Assignment : 1

```
Select e.employee_id, e.first_name,
d.department_name  from Employees e, Departments d
where e.department_id = d.department_id
```

```
SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALL
+OUTLINE'));
```

ID	Operation	Name	Rows	Bytes	Cost	CPU	Time
0	SELECT STATEMENT			6	100		
1	TABLE ACCESS FULL	DEPARTMENTS	2	42	2		00:00:01
2	TABLE ACCESS FULL	EMPLOYEE	10	280	2		00:00:01

```
ALTER session set optimizer_mode=first_rows;
```

First_rows attempts to optimize the query to get the very first row back to the client as fast as possible.

The execution plan changes, the full table scans are replaced by index range scans. The first_rows access incurs additional I/O, but the index access ensures the fastest possible response time.

Assignment 2

```
ALTER SESSION SET SQL_TRACE = TRUE;
SELECT * FROM emp, dept WHERE emp.deptno = dept.deptno;
ALTER SESSION SET SQL_TRACE = FALSE;
```

OUTPUT->

```
SELECT * FROM emp, dept
WHERE emp.deptno = dept.deptno;

call   count   cpu   elapsed   disk    query  current  rows ----   -------
-------  ---------  --------  --------  -------   ------ Parse    1     0.16
0.29       3      13       0       0
Execute  1     0.00     0.00       0      0        0       0
Fetch    1     0.03      0.26       2     2        4      14

Misses in library cache during parse: 1
Parsing user id: (8) USER

Rows     Execution Plan
```

```
-------  ------------------------------------------------
14       MERGE JOIN
 4       SORT JOIN
 4       TABLE ACCESS (FULL) OF 'DEPT'
14       SORT JOIN
14       TABLE ACCESS (FULL) OF 'EMP'
```

For this statement, TKPROF output includes the following information:

```
TKPROF ora53269.trc ora53269.prf SORT = (PRSDSK, EXEDSK, FCHDSK) PRINT = 10
```

- The text of the SQL statement
- The SQL Trace statistics in tabular form
- The number of library cache misses for the parsing and execution of the statement.
- The user initially parsing the statement.
- The execution plan generated by EXPLAIN PLAN.

TKPROF also provides a summary of user level statements and recursive SQL calls for the trace file.

Assignment 3

Optimizer statistic on EMPLOYEE Table of HR schema with individual histograms on MANAGER_ID, SALARY columns

```
SELECT sal, count(*)
FROM   hr.emp_w
GROUP BY sal
ORDER BY 1;

 BEGIN  DBMS_STATS.GATHER_TABLE_STATS (
    ownname         => 'HR' ,
tabname         => 'emp_w'
,   method_opt       => 'FOR COLUMNS sal SIZE 10'
,   estimate_percent => 100
);
END;

SELECT TABLE_NAME, COLUMN_NAME, NUM_DISTINCT, HISTOGRAM
FROM   USER_TAB_COL_STATISTICS
WHERE  TABLE_NAME='emp_w'
AND    COLUMN_NAME='mgr';
*******************************************************
SELECT mgr, count(*)
FROM   hr.emp_w
GROUP BY mgr
ORDER BY 1;
```

```
 BEGIN  DBMS_STATS.GATHER_TABLE_STATS (
    ownname          => 'HR' ,
tabname          => 'emp_w'
,   method_opt       => 'FOR COLUMNS mgr SIZE 10'
,   estimate_percent => 100
);
END;
```

```
SELECT TABLE_NAME, COLUMN_NAME, NUM_DISTINCT, HISTOGRAM
FROM   USER_TAB_COL_STATISTICS
WHERE  TABLE_NAME='emp_w'
AND    COLUMN_NAME='mgr';
```

Optimizer statistic on EMPLOYEE Table of HR schema with extended histogram on (DEPARTMENT_ID, JOB_ID)

```
Select
DBMS_STATS.CREATE_EXTENDED_STATS(null,’EMPLOYEES’,’(DEPARTMENT_ID,JOB_ID)’) ;
```

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ( -ownname  => ‘HR’, -tabname => 'EMPLOYEES’, -estimate_percent => dbms_stats.auto_sample_size, -method_opt => 'for all columns size auto', -cascade => true, -degree  => 5 -)
```

Assignment 4

QUERY:

```
 create index unique_idx on EMPLOYEES;
  exec dbms_stats.gather_index_stats(‘unique_idx');
```

OUTPUT:

```
SELECT /*+ index (EMPLOYEES, unique_idx) */ * from EMPLOYEES where
MANAGER_ID = null;
```