Relational DBMS Internals
Solutions Manual

A. Albano, D. Colazzo, G. Ghelli and R. Orsini

February 10, 2015

CONTENTS

Chapter 2

PERMANENT MEMORY AND BUFFER MANAGEMENT

Exercise 2.1 A disk has the following characteristics:

– bytes per sector (bytes/sector) = 512
– sectors per track (sectors/track) = 50
– tracks per surface (tracks/surface) = 2000
– number of platters = 5
– rotation speed = 5400 rpm (rotations/minutes)
– average seek time = 10 ms

Calculate the following parameters.

1. Tracks capacity (bytes), a surface capacity, total capacity of the disk.
2. Number of disk cylinders.
3. Average rotational latency.
4. Average transfer time of a block of 4096 bytes.

256, 2048 and 51 200 are examples of valid block sizes?

Answer 2.1

1. bytes/track = bytes/sector $\times$ sectors/track = 512 $\times$ 50 = 25 600
   bytes/surface = bytes/track $\times$ tracks/surface = 25 600 $\times$ 2000 = 51,2M.
   bytes/disk = bytes/surface $\times$ surfaces/disk = 51,2M $\times$ 10 = 512M.
2. Number of cylinders = tracks per surface = 2000.
3. Average rotational latency = $1/2$ rotation time = $1/2 \times (1/5400) \times 60$ = 0,006 sec.
4. Average transfer time of a track = 25 600/0,011 = 2 327 272 bytes/sec.
   Average transfer time of a block of 4096 bytes = 4096/2 327 272 = 1,76 ms.

The size of a block must be a multiple of the sector size and less than a track size. 256 and 51 200 are not valid block sizes.

Exercise 2.2 Consider the disk of the previous exercise with blocks of 1024 bytes to store a file with 100 000 records, each of 100 bytes and stored completely in a block,

Calculate the following parameters.

1. Number of records per block.
2. Number of blocks to store the file.
3. Number of cylinders to store the file per cylinders.
4. Number of 100 bytes records stored in the disk.
5. If the pages are stored on the disk by cylinder, with page 1 on block 1 of track 1, which page is stored on block 1 of track 1 of the next disk surface? What will change if the disk can read/write in parallel by all the array of heads?
6. What is the time to read serially a file with 100 000 records of 100 bytes? What will change if the disk is able to read/write in parallel from all the array of heads (with the data stored in the best way)?
7. Suppose that every reading of a block involves a seek time and a rotational latency time, what is the time to read the file randomly?

Answer 2.2

1. Number of record per block = 1024/100 = 10.
2. To store 100 000 record in blocks of 10 record are needed 10 000 blocks.
3. A track contains 25 blocks and so 400 are needed. A cylinder contains 250 blocks and so 100 000 records are stored in 40 cylinder.
4. Total capacity of the disk is 512M, i.e. 500 000 blocks. Each block can contain 10 records, therefore 5M records can be stored in the disk.
5. It is block 26 on block 1 of track 1 on the next disk surface.
   If the disk were capable of reading/writing from all heads in parallel, we can put the first 10 blocks on the first cylinder. Therefore, it is block 2 on block 1 of track 1 on the next disk surface.
6. A file containing 100,000 records of 100 bytes needs 40 cylinders or 400 tracks in this disk. The transfer time of one track of data is 0.011 seconds. Then it takes $400 \times 0,011 = 4,4$ seconds to transfer 400 tracks.
   This access seeks the track 40 times. The seek time is $40 \times 0,01 = 0,4$seconds. Therefore, total access time is $4,4 + 0,4 = 4,8$ seconds.
   If the disk were capable of reading/writing from all heads in parallel, the disk can read 10 tracks at a time. The transfer time is 10 times less, which is 0,44 seconds. Thus total access time is $0,44 + 0,4 = 0,84$ seconds
7. To read a block, the average access time is: $t_s + t_r + t_t$ = 10 ms + 6 ms + 1/2250 ms = $10 + 6 + 0,44 = 16,44$ ms.
   For a file containing 100 000 records of 100 bytes, in 10 000 blocks, the total access time to read serially the file would be 164,4 seconds. To read the file randomly, 100 000 blocks must be read and the total access time would be 1644 seconds.

Exercise 2.3 Consider a disk with the following characteristics:

– $2^9 = 512$ bytes/sector
– 1000 sectors/track
– 10 000 cylinders
– 5 platters and 10 surfaces
– rotation speed 10 000 rpm
– the seek time is of 1 ms per track plus 1 ms per 1000 cylinders skipped.
  Calculate the following parameters.

1. Total capacity of the disk.
2. The average seek time.
3. The average rotational latency.
4. The transfer time of a block ($2^{14} = 16\,384$ bytes).
5. The average time for accessing 10 continuous blocks in one track on the disk.
6. Suppose you are told that half of the data on the disk are accessed much more frequently than another half (*hot* or *cold* data), and you are given the choice to place the data on the disk to reduce the average seek time. Where do you propose to place the hot data, considering each of the following two cases? (Hint: inner-most tracks, outer-most tracks, middle tracks, random tracks, etc). State your assumptions and show your reasoning.
   (a) There are same number of sectors in all tracks (the densities of inner tracks are higher than those of the outer tracks).
   (b) The densities of all tracks are the same (there are less sectors in the inner tracks than in the outer tracks).

Answer 2.3

1. *TotalNumberOfTracks* = 10 000 × 10 = 100 000
   TotalNumberOfSectors = *TotalNumberOfTracks* × 1000 = 100 000 000
   Total capacity of disk = 512 × *TotalNumberOfSectors* = 512 × 10^8 = 51,2 GB
2. *Average seek distance* = 10 000 / 3 = 3333 tracks
   Average seek time = 1 + 3333/1000 = 4,333 ms
3. *Average rotational latency* = $1/2 \times 60/10\,000 = 3$ ms
4. *Number of sectors per block* = $2^{14}/2^9 = 32$
   Transfer time of a block = 2^5 / 1000 × 60/10 000 s = 0,192 ms
5. *Time for accessing 10 continuous blocks* = $t_s + t_r + t_t$ = 4,33 + 3 + 0,192 × 10 = 9,25 ms.
6.
   (a) Put the hot data on the middle tracks. Thus, the average seek distance between the hot data and other (cold) data is minimized, and thereby minimizing the overall average seek time. Note that for part (a), the average seek distance among the hot data is the same no matter where you put the hot data.
   (b) Put the hot data on the outer tracks. Thus, the average seek distance among the hot data is minimized, because the hot data fits in fewer tracks. Note that since the hot data is frequently accessed, the average seek distance among the hot data is a dominating factor for the overall average seek time. So, minimizing the seek distance among the hot data also minimizes the overall average seek time.

Exercise 2.4 Give a brief answer to the following questions:

1. Explain how the read of a page is executed by the buffer manager.
2. When the buffer manager writes a page to the disk?
3. What does it mean that a page is pinned in the buffer? Who puts the pins and who takes them off?

Answer 2.4 See the textbook.

Chapter 3

HEAP AND SEQUENTIAL ORGANIZATIONS

Exercise 3.1 Explain what is meant by *file reorganization* and give an example of file organization that requires it, specifying which operations motivate it.

Answer 3.1 The performance of a static data organization tend to degrade when there are several insertions and deletions. An example is a sequential organization with overflow management to deal with several insertions and deletions.

Deletions reduce the pages load factor. Insertions can create long lists of overflow records that increase the cost of a search. In all these cases it is advisable to *reorganize* the file periodically, that is, to unload the records, allocate more space for records, and the reinsert the records back into the file in the required order.

Exercise 3.2 Discuss the advantages and disadvantages of records with fixed fields vs variable fields, and of records with fixed length vs variable length.

Answer 3.2

1. Fixed fields
   (a) Advantages:
      i. Easy to implement.
      ii. Efficient access.
      iii. Easy to update data.
   (b) Disadvantages:
      i. May waste space (when the padding is bigger than the field size)
      ii. Inflexible (need a priori knowledge of field size).
      iii. Can't support repeating fields that don't have an upper limit.
2. Fixed records
   (a) Advantages:
      i. Easy to implement.
      ii. Easy to manipulate.
      iii. Efficient access.
      iv. Efficient usage of disk space (schema is separate from data).

(b) Disadvantages:
   i. Space is waste with fields that are just full of nulls.
   ii. Inflexible schema.

Exercise 3.3 Let $R(K, A, B, \textit{other})$ a relation with $N_{\mathsf{rec}}(R) = 100\,000$, a key K with integer values in the range (1, 100 000), and the attribute A with integer values uniformly distributed in the range (1, 1000). The size of a record of R is $L_r = 100$ bytes. Suppose R stored with heap organization, with data *unsorted* both respect to K and A, in pages with size $D_{\mathsf{pag}} = 1024$ bytes.

The cost estimate C of executing a query is the number of pages read from or written to the permanent memory to produce the result.

Estimate the cost of the following SQL queries, and consider for each of them the cases that *Attribute* is K or A, and assuming that there are always records that satisfy the condition.

1. **SELECT** *
   FROM R
   WHERE Attribute = 50;

2. **SELECT** *
   FROM R
   WHERE Attribute **BETWEEN** 50 **AND** 100;

3. **SELECT** Attribute
   FROM R
   WHERE Attribute = 50
   ORDER BY Attribute;

4. **SELECT** *
   FROM R
   WHERE Attribute **BETWEEN** 50 **AND** 100
   ORDER BY Attribute;

5. **SELECT** Attribute
   FROM R
   WHERE Attribute **BETWEEN** 50 **AND** 100
   ORDER BY Attribute;

6. **INSERT INTO** R **VALUES** (...);
7. **DELETE**
   FROM R
   WHERE Attribute = 50;

8. **UPDATE** R
   SET A = 75
   WHERE K **BETWEEN** 50 **AND** 100;

Answer 3.3

1. With Attribute = K the cost is $C = \lceil N_{\mathsf{pag}}/2 \rceil$. With Attribute = A the cost is $C = N_{\mathsf{pag}}$.
2. With Attribute = K or A the cost is $C = N_{\mathsf{pag}}$.
3. The clause **ORDER BY** is superfluous and the cost is as in case 1.
4. Suppose that the cost is the sum of four quantities:
   a) range search cost.
   b) write cost to a temporary file of the records to sort.

c) a temporary file sort and
d) read cost of the sorted file.

The cost of (a) is $C = N_{\mathsf{pag}}$ as in case 2.
The cost of (b) is $C = N'_{\mathsf{pag}}$, the number of pages occupied by the E_{rec} records that satisfy the condition: $C = N'_{\mathsf{pag}} = \lceil E_{\mathsf{rec}}/D_{\mathsf{pag}} \rceil$.
With Attribute = K

$$E_{\mathsf{rec}} = \left\lceil N_{\mathsf{rec}} \times \frac{100 - 50}{100\,000 - 1} \right\rceil = 51$$

therefore $N'_{\mathsf{pag}} = 5$.
With Attribute = A

$$E_{\mathsf{rec}} = \left\lceil N_{\mathsf{rec}} \times \frac{100 - 50}{1000 - 1} \right\rceil = 5006$$

therefore $N'_{\mathsf{pag}} = 489$.
The cost of (c) is $C = 4 \times N'_{\mathsf{pag}}$.
The cost of (d) is $C = N'_{\mathsf{pag}}$.

5. As in the previous case, the cost is

$$C = N_{\mathsf{pag}} + N'_{\mathsf{pag}} + 4 \times N'_{\mathsf{pag}} + N'_{\mathsf{pag}}$$

the value of N'_{pag} is different because the result records have one attribute only, suppose with $L_A = 4$:

$$C = N'_{\mathsf{pag}} = E_{\mathsf{rec}} \times L_A / D_{\mathsf{pag}}$$

6. The cost is one page read and write: $C = 2$.
7. With Attribute = K the cost is that of search (case 1), to test that a record with $K = 50$ does not exist, and a page write.
   With Attribute = A the cost is that of search (case 1) and a write of the pages with the records deleted, estimated as $E_{\mathsf{rec}} = N_{\mathsf{rec}}/N_{\mathsf{key}}(A)$, with $N_{\mathsf{key}}(A)$ the number of different A values.
8. The cost is the range search by the primary key, as in the case 2, and the write of N'_{pag} with the 51 records that satisfy the condition: $N'_{\mathsf{pag}} = 51$.

Exercise 3.4 Assuming that a page access requires 10 ms, estimate the execution time of the SQL queries of the previous exercise in the case of a sequential organization with records *sorted* on the key K values.

Answer 3.4 The execution time is estimated as $C = C_D \times t_a$, where C_D is the search cost and $t_a = 10$ ms. Let us estimate C_D in the different cases with the selection Attribute = K only because with Attribute = A the result of the previous exercise hold.

1. $C_D = \lceil N_{\mathsf{pag}}/2 \rceil$.
2. $C_D = C_r + \lceil f_s \times N_{\mathsf{pag}} \rceil - 1$, with C_r the cost of case 1 and $f_s = 51$
3. The clause **ORDER BY** is superfluous and the cost is as in case 1.
4. The clause **ORDER BY** is superfluous and the cost is as in case 2.
5. As in the previous case.

6. To insert a new record, we must find the correct position in the ordering for the record and then find space to insert it. If there is sufficient space in the required page for the new record, then the single page can be reordered and written back to disk: $C_D = C_r + 1$. If this is not the case, then it would be necessary to move one or more records on to the next page. Again, the next page may have no free space and the records on this page must be moved, and so on: $C_D = C_r + N_{\mathsf{pag}}$. If the insertion requires an update of the overflow record list, then $C_D = C_r + 3$.
7. $C_D = C_r + 1$
8. The cost is the range search by the primary key, as in the case 2, and the write of $N'_{\mathsf{pag}} = 6$ with the 51 records that satisfy the condition.

Exercise 3.5 Consider a file R with 10 000 pages to sort using 3 pages in the buffer, and to write the sorted file to the disk.

1. How many runs are produced in the first pass?
2. How many 2-way merge phases are needed to sort the file?
3. How much time does it take to sort the file if a page access requires 10 ms?
4. How many buffer page B are needed to sort the file with one merge phase?

Answer 3.5

1. Let S be the initial number of runs: $S = \lceil N_{\mathsf{pag}}/B \rceil = 3334$
2. $k = \lceil \lg S \rceil = 12$
3. $t = t_a \times 2 \times N_{\mathsf{pag}} \times (k+1) = 0,010 \times 2 \times 10\,000 \times 13 = 0,010 \times 260\,000 = 2600$
4. $B - 1 \geq S$.
   $B \times (B-1) \geq N_{\mathsf{pag}}$
   $B = 101$

Exercise 3.6 Consider a file R with $N_{\mathsf{rec}}(R) = 10\,000$ records of 100 bytes stored in pages with size 1K. Assume that there are 5 buffer pages to sort the file, and to write the sorted file to the disk.

1. How many runs are produced in the first pass, and how long will each run be?
2. How many passes are needed to sort the file completely?
3. Which is the cost of sorting the file?
4. What is the number of records N_{rec} of the largest file that can be sorted in just two passes?

Answer 3.6

1. Let $N_{\mathsf{pag}} = \lceil N_{\mathsf{rec}} \times L_r / D_{pag} \rceil = 1000$ and the initial number of runs $S = \lceil N_{\mathsf{pag}}/B \rceil = 200$ of 5 pages.
2. $k = \lceil \log_{B-1} S \rceil + 1 = 5$
3. $C = 2 \times N_{\mathsf{pag}} \times k = 10\,000$
4. In the second pass we must merge $\lceil N_{\mathsf{pag}}/B \rceil$ runs, i.e. $\lceil N_{\mathsf{pag}}/B \rceil \leq (B-1)$. With $B = 5$, we get $N_{\mathsf{pag}} = 20$. The maximum number of records for page is 10 and so $N_{\mathsf{rec}} = 200$.

Chapter 4

HASHING ORGANIZATIONS

Exercise 4.1 The **CREATE TABLE** statement of a relational system creates a heap-organized table by default, but the DBA can use the following command to transform a heap organization into a hash primary organization:

MODIFY Table **TO HASH ON** Attribute;

The manual contains the following warning: "Do not modify a table's structure from its default heap structure to a keyed (i.e. hash) structure until the table contains most, if not all, of its data, ..., (otherwise) query processing performance degrade upon adding extra data". Explain what determines the performance degradation.

Answer 4.1 Performance degrades upon adding extra records, since the hash organization is static and overflow are created.

Exercise 4.2 Let $R(K, A, B, \mathit{other})$ a relation with key K with integer values. In the text it has been shown how the relation is stored with a *primary static hashing organization*. Explain how to modify the operations when the static hashing organization is made using the non-key attribute A.

Answer 4.2 An insertion does not present problems: a record with the same value of A is assigned to the same page. A search operation, or deletion, requires that once a record with the specified A value has been found, it is necessary to check if there are others records with the same attribute value.

A problem occurs when an overflow record is inserted in a page of a separate overflow area, and A has few different values in number less than the number of pages. In this case part of the primary area is not used and the lists of overflow become long, with a performance degradation.

Exercise 4.3 Let $R(K, A, B, \mathit{other})$ a relation with $N_{\mathsf{rec}}(R) = 100\,000$, $L_r = 100$ bytes, and a key K with integer values in the range (1, 100 000). Assume the relation stored with a *primary static hashing organization* using pages with size $D_{\mathsf{pag}} = 1024$ bytes and a loading factor $d = 0,80$.

Estimate the cost of the following SQL queries, assuming that there are always records that satisfy the condition.

1. **SELECT** *
   FROM R;

2.
```
SELECT   *
FROM     R
WHERE    K = 50;
```

3.
```
SELECT   *
FROM     R
WHERE    K BETWEEN 50 AND 100;
```

4.
```
SELECT     *
FROM       R
WHERE      K BETWEEN 50 AND 100
ORDER BY K;
```

Answer 4.3

1. The cost is that of primary area scan:
   $C_D = (N_{\mathsf{rec}} \times L_r)/(D_{\mathsf{pag}} \times d) = 12\,208$
2. $C_D = 1$
3. Since the number of records to find is 51, less than the number pages of the primary area, they can be found by applying 51 times the hash function to the values of the keys with a cost of 51.
4. As in the previous case because the clause **ORDER BY** is superfluous.

Exercise 4.4 Let $R(K, A, B, \mathit{other})$ a relation with $N_{\mathsf{rec}}(R) = 100\,000$, $L_r = 100$ bytes, and a key K with integer values in the range (1, 100 000), and the attribute A with integer values uniformly distributed in the range (1, 1 000) and $L_A = 4$. Assume the relation stored using pages with size $D_{\mathsf{pag}} = 1024$ bytes, and the following queries must be executed:

1. Find all R records.
2. Find all R records such that $A = 50$.
3. Find all R records such that $K \geq 50$ and $K < 100$.

Which of the following organizations is preferable to perform each operation?

1. A serial organization.
2. A static hashing organization

Answer 4.4 Let $N_{\mathsf{pag}}(R) = \lceil (N_{\mathsf{rec}} \times L_r)/D_{\mathsf{pag}} \rceil = 10^4$ be the number of pages needed to store data with a serial organization, and $\lceil 1,25 \times N_{\mathsf{pag}}(R) \rceil$ the number of pages needed by a static hashing organization. Table 4.1 shows the costs of operations for the two organizations.

– To find all records of R a serial organization is preferable.
– To find the record with $A = 50$ a static hashing organization is preferable.
– To find the records in the range $K \geq 50$ and $K < 100$, with 50 key values, a static hashing organization is preferable (remember, however, that a file scan is usually faster than a random access to half of its pages).

Operation	Serial	Hash
1	$N_{\mathsf{pag}}(R) = 10^4$	$\lceil 1,25 \times N_{\mathsf{pag}}(R) \rceil$
2	$N_{\mathsf{pag}}(R)$	$N_{\mathsf{pag}}(R)/N_{\mathsf{key}}(A) = 10$
3	$N_{\mathsf{pag}}(R)$	50

Table 4.1: Table for Exercise 4.4.

Operation	Sequential	Hash	Hash index
1	$N_{\mathsf{pag}}(R) = 10^4$	$\lceil 1,25 \times N_{\mathsf{pag}}(R) \rceil$	$N_{\mathsf{pag}}(I) + N_{\mathsf{rec}}$
2	$\lceil \lg N_{\mathsf{pag}}(R) \rceil = 14$	1	2
3	$14 + \lceil f_{s_1} \times N_{\mathsf{pag}} \rceil - 1 = 18$	50	$50 + 50$
4	$14 + \lceil f_{s_2} \times N_{\mathsf{pag}} \rceil - 1 = 14$	5	$5 + 5$
5	$N_{\mathsf{pag}}(R)$	$1,25 \times N_{\mathsf{pag}}(R)$	$N_{\mathsf{pag}}(I)$

Table 4.2: Costs of operations.

Exercise 4.5 Let $R(K, A, B, \mathit{other})$ a relation with $N_{\mathsf{rec}}(R) = 100\,000$, $L_r = 100$ bytes, and a key K with integer values in the range (1, 100 000), and $L_K = 4$. Assume the relation stored using pages with size $D_{\mathsf{pag}} = 1024$ bytes, and the following queries must be executed:

1. Find all R records.
2. Find all R records such that $K = 50$;
3. Find all R records such that $K \geq 50$ and $K < 100$.
4. Find all R records such that $K \geq 50$ and $K < 55$.
5. Find all K values.

Which of the following organizations is preferable to perform each operation?

1. A sequential organization.
2. A static hashing organization.
3. An unclustered hash index I.

Answer 4.5

– Let $N_{\mathsf{pag}}(R) = \lceil (N_{\mathsf{rec}} \times L_r)/D_{\mathsf{pag}} \rceil = 10^4$ be the pages occupied by the data organized with a sequential organization,
– Let $N_{\mathsf{pag}}(I) = \lceil 1,25 \times (N_{\mathsf{rec}} \times 4)/D_{\mathsf{pag}} \rceil = 500$ be the pages occupied by the hash index,
– Let $f_{s_1} = 5 \times 10^{-4}$ the selectivity factor of the condition $K \geq 50$ and $K < 100$,
– Let $f_{s_2} = 5 \times 10^{-5}$ the selectivity factor of the condition $K \geq 50$ and $K < 55$,

Table 4.2 shows the costs of operations for the three organizations, assuming that there are no overflow.

1. To find all the R records a sequential organization is preferable.

2. To find the records with $K = 50$ a static hashing organization is preferable.
3. To find all R records such that $K \geq 50$ and $K < 100$, a range with many key values, a sequential organization is preferable.
4. To find all R records such that $K \geq 50$ and $K < 55$, a range with only few key values, a static hashing organization is preferable.
5. To find all K values, an hash index is preferable.

Exercise 4.6 Consider a *linear hashing* organization, with $M = 3$ and each page holding 2 data entries. The following figure shows how the keys $\{4, 18, 13, 29, 32\}$ are stored.

0	18	
1	4	13
2	29	32

Show how the structure changes by inserting then the following keys in the order (9, 22, 44, 35).

Answer 4.6 See the textbook.

Chapter 5

DYNAMIC TREE-STRUCTURE ORGANIZATIONS

Exercise 5.1 The **CREATE TABLE** statement of a relational system creates a heap-organized table by default, but provides the DBA the following command to transform a heap organization into a tree-structure organization:

MODIFY Table **TO ISAM ON** Attribute;

The manual contains the following warning: "Do not modify a table's structure from its default heap structure to a keyed (i.e. ISAM) structure until the table contains most, if not all, of its data, ..., (otherwise) query processing performance degrade upon adding extra data". Explain what determines the performance degradation.

Answer 5.1 Performance degrades upon adding extra records, since the ISAM organization is static and overflow are created.

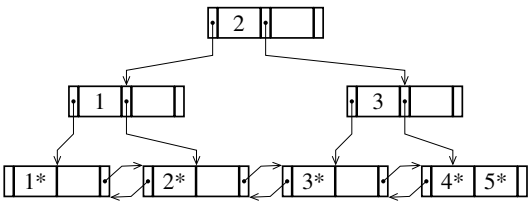
Exercise 5.2 Answer the following questions about index and tree organizations:

- What is the difference between an index *secondary organization* and *index sequential. organization*?
- What is the difference between a *clustered index* and an *unclustered index*? If an index contains data records as 'data entries' can it be unclustered?

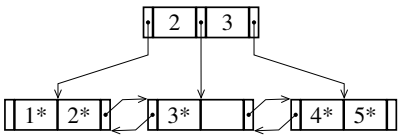
Answer 5.2 See the textbook.

Exercise 5.3 Show the result of entering the records with keys in the order (1, 2, 3, 4, 5) to an initially empty B^+–tree of order $m = 3$. In case of overflow, split the node and do not re-distribute keys to neighbors. Is it possible to enter the records with keys in a different order to have a tree of less height?

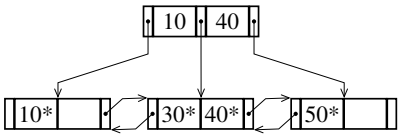
Answer 5.3 The result of entering the records with keys in the order (1, 2, 3, 4, 5) is the following B^+–tree.



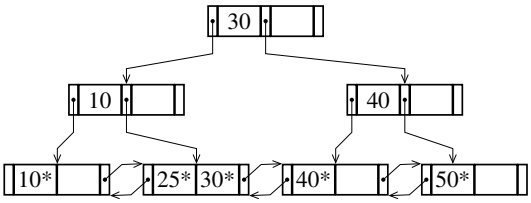
The result of entering the records with keys in the order (2, 3, 4, 1, 5) is the following B^+–tree.



Exercise 5.4 Show how the following B^+–tree changes after the insertion of the record with key 25.



Answer 5.4



Exercise 5.5 Consider a DBMS with the following characteristics: a) file pages with size 2048 bytes, b) pointers of 12 bytes, c) the page header of 56 bytes. A secondary index is defined on a key of 8 bytes. Compute the maximum number of records that can be indexed with

1. A three levels B–tree.
2. A three levels B^+–tree. For simplicity, assume that the leaf nodes are organized into a singly linked list.

Answer 5.5

1. An internal node of a B–tree of order n has at most n children and $(n-1)$ pairs (k_i, rid_i), e.g. keys and pointers to data: $8 \times (n-1) + 12 \times (2n-1) + 56 \leq 2.048$ and so $n \leq 62$. The root has at most 61 keys, the second level at most 62×61 keys, the leaves at most $62 \times 62 \times 61$ keys. Therefore the maximum number of keys is $61 + 62 \times 61 + 62 \times 62 \times 61 = 238\,327$.
2. An internal node of a B^+–tree of order n has at most n children and $n-1$ keys: $8 \times (n-1) + 12 \times n + 56 \leq 2048$ and so $n \leq 100$. The root has at most 99 keys. The number of leaves is $100 \times 100 = 10\,000$ and have at most $99 \times 10\,000 = 990\,000$ keys.

Exercise 5.6 Consider a secondary index on a primary key of a table with N records. The index is stored with a B^+–tree of order m. What is the minimum number of nodes to visit to search a record with a given key value?

Answer 5.6 First let us calculate the maximum number of keys that can be indexed with a B^+–tree of order m and height h. Each internal node has m children and each leaf node has at most $m-1$ pointers to a record, and so with h levels $m^{h-1} \times (m-1)$ keys can be indexed. Therefore $m^{h-1} \times (m-1) \geq N$ and so $h \geq \lceil \log_m(N/m-1) \rceil + 1$.

Exercise 5.7 Discuss the advantages and disadvantages of a B–tree and a *static hashing* primary organizations.

Answer 5.7 A B–tree is a typical dynamic structure that does not require a reorganization when there are several insertions and deletions. The operations cost depend on the tree height, which is usually small, and the structure allows both key search and range key search. The cost of an operation is easily estimate both in the best and in the worse case. The memory occupation is good because the nodes are 70% filled on average.

A static hashing is a typical structure that requires a reorganization when there are several insertion and deletions. The cost of operations depend on the loading factor of the primary area: for values lesser than 0.8 the average key search cost is estimated 1, the optimum. The cost of the worst case cannot always be easily estimated significantly. This organization does not allow range key search, except in cases where they are few and well-known key values, because it is enough to repeat the search for each value. The memory occupation is comparable to that of a B–tree. Another advantage is that it is easily implemented.

Exercise 5.8 Let $R(K, A, B, \textit{other})$ be a relation with $N_{\mathsf{rec}}(R) = 100\,000$, $L_r = 100$ bytes, a key K with integer values in the range (1, 100 000) and $L_K = 4$. Suppose R stored with heap organization in pages with size $D_{\mathsf{pag}} = 1024$ bytes and a loading factor $f_r = 0,8$, and an index exists on the key K stored as B^+–tree,

Estimate the cost of the following SQL queries, assuming that there are always records that satisfy the **WHERE** condition.

1. ```
   SELECT  *
   FROM    R;
   ```

2. ```
   SELECT  *
   FROM    R
   WHERE   K = 50;
   ```

3. ```
   SELECT  *
   FROM    R
   WHERE   K BETWEEN 50 AND 100;
   ```

4. ```
   SELECT    *
   FROM      R
   WHERE     K BETWEEN 50 AND 100
   ORDER BY K;
   ```

Answer 5.8

1. The cost of the data file scan is $(N_{\mathsf{rec}} \times L_r)/(D_{\mathsf{pag}} \times f_r) = 12\,208$. With the unclusterd index the cost is $N_{\mathsf{leaf}} + N_{\mathsf{rec}}$.
2. $C = 1 + 1$

3. The operation requires the index search to get the record rids with the cost $\lceil f_s \times N_{\mathsf{leaf}} \rceil$ where

$$f_s = \frac{100 - 50}{100\,000 - 1}$$

and the access cost to $E_{\mathsf{rec}} = 50$ records.

4. As in the previous case because the clause **ORDER BY** is superfluous.

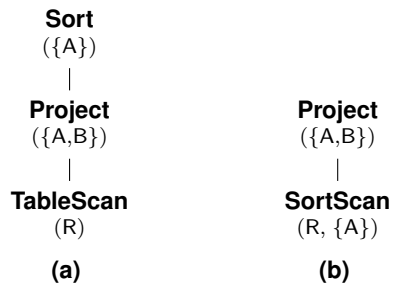
Exercise 5.9 Let $R(A, B, C, D, E)$ be a relation with key A, $N_{\mathsf{pag}}(R) = 10\,000$, $N_{\mathsf{rec}}(R) = 100\,000$ and $D_{\mathsf{pag}} = 500$ bytes. The values of all attributes are strings with length 10 bytes. Consider the query

```
SELECT    A, B
FROM      R
ORDER BY  A;
```

1. Estimate the query execution cost without indexes.
2. Estimate the query execution cost with a clustered index on A stored as B^+–tree with $N_{\mathsf{leaf}} = 3500$.

Answer 5.9

1. The query can be executed in two ways:



Let us assume that merge sort of a file with N pages is implemented so that it returns the final result of the merge without first writing it to a temporary file (*piped merge sort*), so the cost is $3 \times N$.
Let T be the result of the projection on (A, B), and

$$N_{\mathsf{pag}}(T) = \frac{20 \times 100\,000}{500} = 4000$$

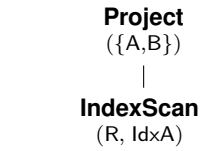
The first solution has cost

$$C = N_{\mathsf{pag}}(R) + \mathsf{sort}(T) = 10\,000 + 3 \times 4000 = 22\,000$$

while the second solution has cost

$$C = 3 \times N_{\mathsf{pag}}(R) = 3 \times 10\,000 = 30\,000$$

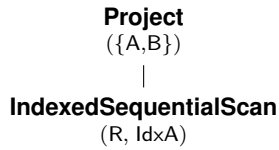
2. With the use of the clustered index on A



the cost is

$$C = N_{\mathsf{leaf}}(A) + N_{\mathsf{pag}}(R) = 3500 + 10\,000 = 13\,500$$

If the data were stored with the *index sequential organization*,



the cost would be $C = N_{\mathsf{pag}}(R) = 10\,000$, considering only the cost of data access.

Chapter 6

NON-KEY ATTRIBUTE ORGANIZATIONS

Exercise 6.1 To speed up the search for records in a table with an equality predicate on a non-key attribute A, and selectivity factor f_s, is preferable:

1. A sequential organization on a key attribute.
2. A static hash organization on a key attribute.
3. An inverted index on A.

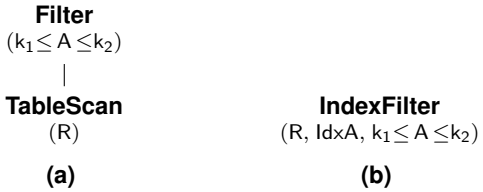
Briefly justify the answer and give an estimate of query execution cost in all three cases.

Answer 6.1 A sequential or a static hash organization requires a data file scan with cost N_{pag}.

An inverted index on A has the cost $C = C_I + C_D$, where $C_I = \lceil f_s \times N_{\mathsf{leaf}} \rceil$.
The data access cost C_D depends on the type of index.
If the index is clustered, $C_D = \lceil f_s \times N_{\mathsf{pag}} \rceil$.
If the index is unclustered with sorted rid-lists, $C_D = \lceil \Phi(f_s \times N_{\mathsf{rec}}, N_{\mathsf{pag}}) \rceil$.
If the index is unclustered with unsorted rid-lists, $C_D = \lceil f_s \times N_{\mathsf{rec}} \rceil$.

Exercise 6.2 Consider a relation R with N_{rec} records stored in N_{pag} of a heap file, and an inverted index on the attribute A with N_{key} integer values in the range A_{min} and A_{max}. Show two different execution plans to evaluate a non-key range query with condition $k_1 \leq A \leq k_2$, and their estimated cost. Explain in which case one is better than the other.

Answer 6.2 Let us consider the following plans



The cost of plan (a) is N_{pag}.

The cost of plan (b), with the index stored in a B^+–tree, with sorted rid-lists in the leaf nodes, is:

$$C = C_I + C_D = \lceil f_s \times N_{\mathsf{leaf}} \rceil + \left\lceil f_s \times N_{\mathsf{key}} \times \Phi(N_{\mathsf{rec}}/N_{\mathsf{key}}, N_{\mathsf{pag}}) \right\rceil$$

with $f_s = (k_2 - k_1)/(A_{\mathsf{max}} - A_{\mathsf{min}})$

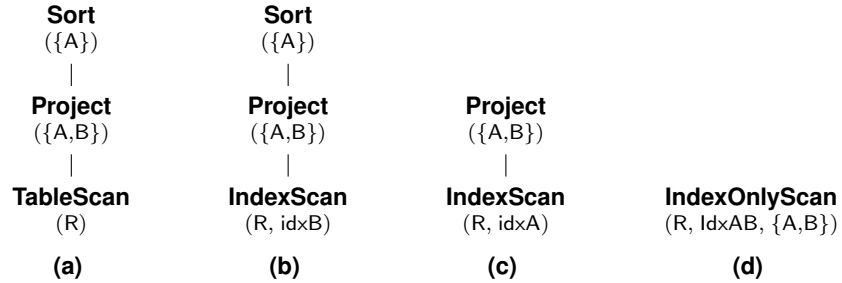
The plan (b) has a lower cost only if f_s is very small and N_{rec} is large.

Exercise 6.3 Consider the relation $R(A, B, C, D, E)$ with the key A, and each attribute a string 10 characters long. Assume that $N_{pag}(R) = 10\,000$, $N_{\mathsf{rec}} = 100\,000$ and $D_{\mathsf{pag}} = 500$ bytes. Consider the following query:

```
SELECT    A, B
FROM      R
ORDER BY  A;
```

a. Estimate the cost of a plan without the use of indexes.
b. Estimate the cost of a plan with the use of a clustered index on B stored with a B^+–tree with $N_{\mathsf{leaf}} = 2500$.
c. Estimate the cost of a plan with the use of a clustered index on A stored with a B^+–tree with $N_{\mathsf{leaf}} = 2500$.
d. Estimate the cost of a plan with the use of a clustered index on A, B stored with a B^+–tree with $N_{\mathsf{leaf}} = 5000$.

Answer 6.3 Let us consider the following plans for the four cases:



Let T be the result of the projection on (A, B), and $N_{\mathsf{pag}}(T) = \dfrac{20 \times 100\,000}{500} = 4000$

a. $C = N_{\mathsf{pag}}(R) + \mathsf{sort}(T) = 10\,000 + 3 \times 4000 = 22\,000$

b. $C = N_{\mathsf{leaf}}(B) + N_{\mathsf{pag}}(R) + \mathsf{sort}(T) = 2500 + 10\,000 + 3 \times 4000 = 24\,500$
c. $C = N_{\mathsf{leaf}}(A) + N_{\mathsf{pag}}(R) = 2500 + 10\,000 = 12\,500$
d. $C = N_{\mathsf{leaf}}(A, B) = 5000$

Exercise 6.4 Which of the following SQL queries execution takes less advantage from the presence of a multi-attribute index on R with A as the first attribute and B as the second attribute?

1. **SELECT** $*$ **FROM** R **WHERE** A = 10;
2. **SELECT** $*$ **FROM** R **WHERE** B = 20;
3. **SELECT** $*$ **FROM** R **WHERE** A $<$ B;
4. **SELECT** $*$ **FROM** R **WHERE** A $<$ C;
5. **SELECT** $*$ **FROM** R **WHERE** C $<$ 100 **AND** A = 10;

Answer 6.4 The index is useful for queries 1 and 5.

Exercise 6.5 Discuss the advantages and disadvantages of bitmap indexes.

Answer 6.5 See textbook.

Exercise 6.6 Consider a relation R with an unclustred index on the numerical non-key attribute B. Explain whether to find all records with $B > 50$ is always less costly to use the index.

Answer 6.6 No. In fact, since the index is not clustered and more rid-lists must be used, a page file might be read several times and so a file scan has a lower cost.

Chapter 9

TRANSACTION MANAGEMENT

Exercise 9.1 Define the concepts of transaction, of transaction failure and system failure. Describe the algorithm *NoUndo-Redo*.

Answer 9.1 See textbook.

Exercise 9.2 Consider a DBMS that uses the recovery algorithm *Undo-Redo*. Which of the following statements are true? Briefly justify your answers.

Exercise 9.3 Consider a DBMS that uses the recovery algorithm *Undo-Redo*. Which of the following statements are true? Briefly justify your answers.

(a1) All the updates of a transaction must be transferred to the database *before* the successful termination of the transaction (i.e. before the commit record is written to the log).

(a2) All the updates of a transaction must be transferred to the database *after* the successful termination of the transaction (i.e. after the commit record is written to the log).

(a3) The updates of a transaction may be transferred to the database *before* or *after* the successful termination of the transaction (i.e. before or after the commit record is written to the log).

(b1) The updates of a transaction must be transferred to the database *before* their before-images have been previously written to the log in the persistent memory.

(b2) The updates of a transaction must be transferred to the database *after* their before-images have been previously written to the log in the permanent memory.

(b3) The updates of a transaction may be transferred to the database *before* or *after* their before-images have been previously written to the log in the permanent memory.

Answer 9.2

1. Group a: only a3 is true. With redo, it is not necessary to transfer the updates to the database before the commit. With undo it is not necessary to wait for the commit to transfer the updates to the database.
2. Group b: only b2 is true. In fact, if a data has been updated before transferring its old value in the log, a failure between the two events would lead to the impossibility of retrieve the old value, and then the inability to undo.

Exercise 9.4 Describe the *Undo-Redo* algorithm and how the commit and the abort are implemented.

Answer 9.3 See the textbook.

Exercise 9.5 Describe the *NoUndo-Redo* algorithm and how the commit and the abort are implemented.

Answer 9.4 See the textbook.

Exercise 9.6 Consider the following log records, assuming that A, B, C and D are the pages with integer values. Assume the log entries are in the format (W, Trid, Variable, Old value, New value).

```
(BEGIN T1)
(W, T1, A, 20, 50)
      (BEGIN T2)
      (W, T2, B, 20, 10)
      (COMMIT T2)
(CKP, {T1})
(W, T1, C, 10, 5)
      (BEGIN T4)
      (W, T4, D, 30, 5)
(COMMIT T1)
SYSTEM FAILURE
```

Suppose that the transactions are managed with the *Undo-Redo* algorithm, and the checkpoint with the *Buffer-consistent checkpoint – Version 1*. Show the actions made with the system *restart*.

Answer 9.5 Not yet done.

Exercise 9.7 Consider the following log records from the start of the run of a database system, and suppose that the transactions are managed with the *Undo-Redo* algorithm, and the checkpoint with the *Buffer-consistent checkpoint – Version 1*.

```
 1)  (BEGIN T1)
 2)  (W, T1, A, 25, 50)
 3)  (W, T1, B, 25, 250)
 4)        (BEGIN T2)
 5)  (W, T1, A, 50, 75)
 6)        (W, T2, C, 25, 55)
 7)  (COMMIT T1)
 8)              (BEGIN T3)
 9)              (W, T3, E, 25, 65)
10)        (W, T2, D, 25, 35)
11)  (CKP {T2,T3})
12)        (W, T2, C, 55, 45)
13)        (COMMIT T2)
14)        (BEGIN T4)
15)        (W, T4, F, 25, 120)
16)              (COMMIT T3)
17)        (W, T4, F, 120, 150)
18)        (COMMIT T4)
```

Assume the log entries are in the format (W, Trid, Variable, Old value, New value). What is the value of the data items A, B, C, D, E on F on disk after the *restart* if the system crashes

1. just before line 10 is written to disk.
2. just before line 13 is written to disk.
3. just before line 14 is written to disk.
4. just before line 18 is written to disk.
5. just after line 18 is written to disk.

Answer 9.6

Case	A	B	C	D	E	F
1	75	250	25	25	25	25
2	75	250	25	25	25	25
3	75	250	45	35	25	25
4	75	250	45	35	65	25
5	75	250	45	35	65	150

Chapter 10

CONCURRENCY CONTROL

Exercise 10.1 Consider the following transactions and the history H:

$T_1 = r_1[a], w_1[b], c_1$
$T_2 = w_2[a], c_2$
$T_3 = r_3[a], w_3[b], c_3$
$H = r_1[a], w_2[a], c_2, r_3[a], w_1[b], w_3[b], c_3, c_1$

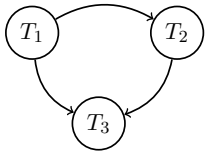
Answer the following questions:

1. Is H c-serializable?
2. Is H a history produced by a strict 2PL protocol?

Answer 10.1

T_1	T_2	T_3
$r_1[a]$		
	$w_2[a]$	
	c_2	
		$r_3[a]$
$w_1[b]$		
		$w_3[b]$
		c_3
c_1		

The history is c-serializable because its serialization graph is acyclic.



A strict 2PL serializer cannot generate the history H, because T_2 writes a before T_1 has released the read lock on the same data that it holds.

Exercise 10.2 Consider the following transactions:

$T_1 = r_1[X], r_1[Y], w_1[X]$
$T_2 = r_2[X], w_2[Y]$

and the history $S = r_1[X], r_2[X], r_1[Y] \ldots$

Show how the history can continue on a system that adopts the strict 2PL protocol.

Answer 10.2 A system that adopts the strict 2PL protocol will produce a deadlock when T_1 and T_2 will execute in any order the operations $W_1(X)$ and $W_2(Y)$, because T_2 has a read lock on X and T_1 has a read lock on Y. The system must abort a transaction, typically the youngest T_2, and T_1 will commit.

Exercise 10.3 Consider the following transactions and the history H:

$T_1 = r_1[a], w_1[a], c_1$
$T_2 = r_2[b], w_2[a], c_2$
$H = r_1[a], r_2[b], w_2[a], c_2, w_1[a], c_1$

Answer the following questions:

1. Is H c-serializable?
2. Is H a history produced by a strict 2PL protocol?
3. Suppose that a strict 2PL serializer receives the following requests (where rl and wl means *read lock* and *write lock*):

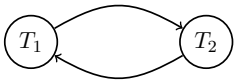
   $rl_1[a], r_1[a], rl_2[b], r_2[b], wl_2[a], w_2[a], c_2, wl_1[a], w_1[a], c_1$

   Show the history generated by the serializer.

Answer 10.3

T_1	T_2
$r_1[a]$	
	$r_2[b]$
	$w_2[a]$
	c_2
$w_1[a]$	
c_1	

1. The history is not c-serializable because its serialization graph is cyclic.



2. No, because the *strict 2PL* protocol produces only c-serializable histories .
3. The serializer executes the commands in the order they are received until it receives the request for a write lock $wl_2[a]$; this request is not granted because is in conflict with the granted read lock $rl_1[a]$. T_2 is is blocked, while the other T_1 requests $(wl_1[a], w_1[a], c_1)$ are accepted and executed. The commit c_1 causes the release of the blocks assigned to the transaction T_1, and therefore the serializer unblocks T_2 and executes the related commands. The final history is therefore the following:

$$rl_1[a], r_1[a], rl_2[b], r_2[b], wl_1[a], w_1[a], c_1, wl_2[a], w_2[a], c_2$$

Exercise 10.4 Consider the following history H of transactions T_1, T_2 and T_3 initially arrived at time 10, 20, 30, respectively.

$$H = r_3[B], r_1[A], r_2[C], w_1[C], w_2[B], w_2[C[, w_3[A]$$

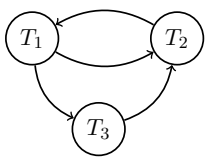
We make the following assumptions:

1. A transaction requests the necessary lock (shared lock for read and exclusive lock for write) on a data item right before its action on that item is issued,
2. If a transaction ever gets all the locks it needs, then it instantaneously completes work, commits, and releases its locks,
3. If a transaction dies or is wounded, it instantaneously gives up its locks, and restarts only after all current transactions commit or abort,
4. When a lock is released, it is instantaneously given to any transaction waiting for it (in a first-come-first-serve manner).

Answer the following questions:

1. Is H c-serializable?
2. If the *strict 2PL* is used to handle lock requests, in what order do the transactions finally commit?
3. If the *wait-die* strategy is used to handle lock requests, in what order do the transactions finally commit?
4. If the *wound-wait* strategy is used to handle lock requests, in what order do the transactions finally commit?
5. If the *snapshot* strategy is used, in what order do the transactions finally commit?

Answer 10.4 1. H is not c-serializable because its serialization graph is cyclic.



2. **Strict 2PL**

	T_1	T_2	T_3
1)			$rl[B], r[B]$
2)	$rl[A], r[A]$		
3)		$rl[C], r[C]$	
4)	$wl[C]$ **Denied**		
5)		$wl[B]$ **Denied**	
6)			$wl[A]$ **Denied**
7)			**Deadlock, Restart**
8)		$wl[B], w[B]$	
9)		$wl[C], w[C]$ **Commit**	
10)	$wl[C], w[C]$ **Commit**		
11)			$rl[B], r[B]$
			$wl[A], w[A]$ **Commit**

The transactions commit in the order T_2, T_1, T_3.

3. **Waits-die**

	T_1	T_2	T_3
1)			$rl[B], r[B]$
2)	$rl[A], r[A]$		
3)		$rl[C], r[C]$	
4)	$wl[C]$ **Waits**		
5)		$wl[B]$ **Waits**	
6)			$wl[A]$ **Dies**
7)		$w[B]$	
		$wl[C], w[C]$ **Commit**	
8)	$w[C]$ **Commit**		
9)			$rl[B], r[B]$
			$wl[A], w[A]$ **Commit**

The transactions commit in the order T_2, T_1, T_3.

4. **Wound-wait**

	T_1	T_2	T_3
1)			$rl[B], r[B]$
2)	$rl[A], r[A]$		
3)		$rl[C], r[C]$	
4)	$wl[C]$	**Wounded**	
5)	$w[C]$ **Commit**		
6)			$rl[B], r[B]$
			$wl[A], w[A]$ **Commit**
7)		$rl[C], r[C]$	
		$wl[B], w[B]$	
		$wl[C], w[C]$ **Commit**	

The transactions commit in the order T_1, T_3, T_2.

5. **Snapshot**

	T_1	T_2	T_3
1)			$r[B]$
2)	$r[A]$		
3)		$r[C]$	
4)	$w[C]$ **Commit**		
5)		$w[B]$	
		$w[C]$ **Commit**	
		Write set conflict	
		Abort, Restart	
6)			$w[A]$ **Commit**
7)		$r[C], w[B], w[C]$ **Commit**	

The transactions commit in the order T_1, T_3, T_2.

Exercise 10.5 Consider the transactions:

$T_1 = r_1[x], w_1[x], r_1[y], w_1[y], c_1$
$T_2 = r_2[y], w_2[y], r_2[x], w_2[x], c_2$

1. Compute the number of possible histories.
2. How many of the possible histories are c-equivalent to the serial history (T_1, T_2) and how many to the serial history (T_2, T_1)?

Answer 10.5

1. In general, given m transactions with number of operations n_1, n_2, ..., n_m, the number of possible histories is:

$$\frac{(n_1 + n_2 + \ldots + n_m)!}{(n_1! \times n_2! \times \ldots \times n_m!)}$$

where the numerator gives the number of possible permutations and the denominator is the product of the permutations of the operations of individual transactions to be excluded from the result. In our case, $m = 2$, $n_1 = 5$ and $n_2 = 5$, so the number of possible histories is

$$\frac{(5+5)!}{(5! \times 5!)} = 252$$

2. Given a history, another c-equivalent to it can be obtained by inverting the order of non-conflicting operations of different transactions. Therefore, instead of searching among the possible histories for those equivalent to a serial history, we use the inverse procedure: we calculate in how many ways it is possible to invert the order of non-conflicting operations of different transactions to get other c-equivalent histories.
In the history (T_1, T_2) the fourth and fifth operation are $w_1[y]$ and $r_2[y]$ that can not be inverted. The other pairs of adjacent operations are of the same transaction and can not be inverted. Therefore there is only one history c-equivalent to (T_1, T_2). Same conclusion for (T_2, T_1).

Exercise 10.6 The transaction T_1 *precedes* T_2 in the history S if all actions of T_1 precede actions of T_2. Give an example of a history S that has the following properties:

1. T_1 precedes T_2 in S,
2. S is c-serializable, and
3. in every serial history c-equivalent to S, T_2 precedes T_1.

The schedule may include more than 2 transactions and you do not need to consider locking actions. Please use as few transactions and read or write actions as possible.

Answer 10.6 Note that if T_1 and T_2 are the only transactions in S, then say that T_1 precedes T_2 is equivalent to saying that S is the serial history (T_1, T_2), but it does not satisfy the third condition. The story S must therefore include at least one transaction T_3 with an operation that precedes and conflicts with an operation of T_1, and T_2 must have an operation that precedes and conflicts with operation of T_3. Finally, T_1 and T_2 should not have operations in the conflict.

1. Assume $S = w_3[x], r_1[x], w_2[y], w_3[y]$. In S, T_1 precedes T_2;
2. The serialization graph is $T_2 \rightarrow T_3 \rightarrow T_1$, and therefore S is c-serializzable;
3. T_2, T_3, T_1 is the only serial history c-equivalent to S and T_2 precedes T_1.

Exercise 10.7 Assume the transactions are managed with the *undo-redo* algorithm, the concurrency mechanism used is strict 2PL protocol, there are only read and write locks, and the checkpoint method used is the *Buffer-consistent – Version 1*.

Assume the log contents shown below (the sequence is ordered left to right, top to bottom) when a system failure occurs. Assume the log entries are in the format (W, Trid, Variable, Old value, New value) and for simplicity the variables are pages with an integer value:

(BEGIN T1)	(W, T1, X, 5, 10)	(BEGIN T2)
(W, T2, X, 10, 20)	(COMMIT T1)	(W, T2, Y, 30, 60)
(CKP {T2})	(W, T2, W, 35, 70)	(BEGIN T3)
(W, T3, Z, 60, 40)	(COMMIT T2)	

1. Is it possible for the log to have the above contents? Explain briefly. If the answer is yes, give a possible sequence of actions of a possible schedule based on the log. If the answer is no, remove the first "impossible" log entry and repeat the process until you get a possible sequence of log entries.
2. For the sequence of entries you got in the previous point, what are the possible values of X, Y, W and Z after the last of these records is written to the permanent memory and before recovery.

Answer 10.7

1. The schedule is not possible: T_2 obtains a lock on X and it changes X while T_1 is still locking X. Assume that the (W, T2, X, 10, 20) is in the log after (COMMIT T1). The following sequence of actions is one possible schedule based on the log.

T1 starts
T1 locks X
T1 changes X from 5 to 10
T2 starts
T1 commits
T2 locks X
T2 changes X from 10 to 20
T2 locks Y
T2 changes Y from 30 to 60
CKP {T2}
T2 locks W
T2 changes W from 35 to 70
T3 starts
T3 locks Z
T3 changes Z from 60 to 40
T2 commits

2. The values written to the checkpoint (CKP) are certainly in the permanent memory: $X = 20, Y = 60$.
   For the other variables, since *undo-redo* logging does not have any restrictions on whether to flush data before or after the commit, the new values of W and Z may or may not be flushed: $W = 35/70$ e $Z = 60/40$.

Chapter 11

IMPLEMENTATION OF RELATIONAL OPERATORS

Exercise 11.1 Briefly answer the following questions:

1. Define the term *useful index for a query*.
2. Which relational algebra operators can be implemented with a sorting operator.
3. Describe an algorithm for the join implementation and give an estimate of its cost.
4. Compare two algorithms for the join implementation.
5. If the join condition is not equality, which join algorithms cannot be used?

Answer 11.1 See the textbook.

Exercise 11.2 Consider the relation $R(A, B, C, D)$ with key A and the following SQL query. All the attributes have a type string of the same length.

```
SELECT DISTINCT A, B FROM R;
```

1. Estimate the cost of an access plan without the use of indexes.
2. Estimate the cost of an access plan with the use of a clustered B^+–tree index on B.
3. Estimate the cost of an access plan with the use of an unclustered B^+–tree index on A.
4. Estimate the cost of an access plan with the use of a multi-attribute clustered B^+–tree index on A, B.

Answer 11.2 Not yet done.

Exercise 11.3 Consider the relation $R(A, B, C, D)$ with key A. All the attributes have a type string of the same length.

Suppose that a B^+–tree inverted index on C is available. Estimate the cost of an access plan for the following SQL query:

```
SELECT   *
FROM     R
WHERE    C BETWEEN 'C1' AND 'C10';
```

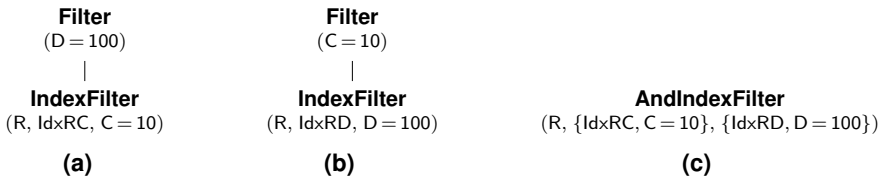
Answer 11.3 Since the values of C have a type string, the selectivity factor of the condition is estimated as 1/4.

$$C_A = C_I + C_D = \lceil 0,25 \times N_{\mathsf{leaf}} \rceil + \left\lceil 0,25 \times N_{\mathsf{key}} \times \Phi(N_{\mathsf{rec}}/N_{\mathsf{key}}, N_{\mathsf{pag}}) \right\rceil$$

Exercise 11.4 Consider the relation $R(A, B, C, D)$ with the attributes C and D of type integer. Suppose there is a clustered inverted index on C and an unclustered inverted index on D. Estimate the cost of an access plan for the following SQL query:

```
SELECT   *
FROM     R
WHERE    C = 10 AND D = 100;
```

Answer 11.4 Let us consider the following physical plans:



The estimated cost of each plan is $C = C_I + C_D$.

(a) Only the clustered index I_C on C is used. Let $\psi = (C = 10)$.

$$C_I = \lceil s_f(\psi) \times N_{\mathsf{leaf}}(I_C) \rceil \qquad C_D = \left\lceil s_f(\psi) \times N_{\mathsf{pag}}(R) \right\rceil$$

(b) Only the unclustered index I_D on D is used. Let $\psi = (D = 100)$.

$$C_I = \lceil s_f(\psi) \times N_{\mathsf{leaf}}(I_D) \rceil \qquad C_D = \left\lceil \Phi(s_f(\psi) \times N_{\mathsf{rec}}(R), N_{\mathsf{pag}}(R)) \right\rceil$$

where Φ is the Cardenas formula.

(c) Both indexes are used. Let $\psi_C = (C = 10)$ and $\psi_D = (D = 100)$.

$$C_I = \lceil s_f(\psi_C) \times N_{\mathsf{leaf}}(I_C) \rceil + \lceil s_f(\psi_D) \times N_{\mathsf{leaf}}(I_D) \rceil$$

$$C_D = \left\lceil \Phi(s_f(\psi_C) \times s_f(\psi_D) \times N_{\mathsf{rec}}(R), N_{\mathsf{pag}}(R)) \right\rceil$$

Exercise 11.5 Consider the relation R(K:int, A:int, B:int) organized as a *sorted file* on the attribute A in N_{pag} pages. R contains N_{rec} records. The attributes A and B have $N_{\mathsf{key}}(A)$ and $N_{\mathsf{key}}(B)$ values.

Suppose that the following indexes are available:

1. An hash index on the primary key K.
2. Two B^+–tree inverted indexes on A and B.

For each of the following relational algebra queries, estimate the cost of an access plan to execute the queries with the use of only one index:

1. $\sigma_{\text{(K isin [k1; k2; k3]) AND (B = 10)}}(R)$
2. $\sigma_{\text{(A = 100) AND (B = 10)}}(R)$.

Answer 11.5

1. A strategy that uses at most one index may (a) use only the index on K, or (b) use only the index on B. In the first case

$$C^K = C_I + C_D = 3 + 3 = 6$$

Where we have estimated 1 the average cost of an access to a hash index. In the second case, we have the following estimate (assuming sorted rid-lists):

$$\begin{aligned} C^B &= C_I + C_D \\ &= \lceil s_f(B = 10) \times N_{\text{leaf}}(B) \rceil + \left\lceil \Phi(s_f(B = 10) \times N_{\text{rec}}, N_{\text{pag}}) \right\rceil \end{aligned}$$

In practice, $C^K < C^B$, unless the condition $(B = 10)$ is not extremely selective, or $N_{\text{rec}}/N_{\text{key}}(B)$ is a rather low value, in this case we may approximate C^B as follows:

$$C^B \approx 1 + N_{\text{rec}}/N_{\text{key}}(B)$$

and therefore $C^B < C^K$ if $N_{\text{rec}}/N_{\text{key}}(B) < 5$.

2. A strategy that uses at most one index may (a) use only the index on A, or (b) use only the index on B. In the first case, since the index is clustered, the cost is estimated as:

$$C^A = C_I + C_D = \lceil f_s(A = 100) \times N_{\text{leaf}}(A) \rceil + \left\lceil f_s(A = 100) \times N_{\text{pag}} \right\rceil$$

The value of C^B is the same as previously calculated. If $N_{\text{key}}(A)$ and $N_{\text{key}}(B)$ are similar, C^A will be much less than C^B. The approach (b) is instead convenient if $N_{\text{key}}(A) \ll N_{\text{key}}(B)$, more precisely, if $N_{\text{key}}(A)(N_{\text{rec}}/N_{\text{pag}}) < N_{\text{key}}(B)$.

Exercise 11.6 Consider the relations $R(A, B)$, $S(B, C)$, $T(C, D)$ and the following information about them:

$N_{\text{rec}}(R) = 200, N_{\text{key}}(R.A) = 50, N_{\text{key}}(R.B) = 100$
$N_{\text{rec}}(S) = 300, N_{\text{key}}(S.B) = 50, N_{\text{key}}(S.C) = 50$
$N_{\text{rec}}(T) = 400, N_{\text{key}}(T.C) = 40, N_{\text{key}}(T.D) = 100$

For each of the following relational algebra queries, estimate the size of the results:

1. $(\sigma_{S.B=20}(S)) \bowtie T$;
2. $\sigma_{R.A \neq S.C}(R \bowtie S)$.

Answer 11.6

1. $(\sigma_{S.B=20}(S)) \bowtie T$.

   Let $U = \sigma_{S.B=20}(S)$.

   $E_{\mathsf{rec}}(U) = \dfrac{N_{\mathsf{rec}}(S)}{N_{\mathsf{key}}(S.B)} = \dfrac{300}{50} = 6.$

   We can assume that $N_{\mathsf{key}}(C)$ of U is 6,

   $E_{\mathsf{rec}}(U \bowtie T) = \dfrac{N_{\mathsf{rec}}(U) \times N_{\mathsf{rec}}(T)}{max\{N_{\mathsf{key}}(U.C), N_{\mathsf{key}}(T.C)\}} = \dfrac{6 \times 400}{40} = 60.$

2. $\sigma_{R.A \neq S.C}(R \bowtie S)$

   Let $U = R \bowtie S$.

   $E_{\mathsf{rec}}(U) = \dfrac{N_{\mathsf{rec}}(R) \times N_{\mathsf{rec}}(S)}{max\{N_{\mathsf{key}}(R.B), N_{\mathsf{key}}(S.B)\}} = \dfrac{200 \times 300}{100} = 600.$

   $E_{\mathsf{rec}}(\sigma_{R.A \neq S.C}(R \bowtie S)) = f_s(R.A \neq S.C) \times E_{\mathsf{rec}}(U)$

   $E_{\mathsf{rec}}(\sigma_{R.A \neq S.C}(R \bowtie S)) = E_{\mathsf{rec}}(U) \times \left(1 - \dfrac{1}{max\{N_{\mathsf{key}}(R.A), N_{\mathsf{key}}(S.C)\}}\right)$

   $E_{\mathsf{rec}}(\sigma_{R.A \neq S.C}(R \bowtie S)) = \dfrac{600 \times 49}{50} = 588.$

Chapter 12

QUERY OPTIMIZATION

Exercise 12.1 Briefly answer the following questions:

1. Define the term *selectivity factor*.
2. Explain the role of *interesting orders* in a *System R* like optimizer.
3. Describe *left-deep plans* and explain why optimizers typically consider only such plans.

Answer 12.1 See textbook.

Exercise 12.2 Consider the following schema, where keys are underlined:

Students(Id, Name, BirthYear, Status, Other)
Transcripts(Subject, StudId, Grade, Year, Semester)

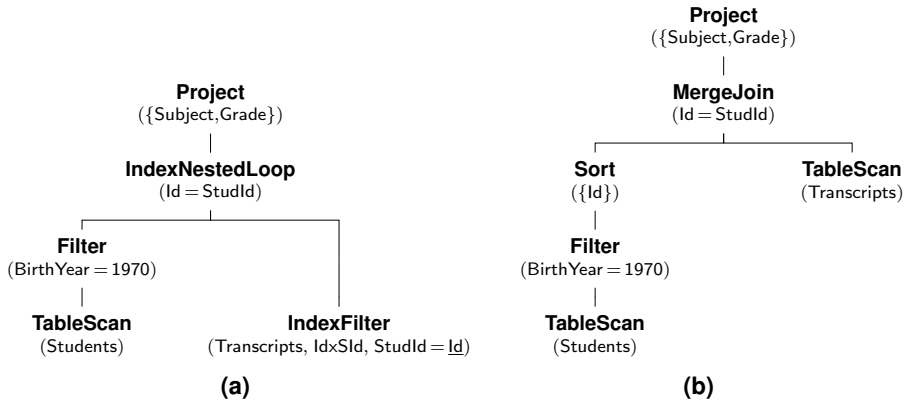
Consider the following query:

```
SELECT   Subject, Grade
FROM     Students, Transcripts
WHERE    BirthYear = 1990 AND Id = StudId
```

Suppose that a clustered B^+–tree index on StudId is available.

Show two physical plans, and the estimated costs, one with the use of the join physical operator **IndexNestedLoop** and the other with the join physical operators **MergeJoin**.

Answer 12.2
Taking into account the presence of the clustered index on StudId, two possible physical plans with the join physical operators **IndexNestedLoop** and **MergeJoin** are the following.



The join cost with the **IndexNestedLoop** is estimated as follows.

$$\begin{aligned} C_{\mathsf{INL}} &= C_A(\mathsf{Students}) + E_{\mathsf{rec}}(\mathsf{Students}) \times C_A(\mathsf{Transcripts}) \\ &= N_{\mathsf{pag}}(\mathsf{Students}) + N_{\mathsf{rec}}(\mathsf{Students})/N_{\mathsf{key}}(\mathsf{BirthYear}) \\ &\quad \times (C_I(\mathsf{StudId}) + C_D(\mathsf{Transcripts})) \\ &= N_{\mathsf{pag}}(\mathsf{Students}) + N_{\mathsf{rec}}(\mathsf{Students})/N_{\mathsf{key}}(\mathsf{BirthYear}) \\ &\quad \times (\lceil f_s \times N_{\mathsf{leaf}}(\mathsf{StudId}) \rceil + \lceil f_s \times N_{\mathsf{pag}}(\mathsf{Transcripts}) \rceil) \\ &\approx N_{\mathsf{pag}}(\mathsf{Students}) + N_{\mathsf{rec}}(\mathsf{Students})/N_{\mathsf{key}}(\mathsf{BirthYear}) \times 2 \end{aligned}$$

where $f_s = 1/N_{\mathsf{key}}(\mathsf{StudId})(= 1/N_{\mathsf{rec}}(\mathsf{Students}))$, and the last approximation is valid if $N_{\mathsf{key}}(\mathsf{StudId}) > N_{\mathsf{leaf}}(\mathsf{IdxSId})$ and $N_{\mathsf{key}}(\mathsf{StudId}) > N_{\mathsf{pag}}(\mathsf{Transcripts})$.

The join cost with the **MergeJoin** is estimated as follows taking into account the fact that the table Transcripts is sorted on StudId, and that it is only necessary to sort a temporary table S' containing only the Id of the students who were born in 1970, with

$$N_{\mathsf{pag}}(S') = \frac{N_{\mathsf{pag}}(\mathsf{Students})}{N_{\mathsf{key}}(\mathsf{BirthYear})}$$

$$\begin{aligned} C_{\mathsf{MJ}} &= C_A(\mathsf{Students}) + C_{\mathsf{sort}}(S') + N_{\mathsf{pag}}(\mathsf{Transcripts}) \\ &= N_{\mathsf{pag}}(\mathsf{Students}) + 3 \times N_{\mathsf{pag}}(S') + N_{\mathsf{pag}}(\mathsf{Transcripts}) \end{aligned}$$

Exercise 12.3 Consider the following schema, where the keys are underlined:

Students(Id, Name, BirthYear, Status, Other)
Transcripts(Subject, StudId, Grade, Year, Semester)

Consider the following query:

```
SELECT     Subject, COUNT(*) AS NExams
FROM       Students, Transcripts
WHERE      Year = 2012 AND Id = StudId
GROUP BY   Subject
HAVING     AVG(Grade) > 25;
```

1. Suppose that no indexes are available. Show the physical plan with the lowest estimated cost.
2. If there is a B^+–tree index on Subject, and a B^+–tree index on Id, what is the physical plan with lowest estimated cost?

Answer 12.3 Not yet done.

Exercise 12.4 Consider the following schema, where the keys are underlined (different keys are underlined differently):

Customer(PkCustPhoneNo, CustName, CustCity)
CallingPlans(PkPlanId, PlanName)
Calls(PkCustPhoneNo, FkPlanId,
Day, Month, Year, Duration, Charge)

where PkPlanId e PlanName are two different keys, and the following query

```
Q:  SELECT      Year, PlanName, SUM(Charge) AS TC
    FROM        Calls, CallingPlans
    WHERE       FkPlanId = PkPlanId AND Year >= 2000 AND Year <=2005
    GROUP BY    Year, PlanName
    HAVING      SUM(Charge) > 1000;
```

Give the initial logical query plan. Can the **GROUP BY** be pushed on the relation Calls?

Answer 12.4

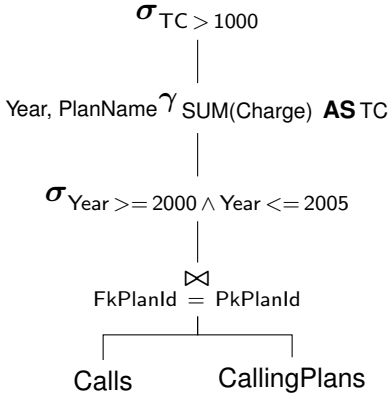


Figure 12.1: Logical query plan

The selection on Year can be pushed below the join.

The group-by can be pushed below the join because the *invariant grouping* property holds: (a) PlanName → PkPlanId, FkPlanId = PkPlanId, and so PlanName → FkPlanId; (b) SUM(Charge) uses an attribute of Calls. The rewriting of the group-by can be done together with the selection on TC.

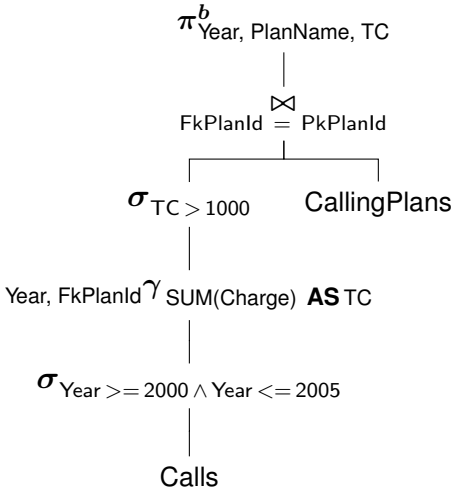


Figure 12.2: Logical query plan with the group-by pushed below the join

Exercise 12.5 Consider the following schema without null values, where the keys are underlined:

Customer(PkCustomer, CName, CCity)
Order(PkOrder, FkCustomer, ODate)
Product(PkProduct, PName, PCost)
OrderLine(LineNo, FkOrder, FkProduct, Quantity, ExtendedPrice, Discount, Revenue)

Consider the following query:

```
SELECT      CCity, AVG(Revenue) AS avgR
FROM        OrderLine, Order, Customer
WHERE       FkOrder = PkOrder AND FkCustomer = PkCustomer
GROUP BY    CCity, FkCustomer
HAVING      SUM(Revenue) > 1000;
```

Give the logical query plan and show how the **GROUP BY** can be pushed on the join (OrderLine $\bowtie$ Order).

Can the **GROUP BY** be pushed on the relation OrderLine?

Answer 12.5

The group-by can be pushed below the join (OrderLine $\bowtie$ Order) because the *invariant grouping* property holds.

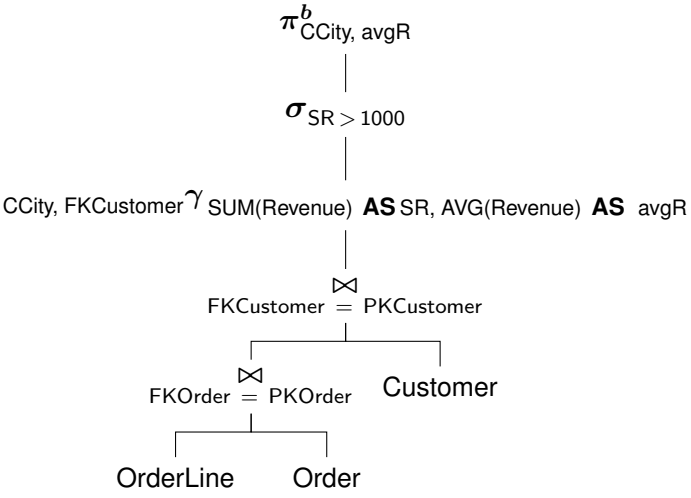


Figure 12.3: Logical query plan

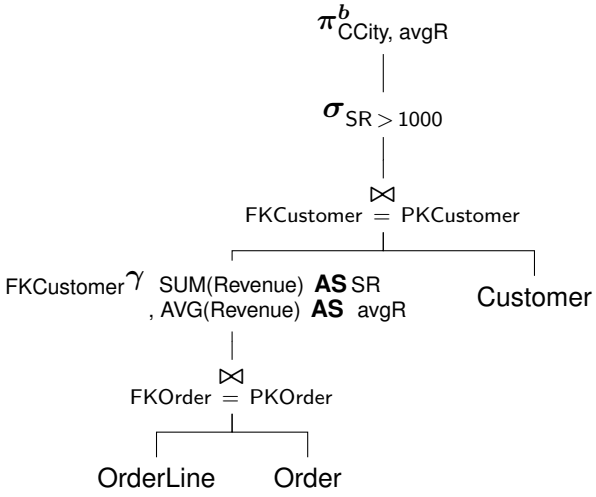


Figure 12.4: Logical query plan: the **GROUP BY** is pushed below the first join

Can the **GROUP BY** be pushed on the relation OrderLine?

The group-by cannot be pushed below the join (OrderLine $\bowtie$ Order)
because the *invariant grouping* property does not hold: FKCustomer $\not\rightarrow$ FKOrder.

However, the group-by can rewritten as a double grouping, with the rewriting of the not decomposable aggregation function AVG(Revenue) as SUM(Revenue) / COUNT(Revenue), equivalent to SUM(Revenue) / COUNT(*) because the database is without null values (Figure 12.5a).

Now the second group-by can be pushed below the join (OrderLine $\bowtie$ Order) on OrderLine because the *invariant grouping* property holds (Figure 12.5b).

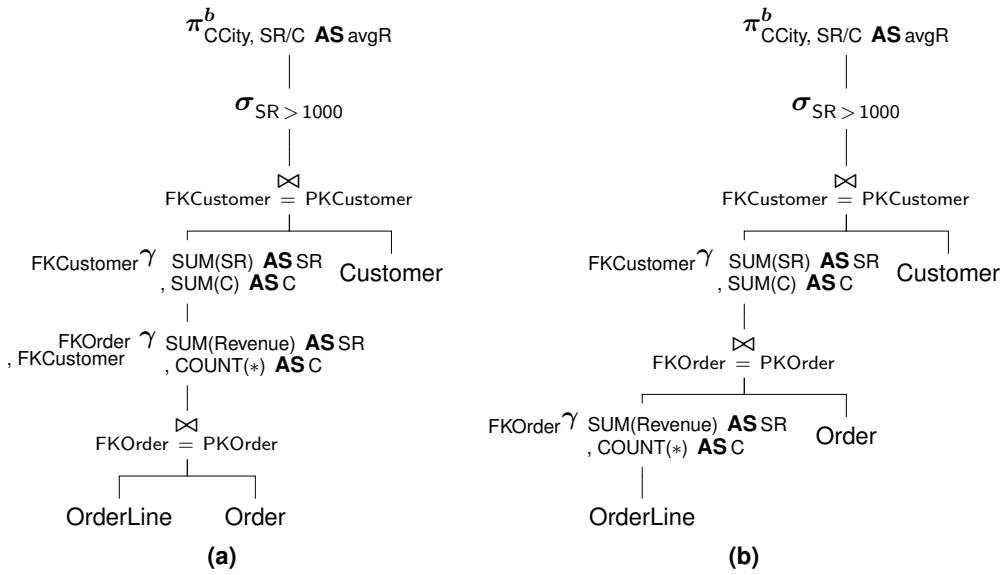


Figure 12.5: Logical query plans: (a) the **GROUP BY** is rewritten with the double grouping and the rewriting of AVG, (b) the second group-by is pushed below the second join

Exercise 12.6 Consider the following schema with attributes of type integer without null values, where the keys are underlined:

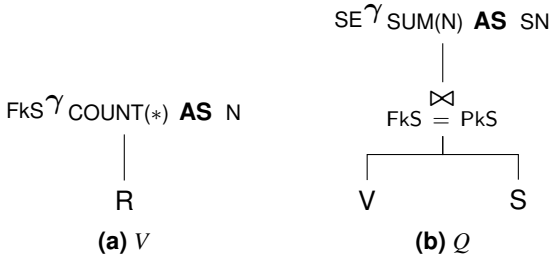
R(PkR, FkS, RC)
S(PkS, SE)

Show how the following query Q can be rewritten in SQL without the use of the view V.

```
CREATE  VIEW V AS
SELECT  FkS, COUNT(*) AS N
FROM    R
GROUP BY FkS;
```

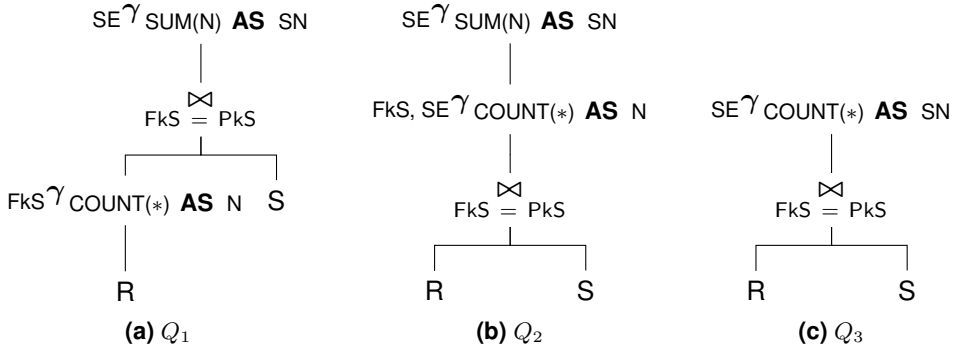
```
Q: SELECT  SE, SUM(N) AS SN
   FROM    V, S
   WHERE   FkS = PkS;
   GROUP BY SE;
```

Answer 12.6 Let us consider the logical query plans of V and Q:



(a) V **(b)** Q

Let us rewrite the Q logical plan, first by replacing the view V with its logical plan, then by pulling the view γ up above the join using the *pulling up grouping rule*, and finally by combining the two γ:



(a) Q_1 **(b)** Q_2 **(c)** Q_3

Now the query Q can be rewritten in SQL without the use of the view V.

```
Q:  SELECT    SE, COUNT(∗) AS SN
    FROM      R, S
    WHERE     FkS = PkS;
    GROUP BY  SE;
```

Exercise 12.7 Consider the following schema, where the keys are underlined:

R(PkR integer, FkRS integer, RA varchar(10), RB integer)
S(PkS integer, SA varchar(20), SB varchar(10), SC varchar(10))
T(FkTS integer, TA integer, TB integer)
FkTS is both a primary key for T and a foreign key for S.

and the following query:

```
SELECT  SA, TA
FROM    R, S, T
WHERE   FkRS = PkS  AND  PkS = FkTS
        AND  SC = 'P'  AND  RB > 130  AND  RA = 'B';
```

Suppose the following indexes exist on the relations:

- R: four unclustered B^+–tree indexes on PkR, FkRS, RA and RB.
- S: two unclustered B^+–tree indexes on PkS and SC.
- T: one unclustered B^+–tree index on FkTS.

The following information about the relations are available:

	S	R	T
N_{rec}	300	10 000	300
N_{pag}	66	110	18
N_{key}(IdxRB)		50 (min = 70, max = 160)	
N_{key}(IdxRA)		200	
N_{key}(IdxSC)	15		

Table 12.1: Statistics

Assume that the DBMS uses only the join physical operators **NestedLoop** and **IndexNestedLoop**, and only one index for selections.

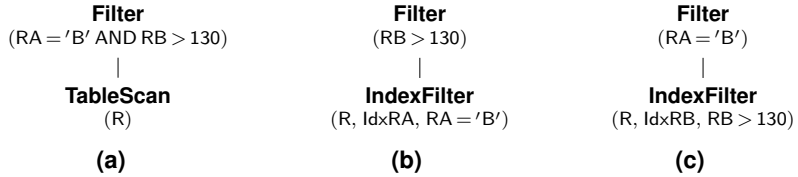
1. Give the query logical plan that the optimizer uses to find the physical plan.
2. Assuming that the optimizer uses a greedy search algorithm, give an estimate of the cost and of the result size of the physical plan for each relation, approximating and index access cost with only the C_D.
3. Which join physical plan for two relations will be selected in the second query optimization step?
4. What is the cost and the result size of the final best physical query plan?

Answer 12.7

1. The initial logical plan is transformed by pushing projections and selections below joins.
2. Assuming that the optimizer uses a greedy search algorithm, give an estimate of the cost and of the result size of the physical plan for each relation, approximating and index access cost with only the C_D.

   (a) Physical plans for the subexpression on R, $\sigma_{RA='B' \wedge RB>130}(R)$:
   The plans have the cost:

   $C_a = N_{\text{pag}}(\text{R}) = 110$



$$C_b = C_D$$
$$= \Phi(\lceil N_{\mathsf{rec}}(\mathrm{R})/N_{\mathsf{key}}(\mathrm{IdxRA})\rceil, N_{\mathsf{pag}}(\mathrm{R}))$$
$$= \Phi(50, 110) = 41$$

$$C_c = C_D$$
$$= f_s(\mathrm{RB} > 130) \times N_{\mathsf{key}}(\mathrm{RB})$$
$$\times \Phi(\lceil N_{\mathsf{rec}}(\mathrm{R})/N_{\mathsf{key}}(\mathrm{IdxRB})\rceil, N_{\mathsf{pag}}(\mathrm{R}))$$
$$= 1/3 \times 50 \times \Phi(200, 110) = 1550$$

The best physical plan is (b).

$$E_{\mathsf{rec}} = f_s(\mathrm{RB} > 130) \times f_s(\mathrm{RA} = \mathrm{'B'}) \times N_{\mathsf{rec}}(\mathrm{R})$$
$$= 1/200 \times 1/3 \times 10\,000 = 17$$

(b) Physical plans for the subexpression on S, $\sigma_{SC='P'}(S)$:

The plans have the cost:

$$C_a = N_{\mathsf{pag}}(\mathrm{S}) = 66$$

$$C_b = C_D$$
$$= \Phi(\lceil N_{\mathsf{rec}}(\mathrm{S})/N_{\mathsf{key}}(\mathrm{IdxSC})\rceil, N_{\mathsf{pag}}(\mathrm{S}))$$
$$= \Phi(20, 66) = 18$$

The best physical plan is (b).

$$E_{\mathsf{rec}} = f_s(\mathrm{SC} = \mathrm{'P'}) \times N_{\mathsf{rec}}(\mathrm{S}) = 1/15 \times 300 = 20$$

(c) Physical plans for the subexpression on T:

The plan has the cost:

$$C = N_{\mathsf{pag}}(\mathrm{T}) = 18$$

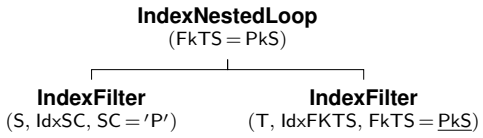
$$E_{\mathsf{rec}} = N_{\mathsf{rec}}(\mathrm{T}) = 300$$

3. Which join physical plan for two relations will be selected in the second query optimization step?

   The subexpression with the lowest physical plan cost is selected and, at the same cost, the one the produce less records: the subexpression on S. Then the optimization proceeds by expanding the best subexpression with the possible joins of two relations.

   For simplicity, for each possible join, if a physical plan with the **IndexNestedLoop** is possible, we will not show also the physical plan with the **NestedLoop**, if it has an higher cost.

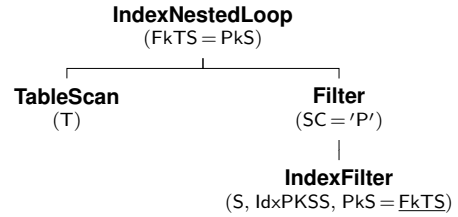
   (a) Physical plan for $(\sigma_{SC='P'}(S)) \overset{\bowtie}{\underset{C_j}{}} T$:



$$C_{\mathrm{INL_{ST}}} = C(O_E) + E_{\mathsf{rec}}(O_E) \times C(O_I) = 18 + 20 \times 1 = 38$$

$$\begin{aligned} E_{\mathsf{rec}}(\mathrm{INL_{ST}}) &= f_s(\mathrm{PkS} = \mathrm{FkTS}) \\ &\quad \times E_{\mathsf{rec}}(O_E) \times E_{\mathsf{rec}}(O_I) \\ &= 1/300 \times 20 \times 300 = 20 \end{aligned}$$

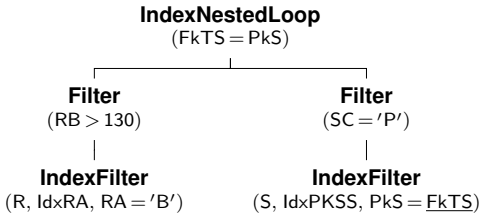
   (b) Physical plan for $T \overset{\bowtie}{\underset{C_j}{}} (\sigma_{SC='P'}(S))$:



$$C_{\mathrm{INL_{TS}}} = C(O_E) + E_{\mathsf{rec}}(O_E) \times C(O_I) = 18 + 300 \times 1 = 318$$

$$E_{\mathsf{rec}}(\mathrm{INL_{TS}}) = E_{\mathsf{rec}}(\mathrm{INL_{TS}}) = 20$$

   (c) Physical plan for $(\sigma_{RA='B' \wedge RB>130}(R)) \overset{\bowtie}{\underset{C_j}{}} (\sigma_{SC='P'}(S))$:



$$\begin{aligned} C_{\mathrm{INL_{RS}}} &= C(O_E) + E_{\mathsf{rec}}(O_E) \times C(O_I) \\ &= 41 + 17 \times 1 = 58 \end{aligned}$$

$$E_{\text{rec}}(\text{INL}_{\text{ST}}) = f_s(\text{FkRS = PkS}) \times E_{\text{rec}}(O_E) \times E_{\text{rec}}(O_I) = 1/300 \times 17 \times 20 = 2$$

(d) The physical plan for $(\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} (\sigma_{RA='B' \wedge RB>130}(R))$ has the higher cost 618.

The subexpression with the lowest cost is $(\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} T$, and the optimization proceeds by expanding it with the possible joins with R to find the final physical query plan.

4. What is the cost and the result size of the final best physical query plan?

The subexpression with the lowest cost $(\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} T$ it is expanded with the join with the subexpression on R

$$((\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} T) \overset{\bowtie}{}_{C_j} (\sigma_{RA='B' \wedge RB>130}(R))$$

The physical query plan with the best physical plan for $(\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} T$ as external operand, and a **NestedLoop** for the internal operand (Figure 12.6), has the cost

$$\begin{aligned} C &= C(O_E) + E_{\text{rec}}(O_E) \times C(O_I) \\ &= C_{\text{INL}_{\text{ST}}} + E_{\text{rec}}(\text{INL}_{\text{ST}}) \times C(O_I) \\ &= 38 + 20 \times 41 = 858 \end{aligned}$$

$$\begin{aligned} E_{\text{rec}} &= f_s(\text{PkS = FkRS}) \times E_{\text{rec}}(\text{INL}_{\text{ST}}) \times E_{\text{rec}}(O_I) \\ &= 1/300 \times 20 \times 17 = 2 \end{aligned}$$

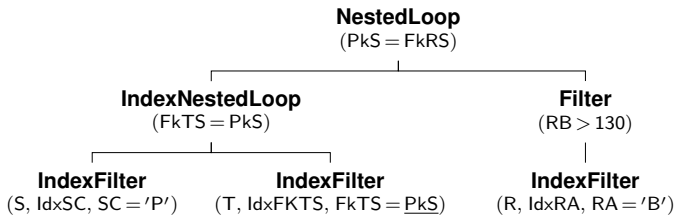


Figure 12.6: Physical query plan.

The physical query plan with the best physical plan for $(\sigma_{SC='P'}(S)) \overset{\bowtie}{}_{C_j} T$ as external operand, and an **IndexNestedLoop** for the internal operand (Figure 12.7), has the lower cost

$$\begin{aligned} C &= C(O_E) + E_{\text{rec}}(O_E) \times C(O_I) \\ &= C_{\text{INL}_{\text{ST}}} + E_{\text{rec}}(\text{INL}_{\text{ST}}) \times C(O_I) \\ &= C_{\text{INL}_{\text{ST}}} + E_{\text{rec}}(\text{INL}_{\text{ST}}) \times \Phi(\lceil N_{\text{rec}}(\text{R})/N_{\text{key}}(\text{IdxFKRS}) \rceil, N_{\text{pag}}(\text{R})) \\ &= 38 + 20 \times \Phi(34, 110) = 38 + 20 \times 30 = 638 \end{aligned}$$

The final physical query plan is the one in Figure 12.7 extended with the **Project**({SA, TA}).

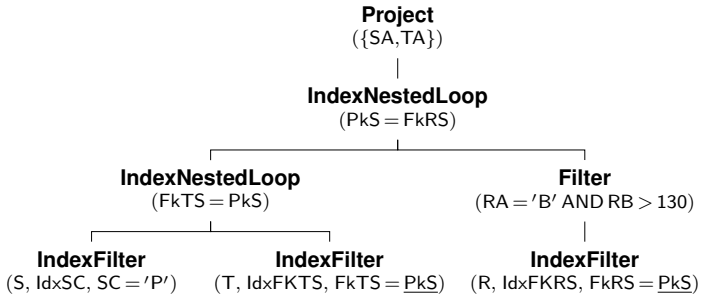


Figure 12.7: Final physical query plan.

Note that if the optimization had been done using the full search instead of the greedy search algorithm, the best physical query plan would be the one in Figure 12.8.

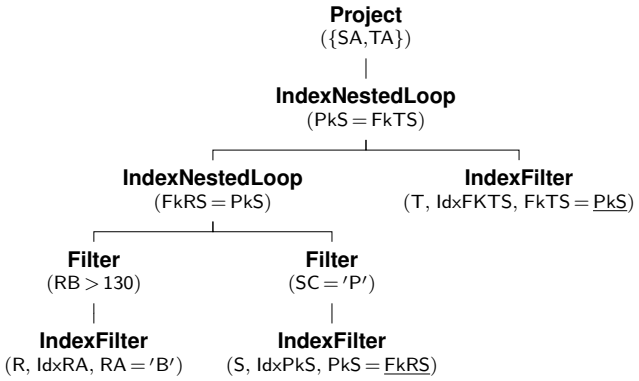


Figure 12.8: The best physical query plan.

$$
\begin{aligned}
C &= C(O_E) + E_{\text{rec}}(O_E) \times C(O_I) \\
&= C_{\text{INL}_{\text{RS}}} + E_{\text{rec}}(\text{INL}_{\text{RS}}) \times C(O_I) \\
&= 58 + 2 \times 1 = 60
\end{aligned}
$$