stmt>>;

4. Select the output from PLAN_TABLE
   select operation,options,object_name,id,parent_id
   from plan_table where statement_id = ‘my_id’;

   To get the formatted output from plan table ,you can
   use utlxpls.sql provided by oracle.

Explain Plan (contd..)

Example: Run Explain Plan for this query:

```
SELECT    a.customer_name,   a.customer_number,
          b.invoice_number,  b.invoice_type,
          b.invoice_date,    b.total_amount,
          c.line_number,     c.part_number,
          c.quantity,        c.unit_cost
FROM      customers a,
          invoices b,
          invoice_items c
WHERE     c.invoice_id = :b1
AND       c.line_number = :b2
AND       b.invoice_id = c.invoice_id
AND       a.customer_id = b.customer_id;
```

Explain Plan (contd..)

Interpreting PLAN_TABLE output

ID	PARENT	OPERATION	OBJECT_NAME
0		SELECT STATEMENT	
1	0	NESTED LOOPS	
2	1	NESTED LOOPS	
3	2	TABLE ACCESS BY INDEX ROWID	INVOICE_ITEMS
4	3	INDEX UNIQUE SCAN	INVOICE_ITEMS_PK
5	2	TABLE ACCESS BY INDEX ROWID	INVOICES
6	5	INDEX UNIQUE SCAN	INVOICES_PK
7	1	TABLE ACCESS BY INDEX ROWID	CUSTOMERS
8	7	INDEX UNIQUE SCAN	CUSTOMERS_PK

SQL Tuning Tools (contd..)

Auto Trace

- Auto trace feature of SQL*PLUS allows automatic display of execution plans and helpful statistics.
- When you turn on this feature,the statement is executed and the results are displayed followed by the execution plan and resource statistics.

Setting AUTOTRACE

```
SET AUTOTRACE OFF|ON|TRACEONLY
                              [EXPLAIN] [STATISTICS]
<<SELECT statement>>
SET AUTOTRACE OFF
```

Auto Trace (contd..)

Sample Autotrace Output

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=4 Card=1 Bytes=39)
   1    0   NESTED LOOPS (Cost=4 Card=1 Bytes=39)
   2    1     NESTED LOOPS (Cost=3 Card=1 Bytes=27)
   3    2       TABLE ACCESS (BY INDEX ROWID) OF 'INVOICE_ITEMS' (Cost
          =2 Card=1 Bytes=15)

   4    3         INDEX (UNIQUE SCAN) OF 'INVOICE_ITEMS_PK' (UNIQUE) (
          Cost=1 Card=2)

   5    2       TABLE ACCESS (BY INDEX ROWID) OF 'INVOICES' (Cost=1 Ca
          rd=2 Bytes=24)

   6    5         INDEX (UNIQUE SCAN) OF 'INVOICES_PK' (UNIQUE)
   7    1     TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (Cost=1 Car
          d=100 Bytes=1200)

   8    7       INDEX (UNIQUE SCAN) OF 'CUSTOMERS_PK' (UNIQUE)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          8  consistent gets
          0  physical reads
          0  redo size
        517  bytes sent via SQL*Net to client
        424  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

SQL Tuning Tools (Contd..)

Exercise

1.Run Explain Plan for the following query and analyze the plan output.

```
SELECT ename, job_code, salary, deptno
FROM trg_emp_sm
WHERE empno = 1978;
```

Run Explain Plan for the following:

- ✓ Query performing full table scan
- ✓ Query performing Index Range scans(non unique index)
- ✓ Query joining 2 or more row sources
- ✓ Query using optimizer hints

2. Auto Trace

(a) Set auto trace on in SQL*PLUS prompt and run the above mentioned queries and analyze the auto trace output.

(b) Try running the queries with various auto trace options.

SQL Tuning Tools (Contd

Exercise

3.Run the following Query

```
SELECT empno, ename, sal, job
FROM emp
WHERE empno = 7566;
```

Use Hints and check the explain plan

Session

Indexing Principles

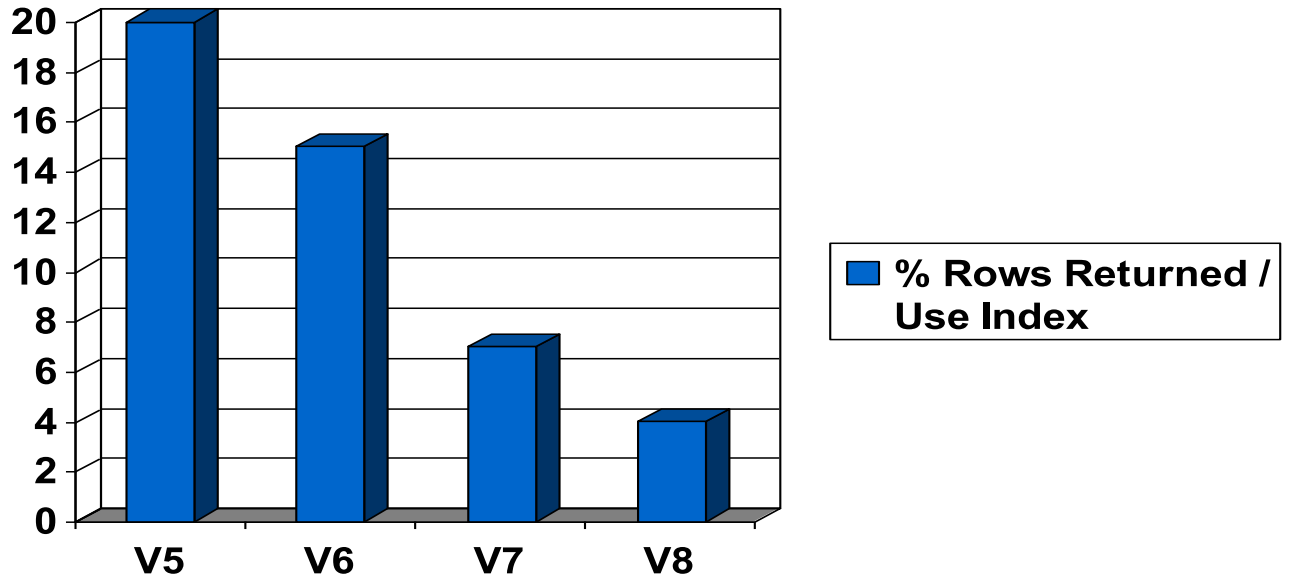
Basic Index Concepts

- Index generally increases the performance for SELECT,UPDATE, DELETE statements (when few records are accessed) and decreases performance for INSERT statements.
- In general, every index on a table slows INSERT into the table by a factor of three;two indexes generally make the insert twice as slow as one index.
- An index on a table column makes the UPDATE to that column slower by a factor of about three depending on storage of data. Similarly INSERT into indexed column is also slower because you need to insert the data record as well as the index record.
- You need to balance the index performance benefits of row retrieval against their negative effect on INSERT,UPDATE,DELETE

Basic Index concepts (contd..)

Determining when to use an index

When accessing less than 5 percent of the blocks of table,you want to use an index.If the selected values are distributed across many blocks of the table,you may get better performance from a full table scan than from an index based access.



Basic Index concepts (contd..)

Selectivity

- The selectivity of the index is the number of distinct values in the indexed column(s) to the number of records in the table.
- Number of distinct keys can be obtained from distinct_keys column in USER_INDEXES view.
- Selectivity of the index helps the CBO determine an execution path.
- The greater the selectivity,the better the index would be for returning small amounts of data.Highly selective column should be used as an index column.
- You can improve the selectivity by creating concatenated index.
- In case of concatenated index, the leading column should be the most selective column

Basic Index concepts (contd..)

Table trg_emp_sm has 80 rows and 16 distinct job codes.

- Determine the index selectivity manually before creating the index on job column.

  Hint: Manually take the count of records as well as distinct job codes from the table.

- Create an index on job column,analyze the table and then determine the selectivity of the index created.

  Hint: You can query user_indexes and user_tables.

- Also how would you determine the selectivity on other columns of this table.

  Hint: Use user_tab_columns.

Types Of Indexes

- **Unique Index**
- **Nonunique Index**
- **Composite Index**
- **Function-Based Index**

Basic Index concepts (contd..)

Fixing a bad index

- Bad indexes (Indexing the wrong columns) can cause as much trouble as forgetting to use indexes on the correct columns.
- Although CBO generally suppresses poor indexes,problems can still develop when a bad index is used at the same time as a good index.

Example 1

In PRODUCT table,there is a company_no column.Because this company's expansion has not occurred,all rows in the table have a company_no = 1.

What if you are a beginner who has heard that indexes are good and you have decided to index the company_no column?

Concatenated Index

Example2:
Consider a million row table (trg_emp).

Case 1: (No index)

```
select ename
from   trg_emp
where deptno = 10;
```

Elapsed time : 55 seconds (Full table scan)

Case 2: (Index on deptno column)

```
select  ename
from   trg_emp
where deptno = 10;
```

Elapsed time : 70 seconds (Index on deptno used)

Concatenated Index (contd..)

Case 3:

Drop the index on deptno column and create a concatenated index on (deptno,ename).

```
select  ename
from    trg_emp
where deptno = 10;
```

Elapsed time : Less than 1 second (Concatenated Index used)

Querying with concatenated index is much faster as the table itself did not have to be accessed to increase the query's speed. Indexing both the column in SELECT and WHERE clause allows the query to access only the index.

Multiple Indexes

- When multiple indexes on a single table are used within a query use the most restrictive index when you need to override an optimizer choice.
- Although Oracle's CBO generally forces the use of the most restrictive index,variations may occur based on the version.

Example3:

Case1:

Product table has index on product_id.

```
Select product_id,qty
from    trg_product
where  company_no = 1
and      product_id     = 167;
```

Elapsed time : 1 second (Index on product_id is used.1 record fetched)

Multiple Indexes (contd..)

Case2:

Create an index on company_no also.

```
Select product_id,qty
from    trg_product
where  company_no = 1
and      product_id     = 167;
```

Elapsed time : 725 seconds (Full table scan.1 record retrieved)

Rewrite the query to force the use of the correct index.

```
Select  /*+ INDEX( product prod_idx1) */ product_id,qty
from    trg_product
where  company_no = 1
and      product_id     = 167;
```

Elapsed time : 1 second (Index on product_id used.1 record retrieved)

Exercise

1.a) Run the following query with an index on company_no .
Check if the query is using the index or doing a full table scan.

```
SELECT  prod_id, prod_descr
FROM    trg_product
WHERE   company_code = 1;
```

b) Run the query again after dropping the index on company_code
c) Force the usage of index or full table scan by specifying appropriate optimizer hints.

Compare the performance(elapsed time) and also the execution plan in all the above cases.

2. Run the query on trg_emp table with and without composite index as mentioned in this section in example2.
Compare the results for both the scenarios.

Exercise(contd..)

3.Comparing mismatched data types

Run the following query and check if the index is being used.

There is an non-unique index on active_flag column.

```
SELECT  em_id,ename,age
FROM     trg_product
WHERE  active_flag = '1';
```

Try running the above by removing the single quotes and check if its performing index scan.

4. Run the query with multiple indexes on trg_product table as mentioned in example3 in this section and compare the performance of the query under different scenarios.

Session

Tuning SQL

Tips for writing effective sql

- **Avoid using * in SELECT clause**
- **Use TRUNCATE instead of DELETE,wherever possible**
- **Use WHERE in place of HAVING clause**
- **Use Table aliases**
- **Use EXISTS in place of IN and NOT EXISTS in place of NOT IN (wherever possible)**
- **Use joins in place of subquery**
- **Use bind variables.**
- **Use scalar sub queries in place of outer joins.**
- **Set arraysize to a larger value.**
- **Use function based index for queries having functions on indexed columns.**
- **Use inline view in place of complex joins**
- **Use static SQL in place of dynamic SQL (wherever possible)**

Tips for writing effective sql

USE IN

When you have index on BIG (larger table) table in the query.

USE EXISTS

When you have outer result set small and result set of subquery is large with appropriate index.

USE NOT IN:

a) When sub query or IN lists has no NULLS being returned.
b) When you have index on BIG table.

USE NOT EXISTS:

c)When outer query has no NULLS being returned.

Note:

NOT IN results in zero rows when the IN list has NULL values and NOT EXIST provides wrong result when the outer query has NULL

Tuning SQL (contd..)

Reduce the number of trips to database

The more you can reduce the number of database accesses, the more overhead you can save.

Example

Instead of having 2 different selects for fetching details of 2 employees , 342 and 291,we can have 1 select having a self join on emp table.

Least Efficient:

```
SELECT emp_name, salary, grade
FROM EMP
WHERE empno = 342;
SELECT emp_name, salary, grade
FROM EMP
WHERE empno = 291;
```

Tuning SQL (contd..)

Minimize table lookups in queries

Example:

Least Efficient :

```
SELECT tab_name  FROM tables
WHERE tab_name = (SELECT TAB_NAME
                  FROM TAB_COLUMNS
                  WHERE VERSION = 604)
AND DB_VER = (SELECT DB_VER
              FROM TAB_COLUMNS
              WHERE VERSION = 604)
```

Most Efficient :

```
SELECT TAB_NAME FROM TABLES
WHERE (TAB_NAME, DB_VER) = (SELECT TAB_NAME, DB_VER
                            FROM TAB_COLUMNS
                            WHERE VERSION = 604)
```

Tuning SQL (contd..)

Avoid calculation on Indexed columns

If the indexed column is a part of a function (in the WHERE clause), the optimizer does not use an index and will perform a full-table scan instead.

For example:

Least Efficient :

```
SELECT . . .
FROM DEPT
WHERE SAL * 12 > 25000;
```

Most Efficient :

```
SELECT . . .
FROM DEPT
WHERE SAL > 25000 / 12;
```

Tuning SQL (contd..)

Avoid NOT on Indexed columns

Example:

Least Efficient : (Here, index will not be used)

```
SELECT . . .
FROM DEPT
WHERE DEPT_CODE ! = 0;
```

Most Efficient : (Here, index will be used)

```
SELECT . . .
FROM DEPT
WHERE DEPT_CODE > 0;
```

Tuning SQL (contd..)

Use >= instead of >

If there is an index on DEPTNO, then try:

```
SELECT * FROM EMP
WHERE DEPTNO >= 4
```

Instead of

```
SELECT * FROM EMP
WHERE DEPTNO > 3
```

Because instead of looking in the index for the first row with column = 3 and then scanning forward for the first value that is > 3, the DBMS may jump directly to the first entry that is = 4.

Tuning SQL (contd..)

Use UNION in place of OR (In case of indexed columns)

Using OR on an indexed column causes the optimizer to perform a full-table scan rather than an indexed retrieval.

In the following example, both LOC_ID and REGION are indexed.

Query 1

```
SELECT LOC_ID, LOC_DESC, REGION
FROM LOCATION
WHERE LOC_ID = 10
OR REGION = 'MELBOURNE';
```

Tuning SQL (contd..)

Using UNION in place of OR :

Query2:

```
SELECT LOC_ID, LOC_DESC, REGION
FROM LOCATION
WHERE LOC_ID = 10
UNION
SELECT LOC_ID, LOC_DESC, REGION
FROM LOCATION
WHERE REGION = 'MELBOURNE'
```

Tuning SQL (contd..)

Use IN in place of OR

IN should be considered in place of OR condition

Example:

Least Efficient :

```
SELECT . . .
FROM LOCATION
WHERE LOC_ID = 10
OR LOC_ID = 20
OR LOC_ID = 30
```

Most Efficient :

```
SELECT . . .
FROM LOCATION
WHERE LOC_IN IN (10,20,30)
```

Tuning SQL (contd..)

Avoid IS NULL and IS NOT NULL on Indexed columns

For example:

Least Efficient : (Here, index will not be used)

```
SELECT . . .
FROM DEPARTMENT
WHERE DEPT_CODE IS NOT NULL;
```

Most Efficient : (Here, index will be used)

```
SELECT . . .
FROM DEPARTMENT
WHERE DEPT_CODE >= 0;
```

Tuning SQL (contd..)

Use UNION ALL in place OF UNION (where possible)

Replacing UNION with UNION ALL will improve the performance

```
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
      UNION   -- Change to UNION ALL
SELECT ACCT_NUM, BALANCE_AMT
FROM CREDIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
```

Tuning SQL (contd..)

Avoid converting Index Column Types

Assume that EMPNO is an **indexed numeric column.**

```
SELECT . . .
FROM EMP
WHERE EMPNO = '123'  --should be 123
```

In fact, because of conversion, this statement will actually be processed as:

```
SELECT . . .
FROM EMP
WHERE EMPNO = TO_NUMBER('123')
```

Tuning SQL (contd..)

Now assume that EMP_TYPE is an **indexed CHAR column.**

```
SELECT . . .
FROM EMP
WHERE EMP_TYPE = 123
```

This statement will actually be processed as:

```
SELECT . . .
FROM EMP
WHERE TO_NUMBER(EMP_TYPE) = 123
```

Indexes cannot be used, if they are included in a function. Therefore, this internal conversion will keep the index from not being used.

Tuning SQL (contd..)

Avoid using != or <>

Do Not Use:

```
SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT != 0;  --> will not use index
```

Use:

```
SELECT ACCOUNT_NAME
FROM TRANSACTION
WHERE AMOUNT > 0;
```

Tuning SQL (contd..)

Using Dates

Unless you are using function based indexes, using functions on indexed columns in the WHERE clause of a SQL statement causes the optimizer to bypass indexes.

Example

```
select  empno,ename,deptno
from    emp
where  trunc(hiredate) = '19-DEC-1978'; ( Full table scan )
```

```
select  empno,ename,deptno
 from    emp
where  hiredate  >=  '19-DEC-1978'
and     hiredate  < (to_date('19-DEC-1978') + 0.99999);
```

Tuning SQL (Contd..)

Exercise:

1. Modify this query to make use of index on hiredate column.

```
select  em_id,ename,deptno
from    trg_emp
where  trunc(hr_dt) = '01-JAN-2001';
```

2. a) Run the following query with and without parallel hint and then compare the execution plan and the elapsed time.

```
Select  e.em_id, e.deptno,e.ename,e.age,d.dname
from     trg_emp e ,trg_dept d
where   e.deptno = d.deptno;
```

Hint: Script utlxplp.sql can be used for parallel queries.

Tuning SQL (Contd..)

b) Create the trg_emp table with parallel option and run the query without parallel hint and check the execution plan.

3. Optimize this query:

```
SELECT  em_id,lst_nm
FROM    trg_emp
WHERE  SUBSTR(ename,1,2) = 'Jo';
```

Run explain plan for the above query and compare it against the tuned one.

4. Run the following query

```
Select count (*)
from    trg_tab2
where col1 in (select col1 from trg_tab1);
```

Modify the IN clause to EXISTS and check the performance with and without indexes on both the tables.

Tuning SQL (Contd..)

5. Run the following query.

   Create index on deptno column in emp_trg

```
SELECT  *
FROM    trg_emp
WHERE deptno IS NOT NULL;
```

   Optimize the above query and run it.

   Run explain plan for both the scenarios and compare the execution path and performance

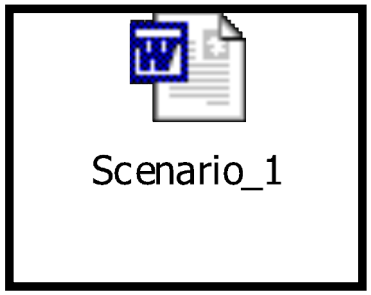
Generic Steps in SQL Tuning

Generic Steps to analyze Performance Bottleneck

- Check the Table Structure, Indexes available, etc.
- Follow best practices
- Check volume of data in all the tables involved
- Check Optimizer mode
- If CBO is used, analyze all related tables
- Check number of records returned and selectivity

Case Study

Scenario – 1



Scenario – 2:

One record is inserted into table A, which fires a trigger that inserts into 9 different tables (1 record each). Total time taken was 30-40 seconds.

Query: what all steps needs to be considered such that the insertion takes place faster?

Case Study contd..

Scenario – 3

A search query without any filter criteria which was running faster now takes 40 sec to 1 minute. But if we use RULE based optimizer, it was executing faster.

Query: what all steps needs to be considered such that the execution takes place faster?

MOCK Session

Summary

- Select the best indexing options for a variety of situations
- Tune crucial init.ora parameters for peak database performance
- Use Explain Plan, Trace, Tkprof and other tools
- Specify hints to override the optimizer as necessary.
- Use the Parallel Executions Option (PEO) for enhanced performance.
- Consider Parallel execution of batch processes.
- Analyze the tables and indexes and use the CBO
- Group related procedures into packages. It improves manageability as well as performance since the package is loaded at one shot.
- Consider dropping indexes before inserting huge data in to a table and recreate it after the insert.

References

Oracle9i Performance Tuning Tips and Techniques,
Richard J.Niemiec
Effective Oracle by Design , Thomas Kyte
Oracle SQL High-Performance Tuning , Guy Harrison
Expert one-on-one Oracle , Thomas Kyte
Oracle9i Performance Tuning Guide and Reference.