Optimization Exercises

Question 1

- How do you think the following query should be computed?
- What indexes would you suggest to use?

```
SELECT E.ename, D.mgr
FROM Employees E, Departments D
WHERE D.dname='Toy' AND
        E.dno=D.dno
```

Question 2

- How do you think the following query should be computed?
- What indexes would you suggest to use?

```
SELECT E.ename, D.mgr
FROM Employees E, Departments D
WHERE D.dname='Toy' AND
        E.dno=D.dno AND E.age = 25
```

Question 3

- How do you think the following query should be computed?
- What indexes would you suggest to use?
- Must both tables be accessed?

```
SELECT    D.mgr
FROM      Departments D, Employees E
WHERE     D.dno=E.dno
```

Question 4

- Emp(eid, age, sal, did)
- Dept(did, projid, budget, status)
- Proj(projid, code, report)
- Length of tuples, Number of tuples
  - Emp: 20 bytes, 20000 tuples
  - Dept: 40 bytes, 5000 tuples
  - Proj: 2000 bytes, 1000 tuples
- Pages contain 4000 bytes; 12 buffer pages

Question 4 (cont)

- Consider the queries:
  - find employees with age = 30
  - find projects with code = 20
- Assume that there are the same number of qualifying tuples in both.
- For which query is a *clustered* index more important?

Question 4 (cont)

- Consider the query: find employees with age > 30
- Assume that there is an unclustered index on age. Suppose that there are *N* tuples for which age > 30
- For which values of *N* is it cheaper to do a sequential scan?

Database Tuning

Database Tuning

- Problem: Make database run efficiently
- 80/20 Rule: 80% of the time, the database is running 20% of the queries/operations
  - find what is taking all the time, and tune these operations

Solutions

- Indexing
  - this can sometimes degrade performance. why?
- Tuning queries (using hints)
- Reorganization of tables; perhaps "denormalization"
- Changes in physical data storage

Denormalization

- Suppose you have tables:
  - emp(<u>eid</u>, ename, salary, did)
  - dept(<u>did</u>, budget, address, manager)
- Suppose you often ask queries which require finding the manager of an employee. You might consider changing the tables to:
  - emp(<u>eid</u>, ename, salary, did, manager)
  - dept(<u>did</u>, budget, address, manager)
- How will you ensure that the redundancy does not introduce errors into the database?

Creating Indexes Using Oracle

Creating an Index

- Syntax:

```
create [bitmap] [unique] index index on
        table(column [,column] . . .)
```

- Notes: Oracle has two kinds of indexes
  - b-tree
  - bitmaps
  - No hash indexes! (Why?)

Automatically Created Indexes

- When you create a table with a
  - primary key constraint *or*
  - unique constraint

  a "unique" index is created automatically

Bitmap Indexes

- Appropriate for columns that may have very few possible values
- For each value *c* that appears in the column, a vector *v* of bits is created, with a 1 in *v[i]* if the *i*-th row has the value *c*
- Oracle can automatically compute the RowIDs for the bitmap entries during query processing (so they do not have to be explicitly stored)

Bitmap Indexes: Example

Sid	Sname	age	rating
12	Jim	55	3
13	John	46	7
14	Jane	46	10
15	Sam	37	3

```
create bitmap index rating_bit on Sailors(rating);
```

- Corresponding bitmaps:
  - 3: <1 0 0 1>
  - 7: <0 1 0 0>
  - 10: <0 0 1 0>

Bitmap Indexes: Think About it

- How do you think that Oracle finds the RowIDs?
- What types of operations can be efficiently computed?
  - hint: aggregation functions
  - hint: logical conditions

Function-Based Indexes

- You can't use an index on sname for the following query:

```
select *
from Sailors
where UPPER(sname) = 'SAM';
```

- You can create a function-based index to speed up the query:

```
create index upp_sname on Sailors(UPPER( sname));
```

Index-Organized Tables

- An index organized table keeps its data sorted by the primary key
- They store their data as if they were an index

```
create table Sailors(
    sid number primary key,
    sname varchar2(30),
    age number,
    rating number)
    organization index;
```

Index-Organized Tables (2)

- What advantages does this have?
- What disadvantages?
- When to use?

Clustering Tables Together

- You can ask Oracle to store several tables close together on the disk
- This is useful if you usually query these tables together
- Note that the index created is actually a cluster index

Clustering Tables Together: Syntax

- create cluster sailor_reserves (X number);
- create table Sailors(
```
    sid number primary key,
    sname varchar2(30),
    age number,
    rating number)
    cluster sailor_reserves(sid);
```
- create index sailor_reserves_index on cluster sailor_reserves

Clustering Tables Together: Syntax (cont.)

- create table Reserves(
```
    sid number,
    bid number,
    day date,
    primary key(sid, bid, day) )
    cluster sailor_reserves(sid);
```
- *Note:* We end up getting a clustered index!

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Reserves

sid	bid	day
22	102	7/7/97
22	101	10/10/96
58	103	11/12/96

Stored

sid	sname	rating	age	bid	day
22	Dustin	7	45.0	102	7/7/97
				101	10/10/96
31	Lubber	8	55.5		
58	Rusty	10	35.0	103	11/12/96

The Oracle Optimizer

Types of Optimizers

- There are different modes for the optimizer
- RULE: Rule-based optimizer; picks a plan based on syntactic rules
- CHOOSE: Cost-based optimizer; picks a plan based on statistics about database contents
  - Need to analyze the data in the database for cost-based optimizer
- ALL: Optimize for total throughput
- FIRST_ROWS: Optimize for first rows

```
ALTER SESSION SET optimizer_mode =
      {choose|rule|first_rows(_n)|all_rows}
```

Analyzing the Data

- You can also use the following command:

```
analyze table | index
       <table_name> | <index_name>
compute statistics |
estimate statistics [sample <integer>
                      rows | percent] |
 delete statistics;
```

```
analyze table Sailors estimate statistics sample
25 percent;
```

Viewing the Execution Plan

- First, you need a PLAN table. So, the first time that you want to see execution plans, run the command:

```
@$ORACLE_HOME/rdbms/admin/utlxplan.sql
```

- Now, you can run: set autotrace on to see all plans

Viewing the Execution Plan (Option 2)

```
explain plan set statement_id= <name>
for <statement>
```

```
explain plan
set statement_id='test'
for
SELECT *
FROM Sailors S
WHERE sname='Joe';
```

Viewing the Execution Plan (Option 2, cont.)

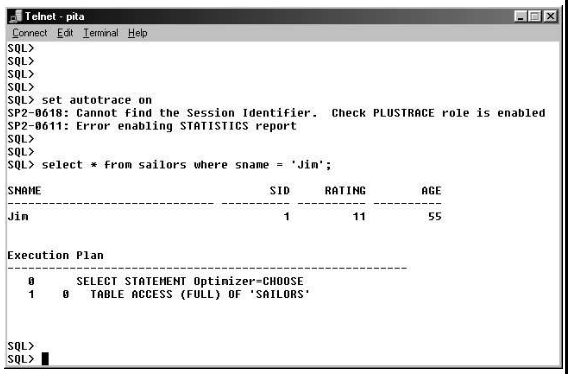
```
SELECT lpad(' ',2*Level)||Operation
||' '||Options||' '||Object_Name
Execution_Path
FROM Plan_Table
WHERE Statement_Id='test'
connect by prior
ID = parent_id and statement_id = 'test'
start with id = 1;
```

Operations that Access Tables

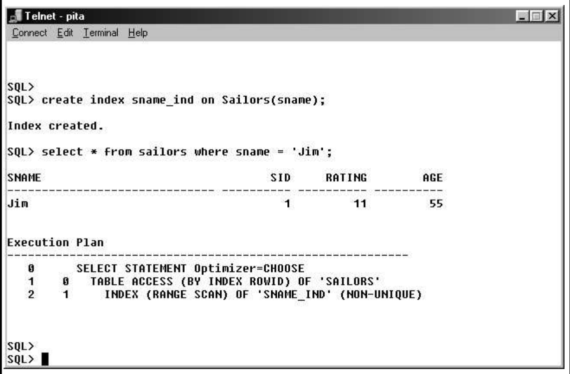
- TABLE ACCESS FULL: sequential table scan
- TABLE ACCESS BY ROWID: Given a rowid, you can access the table according to the rowid.
  - How do you get the rowid? From an index!

Operations that Use Indexes

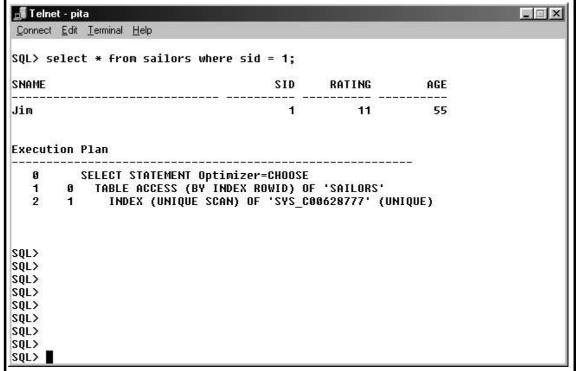
- INDEX UNIQUE SCAN: Access of an index that is defined to be unique
- INDEX RANGE SCAN: Access of an index that is not unique or access of a unique index for a range of values



```
SQL>
SQL>
SQL>
SQL>
SQL> set autotrace on
SP2-0618: Cannot find the Session Identifier.  Check PLUSTRACE role is enabled
SP2-0611: Error enabling STATISTICS report
SQL>
SQL>
SQL> select * from sailors where sname = 'Jim';

SNAME                                 SID     RATING        AGE
------------------------------ ---------- ---------- ----------
Jim                                     1         11         55


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'SAILORS'



SQL>
SQL>
```



```
SQL>
SQL> create index sname_ind on Sailors(sname);

Index created.

SQL> select * from sailors where sname = 'Jim';

SNAME                                 SID     RATING        AGE
------------------------------ ---------- ---------- ----------
Jim                                     1         11         55


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'SAILORS'
   2    1     INDEX (RANGE SCAN) OF 'SNAME_IND' (NON-UNIQUE)



SQL>
SQL>
```



```
SQL> select * from sailors where sid = 1;

SNAME                                 SID     RATING        AGE
------------------------------ ---------- ---------- ----------
Jim                                     1         11         55


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'SAILORS'
   2    1     INDEX (UNIQUE SCAN) OF 'SYS_C00628777' (UNIQUE)



SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
```

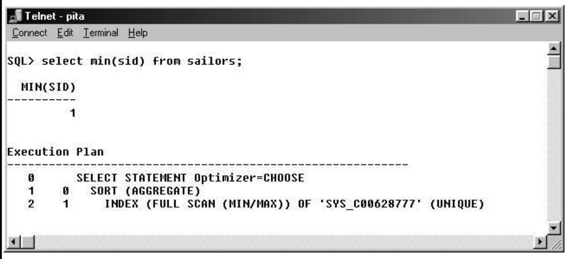
When are Indexes Used/Not Used?

- If you set an indexed column equal to a value, e.g., *sname = 'Jim'*
- If you specify a range of values for an indexed column, e.g., *sname like 'J%'*
  - *sname like '%m':* will not use an index
  - *UPPER(sname) like 'J%'* : will not use an index
  - *sname is null:* will not use an index, since null values are not stored in the index
  - *sname is not null:* will not use an index, since every value in the index would have to be accessed

When are Indexes Used? (cont)

- *sname != 'Jim':* Index will not be used. Why?
- MIN and MAX functions: Index will be used
- Equality of a column in a leading column of a multicolumn index. For example, suppose we have a multicolumn index on (sid, bid, day)
  - *sid = 12:* Can use the index
  - *bid = 101:* Cannot use the index
- Remember that indexes will only be used if they are selective

Is the table always accessed?



Telnet - pita

```
SQL> select min(sid) from sailors;

  MIN(SID)
----------
         1


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     INDEX (FULL SCAN (MIN/MAX)) OF 'SYS_C00628777' (UNIQUE)
```

Combining Output From Multiple Index Scans

- AND-EQUAL:
  - select * from sailors
    where sname = 'Jim' and rating = 10
- Suppose we have 2 indexes: sname, rating
  - Can search both and perform intersection
- Suppose we also have an index on (sname, rating)
  - How should the query be performed?

Combining Output From Multiple Index Scans

- CONCATENATION of scans:
  - select * from sailors
    where sname = 'Jim' or sname = 'Sam'

Operations that Manipulate Data Sets

- Up until now, all operations returned the rows as they were found
- There are operations that must find all rows before returning a single row
  - SORT ORDER BY: e.g., query with order by
  - SORT UNIQUE: e.g., query with distinct; query with minus, intersect or union (what about union all?)
  - SORT AGGREGATE, SORT GROUP BY: queries with aggregate functions or with group by functions

Operations that Manipulate Data Sets (cont.)

- Consider the query:
  - select sname from sailors
    union
    select bname from boats;

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (UNIQUE)
   2    1     UNION-ALL
   3    2       TABLE ACCESS (FULL) OF 'SAILORS'
   4    2       TABLE ACCESS (FULL) OF 'BOATS'
```

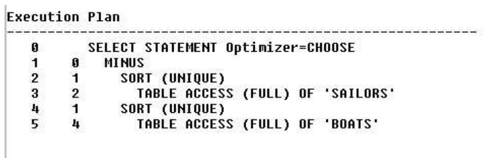
Operations that Manipulate Data Sets (cont.)

- Consider the query:
  - select sname from sailors

    minus

    select bname from boats;

How do you think that Oracle implements intersect? union all?



```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   MINUS
   2    1     SORT (UNIQUE)
   3    2       TABLE ACCESS (FULL) OF 'SAILORS'
   4    1     SORT (UNIQUE)
   5    4       TABLE ACCESS (FULL) OF 'BOATS'
```

Distinct

- What should Oracle do when processing the query (assuming that sid is the primary key):
  - select distinct sid

    from Sailors
- Does it matter if you write DISTINCT?

Join Methods

- Select * from Sailors, Reserves

  where Sailors.sid = Reserves.sid
- Oracle can use an index on Sailors.sid or on Reserves.sid (note that both will not be used)
- Join Methods: MERGE JOIN, NESTED LOOPS, HASH JOIN

Picking a Join Method

- MERGE JOIN: Performs badly when tables are of unequal size. Why?
- NESTED LOOPS: Perform well when tables are of unequal size (only used when there is an index on the join column of the inner table!)
- Appropriate if tables are small (so they fit in memory) or if results should be returned online. Why?

Hints

Hints

- You can give the optimizer hints about how to perform query evaluation
- Hints are written in /*+ */ right after the select
- Note: These are only hints. The oracle optimizer can choose to ignore your hints

Examples

```
Select /*+ FULL (sailors) */ sid
From sailors
Where sname= Joe ;
```

```
Select /*+ INDEX (sailors) */ sid
From sailors
Where sname= Joe ;
```

```
Select /*INDEX (sailors s_ind) */ sid
From sailors S, reserves R
Where S.sid=R.sid AND sname= Joe ;
```

More Examples

inner table

```
Select /*+ USE_NL (sailors) */ sid
From sailors S, reserves R
Where S.sid=R.sid AND sname= Joe ;
```

```
Select /*+ USE_MERGE (sailors, reserves) */
sid
From sailors S, reserves R
Where S.sid=R.sid AND sname= Joe ;
```

```
Select /*+ USE_HASH */ sid
From sailors S, reserves R
Where S.sid=R.sid AND sname= Joe ;
```