Database Tuning
CS5226
Lab 2

A0079953L – LEE ZHONG DE ROLLEI

TABLE OF CONTENT

Question 1: Index-Organized Tables

Start up the database with init.ora, and run setup.sql.

dbtune.comp.nus.edu.sg - PuTTY

```
a0079953@dbtune:~/lab/lab2[158]$ ls
cluster-setup.sql  q1create.sql       q1query.sql        q2query2.sql       rename.sh
init.ora           q1load.sql         q2query1.sql       q2setup.sql        setup.sql
a0079953@dbtune:~/lab/lab2[159]$ sh rename.sh
a0079953@dbtune:~/lab/lab2[160]$ sqlplus / as sysdba

SQL*Plus: Release 10.2.0.1.0 - Production on Wed Mar 13 10:02:48 2013

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Connected to an idle instance.

SQL> startupfile pfile=init.ora
SP2-0734: unknown command beginning "startupfil..." - rest of line ignored.
SQL> startup pfile = init.ora
ORACLE instance started.

Total System Global Area  721420288 bytes
Fixed Size                  2079720 bytes
Variable Size             180306968 bytes
Database Buffers          536870912 bytes
Redo Buffers                2162688 bytes
Database mounted.
@setup
Database opened.
SQL> @setup

User dropped.


User created.


Grant succeeded.

SQL>
User dropped.


User created.


Grant succeeded.

SQL> set linesize 300;
```

Run q1create.sql to create the table lab2user1.item

```
SQL> host more q1create.sql

create table lab2user1.item (
  item_id number,
  item_name varchar2(30),
  price number,
primary key (item_id));

SQL> @q1create

Table created.
```

Run q1load.sql to populate the table lab2user1.item.

```
SQL> host more q1load.sql

declare
  i number;
begin
  for i in 1 .. 10000 loop
    insert into lab2user1.item (item_id, item_name, price) values
    (i, 'Item ' || i, i * 0.5);
  end loop;
  commit;
end;
/


SQL> @q1load

PL/SQL procedure successfully completed.
```

Use the set autotrace command (with the appropriate option) to determine the estimated cost of running the query q1query.sql.

There are several parameters in the autotrace command (as below). However, since we are only interested in the execution cost and not the data nor the statistics, we will use this command “set autotrace traceonly explain” to display the execution plan.

set autotrace off set autotrace on set autotrace traceonly set autotrace on explain set autotrace on statistics set autotrace on explain statistics	set autotrace traceonly explain set autotrace traceonly statistics set autotrace traceonly explain statistics set autotrace off explain set autotrace off statistics set autotrace off explain statistics

```
SQL> set autotrace traceonly explain;
SQL> host more q1query.sql

select *
from lab2user1.item
where item_id between 200 and 500;

SQL> @q1query

Execution Plan
----------------------------------------------------------
Plan hash value: 3354656151

--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |   301 | 12943 |     9   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| ITEM |   301 | 12943 |     9   (0)| 00:00:01 |
--------------------------------------------------------------------------

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

   1 - filter("ITEM_ID">=200 AND "ITEM_ID"<=500)

Note
-----
   - dynamic sampling used for this statement
```

This plan shows the execution of a SELECT statement. The table was accessed using a full table scan. i.e.

- Every row in the table is accessed, and the WHERE clause criteria is evaluated for every row.
- The SELECT statement returns the rows meeting the WHERE clause criteria.

To reduce the running cost, we rebuild the table lab2user1.item as an index-organized table as follows:

```
SQL> host more q1create.sql

create table lab2user1.item (
  item_id number,
  item_name varchar2(30),
  price number,
primary key (item_id))ORGANIZATION INDEX;
```

Repeat the steps above to recreate and reload the data and subsequently examine the cost of execution after an organized index is created

```
SQL> @q1create

Table created.

SQL> @q1load

PL/SQL procedure successfully completed.

SQL> @q1query

Execution Plan
----------------------------------------------------------
Plan hash value: 4177117978

-------------------------------------------------------------------------------------
| Id  | Operation         | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                    |   301 | 12943 |     2   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| SYS_IOT_TOP_47838 |   301 | 12943 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("ITEM_ID">=200 AND "ITEM_ID"<=500)

Note
-----
   - dynamic sampling used for this statement
```

The cost of running q1query.sql has now been reduced. This plan shows execution of a SELECT statement.

- The organized Index (SYS_IOT_TOP_47838) is used in a range scan operation to evaluate the WHERE clause criteria.
- The SELECT statement returns rows satisfying the WHERE clause conditions.

Conclusion

An index-organized table (IOT) stores data in a B-Tree index structure. It also stores all the columns of the row and access the row data via the row's primary key .This makes scanning of data faster because once a search has located the key value all its corresponding row data will exists in the same location. This avoids the need to perform a look up in the table hence making the scanning more efficient. Also, there is no need to store duplicate key in the index and table (since both are in the same segment), hence this will help to save storage space.

IOT are useful when related pieces of data must be stored together or data must be physically stored in a specific order. i.e. if the query pattern is such that all column data must always be returned together or if the query pattern is such that the data must always be sorted in specific order. Tables that are queried by the key, not updated frequently and can fit well into a block are also good candidates for IOT

Question 2: Clusters

Run q2setup.sql.

dbtune.comp.nus.edu.sg - PuTTY

```
SQL> host more q2setup.sql

drop index lab2user1.personnel_index;
drop table lab2user1.person;
drop table lab2user1.person_detail;
drop cluster lab2user1.personnel;


create table lab2user1.person (
  id int,
  name varchar2(30),
  gender varchar2(1),
primary key (id));

declare
  i number;
  gen varchar2(1);
begin
  for i in 1 .. 10000 loop
    if mod(i,1000) = 0 then
      gen := 'M';
    else
      gen := 'F';
    end if;
    insert into lab2user1.person values (i, 'Name ' || i, gen);
  end loop;
  commit;
end;
/

create table lab2user1.person_detail (
  pid int,
  address varchar2(100),
  phone varchar2(10),
  id int,
primary key (pid));

declare
  i number;
  j number;
begin
  for i in 1 .. 25000 loop
        j := mod(i,5000);
        insert into lab2user1.person_detail values (i, 'Address ' || i, 'P ' || i, j);
  end loop;
```

```
SQL> @q2setup
drop index lab2user1.personnel_index
                     *
ERROR at line 1:
ORA-01418: specified index does not exist


drop table lab2user1.person
                     *
ERROR at line 1:
ORA-00942: table or view does not exist


drop table lab2user1.person_detail
                     *
ERROR at line 1:
ORA-00942: table or view does not exist


drop cluster lab2user1.personnel
*
ERROR at line 1:
ORA-00943: cluster does not exist



Table created.


PL/SQL procedure successfully completed.


Table created.


PL/SQL procedure successfully completed.
```

Determine the estimated cost of running the query q2query1.sql.
Determine the estimated cost of running the query q2query2.sql.

dbtune.comp.nus.edu.sg - PuTTY

```
SQL> @q2query1

Execution Plan
----------------------------------------------------------
Plan hash value: 836964621

------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |   101 | 11817 |    52   (4)| 00:00:01 |
|*  1 |  HASH JOIN          |               |   101 | 11817 |    52   (4)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| PERSON         |    99 |  3168 |     9   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| PERSON_DETAIL  |   283 | 24055 |    42   (3)| 00:00:01 |
------------------------------------------------------------------------------------

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

   1 - access("PERSON"."ID"="PERSON_DETAIL"."ID")
   2 - filter("PERSON"."ID"<100)
   3 - filter("PERSON_DETAIL"."ID"<100)

Note
-----
   - dynamic sampling used for this statement

SQL> @q2query2

Execution Plan
----------------------------------------------------------
Plan hash value: 3595854160

----------------------------------------------------------------------------
| Id  | Operation          | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |        |    99 |  3168 |     9   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PERSON  |    99 |  3168 |     9   (0)| 00:00:01 |
----------------------------------------------------------------------------

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

   1 - filter("PERSON"."ID"<100)

Note
-----
   - dynamic sampling used for this statement
```

Modify cluster-setup.sql to create a cluster and a cluster index for the two tables referenced in q2query1.sql using the cluster name ’lab2user1.personnel’.

dbtune.comp.nus.edu.sg - PuTTY

```
-- ---------------------------------------------------
-- insert your create cluster statements begin here
-- ---------------------------------------------------

DROP TABLE lab2user1.person;
DROP TABLE lab2user1.person_detail;


DROP INDEX lab2user1.personnel_index;
DROP CLUSTER lab2user1.personnel including tables;

CREATE CLUSTER lab2user1.personnel (id int);


CREATE TABLE lab2user1.person
(
  id int,
  name varchar2(30),
  gender varchar2(1),
  primary key (id)
)
CLUSTER lab2user1.personnel (id);


CREATE TABLE lab2user1.person_detail
(
  pid int,
  address varchar2(100),
  phone varchar2(10),
  id int,
        primary key (pid)
)
CLUSTER lab2user1.personnel (id);

COMMIT;

CREATE INDEX lab2user1.personnel_index ON CLUSTER lab2user1.personnel;


-- ---------------------------------------------------
-- insert your create cluster statements end here
-- ---------------------------------------------------
```

Execute modified cluster-setup.sql

```
SQL> @cluster-setup

Table dropped.


Table dropped.


Index dropped.


Cluster dropped.


Cluster created.


Table created.


Table created.


Commit complete.


Index created.


PL/SQL procedure successfully completed.


PL/SQL procedure successfully completed.
```

Rerun query 1 and 2 after the creation of cluster and add tables to the cluster created. The cost of the2 queries have then been reduced

```
dbtune.comp.nus.edu.sg - PuTTY

SQL> @q2query1

Execution Plan
----------------------------------------------------------
Plan hash value: 2399653129

--------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |  1162 |  132K|    10   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                  |  1162 |  132K|    10   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| PERSON           |    99 |  3168 |     9   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | SYS_C004577      |    99 |       |     2   (0)| 00:00:01 |
|   4 |   TABLE ACCESS CLUSTER       | PERSON_DETAIL    |    12 |  1020 |     1   (0)| 00:00:01 |
|*  5 |    INDEX UNIQUE SCAN         | PERSONNEL_INDEX |     1 |       |     0   (0)| 00:00:01 |
--------------------------------------------------------------------------------------------------

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

   3 - access("PERSON"."ID"<100)
   5 - access("PERSON"."ID"="PERSON_DETAIL"."ID")
       filter("PERSON_DETAIL"."ID"<100)

Note
-----
   - dynamic sampling used for this statement

SQL> @q2query2

Execution Plan
----------------------------------------------------------
Plan hash value: 709164903

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |    99 |  3168 |     9   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PERSON      |    99 |  3168 |     9   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | SYS_C004577 |    99 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   2 - access("PERSON"."ID"<100)
```

Conclusion

A table cluster is a group of tables that share common columns and store related data in the same blocks. When tables are clustered, a single data block can contain rows from multiple tables. In the example above, both tables person and person_detail are created in the same cluster “lab2user1.personnel”, therefore each oracle block will be able to store data from these 2 tables rather than from a single table.

The cluster key is the column or columns that the clustered tables have in common. For example, person and person_dtail shared the same id column. The cluster key has to be specified when creating the table cluster and when adding new table to the cluster

The cluster key value is the value of the cluster key columns for a particular set of rows. All data that contains the same cluster key value, such as id=20, is physically stored together. Each cluster key value is stored only once in the cluster and the cluster index, no matter how many rows of different tables contain the value.

We can consider clustering tables when they are primarily queried (but not modified) and records from the tables are frequently queried together or joined. Because table clusters store related rows of different tables in the same data blocks, properly used table clusters offer the following benefits over nonclustered tables:

- Disk I/O is reduced for joins of clustered tables.
- Access time improves for joins of clustered tables.
- Less storage is required to store related table and index data because the cluster key value is not stored repeatedly for each row.

Typically, clustering tables is not appropriate in the following situations:

- The tables are frequently updated.
- The tables frequently require a full table scan.
- The tables require truncating.