HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 15 buffer pages the first scan of S splits it into 14 buckets, each containing about 72 pages, so again we have to deal with partition overflow. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit an entire partition of S in memory. The total cost will be the cost of two partioning phases plus the cost of one matching phase.

$$TotalCost = 2 * (2 * (M + N)) + (M + N) = 10,000$$

5. SORT-MERGE: With 52 buffer pages we have $B > \sqrt{M}$ so we can use the "merge-on-the-fly" refinement which costs $3 * (M + N)$.

$$TotalCost = 3 * (1,000 + 1,000) = 6,000$$

HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 52 buffer pages the first scan of S splits it into 51 buckets, each containing about 20 pages. This time we do not have to deal with partition overflow. The total cost will be the cost of one partioning phase plus the cost of one matching phase.

$$TotalCost = (2 * (M + N)) + (M + N) = 6,000$$

Exercise 12.6 Answer each of the questions—if some question is inapplicable, explain why—in Exercise 12.1 again, but using the following information about R and S:

Relation R contains 200,000 tuples and has 20 tuples per page.
Relation S contains 4,000,000 tuples and also has 20 tuples per page.
Attribute a of relation R is the primary key for R.
Each tuple of R joins with exactly 20 tuples of S.
1,002 buffer pages are available.

Answer 12.6 Let M = 10,000 be the number of pages in R, N = 200,000 be the number of pages in S, and B = 1002 be the number of buffer pages available.

1. Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be

$$\#pagesinouter + (\#pagesinouter * \#pagesininner)$$

which is minimized by having the smaller relation be the outer relation.

$$TotalCost = M + (M * N) = 2,000,010,000$$

The minimum number of buffer pages for this cost is 3.

2. This time read the outer relation in *blocks*, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are $\lceil \frac{\#pagesinouter}{B-2} \rceil$.

   $$TotalCost = M + N * \lceil \frac{M}{B-2} \rceil = 2,010,000$$

   The minimum number of buffer pages for this cost is 1002.

3. Since $B > \sqrt{N} > \sqrt{M}$ we can use the refinement to Sort-Merge discussed on pages 254-255 in the text.

   $$TotalCost = 3 * (M + N) = 630,000$$

   NOTE: if $R.a$ were not a key, then the merging phase could require more than one pass over one of the relations, making the cost of merging $M * N$ I/Os in the worst case.

   The minimum number of buffer pages required is 325. With 325 buffer pages, the initial sorting pass will split R into 16 runs of size 650 and split S into 308 runs of size 650 (approximately). These 324 runs can then be merged in one pass, with one page left over to be used as an output buffer. With fewer than 325 buffer pages the number of runs produced by the first pass over both relations would exceed the number of available pages, making a one-pass merge impossible.

4. The cost of Hash Join is $3*(M+N)$ if $B > \sqrt{f * M}$ where f is a 'fudge factor' used to capture the small increase in size involved in building a hash table, and M is the number of pages in the smaller relation, S (see page 258). Since $\sqrt{M} = 100$, we can assume that this condition is met. We will also assume uniform partitioning from our hash function.

   $$TotalCost = 3 * (M + N) = 630,000$$

   Without knowing f we can only approximate the minimum number of buffer pages required, and a good guess is that we need $B > \sqrt{f * M}$.

5. The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer.

   $$TotalCost = M + N = 210,000$$

   The minimum number of buffer pages for this cost is $M + 1 + 1 = 10,002$.

6. Any tuple in S can match at most one tuple in R because *R.a* is a primary key (which means the *R.a* field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in S, which is 4,000,000.

   The size of a tuple in the result could be as large as the size of an R tuple plus the size of an S tuple (minus the size of the shared attribute). This may allow only 10 tuples to be stored on a page. Storing 4,000,000 tuples at 10 per page would require 400,000 pages in the result.

7. If R.b is a foreign key referring to S.a, this contradicts the statement that each R tuple joins with exactly 20 S tuples.

Exercise 12.7 We described variations of the join operation called *outer joins* in Section 5.6.4. One approach to implementing an outer join operation is to first evaluate the corresponding (inner) join and then add additional tuples padded with *null* values to the result in accordance with the semantics of the given outer join operator. However, this requires us to compare the result of the inner join with the input relations to determine the additional tuples to be added. The cost of this comparison can be avoided by modifying the join algorithm to add these extra tuples to the result while input tuples are processed during the join. Consider the following join algorithms: *block nested loops join, index nested loops join, sort-merge join,* and *hash join.* Describe how you would modify each of these algorithms to compute the following operations on the Sailors and Reserves tables discussed in this chapter:

1. Sailors `NATURAL LEFT OUTER JOIN` Reserves
2. Sailors `NATURAL RIGHT OUTER JOIN` Reserves
3. Sailors `NATURAL FULL OUTER JOIN` Reserves

Answer 12.7 Answer not available.

2. Suppose that for each of the preceding selection conditions, you want to retrieve the average salary of qualifying tuples. For each selection condition, describe the least expensive evaluation method and state its cost.

3. Suppose that for each of the preceding selection conditions, you want to compute the average salary for each *age* group. For each selection condition, describe the least expensive evaluation method and state its cost.

4. Suppose that for each of the preceding selection conditions, you want to compute the average age for each *sal* level (i.e., group by *sal*). For each selection condition, describe the least expensive evaluation method and state its cost.

5. For each of the following selection conditions, describe the best evaluation method:

   (a) $sal > 200 \vee age = 20$

   (b) $sal > 200 \vee title = 'CFO'$

   (c) $title = 'CFO' \wedge ename = 'Joe'$

Answer 14.2 The answers are as follows.

1. For this problem, it will be assumed that each data page contains 20 relations per page.

   (a) $sal > 100$ For this condition, a filescan would probably be best, since a clustered index does not exist on sal. Using the unclustered index would accrue a cost of 10,000 pages * $\frac{20bytes}{100bytes}$ * 0.1 for the B+ index scan plus 10,000 pages * 20 tuples per page * 0.1 for the lookup = 22000, and would be inferior to the filescan cost of 10000.

   (b) $age = 25$ The clustered B+ tree index would be the best option here, with a cost of 2 (lookup) + 10000 pages * 0.1 (selectivity) + 10,000 * 0.2 (reduction) * 0.1 = 1202. Although the hash index has a lesser lookup time, the potential number of record lookups (10000 pages * 0.1 * 20 tuples per page = 20000) renders the clustered index more efficient.

   (c) $age > 20$ Again the clustered B+ tree index is the best of the options presented; the cost of this is 2 (lookup) + 10000 pages * 0.1 (selectivity)+ 200 = 1202.

   (d) $eid = 1000$ Since eid is a candiate key, one can assume that only one record will be in each bucket. Thus, the total cost is roughly 1.2 (lookup) + 1 (record access) which is 2 or 3.

   (e) $sal > 200 \wedge age > 30$ This query is similar to the $age > 20$ case if the $age > 30$ clause is examined first. Then, the cost is again 1202.

(f) $sal > 200 \wedge age = 20$ Similar to the previous part, the cost for this case using the clustered B+ index on $< age, sal >$ is smaller, since only 10 % of all relations fulfill $sal > 200$. Assuming a linear distribution of values for *sal* for *age*, one can assume a cost of 2 (lookup) + 10000 pages * 0.1 (selectivity for *age*) * 0.1 (selectivity for *sal*) + 10,000 * 0.4 * 0.1 * 0.1 = 142.

(g) $sal > 200 \wedge title = "CFO"$ In this case, the filescan is the best available method to use, with a cost of 10000. $sal > 200 \wedge age > 30 \wedge title = "CFO"$ Here, an age condition is present, so the clustered B+ tree index on $< age, sal >$ can be used. Here, the cost is 2 (lookup) + 10000 pages * 0.1 (selectivity) = 1002.

(h) $sal > 200 \wedge age > 30 \wedge title = "CFO"$ Similar to the case of $age > 20$; the best access path is again the clustered B+ tree on *age, sal*.

2. (a) $sal > 100$ Since the result desired is only the average salary, an index-only scan can be performed using the unclusterd B+ tree on *sal* for a cost of 2 (lookup) + 10000 * 0.1 * 0.2 (due to smaller index tuples) = 202.

(b) $age = 25$ For this case, the best option is to use the clustered index on $< age, sal >$, since it will avoid a relational lookup. The cost of this operation is 2 (B+ tree lookup) + 10000 * 0.1 * 0.4 (due to smaller index tuple sizes) = 402.

(c) $age > 20$ Similar to the $age = 25$ case, this will cost 402 using the clustered index.

(d) $eid = 1000$ Being a candiate key, only one relation matching this should exist. Thus, using the hash index again is the best option, for a cost of 1.2 (hash lookup) + 1 (relation retrieval) = 2.2.

(e) $sal > 200 \wedge age > 30$ Using the clustered B+ tree again as above is the best option, with a cost of 402.

(f) $sal > 200 \wedge age = 20$ Similarly to the $sal > 200 \wedge age = 20$ case in the previous problem, this selection should use the clustered B+ index for an index only scan, costing 2 (B+ lookup) + 10000 * 0.1 (selectivity for *age*) * 0.1 (selectivity for *sal*) * 0.4 (smaller tuple sizes, index-only scan) = 42.

(g) $sal > 200 \wedge title = "CFO"$ In this case, an index-only scan may not be used, and individual relations must be retrieved from the data pages. The cheapest method available is a simple filescan, with a cost of 10000 I/Os.

(h) $sal > 200 \wedge age > 30 \wedge title = "CFO"$ Since this query includes an age restriction, the clustered B+ index over $< age, sal >$ can be used; however, the inclusion of the title field precludes an index-only query. Thus, the cost will be 2 (B+ tree lookup) + 10000 * 0.1 (selectivity on *age*)+ 10,000 * 0.1 * 0.4 = 1402 I/Os.

3. (a) $sal > 100$ The best method in terms of I/O cost requires usage of the clustered B+ index over $< age, sal >$ in an index-only scan. Also, this assumes

the ablility to keep a running average for each age category. The total cost of this plan is 2 (lookup on B+ tree, find min entry) + 10000 * 0.4 (index-only scan) = 4002. Note that although *sal* is part of the key, since it is not a *prefix* of the key, the entire list of pages must be scanned.

(b) $age = 25$ Again, the best method is to use the clustered B+ index in an index-only scan. For this selection condition, this will cost 2 (*age* lookup in B+ tree) + 10000 pages * 0.1 (selectivity on *age*) * 0.4 (index-only scan, smaller tuples, more per page, etc.) = 2 + 400 = 402.

(c) $age > 20$ This selection uses the same method as the previous condition, the clustered B+ tree index over $< age, sal >$ in an index-only scan, for a total cost of 402.

(d) $eid = 1000$ As in previous questions, *eid* is a candidate field, and as such should have only one match for each equality condition. Thus, the hash index over *eid* should be the most cost effective method for selecting over this condition, costing 1.2 (hash lookup) + 1 (relation retrieval) = 2.2.

(e) $sal > 200 \wedge age > 30$ This can be done with the clustered B+ index and an index-only scan over the $< age, sal >$ fields. The total estimated cost is 2 (B+ lookup) + 10000 pages * 0.1 (selectivity on *age*) * 0.4 (index-only scan) = 402.

(f) $sal > 200 \wedge age = 20$ This is similar to the previous selection conditions, but even cheaper. Using the same index-only scan as before (the clustered B+ index over $< age, sal >$), the cost should be 2 + 10000 * 0.4 * 0.1 (*age* selectivity) * 0.1 (*sal* selectivity) = 42.

(g) $sal > 200 \wedge title = "CFO"$ Since the results must be grouped by age, a scan of the clustered $< age, sal >$ index, getting each result from the relation pages, should be the cheapest. This should cost 2 + 10000 * .4 + 10000 * tuples per page * 0.1 + 5000 * 0.1 (index scan cost) = 2 + 1000(4 + tuples per page). Assuming the previous number of tuples per page (20), the total cost would be 24002. Sorting the filescan alone, would cost 40000 I/Os. However, if the tuples per page is greater than 36, then sorting the filescan would be the best, with a cost of 40000 + 6000 (secondary scan, with the assumption that unneeded attributes of the relation have been discarded).

(h) $sal > 200 \wedge age > 30 \wedge title = "CFO"$ Using the clustered B+ tree over $< age, sal >$ one would accrue a cost of 2 + 10000 * 0.1 (selectivity of *age*) + 5000 * 0.1 = 1502 lookups.

4. (a) $sal > 100$ The best operation involves an external merge sort over $< sal, age >$, discarding unimportant attributes, followed by a binary search to locate minimum $sal < 100$ and a scan of the remainder of the sort. This costs a total of 16000 (sort) + 12 (binary search) + 10000 * 0.4 (smaller tuples) * 0.1 (selectivity of *sal*) + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.

   (b) $age = 25$ The most cost effective technique here employs sorting the clustered B+ index over $< age, sal >$, as the grouping requires that the output be sorted. An external merge sort with 11 buffer pages would require 16000. Totalled, the cost equals 16000 (sort) + 10000 * 0.4 = 20000.

   (c) $age > 20$ This selection criterion works similarly to the previous one, in that an external merge over $< age, sal >$ is required, using the clustered index provided as the pages to sort. The final cost is the same, 20000.

   (d) $eid = 1000$ Begin a candidate key, only one relation should match with a given *eid* value. Thus, the estimated cost should be 1.2 (hash lookup) + 1 (relation retrieval).

   (e) $sal > 200 \wedge age > 30$ This case is similar to the $sal > 100$ case above, cost = 16412.

   (f) $sal > 200 \wedge age = 20$ Again, this case is also similar to the $sal > 100$ case, cost = 16412.

   (g) $sal > 200 \wedge title = "CFO"$ The solution to this case greatly depends of the number of tuples per page. Assuming a small number of tuples per page, the cheapest route is to use the B+ tree index over *sal*, getting each index. The total cost for this is 2 (lookup, $sal > 200$) + 10000 * .2 (smaller size) * .1 (selectivity) + 10000 * .1 (selectivity) * tuples per page. The solution to this case is similar to that of the other requiring sorts, but at a higher cost. Since the sort can't be preformed over the clustered B+ tree in this case, the sort costs 40000 I/Os. Thus, for tuples per page ¡ 40, the B+ index method is superior, otherwise, the sort solution is cheaper.

   (h) $sal > 200 \wedge age > 30 \wedge title = "CFO"$ This solution is the same as the previous, since either the index over *sal* or an external sort must be used. The cost is the cheaper of 2 + 1000 * (.2 + tuples per page) [index method] and 40000 [sort method].

5. (a) $sal > 200 \vee age = 20$ In this case, a filescan would be the most cost effective, because the most cost effective method for satisfying $sal > 200$ alone is a filescan.

   (b) $sal > 200 \vee title = "CFO"$ Again a filescan is the better alternative here, since no index at all exists for *title*.

   (c) $title = "CFO" \wedge ename = "Joe"$ Even though this condition is a conjunction, the filescan is still the best method, since no indexes exist on either *title* or *ename*.

Exercise 14.3 For each of the following SQL queries, for each relation involved, list the attributes that must be examined in order to compute the answer. All queries refer to the following relations:

   (b) An unclustered index would preclude the low cost of the previous plan and necessitate the choice of a simple filescan, cost = 10000, as the best.

   (c) Due to the `WHERE` clause, the clustered B+ index on ename doesn't help at all. The best alternative is to use a filescan, cost = 10000.

   (d) Again, as in the previous answer, the best choice is a filescan, cost = 10000.

   (e) Although the order of the B+ index key makes the tree much less useful, the leaves can still be scanned in an index-only scan, and the increased number of tuples per page lowers the I/O cost. Cost = 10000 * .5 = 5000.

2. (a) A clustered index on *title* would allow scanning of only the 10% of the tuples desired. Thus the total cost is 2 (lookup) + 10000 * 10% + 2500 * 10%= 1252.

   (b) A clustered index on *dname* works functionally in the same manner as that in the previous question, for a cost 1002 + 250 = 1252 . The *ename* field still must be retrieved from the relation data pages.

   (c) In this case, using the index lowers the cost of the query slightly, due to the greater selectivity of the combined query and to the search key taking advantage of it. The total cost = 2 (look up) + 10000 * 5% + 5000 * 5% =752.

   (d) Although this index does contain the output field, the *dname* still must be retrieved from the relational data pages, for a cost of 2 (lookup) + 10000 * 10% + 5000 * 10%= 1502.

   (e) Since this index contains all three indexes needed for an index-only scan, the cost drops to 2 (look up) + 10000 * 5% * .75 (smaller size) = 402.

   (f) Finally, in this case, the prefix cannot be matched with the equality information in the `WHERE` clause, and thus a scan would be the superior method of retrieval. However, as the clustered B+ tree's index contains all the indexes needed for the query and has a smaller tuple, scanning the leaves of the B+ tree is the best plan, costing 10000 * .75 = 7500 I/Os.

3. (a) Since *title* is the only attribute required, an index-only scan could be performed, with a running counter. This would cost 10000 * .25 (index-only scan, smaller tuples) = 2500.

   (b) Again, as the index contains the only attribute of import, an index-only scan could again be performed, for a cost of 2500.

   (c) This index is useless for the given query, and thus requires a sorting of the file, costing 10000 + 3 * 2 * (2500). Finally, a scan of this sorted result will allow us to answer the query, for a cost of 27500.

   (d) This is similar to the previous part, except that the initial scan requires fewer I/Os if the leaves of the B+ tree are scanned instead of the data file. Cost = 5000 + 3 * 2 * (2500) = 22500.

(e) The clustered B+ index given contains all the information required to perform an index-only scan, at a cost of 10000 * .5 (tuple size) = 5000.

4. (a) Using a clustered B+ tree index on *title*, the cost of the given query is 10000 I/Os. The addition of another index would not lower the cost of any evaluation strategy that also utilizes the given index. However, the cost of the query is significantly cheaper if a clustered index on *dname*, *title* is available and is used by itself, and if added would reduce the cost of the best plan to 1500. (See below.)

   (b) The cheapest plan here involves simply sorting the file, at a cost of 10000 + 2 * 2 * (10000 *.25 (size reduction due to elimination of unwanted attributes; the selection can be checked on the fly and we only need to retail the *title* field)) = 20000.

   (c) The optimal plan with the indexes given involves scanning the *dname* index and sorting the (records consisting of the) *title* field of records that satisfy the `WHERE` condition. This would cost 2500 * 10 % [scanning relevant portion of index] + 10000 * 10% [retrieving qualifying records] + 10000 * 10% * .25 (reduction in size) [writing out *title* records] + 3 * 250 [sorting *title* records; result is not written out]. This is a total of 2250.

   (d) We can simply scan the relevant portion of the index, discard tuples that don't satisfy the `WHERE` condition, and write out the *title* fields of qualifying records. The *title* records must then be sorted. Cost = 5000 * 10% + 10000 * 10% * .25 + 3 * 250 = 1500.

   (e) A clustered index on *title*, *dname* supports an index-only scan costing 10000 * .5 = 5000.

Exercise 14.5 Consider the query $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$. Suppose that the projection routine is based on sorting and is smart enough to eliminate all but the desired attributes during the initial pass of the sort, and also to toss out duplicate tuples on-the-fly while sorting, thus eliminating two potential extra passes. Finally, assume that you know the following:

R is 10 pages long, and R tuples are 300 bytes long.
S is 100 pages long, and S tuples are 500 bytes long.
C is a key for S, and A is a key for R.
The page size is 1,024 bytes.
Each S tuple joins with exactly one R tuple.
The combined size of attributes A, B, C, and D is 450 bytes.
A and B are in R and have a combined size of 200 bytes; C and D are in S.

1. What is the cost of writing out the final result? (As usual, you should ignore this cost in answering subsequent questions.)

1.

$$\begin{aligned} TotalCost &= Shipping\ Departments\ Berlin \longrightarrow Naples = 5000t_s \\ &+ Cost\ of\ computing\ query\ at\ Naples = 3*(100,0000+5000)t_d \\ &+ Shipping\ result\ Naples \longrightarrow Delhi = N*t_s \\ &= 5000t_s + 315,000t_d + 10,000*t_s \\ &= 15,000t_s + 315,000t_d \end{aligned}$$

2.

$$\begin{aligned} Total\ Cost &= Shipping\ Employees\ Naples \longrightarrow Berlin = 100,000t_s \\ &+ Cost\ of\ computing\ query\ at\ Berlin = 3*(100,0000+5000)t_d \\ &+ Shipping\ result\ Berlin \longrightarrow Delhi = N*t_s \\ &= 100,000t_s + 315,000t_d + 10,000*t_s \\ &= 110,000t_s + 315,000t_d \end{aligned}$$

3.

$$\begin{aligned} Total\ Cost &= Shipping\ Employees\ Naples \longrightarrow Delhi = 100,000t_s \\ &+ Shipping\ Departments\ Berlin \longrightarrow Delhi = 5000t_s \\ &+ Cost\ of\ computing\ query\ at\ Delhi = 3*(100,0000+5000)t_d \\ &= 100,000t_s + 5000t_s + 315,000t_d \\ &= 105,000t_s + 315,000t_d \end{aligned}$$

4. We need to calculcate the cost of Bloomjoin at Naples.

   The plan is to calculate the bit-vector (corresponding to Employees) at Naples, then ship the bit-vector to Berlin, calculate the reduction of Departments at Berli, ship the reduction to Naples, calculate the join at Naples, and (finally!) ship the result to Delhi.

$$\begin{aligned} Total\ Cost &= Hashing\ Employees\ at\ Naples = 100,000t_d \\ &+ Shipping\ bit-vector\ Naples \longrightarrow Berlin = 5000t_s? \\ &+ Reduction\ of\ Departments\ at\ Berlin = 5000t_d \\ &+ Shipping\ Reduction\ of\ Departments\ Berlin \longrightarrow\ \ Naples = 5000t_s \\ &+ Computing\ join\ at\ Naples = 3*(100,000+5000)t_d \\ &+ Shipping\ result\ Naples \longrightarrow Delhi = N*t_s \\ &= 100,000t_d + 5000t_s? + 5000t_d + 5000t_s + 315,000t_d + 10,000t_s \\ &= 420,000t_d + 20,000t_s? \end{aligned}$$

5. We need to calculcate the cost of Bloomjoin at Berlin.

$$\begin{aligned} Total\ Cost &= Hashing\ Departments\ at\ Berlin = 5000t_d \\ &+ Shipping\ bit-vector\ Berlin \longrightarrow Naples = 250t_s? \end{aligned}$$

$$
\begin{aligned}
&+ \quad \textit{Reduction of Employeesat Naples} = 100,000t_d\\
&+ \quad \textit{Shipping Reduction of Employees Naples} \longrightarrow \ \textit{Berlin} = 100t_s?\\
&+ \quad \textit{Computing join at Berlin} = 3 * (5000 + 100)t_d?\\
&+ \quad \textit{Shipping result Berlin} \longrightarrow \textit{Delhi} = N * t_s\\
&+ \quad 5000t_d + 250t_s? + 100,000t_d + 100t_s? + 15,300t_d? + 10,000t_s\\
&+ \quad 120,300t_d + 10,350t_s??
\end{aligned}
$$

6. We need to calculcate the cost of Semijoin at Naples.

   The plan is to project the *eid* field of Employees at Naples, then ship the projection to Berlin, calculate the reduction of Departments w.r.t. Employees at Berlin, ship the reduction to Naples, calculate the join at Naples, and ship the result to Delhi.

   Let us assume the size of the *eid* field is 10 bytes. Cost of projecting the *eid* field is $100,000t_d$ for the scan of the Employees relation, and $50,000t_d$ for creating a temporary file. (Note the *eid* field is half the length of an Employess record.) If the optimizer to smart to recognize that *eid* is a key field, it will not try to eliminate duplicates. Else the projection will incur additional cost in sorting, and then scanning to eliminate duplicates. For our purposes, let us assume we have a smart optimizer.

$$
\begin{aligned}
\textit{Total Cost} \quad &= \quad \textit{Projecting Employees at Naples} = 150,000t_d\\
&+ \quad \textit{Shipping projection Naples} \longrightarrow \textit{Berlin} = 50,000t_s\\
&+ \quad \textit{Reduction of Departments at Berlin} = 3 * (50,000 + 5000)t_d\\
&+ \quad \textit{Shipping Reduction of Departments Berlin} \longrightarrow \ \textit{Naples} = 5000t_s\\
&+ \quad \textit{Computing join at Naples} = 3 * (100,000 + 5000)t_d\\
&+ \quad \textit{Shipping result Naples} \longrightarrow \textit{Delhi} = N * t_s\\
&= \quad 150,00t_d + 50,000t_s + 165,000t_d + 5000t_s + 315,000t_d + 10,000t_s\\
&= \quad 630,000t_d + 65,000t_s
\end{aligned}
$$

7. We need to calculcate the cost of Semijoin at Berlin.

   The plan is to project the *mgrid* field of Departments at Berlin, then ship the projection to Naples, calculate the reduction of Employeess w.r.t. Departments at Naples, ship the reduction to Berlin, calculate the join at Berlin, and ship the result to Delhi.

   Let us assume the size of the *mgrid* field is 10 bytes. Cost of projecting the *mgrid* field is $5000t_d$ for the scan of the Employees relation, and $2500t_d$ for creating a temporary file. (Note the *mgrid* field is half the length of a Departments record.) Now the *mgrid* field is not a key field, and we need to eliminate duplicates as part of the projection.

$$
\begin{aligned}
\textit{Total Cost} \quad &= \quad \textit{Projecting Departments at Berlin}\\
&+ \quad \textit{Shipping projection Berlin} \longrightarrow \textit{Naples} = 2500t_s\\
&+ \quad \textit{Reduction of Employees at Naples} = 3 * (100,000 + 2500)t_d\\
&+ \quad \textit{Shipping Reduction of Employees Naples} \longrightarrow \ \textit{Berlin} = 1000t_s\\
&+ \quad \textit{Computing join at Berlin} = 3 * (5000 + 1000)t_d
\end{aligned}
$$