Assignment 4: Query and Transaction Processing EotW

Group 10 – Emerald

Kanchan Chowdhari

Payal Kaur

Purva Khandelwal

Samiksha Mhatre

1. Project Ideas: Query Processing
Idea 1: Optimizer Modes: Response Time versus Throughput

1) All_ROWS

It is the default optimizer mode; it gets all rows faster (generally forces index suppression). This is good for untuned, high-volume batch systems.

On executing the query, we see that the cost of query processing is 98.

```
ALTER SESSION SET OPTIMIZER_MODE = ALL_ROWS;

EXPLAIN PLAN FOR
SELECT patient_id,pfname, plname
FROM appointment_dtls
INNER JOIN patient_records USING(patient_id);

SELECT
    PLAN_TABLE_OUTPUT
FROM
    TABLE(DBMS_XPLAN.DISPLAY());
```

Script Output × | Query Result × | Query Result 1 ×

SQL | All Rows Fetched: 15 in 0.089 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 2133296934

-------------------------------------------------------------------------------------
| Id  | Operation          | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                  | 10000 |   224K|    98   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |                  | 10000 |   224K|    98   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| PATIENT_RECORDS  |  4135 | 78565 |    30   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| APPOINTMENT_DTLS | 10000 | 40000 |    68   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

2) FIRST_ROWS_10

It gets the first 10 rows faster. This is good for applications that routinely display partial results to users such as paging data to a user in a web application.

On executing the query, we see that the cost of processing has reduced from 98 to 12.

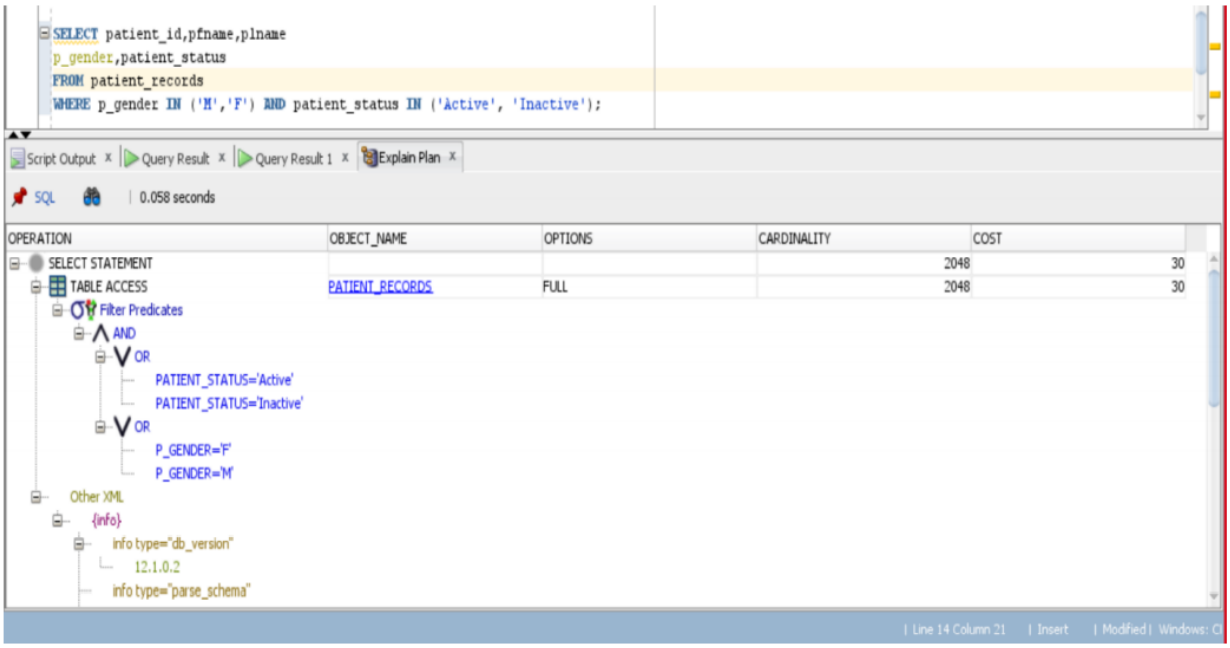
```
PLAN_TABLE_OUTPUT
Plan hash value: 2401842682

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |    10 |   230 |    12   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                    |    10 |   230 |    12   (0)| 00:00:01 |
|   2 |   NESTED LOOPS               |                    |    10 |   230 |    12   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL         | APPOINTMENT_DTLS   |    10 |    40 |     2   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | PATIENT_RECORDS_PK |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| PATIENT_RECORDS    |     1 |    19 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

2. Idea 2: Investigating Selectivity

To demonstrate this experiment, we have queried upon one of our Group project's table- PATIENT_RECORDS.

1. Let's display the Patient name, Patient ID, gender and patient status from Patient_Records table without indexing.

```
SELECT patient_id, pfname,plname,
p_gender,patient_status
FROM patient_records
WHERE p_gender IN ('M','F') AND patient_status IN ('Active','Inactive');
```



2. To check how bitmap indexing affects the performance of the query, we have created indexes on the gender and patient status columns.

   CREATE BITMAP INDEX pat_index ON patient_records (p_gender);
   CREATE BITMAP INDEX status_index

3. Now we get the explain plan for the below query wherein we have eliminated the columns like Patient first name, Patient last name, to get the indexing applied on indexed fields.

Script Output | Query Result | Explain Plan | Query Result 1

SQL | All Rows Fetched: 32 in 0.071 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 311294621

-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                    |  2048 | 30720 |    15   (0)| 00:00:01 |
|*  1 |  VIEW                          | index$ join$ 001   |  2048 | 30720 |    15   (0)| 00:00:0
|*  2 |   HASH JOIN                    |                    |       |       |            |          |
|*  3 |    HASH JOIN                   |                    |       |       |            |          |
|   4 |     INLIST ITERATOR            |                    |       |       |            |          |
|   5 |      BITMAP CONVERSION TO ROWIDS|                   |  2048 | 30720 |     2   (0)| 00:00:01 |
|*  6 |       BITMAP INDEX SINGLE VALUE | PSTATUS_INDX       |       |       |            |          |
|   7 |     INLIST ITERATOR            |                    |       |       |            |          |
|   8 |      BITMAP CONVERSION TO ROWIDS|                   |  2048 | 30720 |     2   (0)| 00:00:01 |
|*  9 |       BITMAP INDEX SINGLE VALUE | PAT_INDEX          |       |       |            |          |
|  10 |    INDEX FAST FULL SCAN        | PATIENT_RECORDS_PK |  2048 | 30720 |    11   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Conclusion-When there were no indexes, there was full access to the table for all the selected fields. After adding the bit map indexes for low cardinality columns like gender, status, we could only select these indexed fields to be able to implement bitmap indexing. Hence by shrinking the selective lookup to indexed fields thereby increased the performance by reducing the query cost from 30 to 15.

3. Project Ideas: Parallel Databases

Idea 1: Basic Parallel Execution

Parallel execution divides the task of executing an SQL statement into multiple small units, each of which is executed by a separate process. Parallel execution is designed to effectively use multiple CPUs and disks to answer queries quickly. When multiple users use parallel execution at the same time, it is easy to quickly exhaust available CPU, memory, and disk resources**.**

Degree of parallelism-The number of parallel execution servers associated with a single operation is known as the **degree of parallelism**.

1) Without Parallelism

Below is the query without any degree of parallelism.

Note- The below query is executed using the group project database.

```
SELECT COUNT(appointment_id),patient_id,pfname, plname
FROM appointment_dtls
INNER JOIN patient_records USING(patient_id)
INNER JOIN patient_treatment_mapping ptm USING(appointment_id)
INNER JOIN treatments t ON(ptm.treatment_id=t.treatement_id)
GROUP BY patient_id,pfname, plname
HAVING COUNT(appointment_id)>=3
ORDER BY COUNT(appointment_id) desc ;
```

```
PLAN_TABLE_OUTPUT
Plan hash value: 1913165403

-----------------------------------------------------------------------------------------------
| Id  | Operation              | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                            |   116 |  3944 |   122   (2)| 00:00:01 |
|   1 |  SORT ORDER BY         |                            |   116 |  3944 |   122   (2)| 00:00:01 |
|*  2 |   FILTER               |                            |       |       |            |          |
|   3 |    HASH GROUP BY       |                            |   116 |  3944 |   122   (2)| 00:00:01 |
|*  4 |     HASH JOIN          |                            |  3447 |   114K|   120   (0)| 00:00:01 |
|*  5 |      HASH JOIN         |                            |  3447 | 51705 |    90   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL| PATIENT_TREATMENT_MAPPING  |  3447 | 24129 |    22   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| APPOINTMENT_DTLS           | 10000 | 80000 |    68   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL | PATIENT_RECORDS            |  4135 | 78565 |    30   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
```

As seen from the above execution plan, the cost of execution for non-parallel process is 122.

2) Parallelism with DEGREE = 4

Below is the execution of above SQL query with degree = 4.

ALTER TABLE appointment_dtls **PARALLEL** 4;

PLAN_TABLE_OUTPUT

```
| Id  | Operation                          | Name                     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
---------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                          |   116 |  3944 |    73   (3)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR                    |                          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (ORDER)               | :TQ10005                 |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,05 | P->S | QC (ORDER) |
|   3 |    SORT ORDER BY                   |                          |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,05 | PCWP |            |
|   4 |     PX RECEIVE                     |                          |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,05 | PCWP |            |
|   5 |      PX SEND RANGE                 | :TQ10004                 |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,04 | P->P | RANGE      |
|*  6 |       FILTER                       |                          |       |       |            |          |  Q1,04 | PCWC |            |
|   7 |        HASH GROUP BY               |                          |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,04 | PCWP |            |
|   8 |         PX RECEIVE                 |                          |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,04 | PCWP |            |
|   9 |          PX SEND HASH              | :TQ10003                 |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,03 | P->P | HASH       |
|  10 |           HASH GROUP BY            |                          |   116 |  3944 |    73   (3)| 00:00:01 |  Q1,03 | PCWP |            |
|* 11 |            HASH JOIN               |                          |  3447 |  114K|    71   (0)| 00:00:01 |  Q1,03 | PCWP |            |
|  12 |             PX RECEIVE             |                          |  3447 | 51705 |    41   (0)| 00:00:01 |  Q1,03 | PCWP |            |
|  13 |              PX SEND HYBRID HASH   | :TQ10001                 |  3447 | 51705 |    41   (0)| 00:00:01 |  Q1,01 | P->P | HYBRID HASH|
|  14 |               STATISTICS COLLECTOR |                          |       |       |            |          |  Q1,01 | PCWC |            |
|* 15 |                HASH JOIN BUFFERED  |                          |  3447 | 51705 |    41   (0)| 00:00:01 |  Q1,01 | PCWP |            |
```

PLAN_TABLE_OUTPUT

```
|  16 |                 PX BLOCK ITERATOR  |                          | 10000 | 80000 |    19   (0)| 00:00:01 |  Q1,01 | PCWC |            |
|  17 |                  TABLE ACCESS FULL | APPOINTMENT_DTLS         | 10000 | 80000 |    19   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|  18 |                 PX RECEIVE         |                          |  3447 | 24129 |    22   (0)| 00:00:01 |  Q1,01 | PCWP |            |
|  19 |                  PX SEND BROADCAST | :TQ10000                 |  3447 | 24129 |    22   (0)| 00:00:01 |  Q1,00 | S->P | BROADCAST  |
|  20 |                   PX SELECTOR      |                          |       |       |            |          |  Q1,00 | SCWC |            |
|* 21 |                    TABLE ACCESS FULL| PATIENT_TREATMENT_MAPPING |  3447 | 24129 |    22   (0)| 00:00:01 |  Q1,00 | SCWP |            |
|  22 |              PX RECEIVE            |                          |  4135 | 78565 |    30   (0)| 00:00:01 |  Q1,03 | PCWP |            |
|  23 |               PX SEND HYBRID HASH  | :TQ10002                 |  4135 | 78565 |    30   (0)| 00:00:01 |  Q1,02 | S->P | HYBRID HASH|
|  24 |                PX SELECTOR         |                          |       |       |            |          |  Q1,02 | SCWC |            |
|  25 |                 TABLE ACCESS FULL  | PATIENT_RECORDS          |  4135 | 78565 |    30   (0)| 00:00:01 |  Q1,02 | SCWP |            |
---------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - filter(COUNT(SYS_OP_CSR(SYS_OP_MSR(COUNT("PTM"."APPOINTMENT_ID")),0))>=3)
  11 - access("APPOINTMENT_DTLS"."PATIENT_ID"="PATIENT_RECORDS"."PATIENT_ID")
  15 - access("APPOINTMENT_DTLS"."APPOINTMENT_ID"="PTM"."APPOINTMENT_ID")
```

SQL History | Messages - Log

From the above explain plan, we see that after setting the degree of parallelism to 4, the cost of execution drastically drops to 73.

4. Project Ideas: Transaction Processing
Idea 5: Transaction Control Language

TCL Statements available in Oracle are-

COMMIT : It **commits** the current **transaction**, making its changes permanent.
ROLLBACK: It rolls back the current **transaction**, canceling its changes.
SAVEPOINT : It is used to specify a point in transaction to which later you can rollback.

1) a) **DELETE without COMMIT**

In this case, we have deleted 4 records having user_id between 119 and 122 from user_master table in session 1

```
delete
from user_master
where user_id between 119 and 122;
```

Script Output | Query Result

Task completed in 0.079 seconds

```
1 row updated.

Rollback complete.

4 rows deleted.
```

b) **Reading 4 records without commit**.

In this case we have read the 4 deleted records without commit in session 2.

```
select * from user_master;
```

Script Output | Query Result

SQL | All Rows Fetched: 24 in 0.034 seconds

	USER_ID	USER_TYPE_ID	UFNAME	ULNAME	UCONTACT_NO	UEMAIL_ID	USER_ADDRESS	USER_STATUS	LICENSE_NO	QUALIFI
15	114	3	Olive	Oyl	8331514233	Olive .Oyl@gmail.com	4769 LEJO Street	ACTIVE	8791070	Assistar
16	115	3	Johnny	Bravo	8435036247	Johnny .Bravo@gmail.com	4929 GJCM Street	ACTIVE	57867605	Assistar
17	116	1	Kanchan	Chowdhari	8135930000	kanchu@gmail.com	GJXX STREET	ACTIVE	7689005	BACHELOI
18	117	4	Micky	Mouse	7835930000	mickymouse@gmail.com	PVXX STREET	INACTIVE	9889005	GRADUATI
19	118	6	Sam	Bhai	7935633141	sambhai@gmail.com	WWDU STREET	ACTIVE	9879005	BACHELOI
20	119	1	Payal	Ahluwalia	8135621111	payal@gmail.com	PKXX STREET	INACTIVE	7799005	BACHELOI
21	120	1	Megh	Vakharia	8135621111	megh@gmail.com	MVXX STREET	INACTIVE	7799005	BACHELOI
22	121	1	Samiksha	Vakharia	8135621111	sami@gmail.com	SVXX STREET	ACTIVE	7799005	BACHELOI
23	122	1	Sorv	Sharma	8135621111	sorv@gmail.com	STRXX STREET	ACTIVE	7799005	BACHELOI
24	123	1	Tyson	Singh	8135621999	tyson@gmail.com	TYRXX STREET	INACTIVE	7799005	BACHELOI

So, unless the commit operation is executed, the other sessions wont be able to see the reflected data with 4 records deleted.

c) **Commit records post deletion.**

Here, we have now committed the deletion operation in the session 1. After this the deleted records will now be reflected across all the sessions of different users. In other words, permanent changes are now done in the database table for other users to see the same copy of committed data.

```
select *
from user_master;


delete
from user_master
where user_id between 119 and 122;

commit;
```

Script Output | Query Result

Task completed in 0.04 seconds

```
4 rows deleted.


Commit complete.
```

d) **Reading the committed data.**

After having committed the data to the table, we read the user_master table in session 2. Now we see the copy of data excluding the deleted 4 records(user_id=119,120,121,122) as shown in the highlighted section.

```
select * from user_master;
```

Script Output | Query Result

SQL | All Rows Fetched: 20 in 0.024 seconds

	USER_ID	USER_TYPE_ID	UFNAME	ULNAME	UCONTACT_NO	UEMAIL_ID	USER_ADDRESS	USER_STATUS	LICENSE_NO	QUALIFI
11	110	3	Martin	Blank	8809912628	Martin .Blank@gmail.com	4790 QVFS Street	ACTIVE	53132024	Assistar
12	111	5	Harry	Potter	8979263570	Harry .Potter@gmail.com	4296 SFLU Street	INACTIVE	22858525	Staff
13	112	2	Katniss	Everdeen	8769549772	Katniss .Everdeen@gmail.com	4874 NTJV Street	ACTIVE	86939322	Bachelor
14	113	2	Shaggy	Rogers	8363546898	Kylie .Jenner@gmail.com	4720 FXYY Street	INACTIVE	65856303	Bachelor
15	114	3	Olive	Oyl	8331514233	Olive .Oyl@gmail.com	4769 LEJO Street	ACTIVE	8791070	Assistar
16	115	3	Johnny	Bravo	8435036247	Johnny .Bravo@gmail.com	4929 GJCM Street	ACTIVE	57867605	Assistar
17	116	1	Kanchan	Chowdhari	8135930000	kanchu@gmail.com	GJXX STREET	ACTIVE	7689005	BACHELOI
18	117	4	Micky	Mouse	7835930000	mickymouse@gmail.com	PVXX STREET	INACTIVE	9889005	GRADUATI
19	118	6	Sam	Bhai	7935633141	sambhai@gmail.com	WWDU STREET	ACTIVE	9879005	BACHELOI
20	123	1	Tyson	Singh	8135621999	tyson@gmail.com	TYRXX STREET	INACTIVE	7799005	BACHELOI

2) a) **UPDATE command without COMMIT**

Here, we have updated the qualification value to ‘BE-CS’ for user_id=123 without COMMIT.

```
update user_master
set qualification = 'BE-CS'
where user_id=123;
```

12	111	5	Harry	Potter	8979263570	Harry .Potter@gmail.com	4296	SFLU Street	INACTIVE	22858525	Staff
13	112	2	Katniss	Everdeen	8769549772	Katniss .Everdeen@gmail.com	4874	NTJV Street	ACTIVE	86939322	Bachelors of
14	113	2	Shaggy	Rogers	8363546898	Kylie .Jenner@gmail.com	4720	FXYY Street	INACTIVE	65856303	Bachelors of
15	114	3	Olive	Oyl	8331514233	Olive .Oyl@gmail.com	4769	LEJO Street	ACTIVE	8791070	Assistant
16	115	3	Johnny	Bravo	8435036247	Johnny .Bravo@gmail.com	4929	GJCM Street	ACTIVE	57867605	Assistant
17	116	1	Kanchan	Chowdhari	8135930000	kanchu@gmail.com	GJXX	STREET	ACTIVE	7689005	BACHELORS OF
18	117	4	Micky	Mouse	7835930000	mickymouse@gmail.com	PVXX	STREET	INACTIVE	9889005	GRADUATE
19	118	6	Sam	Bhai	7935633141	sambhai@gmail.com	WWDU	STREET	ACTIVE	9879005	BACHELORS OF
20	119	1	Payal	Ahluwalia	8135621111	payal@gmail.com	PKXX	STREET	INACTIVE	7799005	BACHELORS OF
21	120	1	Megh	Vakharia	8135621111	megh@gmail.com	MVXX	STREET	INACTIVE	7799005	BACHELORS OF
22	121	1	Samiksha	Vakharia	8135621111	sami@gmail.com	SVXX	STREET	ACTIVE	7799005	BACHELORS OF
23	122	1	Sorv	Sharma	8135621111	sorv@gmail.com	STRXX	STREET	ACTIVE	7799005	BACHELORS OF
24	123	1	Tyson	Singh	8135621999	tyson@gmail.com	TYRXX	STREET	INACTIVE	7799005	BE-CS

b) **ROLLBACK without COMMIT**

Now, we have rolled back the updated data to the previous qualification for user_id= 123.

17	116	1	Kanchan	Chowdhari	8135930000	kanchu@gmail.com	GJXX	STREET	ACTIVE	7689005	BACHELORS OF
18	117	4	Micky	Mouse	7835930000	mickymouse@gmail.com	PVXX	STREET	INACTIVE	9889005	GRADUATE
19	118	6	Sam	Bhai	7935633141	sambhai@gmail.com	WWDU	STREET	ACTIVE	9879005	BACHELORS OF
20	119	1	Payal	Ahluwalia	8135621111	payal@gmail.com	PKXX	STREET	INACTIVE	7799005	BACHELORS OF
21	120	1	Megh	Vakharia	8135621111	megh@gmail.com	MVXX	STREET	INACTIVE	7799005	BACHELORS OF
22	121	1	Samiksha	Vakharia	8135621111	sami@gmail.com	SVXX	STREET	ACTIVE	7799005	BACHELORS OF
23	122	1	Sorv	Sharma	8135621111	sorv@gmail.com	STRXX	STREET	ACTIVE	7799005	BACHELORS OF
24	123	1	Tyson	Singh	8135621999	tyson@gmail.com	TYRXX	STREET	INACTIVE	7799005	BACHELORS OF

This shows that, if we rollback the data without committing the data, it will revert all the transactions that happened in this session to the previous state.

3) **SAVEPOINT usecase-**

In this case,

1. we have created a table**,**

2. created a savepoint named test 1

3. inserted 1 row

4. created a savepoint named test 2

5. inserted 1 row

6. rolled back to savepoint test 2

Queries for the above points are shown below-

create table employee_test
(e_id number(5),
ename varchar2(20));

select * from employee_test order by e_id desc

SAVEPOINT test1

INSERT INTO employee_test (e_id,ename) values (1,'Denver') ;

SAVEPOINT test2

INSERT INTO employee_test (e_id,ename) values (2,'Ron') ;

ROLLBACK to test2

select * from employee_test order by e_id desc;

```
Table EMPLOYEE_TEST created.

Savepoint created.

1 row inserted.

Savepoint created.

1 row inserted.

Rollback complete.
```

Output after rollback savepoint test2

	E_ID	ENAME
1	1	Denver

In this case, we have observed that when we rollback to a particular savepoint test2, the transactions before this point are intact. It is only after this save point that the transactions are reverted.

5. Idea 3: Partitioned Tables

Partitioning of tables can improve the performance of data access substantially. To demonstrate this experiment, we are creating partitions for Beer table

```
CREATE TABLE BEERS_PART
  (BEER_ID NUMBER(10),
        BREWERY_ID NUMBER(10),
        BEER_NAME VARCHAR2(255),
        CAT_ID NUMBER(10),
        STYLE_ID NUMBER(10),
        ABV NUMBER(10,2),
        IBU NUMBER(10,2),
        SRM NUMBER(10,2),
        UPC NUMBER,
        LAST_MOD VARCHAR2(100),
        SRM_RAW NUMBER(10,2),
        ABV_W_NULLS VARCHAR2(20),
        REVIEW_ID NUMBER(10,0))
partition by range (BEER_ID)
(
partition block1 values less than (1000),
partition block2 values less than (2000),
partition block3 values less than (3000),
partition block4 values less than (4000),
partition block5 values less than (5000),
partition block6 values less than (6000));

INSERT INTO BEERS_PART
(BEER_ID,BREWERY_ID,BEER_NAME,CAT_ID,STYLE_ID,ABV,IBU,SRM,UPC,LAST_MOD,SR
M_RAW,ABV_W_NULLS,REVIEW_ID)
SELECT
BEER_ID,BREWERY_ID,BEER_NAME,CAT_ID,STYLE_ID,ABV,IBU,SRM,UPC,LAST_MOD,SR
M_RAW,ABV_W_NULLS,REVIEW_ID FROM BEERS;
```

As seen in below screenshot partition is created for the table.

```
ANALYZE TABLE BEERS_PART COMPUTE STATISTICS;

SELECT
  table_name,
  partition_name,
  num_rows,
  avg_row_len,
  last_analyzed
FROM user_tab_partitions
where table_name like 'BEER%';
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 6 in 0.107 seconds

	TABLE_NAME	PARTITION_NAME	NUM_ROWS	AVG_ROW_LEN	LAST_ANALYZED
1	BEERS_PART	BLOCK6	908	63	24-APR-21
2	BEERS_PART	BLOCK5	994	61	24-APR-21
3	BEERS_PART	BLOCK4	999	56	24-APR-21
4	BEERS_PART	BLOCK3	999	56	24-APR-21
5	BEERS_PART	BLOCK2	999	57	24-APR-21
6	BEERS_PART	BLOCK1	999	59	24-APR-21

Let us execute queries on original Beer table and Partitioned Beer table to see the improvement retrieval time.

Without Partition – We can see from the below screenshot that it takes 0.041 secs to retrieve records.

```
SELECT * FROM BEERS WHERE BEER_ID < 1000
and beer_name like '%Ale';
```

Script Output | Query Result 2

SQL | Fetched 50 rows in 0.041 seconds

	BEER_ID	BREWERY_ID	BEER_NAME	CAT_ID	STYLE_ID	ABV	IBU	SRM	UPC	LAST_MOD	SRM_RAW	ABV_W_NULLS	REVIEW_ID
1	182	810	Double India Pale Ale	3	26	8.6	0	(null)	0	7/22/10 20:00	(null)	8.6	(null)
2	184	397	Extra Strong Vintage Ale	-1	-1	7.5	0	(null)	0	7/22/10 20:00	(null)	7.5	(null)
3	185	481	Certified Organic India Pale Ale	3	26	7	0	(null)	0	7/22/10 20:00	(null)	7	(null)
4	187	481	Certified Organic Extra Pale Ale	-1	-1	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)
5	188	481	Certified Organic Amber Ale	3	33	4.8	0	(null)	0	7/22/10 20:00	(null)	4.8	(null)
6	189	706	Farm House Ale	-1	-1	6.7	0	(null)	0	7/22/10 20:00	(null)	6.7	(null)
7	193	923	NvÉ¬¿gne vÉÀù Brown Ale	3	38	4.5	0	(null)	0	7/22/10 20:00	(null)	4.5	(null)
8	195	1136	Ringwood Brewery Old Thumper Extra Special Ale	-1	-1	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)
9	196	1136	Export Ale	3	26	5.1	0	(null)	0	7/22/10 20:00	(null)	5.1	(null)
10	201	765	Undercover Investigation Shut-Down Ale	3	26	9.28	0	(null)	0	7/22/10 20:00	(null)	9.28	(null)
11	206	779	Sawtooth Ale	3	26	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)

```
SELECT * FROM BEERS WHERE BEER_ID < 1000
and beer_name like '%Ale';
```

ript Output | Query Result 2 | Explain Plan

SQL | 0.055 seconds

ATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			50	
TABLE ACCESS	BEERS	FULL	50	
Filter Predicates				
AND				
BEER_ID<1000				
BEER_NAME LIKE U'%Ale'				
Other XML				

With Partition – We can see from the below screenshot that it takes 0.031 secs to retrieve records and in the explain plan partition range is visible.

```
SELECT * FROM BEERS_PART WHERE BEER_ID < 1000
and beer_name like '%Ale';
```

pt Output | Explain Plan | Query Result

SQL | Fetched 50 rows in 0.031 seconds

	BEER_ID	BREWERY_ID	BEER_NAME	CAT_ID	STYLE_ID	ABV	IBU	SRM	UPC	LAST_MOD	SRM_RAW	ABV_W_NULLS	REVIEW_ID
1	182	810	Double India Pale Ale	3	26	8.6	0	(null)	0	7/22/10 20:00	(null)	8.6	(null)
2	184	397	Extra Strong Vintage Ale	-1	-1	7.5	0	(null)	0	7/22/10 20:00	(null)	7.5	(null)
3	185	481	Certified Organic India Pale Ale	3	26	7	0	(null)	0	7/22/10 20:00	(null)	7	(null)
4	187	481	Certified Organic Extra Pale Ale	-1	-1	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)
5	188	481	Certified Organic Amber Ale	3	33	4.8	0	(null)	0	7/22/10 20:00	(null)	4.8	(null)
6	189	706	Farm House Ale	-1	-1	6.7	0	(null)	0	7/22/10 20:00	(null)	6.7	(null)
7	193	923	NvÉ¬¿gne vÉÀú Brown Ale	3	38	4.5	0	(null)	0	7/22/10 20:00	(null)	4.5	(null)
8	195	1136	Ringwood Brewery Old Thumper Extra Special Ale	-1	-1	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)
9	196	1136	Export Ale	3	26	5.1	0	(null)	0	7/22/10 20:00	(null)	5.1	(null)
10	201	765	Undercover Investigation Shut-Down Ale	3	26	9.28	0	(null)	0	7/22/10 20:00	(null)	9.28	(null)
11	206	779	Sawtooth Ale	3	26	0	0	(null)	0	7/22/10 20:00	(null)	(null)	(null)

```
SELECT * FROM BEERS_PART WHERE BEER_ID < 1000
and beer_name like '%Ale';
```

Script Output | Explain Plan

SQL | 0.106 seconds

OPERATION	OBJECT_NAME	OPTIONS	PARTITION_START	PARTITION_STOP	PARTITION_ID	CAR
SELECT STATEMENT						
PARTITION RANGE		SINGLE	1	1	1	
TABLE ACCESS	BEERS_PART	FULL	1	1	2	
Filter Predicates						
BEER_NAME LIKE '%Ale'						