Ch11 DB Performance Optimization

Exercises (Please disregard question numbers – copy/paste error) ☺

Using Table 11.4 as an example, create two alternative access plans. Use the following assumptions:

a. There are 8,000 employees.
b. There are 4,150 female employees.
c. There are 370 employees in area code 615.
d. There are 190 female employees in area code 615.

TABLE 11.4

COMPARING ACCESS PLANS AND I/O COSTS

PLAN	STEP	OPERATION	I/O OPERATIONS	I/O COST	RESULTING SET ROWS	TOTAL I/O COST
A	A1	Cartesian product (PRODUCT, VENDOR)	7,000 + 300	7,300	2,100,000	7,300
	A2	Select rows in A1 with matching vendor codes	2,100,000	2,100,000	7,000	2,107,300
	A3	Select rows in A2 with V_STATE = 'FL'	7,000	7,000	1,000	**2,114,300**
B	B1	Select rows in VENDOR with V_STATE = 'FL'	300	300	10	300
	B2	Cartesian Product (PRODUCT, B1)	7,000 + 10	7,010	70,000	7,310
	B3	Select rows in B2 with matching vendor codes	70,000	70,000	1,000	**77,310**

Using Table 11.4 as an example, create two alternative access plans. Use the following assumptions:

a. There are 8,000 employees.
b. There are 4,150 female employees.
c. There are 370 employees in area code 615.
d. There are 190 female employees in area code 615.

The solution is shown in Table P11.3.

TABLE P11.3 COMPARING ACCESS PLANS AND I/O COSTS

Plan	Step	Operation	I/O Operations	I/O Cost	Resulting Set Rows	Total I/O Cost
A	A1	Full table scan EMPLOYE Select only rows with EMP_SEX='F' and EMP_AREACODE='615'	8,000	8,000	190	8,000
A	A2	SORT Operation	190	190	190	**8,190**
B	B1	Index Scan Range of EMP_NDX1	370	370	370	370
B	B2	Table Access by RowID EMPLOYEE	370	370	370	740
B	B3	Select only rows with EMP_SEX='F'	370	370	190	930
B	B4	SORT Operation	190	190	190	**1,120**

As you examine Table P11.3, note that in plan A the DBMS uses a full table scan of EMPLOYEE. The SORT operation is done to order the output by employee last name and first name. In Plan B, the DBMS uses an Index Scan Range of the EMP_NDX1 index to get the EMPLOYEE RowIDs. After the EMPLOYEE RowIDs have been retrieved, the DBMS uses those RowIDs to get the EMPLOYEE rows. Next, the DBMS selects only those rows with SEX = 'F.' Finally, the DBMS sorts the result set by employee last name and first name.

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1966;
```

What is the likely data sparsity of the EMP_DOB column?

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1966;
```

What is the likely data sparsity of the EMP_DOB column?

Because the EMP_DOB column stores employee's birthdays, this column is very likely to have high data sparsity.

```
SELECT      EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM        EMPLOYEE
WHERE       YEAR(EMP_DOB) = 1966;
```

Should you create an index on EMP_DOB? Why or why not?

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1966;
```

Should you create an index on EMP_DOB? Why or why not?

Creating an index in the EMP_DOB column would not help this query, because the query uses the YEAR function. However, if the same column is used for other queries, you may want to re-evaluate the decision not to create the index.

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1966;
```

What type of database I/O operations will likely be used by the query? (See Table 11.3.)

TABLE 11.3 Sample DBMS Access Plan I/O Operations

Operation	Description
Table Scan (Full)	Reads the entire table sequentially, from the first row to the last row, one row at a time (slowest)
Table Access (Row ID)	Reads a table row directly, using the row ID value (fastest)
Index Scan (Range)	Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan)
Index Access (Unique)	Used when a table has a unique index in a column
Nested Loop	Reads and compares a set of values to another set of values, using a nested loop style (slow)
Merge	Merges two data sets (slow)
Sort	Sorts a data set (slow)

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1966;
```

What type of database I/O operations will likely be used by the query? (See Table 11.3.)

This query more than likely uses a full table scan to read all rows of the EMPLYEE table and generate the required output. We have reproduced the table here to facilitate your discussion:

TABLE 11.3 Sample DBMS Access Plan I/O Operations

Operation	Description
Table Scan (Full)	Reads the entire table sequentially, from the first row to the last row, one row at a time (slowest)
Table Access (Row ID)	Reads a table row directly, using the row ID value (fastest)
Index Scan (Range)	Reads the index first to obtain the row IDs and then accesses the table rows directly (faster than a full table scan)
Index Access (Unique)	Used when a table has a unique index in a column
Nested Loop	Reads and compares a set of values to another set of values, using a nested loop style (slow)
Merge	Merges two data sets (slow)
Sort	Sorts a data set (slow)

```
SELECT      CUS_CODE, MAX(LINE_UNITS*LINE_PRICE)
FROM        CUSTOMER NATURAL JOIN INVOICE NATURAL JOIN LINE
WHERE       CUS_AREACODE = '615'
GROUP BY    CUS_CODE;
```

Assuming that you generate 15,000 invoices per month, what recommendation would you give the designer about the use of derived attributes?

```
SELECT      CUS_CODE, MAX(LINE_UNITS*LINE_PRICE)
FROM        CUSTOMER NATURAL JOIN INVOICE NATURAL JOIN LINE
WHERE       CUS_AREACODE = '615'
GROUP BY    CUS_CODE;
```

Assuming that you generate 15,000 invoices per month, what recommendation would you give the designer about the use of derived attributes?

This query uses the MAX aggregate function to compute the maximum invoice line value by customer. Because this table increases at a rate of 15,000 rows per month, the query would take considerable amount of time to run as the number of invoice rows increases. Furthermore, because the MAX aggregate function uses an expression (LINE_UNITS*LINE_PRICE) instead of a simple table column, the query optimizer is very likely to perform a full table scan in order to compute the maximum invoice line value. One way to speed up the query would be to store the derived attribute LINE_TOTAL in the LINE_TABLE and create an index on LINE_TOTAL. This way, the query would benefit by using the index to execute the query.

```
SELECT      CUS_CODE, MAX(LINE_UNITS*LINE_PRICE)
FROM        CUSTOMER NATURAL JOIN INVOICE NATURAL JOIN LINE
WHERE       CUS_AREACODE = '615'
GROUP BY    CUS_CODE;
```

Assuming that you follow the recommendations you gave in Problem 29, how would you rewrite the query?

```
SELECT      CUS_CODE, MAX(LINE_UNITS*LINE_PRICE)
FROM        CUSTOMER NATURAL JOIN INVOICE NATURAL JOIN LINE
WHERE       CUS_AREACODE = '615'
GROUP BY    CUS_CODE;
```

Assuming that you follow the recommendations you gave in Problem 29, how would you rewrite the query?

```
SELECT      CUS_CODE, MAX(LINE_TOTAL)
FROM        CUSTOMER NATURAL JOIN INVOICE NATURAL JOIN LINE
WHERE       CUS_AREACODE = '615'
GROUP BY    CUS_CODE;
```