Exercise 1

Text

The table on which queries have to be performed is:

FINE(IDFine, Date_f, Time, ViolationType, Numberplate)

with the following cardinalities:

. card(FINE)$\approx 10^5$ tuples;

. card(Date_f $\geq$ 1/1/2005 and Date_f $\leq$ 31/12/2005 FINE) $\approx 10^5$ tuples;

. card(ViolationType='Type10' FINE) $\approx 10^3$ tuples;

. card(ViolationType='Type50' FINE) $\approx 9 \cdot 10^4$ tuples.

Query 1

```
SELECT Date_f, COUNT(*)
FROM FINE
WHERE Date_f >=  1/1/2005 and Date_f <= 31/12/2005
GROUP BY Date_f;
```

Query 2

```
SELECT Date_f, COUNT(*)
FROM FINE
WHERE Date_f >= 1/10/2005 and Date_f <= 1/12/2005
and ViolationType = 'Type50'
GROUP BY Date_f;
```

Solution

Query 1

The algebraic expression is:

$$\pi_{Date\_f, COUNT(*)}$$
$$|$$
$$\text{GB}_{Date\_f}$$
$$|$$
$$\sigma_{Date\_f \geq 1/1/2005 \wedge Date\_f \leq 31/12/2005}$$
$$|$$
$$FINE$$

Now cardinalities are computed:

$$
\begin{array}{rl}
 & \pi_{Date\_f, COUNT(*)} \\
 & \quad | \qquad \approx 60 \\
\approx \left(\frac{1}{10^2}\right) & \mathrm{GB}_{Date\_f} \\
 & \quad | \qquad \approx 2 \cdot 10^4 \\
\left(\frac{1}{6}\right) & \sigma_{Date\_f \geq 1/1/2005 \,\wedge\, Date\_f \leq 31/12/2005} \\
 & \quad | \qquad \approx 10^5 \\
 & FINE
\end{array}
$$

where in red are marked cardinalities in each point and in blue reduction factors.

Reduction factors The reduction factor for the where close is compute in the following way: from 1/10/2005 to 31/12/2005 there are 2 months, so:

$$R_f = \frac{2}{12} = \frac{1}{6}$$

Then, in order to calculate the cardinality after the where close, is sufficient multiplicate the reduction factor with the cardinality of table FINE:

$$\frac{1}{6} \cdot 10^5 \approx 2 \cdot 10^4$$

The reduction factor for the group by close is more immediatly: since group by takes care only of different values of dates from 1/10/2005 to 31/12/2005 the reduction factor will be simply the number of days:

$$60\ days \approx 100 \quad \Longrightarrow \quad \approx \frac{1}{100}$$

Constraints In this section all constraints are discussed:

. a table is considered small if the number of tuples are less than 10^3;

. indices are required if the table is big and the attribute has an high selectivity in order to have significant reduction factor:

$$\frac{1}{10^N} \qquad N > 1$$

. for group by close:

Kinds of GROUP BY	Utilization
GB+SORT	small tables/when data is not already sorted
GB+NO SORT	index which sort data on the same attribute
GB+HASH	big tables
GB+NO HASH	hash index which sort data on the same attribute

Access path The access path without consider index is:

. FINE: TABLE ACCESS FULL + FILTER [‡]

. GB: HASH because the table is not small.

Although it is not necessary (the size of reduction factor is too low), index can be realized on the table FINE in two ways:

. SECONDARY B$^+$TREE on Date_f;

. PRIMARY because the query is based only on the field Date_f.

The access path will be:

. FINE: INDEX FAST SCAN/RANGE SCAN;

. GB: NO HASH because the table is not small but it is already sorted on the attribute join.

Query 2

$$\pi_{Date\_f, COUNT(*)}$$
$$|$$
$$\text{GB}_{Date\_f}$$
$$|$$
$$\sigma_p$$
$$|$$
$$FINE$$

where:

$p = Date\_f \geq 1/1/2005 \wedge Date\_f \leq 31/12/2005 \wedge ViolationType =' Type50'$

[‡] The filter is due to the WHERE close.

The order in which WHERE conditions are reported is very important for indices.

Now cardinalities are computed:

$$\begin{array}{lll} & \pi_{Date\_f, COUNT(*)} & \\ & | & \approx 60 \\ \approx \left(\frac{1}{10^2}\right) & \mathrm{GB}_{Date\_f} & \\ & | & \approx 10^4 \\ \left(\frac{3}{20}\right) \approx \left(\frac{1}{7}\right) & \sigma_p & \\ & | & \approx 10^5 \\ & FINE & \end{array}$$

Reduction factors The reduction factor for the WHERE close is computed in the following way: the condition on the field Date_f presents the same reduction factor already seen while the condition on ViolationType:

$$\frac{\approx 9 \cdot 10^4}{10^5} \frac{\text{(cardinality of tuples)}}{\text{(cardinality of table tuples)}} = \frac{9}{10}$$

So the total reduction factor is:

$$R_f = \frac{1}{6} \cdot \frac{9}{10} = \frac{3}{20}$$

The reduction factor for the GROUP BY close is the same previously computed.

Access path The access path without consider index is, again, the same already computed:

. FINE: TABLE ACCESS FULL + FILTER [‡]

. GB: HASH because the table is not small.

Also in this case index is not necessary: the reduction factor is lower than $\frac{1}{10}$. Indices can be realized on FINE as:

. SECONDARY B$^+$TREE on Date_f;

. SECONDARY HASH on ViolationType (HASH because it is a string);

. SECONDARY B$^+$TREE COMPOSED on Date_f, ViolationType;

. SECONDARY B$^+$TREE COMPOSED on ViolationType, Date_f.

[‡] The filter is due to the WHERE close.

The HASH index on ViolationType has not a good selectivity so will never be used; the best index is probably the first one and also the composed one on Date_f, ViolationType can be used: the order in which conditions are specified is very important so the second composed index can not be used because the first predicate presents a bad selectivity.

■

Exercise 2

Text

Startin from the following two tables:

STUDENT(StudentCode, Name, Surname, Birthdate)

EXAM(StudentCode, CourseExam, Date, Score)

with cardinalities:

. card(STUDENT) $\approx 10^4$ tuples;

. card(EXAM) $\approx 3 \cdot 10^5$ tuples;

. min{EXAM.*score*}=1;

. max{EXAM.*score*}=30;

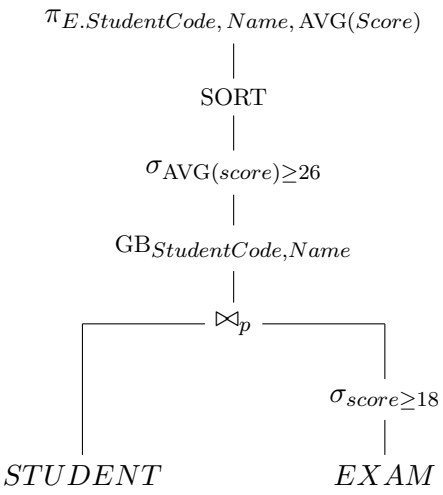
. selectivity HAVING AVG(*score*) $\geq$ 26 equal to $\frac{1}{50}$.

the query to be analized is:

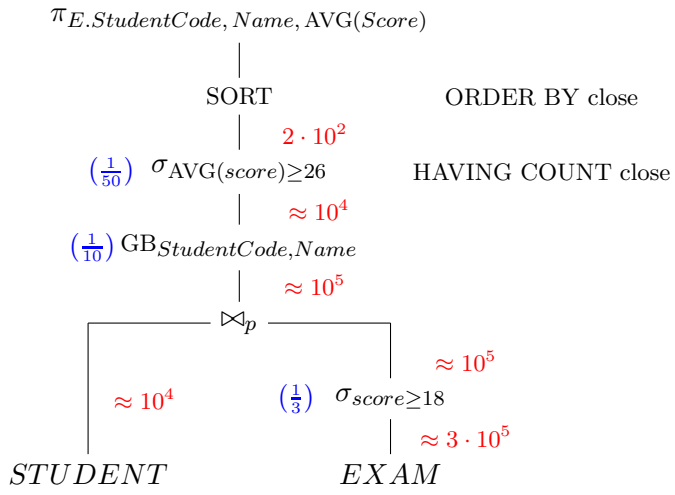
```
SELECT E.StudentCode, Name, AVG (Score)
FROM EXAM E, STUDENT S
WHERE E.StudentCode = S.StudentCode
and Score >= 18
GROUP BY E.StudentCode, Name
HAVING AVG (Score) >= 26
ORDER BY E.StudentCode;
```

Solution

The algebraic expression is:



where $p = E.StudentCode = S.StudentCode$. Now cardinalities are computed:



Reduction factors The reduction factor for the condition of the score in the WHERE close is computed in the following way: first is found how many exames have each student then the number of graduation greater than 18 are computed.

$$\frac{3 \cdot 10^5}{10^4} \frac{\#\text{ exams}}{\#\text{ students}} = 30 \;\#\text{ exams for each student}$$

$$\frac{30 - 18}{30} = \frac{12}{30} \approx \frac{1}{3}$$

The reduction factor for the HAVING COUNT CLOSE is a specification of the text.

After the join operation there are 10^5 tuples because the table which have to be considered is EXAMS: it contains all information needed by the following operations (GROUP BY, HAVING COUNT, AVG).

The GROUP BY close present a reduction factor of $\frac{1}{10}$ because, since there are $\approx 10^4$ distinct students in table STUDENT and after the join operation $\approx 10^5$ the reduction factor is easily to compute.

Access path Without considering indices:

. STUDENT: TABLE ACCESS FULL;

. EXAM: TABLE ACCESS FULL + FILTER [‡];

. JOIN:

  . nested loop choosing as *inner table* STUDENT because is smaller than EXAM which is the *outer table*; this approach can not be used because the inner table violate the constraint of size (10^3), so it can be considered a big table;

  . merge scan requires the sort of two tables and, since they are both big, it can not be used;

  . hash join is the method really used and it is a very good choice because there is a following operation like group by which can be improved (this is true only because both join and group by are performed over the same attribute);

. GROUP BY: NO HASH.

Considering indices, it is possible realize:

. on table EXAM:

  . SECONDARY B^+TREE on score: the reduction factor is only $\frac{1}{3}$ so selectivity is not so high;

  . SECONDARY B^+TREE on StudentCode: this attribute is the natural ordered in which the table STUDENT is create so it is not useful for table EXAM;

  . PRIMARY B^+TREE on StudentCode: useless for the same reason explained before;

[‡] The filter is due to the WHERE close.

. on table STUDENT:

  . PRIMARY B$^+$TREE on StudentCode: very convenient because is the natural ordered in which the table STUDENT is create and full;

  . SECONDARY B$^+$TREE on StudentCode: convenient if nested loop is used but it has a very reduced selectivity because each student code is different among students;

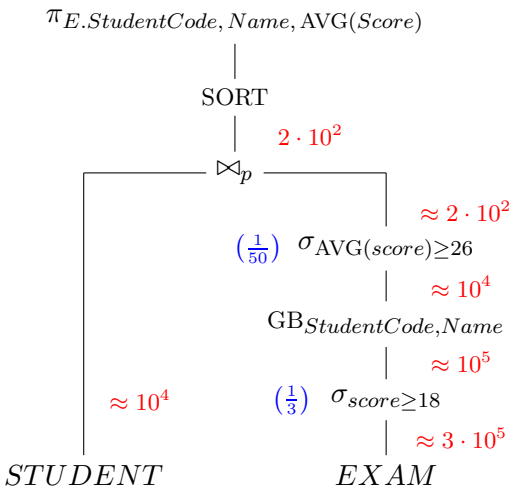
  . SECONDARY B$^+$TREE COMPOSED on StudentCode, Name: it can be used but it is very expensive to maintain.

So the access path becames:

. STUDENT: INDEX UNIQUE SCAN using the primary index;

. EXAM: TABLE ACCESS FULL + FILTER;

. it is possible anticipate the GROUP BY close and the HAVING COUNT close:



with this operation the join operation have to be performed over an amount of tuples very reduced ($\approx 2 \cdot 10^2$ instead $\approx 10^5$) so it is possible to use the nested loop algorithm choosing EXAM as the *inner table*; since the GROUP BY is anticipate, there is no point to perform it NO HASH because this operation is not improved by the previous JOIN realized with HASH algoritm, so GROUP BY: HASH.

■

Exercise 3

Text

The following tables are given:

BRANCH(<u>BCode</u>, BName, City, State)
CLERK(<u>SSN</u>, CName, CSurname, Role, BCode)
PROJECT(<u>PCode</u>, PName, StartDate, EndDate)
MEETING(<u>MCode</u>, PCode, Date, Room, StartTime, DurationInHours)
MEETING PARTICIPATION(<u>MCode</u>, <u>SSN</u>)

with the following cardinalities:

. card(BRANCH)$\approx 10^2$ tuples;

. number of states ≈ 10;

. card(CLERK) $\approx 10^3$ tuples;

. number of roles ≈ 10;

. card(PROJECT) $\approx 10^4$ tuples;

. min$\{StartDate\}$= 1-1-1979 and max$\{StartDate\}$= 31-12-2008;

. card(MEETING) $\approx 10^5$ tuples;

. min$\{Date\}$= 1-1-1979 and max$\{Date\}$= 31-12-2008;

. min$\{DurationInHours\}$= 2 and max$\{DurationInHours\}$= 6;

. card(MEETING PARTICIPATION) $\approx 5 \cdot 10^5$ tuples.

In addition the following reduction factors are given:

. for the query 1, *having count*`(*)`$> 10 \approx \frac{1}{10}$;

. for the query 2, *having sum*`(M.DurationInHours)`$> 500 \approx \frac{1}{500}$.

The two following queries have to be analized:

Query 1

```
SELECT C.CName, M.PCode, SUM (M.DurationInHours)
FROM MEETING M, CLERK C, MEETING PARTICIPATION MP
WHERE MP.MCode=M.MCode and MP.SSN=C.SSN
and M.Date$>=$1-1-1990 and M.Date$<=$31-12-1992
and (C.Role='Manager' or C.Role='Secreatary')
GROUP BY C.SSN, C.CName, M.PCode
HAVING COUNT (*) > 10
```

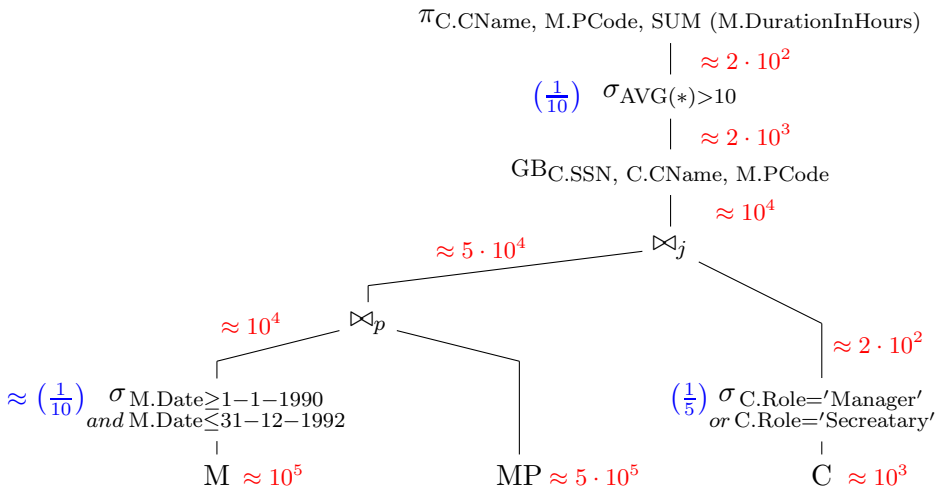
Query 2

```
SELECT C.CName
FROM CLERK C, BRANCH B
WHERE C.BCode=B.BCode and B.State='Italy'
AND C.SSN not in (SELECT MP.SSN
        FROM MEETING M, MEETING PARTECIPATION MP, PROJECT P
        WHERE MP.MCode=M.MCode and M.PCode=P.PCode
        and P.StartDate $>=$ 1-1-2006
        and M.Date $>=$ 1-1-2006 and M.Date $<=$ 31-12-2008
        GROUP BY MP.SSN
        HAVING SUM (M.DurationInHours) > 500)
```

Solution

Query 1

Algebraic expression and cardinality



where in red are highlighted cardinalities in each point and in blue reduction factors. For joins operations:

. condition j is MP.SSN=C.SSN;

. condition p is MP.MCode=M.MCode.

The number of tuples after join p is computed as:

$$(5 \cdot 10^5) \cdot \left(\frac{1}{10}\right) = 5 \cdot 10^4$$

while number of tuples after join j is:

$$(5 \cdot 10^4) \cdot \left(\frac{1}{5}\right) = 10^4$$

Reduction factors In order to compute reduction factor I assume that all data distributions are uniform.

The reduction factor for the condition on Date:

. from 1-1-1979 to 31-12-2008 there are 29 years;

. from 1-1-1990 to 31-12-1992 there are 3 years;

. the reduction factor is:

$$R_f = \frac{3}{29} \approx \frac{1}{10}$$

To compute the reduction factor for the condition on Role, I consider that there are 10 different roles and only two are considered:

$$R_f = \frac{2}{10} = \frac{1}{5}$$

The reduction factor for the HAVING close is specified by text and it is $\frac{1}{10}$. For GROUP BY close I have considered that since the principal condition it is on the fied SSN of table CLERK the number of tuples present after that operation will be exactly equal to the number of tuples of that table, so:

$$R_f = \frac{\approx 10^3}{\approx 2 \cdot 10^3} \left(\frac{\text{tuples after GB}}{\text{tuples before GB}} \right) = \frac{1}{2}$$

Execution plan without indices

. MEETING_PARTICIPATION: TABLE ACCESS FULL;

. MEETING: TABLE ACCESS FULL + FILTER (the filter is due to the WHERE close);

. CLERK: TABLE ACCESS FULL + FILTER (the filter is due to the WHERE close);

. JOIN (condition p): since the number of tuples for both tables is high *nested loop* and *merge scan* can not be used; algorithm implemented will be *hash*;

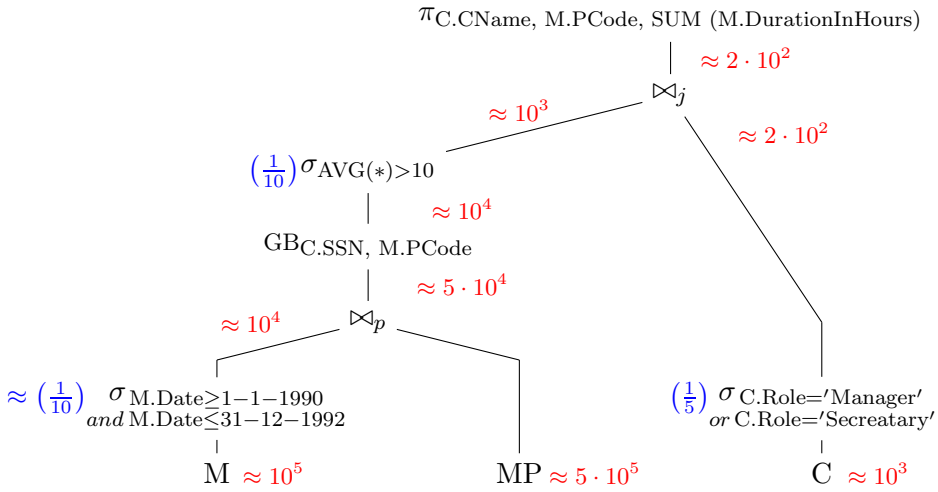
. JOIN (condition j):

  . now *nested loop* can be implemented (actually is that used) since tuples of CLERK table are less than the costraint (10^3) so that table can be the *inner table* and as *outer table* can be chosen the result of the previous JOIN;

  . *merge scan* is not a good choice because sort operation on CLERK is not expensive but on the result of previous JOIN is high since there are more than 10^3 tuples;

. *hash* can be used;

. GROUP BY: since the number of tuples is not too high and the attribute have to be sort this close will be GROUP BY+HASH; anticipate this operation is possible only because JOIN operations have this order and if the condition on CName is removed, otherwise some information is lost:



The number of tuples after join p is computed as before:

$$(5 \cdot 10^5) \cdot \left(\frac{1}{10}\right) = 5 \cdot 10^4$$

while number of tuples after join j is:

$$(10^3) \cdot \left(\frac{1}{5}\right) = 2 \cdot 10^2$$

Execution plan with indices

. on table CLERK index is not required because the number of tuples is 10^3 and the reduction factor is only $\frac{1}{5}$, but it is possible create a PRIMARY INDEX on attribute SSN because is the natural order in which tuples are inserted: this index helps JOIN j;

. on table MEETING_PARTICIPATION secondary index on both attribute can not be built because it is too expensive to maintain;

. on table MEETING: SECONDARY B$^+$TREE INDEX on Date because the predicate is a range predicate and that attribute has an high selectivity.

Access path with index

. MEETING_PARTICIPATION: TABLE ACCESS FULL;

. MEETING: INDEX RANGE SCAN+ACCESS BY ROW ID because the entire row has to be read in order to retrieve information needed;

. CLERK: TABLE ACCESS FULL + FILTER (or INDEX UNIQUE SCAN if the PRIMARY index is present);

. JOIN p: HASH;

. JOIN j: NESTED LOOP selecting as *inner table* CLERK;

. GROUP BY: HASH.

Query 2

Algebraic expression and cardinality From a logical point of view is possible divide the query into:

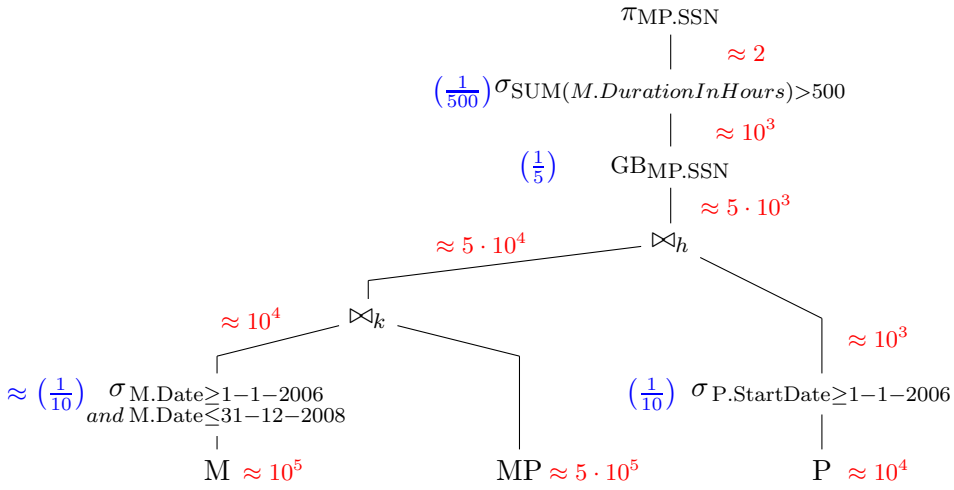
. inner query:

```
SELECT MP.SSN
FROM MEETING M, MEETING PARTECIPATION MP, PROJECT P
WHERE MP.MCode=M.MCode and M.PCode=P.PCode
AND P.StartDate $>=$ 1-1-2006
AND M.Date $>=$ 1-1-2006 AND M.Date $<=$ 31-12-2008
GROUP BY MP.SSN
HAVING SUM (M.DurationInHours) > 500
```

. outer query:

```
SELECT C.CName
FROM CLERK C, BRANCH B
WHERE C.BCode=B.BCode and B.State='Italy'
AND C.SSN not in (inner query)
```

The inner query presents the following algebra:



where:

. condition k is MP.MCode=M.MCode;

. condition h is M.PCode=P.PCode.

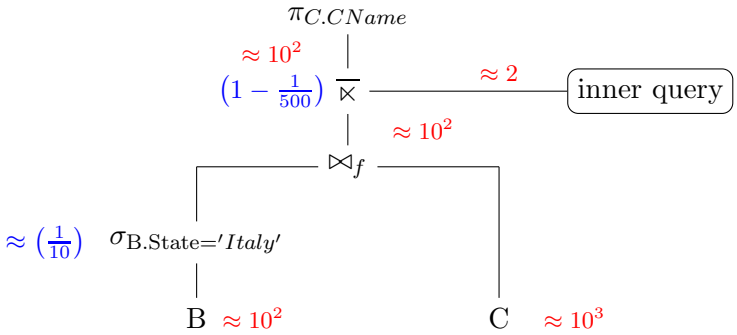
The number of tuples after join k is computed as:

$$(5 \cdot 10^5) \cdot \left(\frac{1}{10}\right) = 5 \cdot 10^4$$

while number of tuples after join h is:

$$(5 \cdot 10^4) \cdot \left(\frac{1}{10}\right) = 5 \cdot 10^3$$

For the outer query:



where f= C.BCode=B.BCode.

The number of tuples after join f is:

$$(10^3) \cdot \left(\frac{1}{10}\right) = 10^2$$

Reduction factors In order to compute reduction factor I assume that all data distributions are uniform.

Inner query Considering the inner query, the reduction factor for the condition on Date:

. from 1-1-1979 to 31-12-2008 there are 29 years;

. from 1-1-2006 to 31-12-2008 there are 3 years;

. the reduction factor is:

$$R_f = \frac{3}{29} \approx \frac{1}{10}$$

To compute the reduction factor for the condition on StartDate, I consider that:

. MIN (StartDate) is 1-1-1979 and MAX(StartDate) is 31-12-2008 so there are 29 years;

. from 1-1-2006 to 31-12-2008 there are 3 years;

. the reduction factor is:

$$R_f = \frac{3}{29} \approx \frac{1}{10}$$

For GROUP BY close I have considered that since there are 10 project for a single clerk, the number of tuples after this operation is $\approx 10^3$, so the reduction factor is:

$$R_f = \frac{10^3}{5 \cdot 10^3} \frac{\text{tuples after GB}}{\text{tuples before GB}} = \frac{1}{5}$$

The reduction factor for HAVING SUM close is already present in the exercise's text.

Outer query The outer query presents two reduction factors: the first one is on the condition about State. Since the number of states is ~ 10 and only one is selected:

$$R_f \approx \frac{1}{10}$$

The second reduction factor have to be computed for *not in* operation: first I consider the cardinality of the inner query (≈ 2) divided by the cardinality of table CLERK since information required is contained in that table:

$$R_f = \frac{2}{10^3} = \frac{1}{500}$$

This is the reduction factor for semi-join operation; the not is is obtained:

$$R_{f_{\text{not-in}}} = \left(1 - \frac{1}{500}\right) \quad \Longrightarrow \quad \approx 1^{\ddagger}$$

‡ This result is true if we consider negligible the second term.

Execution plan without indices

. MEETING_PARTICIPATION: TABLE ACCESS FULL;

. MEETING: TABLE ACCESS FULL + FILTER;

. PROJECT: TABLE ACCESS FULL + FILTER;

. CLERK: TABLE ACCESS FULL;

. BRANCH: TABLE ACCESS FULL + FILTER;

. JOIN (condition k): since the number of tuples for both tables is high *nested loop* and *merge scan* can not be used; algorithm implemented will be *hash*;

. JOIN (condition h): *nested loop* can be implemented since tuples of table PROJECT are the same number than the costraint (10^3) so that table can be the *inner table* and as *outer table* can be chosen the result of the previous JOIN; otherwise also *hash* algorithm can be realized;

. JOIN (condition f): since the number of tuples for both tables is low *nested loop* is the best choice selecting as *inner table* BRANCH which present the lower cardinality and as *outer* table CLERK;

. NOT IN operation can be performed with *nested loop* since the result of the inner query is lower than the constraint (10^3);

. GROUP BY: since the number of tuples is high and the attribute have to be sort this close will be GROUP BY+HASH; anticipate this operation is not possible because JOIN operation will not be performed since some information is lost.

Execution plan with indices

. on table CLERK index is not required because the number of tuples is too low;

. on table MEETING_PARTICIPATION secondary index on both attribute can not be built because it is too expensive to maintain;

. on table PROJECT: SECONDARY B^+TREE INDEX on StartDate because the predicate is a range predicate; also a PRIMARY B^3TREE INDEX on Pcode can be realized since it is the natural order of inserting new tuples: it help the JOIN h if it is implemented with *nested loop* because this table is the *inner table*;

. on table MEETING: SECONDARY B^+TREE INDEX on Date because the predicate is a range predicate;

. on table BRANCH: SECONDARY HASH INDEX on State because the predicate is an equal predicate: this should improve the nested loop because BRANCH is the *inner table* but the index can not be realized since the constraint on the size of the table is not respected.

Access path with index

. MEETING_PARTICIPATION: TABLE ACCESS FULL;

. MEETING: INDEX RANGE SCAN+ACCESS BY ROWID;

. PROJECT: INDEX RANGE SCAN+ACCESS BY ROWID;

. CLERK: TABLE ACCESS FULL;

. BRANCH: INDEX FAST SCAN;

■