CSE2007 DBMS LAB

SLOT: L39+L40
NAME - AMAN SAHU
REG. NO – 22BCE7224
EXPERIMENT NO.-7

1.

transactions

Trans_date	Transaction_id	account_no	name	cr_dr	amount	balance_amount
4/12/2018	12900999099	5694565556	Rakesh	cr	1000	10100
4/12/2018	12900999100	8965347594	John	cr	12000	14000
4/15/2018	12900999101	4938445044	Charles	dr	10000	60000
4/17/2018	12900999102	4834757489	Ravi	cr	5000	54000
4/23/2018	12900999103	4545048888	Sneha	cr	2000	22000
4/23/2018	12900999104	4938445044	Charles	cr	5000	65000
4/28/2018	12900999105	5694565556	Rakesh	dr	5000	5100
4/28/2018	12900999106	4545048888	Sneha	cr	4000	26000
4/28/2018	12900999107	8965347594	John	dr	1500	12500
4/28/2018	12900999108	4545048888	Sneha	dr	1000	25000
4/28/2018	12900999109	5694565556	Rakesh	cr	7000	12100
5/5/2018	12900999110	4545048888	Sneha	cr	2000	27000
5/5/2018	12900999111	4545048888	Sneha	cr	25000	52000
5/18/2018	12900999112	5694565556	Rakesh	cr	900	13000
5/18/2018	12900999113	4938445044	Charles	cr	10000	75000
5/18/2018	12900999114	4545048888	Sneha	dr	2000	50000

a. Select Top 5 Balanced accounts

```
SELECT * FROM (
  SELECT a.*, ROW_NUMBER() OVER (PARTITION BY account_no ORDER BY balance_amount DESC) rn
  FROM transactions a
) WHERE rn <= 5;
```

	TRANS_DATE	TRANSACTION_ID	ACCOUNT_NO	NAME	CR_DR	AMOUNT	BALANCE_AMOUNT
1	18-MAY-18	12900999113	5694565556	Charles	cr	10000	75000
2	23-APR-18	12900999104	9834545044	Charles	cr	5000	65000
3	15-APR-18	12900999101	9834545044	Charles	dr	10000	60000
4	17-APR-18	12900999102	4834757489	Ravi	cr	5000	54000
5	05-MAY-18	12900999111	4545048888	Sneha	cr	25000	52000

b. Select latest 5 transactions for each account

```
SELECT account_no, name, trans_date, transaction_id, cr_dr, amount, balance_amount
```

```
FROM (
  SELECT account_no, name, trans_date, transaction_id, cr_dr, amount, balance_amount,
      ROW_NUMBER() OVER (PARTITION BY account_no ORDER BY trans_date DESC) rn
  FROM transactions
)
WHERE rn <= 5
ORDER BY account_no, trans_date DESC;
```

	ACCOUNT_NO	NAME	TRANS_DATE	TRANSACTION_ID	CR_DR	AMOUNT	BALANCE_AMOUNT
1	4545048888	Sneha	18-MAY-18	12900999114	dr	2000	50000
2	4545048888	Sneha	05-MAY-18	12900999110	cr	2000	27000
3	4545048888	Sneha	05-MAY-18	12900999111	cr	25000	52000
4	4545048888	Sneha	28-APR-18	12900999106	cr	4000	26000
5	4545048888	Sneha	28-APR-18	12900999108	dr	1000	25000
6	4834757489	Ravi	17-APR-18	12900999102	cr	5000	54000
7	5694565556	Rakesh	18-MAY-18	12900999112	cr	900	13000
8	5694565556	Charles	18-MAY-18	12900999113	cr	10000	75000
9	5694565556	Rakesh	28-APR-18	12900999105	dr	5000	5100
10	5694565556	Rakesh	28-APR-18	12900999109	cr	7000	12100
11	5694565556	Rakesh	12-APR-18	12900999099	cr	1000	10100
12	6965347594	John	28-APR-18	12900999107	dr	1500	12500
13	6965347594	John	12-APR-18	12900999100	cr	12000	14000
14	9834545044	Charles	23-APR-18	12900999104	cr	5000	65000
15	9834545044	Charles	15-APR-18	12900999101	dr	10000	60000

c. Which is the top amount transaction in each day

```
SELECT Trans_date, MAX(amount) AS
top_amount
 FROM transactions
 GROUP BY Trans_date;
```

	TRANS_DATE	TOP_AMOUNT
1	15-APR-18	10000
2	28-APR-18	7000
3	23-APR-18	5000
4	18-MAY-18	10000
5	17-APR-18	5000
6	12-APR-18	12000
7	05-MAY-18	25000

2.

a. Create view on sailors table with all the attributes as sailors_v and perform DML operations. Note down your observations in base table.

CREATE VIEW sailors_u AS SELECT * FROM sailors;

select * from sailors_u;

	SID	SNAME	RATING	AGE
1	22	Dustin	7	45
2	29	Brutus	1	33
3	31	Lubber	8	55.5
4	32	Andy	8	25.5
5	58	Rusty	10	35
6	64	Hozatio	7	35
7	71	Zorba	10	16
8	74	Horatio	9	35
9	83	Art	3	25.5
10	95	Bob	3	63.5

b. Create view on sailors table with sname, rating, age attributes as sailors_v1 and perform DML operations. Justify your answer with proper observation on base table.

CREATE VIEW sailors_v1 AS SELECT sname, rating, age FROM sailors;

select * from sailors_v1;

	SNAME	RATING	AGE
1	Dustin	7	45
2	Brutus	1	33
3	Lubber	8	55.5
4	Andy	8	25.5
5	Rusty	10	35
6	Hozatio	7	35
7	Zorba	10	16
8	Horatio	9	35
9	Art	3	25.5
10	Bob	3	63.5

c. Create a view ‘sailor_boat’ to display the all details of sailors and boats who have reserved boats 103 or 104. Justify your answer about whether we able to apply DML operations on this view.

CREATE VIEW sailor_boats AS

SELECT s.*, b.*

FROM sailors s

JOIN reserves r ON s.sid = r.sid

JOIN boats b ON r.bid = b.bid

WHERE b.bid IN (103, 104);

select * from sailor_boats;

	SID	SNAME	RATING	AGE	BID	BNAME	COLOR
1	22	Dustin	7	45	103	Clipper	green
2	22	Dustin	7	45	104	Marine	red
3	31	Lubber	8	55.5	103	Clipper	green
4	31	Lubber	8	55.5	104	Marine	red
5	74	Horatio	9	35	103	Clipper	green

d. Drop all non-updatable views created by you.

DROP VIEW sailor_boats;

DROP VIEW sailors_v1;

DROP VIEW sailors_v;

3

a.Create a non-unique index on color attribute

of Boats.

CREATE INDEX idx_color ON boats(color);

	TRANS_DATE	TOP_AMOUNT
1	15-APR-18	10000
2	28-APR-18	7000
3	23-APR-18	5000
4	18-MAY-18	10000
5	17-APR-18	5000
6	12-APR-18	12000
7	05-MAY-18	25000

b.Create a function-based index on name column of employee table. Function to be used is *substr* on the 3rd and 4th characters.

CREATE INDEX idx_emp_name ON emp (SUBSTR(ename, 3, 4));
SELECT index_name, table_name, uniqueness
FROM user_indexes
WHERE table_name = 'EMP' AND index_name = 'IDX_EMP_NAME';

c. Create a composite index on department and salary attributes of employee.
CREATE INDEX idx_emp_dept_sal ON emp (deptno, sal);

SELECT index_name, table_name, uniqueness

FROM user_indexes

WHERE table_name = 'EMP' AND index_name =

'IDX_EMP_DEPT_SAL';

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	IDX_EMP_DEPT_SAL	EMP	NONUNIQUE

d. Refer the system table to check the status of created indexes.

SELECT index_name, table_name, status

FROM user_indexes;

	INDEX_NAME	TABLE_NAME	STATUS
1	SYS_C008794	CLASS	VALID
2	IDX_EMP_DEPT_SAL	EMP	VALID
3	IDX_NAME_SUBSTR	EMP	VALID
4	SYS_C008797	ENROLLED	VALID
5	SYS_C008792	FACULTY	VALID
6	SYS_C008767	STUDENT	VALID

4.

a. Copy the structure of employee table in to emp_v2 table.
CREATE TABLE emp_v2 AS SELECT * FROM emp WHERE 1=0;
SELECT * FROM EMP_V2;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

b. Use the **merge** statement to copy the contents of employee table into emp_v2.

MERGE INTO emp_v2 e

USING (SELECT * FROM emp) s

ON (e.empno = s.empno)

WHEN NOT MATCHED THEN

INSERT (e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm, e.deptno)

VALUES (s.empno, s.ename, s.job, s.mgr, s.hiredate, s.sal, s.comm, s.deptno);

SELECT * FROM emp_v2;

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7782	CLARK	MANAGER	7839	09-JUN-81	2450	(null)	10
2	7788	SCOTT	ANALYST	7566	19-APR-87	3000	(null)	20
3	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	(null)	30
4	7876	ADAMS	CLERK	7788	23-MAY-87	1100	(null)	20
5	7521	WARD	SALESMAN	7698	22-FEB-81	1250	(null)	30
6	7566	JONES	MANAGER	7839	02-APR-81	2975	(null)	20
7	7902	FORD	ANALYST	7566	03-DEC-81	3000	(null)	20
8	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	500	30
9	7900	JAMES	CLERK	7698	03-DEC-81	950	(null)	30
10	7369	SMITH	CLERK	7902	17-DEC-80	800	300	20
11	7934	MILLER	CLERK	7782	23-JAN-82	1300	(null)	10
12	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
13	7839	KING	PRESIDENT	(null)	17-NOV-81	5000	(null)	10
14	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

c. Change the commission value of an employee in employee table and update it in emp_v2 using merge statement

UPDATE emp SET comm = ... WHERE empno = ...;

MERGE INTO emp_v2 e

USING (SELECT * FROM emp) s

ON (e.empno = s.empno)

WHEN MATCHED THEN

UPDATE SET e.comm = s.comm;

SELECT * FROM emp_v2;

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7782	CLARK	MANAGER	7839	09-JUN-81	2450	(null)	10
2	7788	SCOTT	ANALYST	7566	19-APR-87	3000	(null)	20
3	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	(null)	30
4	7876	ADAMS	CLERK	7788	23-MAY-87	1100	(null)	20
5	7521	WARD	SALESMAN	7698	22-FEB-81	1250	(null)	30
6	7566	JONES	MANAGER	7839	02-APR-81	2975	(null)	20
7	7902	FORD	ANALYST	7566	03-DEC-81	3000	(null)	20
8	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	500	30
9	7900	JAMES	CLERK	7698	03-DEC-81	950	(null)	30
10	7369	SMITH	CLERK	7902	17-DEC-80	800	300	20
11	7934	MILLER	CLERK	7782	23-JAN-82	1300	(null)	10
12	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
13	7839	KING	PRESIDENT	(null)	17-NOV-81	5000	(null)	10
14	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

d. Delete a employee in source table and update it in emp_v2 using merge statement.

DELETE FROM emp WHERE empno = ...;

MERGE INTO emp_v2 e

USING (SELECT * FROM emp) s

ON (e.empno = s.empno)

WHEN NOT MATCHED THEN

DELETE;

SELECT * FROM emp_v2;

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7782	CLARK	MANAGER	7839	09-JUN-81	2450	(null)	10
2	7788	SCOTT	ANALYST	7566	19-APR-87	3000	(null)	20
3	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	(null)	30
4	7876	ADAMS	CLERK	7788	23-MAY-87	1100	(null)	20
5	7521	WARD	SALESMAN	7698	22-FEB-81	1250	(null)	30
6	7566	JONES	MANAGER	7839	02-APR-81	2975	(null)	20
7	7902	FORD	ANALYST	7566	03-DEC-81	3000	(null)	20
8	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	500	30
9	7900	JAMES	CLERK	7698	03-DEC-81	950	(null)	30
10	7369	SMITH	CLERK	7902	17-DEC-80	800	300	20
11	7934	MILLER	CLERK	7782	23-JAN-82	1300	(null)	10
12	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
13	7839	KING	PRESIDENT	(null)	17-NOV-81	5000	(null)	10
14	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

5.

a. Do self-study on ‘explain plan for’ command and write your observations about the output. Identify how indexing can improve the performance of query with query optimizer.

Ans. The EXPLAIN PLAN FOR command in Oracle SQL is a powerful tool for understanding the execution plan that the Oracle optimizer would use to execute a query. It provides a detailed breakdown of the operations the database would perform, such as table access or sort, the options describing the operation, the name of the database object acted upon by the step, and an estimated cost proportional to the expected resource use needed to execute the operation.

Indexing can significantly enhance the performance of a query. When an index is created on a column, the database can use the index to quickly locate the data without having to scan the entire table. This can result in a substantial performance boost, especially for large tables. The Oracle optimizer can leverage these indexes to create more efficient execution plans, reducing the time and resources required to execute queries.

b. Do self-study on how partitioning of real time databases helps in query processing? What are the different partitioning types? Write your observations with the examples.

Ans. Partitioning is a technique used in databases to divide a large table into smaller, more manageable pieces called partitions. Each partition is stored separately, leading to improved query performance because a query can scan a single partition instead of the entire table. This is particularly beneficial in real-time databases where performance and speed are critical.

There are several types of partitioning:

Range Partitioning: The data is partitioned according to a specified range. For example, a table that stores sales data could be partitioned by date, with each partition containing a month of data.

List Partitioning: The data is partitioned according to a list of values. This is useful when the partition key has a discrete set of known values.

Hash Partitioning: The data is partitioned using a hash function on the partition key. This ensures a more even distribution of data among partitions, which can lead to improved performance.

Composite Partitioning: This is a combination of other partitioning types, allowing for more complex partitioning schemes.

By partitioning the data, queries that filter by the partition key can run faster because they need to scan less data. This can significantly improve the performance of real-time databases. For example, a query that retrieves sales data for a specific month would only need to scan the partition for that month, rather than the entire sales table. This can lead to faster query execution and more efficient use of resources.