COMP2411 Database System Homework Assignment 2

Wang Ruijie

November 25, 2023

1 Question A

1.1 Answer of Question A (1)

A block anchor is the first record in a block. An index file for primary indexing of a data file is composed of block anchor primary key values and block pointers in pairs. That is, during a record query operation, the primary key value of the record being queried will be compared with the block anchor primary key values in the index file. Once the relevant block anchor is identified, the block pointer associated with the identified block anchor will locate the disk block of the queried record.

1.2 Answer of Question A (2)

A file can not have more than one primary or clustering index for primary or clustering index directly determines the physical order in the file based on one attribute, and a file in order can not have two kinds of sequences ordered by two or more irrelevant attributes. For instance, a student information sheet can not be ordered by students' HK IDs and passport IDs at the same time with the same sequence. If there are multiple primary or clustering indexes, they will disturb the others and destroy the physical order of the file.

However, secondary indexes do not define the physical order of the file, which provide diverse query accesses according to the given patterns. Multiple secondary indexes will not lead to contradiction among themselves.

1.3 Answer of Question A (3)

1. The number of index blocks of the clustering index based on *Category ID* B_i is $\lceil {}^{10\times(4+6)}/_{600} \rceil = 1$. The blocking factor *bf* is $\lfloor {}^{600}/_{100} \rfloor = 6$ ${}^{records}/_{block}$. Therefore, the number of blocks storing records with a specific *Category ID* $B_{CategoryID}$ is $\lceil {}^{4000}/_{10} \times {}^{1}/_{6} \rceil = 67$, and the total number of block accesses is $1 + 67 = 68$.

2. The number of index blocks of the secondary index based on *Director ID* B_i is $\lceil {}^{200\times(4+6)}/_{600} \rceil = 4$. the number of blocks storing records with a specific *Director ID* $B_{DirectorID}$ is $\lceil {}^{4000}/_{200} \times {}^{1}/_{6} \rceil = 4$, and the number of block accesses on the index is $\lceil log_2 4 \rceil = 2$. As the query request for all records with both a specific *Director ID* and a *Category ID*, we need to do traversal on all records with the given *Director ID*. Hence, the total number of block access is $2 + 4 = 6$.

3. Since there are 4000 records in total, a the size of a bitmap of a specific distinct value is 4000 *bits* = 500 *bytes*. Since each bitmap is stored as a fix-length record, two blocks is needed to store two bitmaps, as $\lceil {}^{2\times500}/_{600} \rceil = 2$. In all, only 2 block accesses is enough.

From the analysis above, the bitmap indexing is the most efficient one in conclusion.

2 Question B

2.1 Answer of Question B (1)

1. 1^{st} tuple:

A	B	A	C
2	z	2	6

2. 2^{nd} tuple:

A	B	A	C
7	x	7	8

3. 3^{rd} tuple:

A	B	A	C
9	y	9	2

4. 4^{th} tuple:

A	B	A	C
8	y	8	1

2.2 Answer of Question B (2)

1. The blocking factor of R $bf_R = \lfloor 1000/20 \rfloor = 50$ $^{records}/_{block}$. Hence, the number of blocks storing R $B_R = \lceil 100000/50 \rceil = 2000$.

2. The blocking factor of R $bf_S = \lfloor 1000/50 \rfloor = 20$ $^{records}/_{block}$. Hence, the number of blocks storing S $B_S = \lceil 50000/20 \rceil = 2500$.

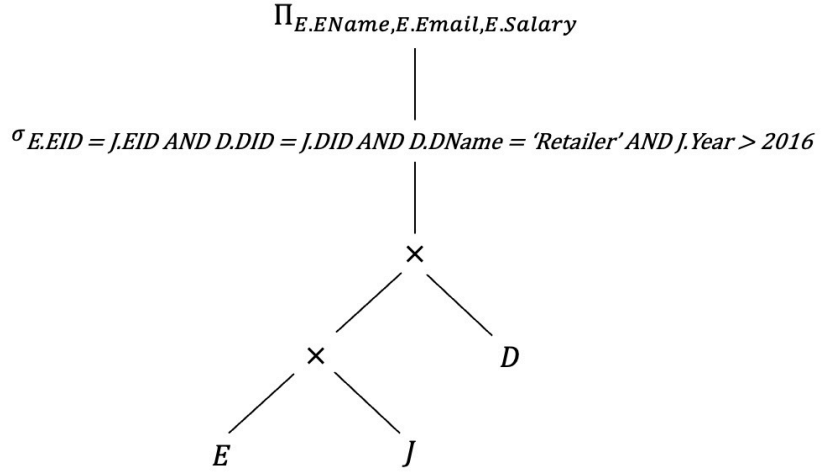
3. Therefore, the number of block accesses required is $B_R + B_R \cdot B_S = 5,002,000$

2.3 Answer of Question B (3)

The efficiency can not be improved. After the changing of the join order, The number of blocks for storing R and S, B_R and B_S will not alter. Hence, the block accesses should be $B_S + B_R \cdot B_S = 5,002,500 > 5,002,000$. The efficiency is deducted.

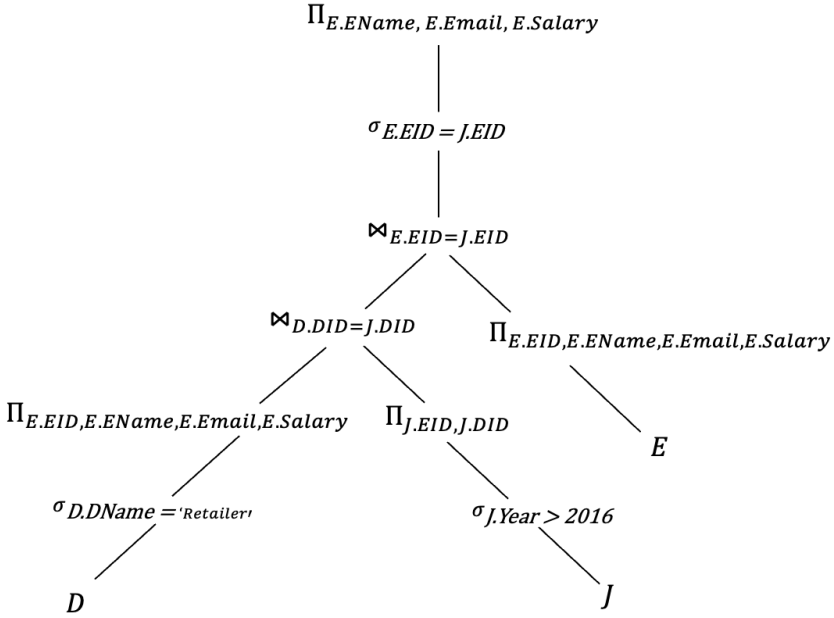
2.4 Answer of Question B (4)

The initial query tree generated by the conceptual evaluation strategy:



2.5 Answer of Question B (5)

The optimized query tree is as follows:



3 Question C

3.1 Answer of Question C (1)

1. *Basic 2PL* divides a transaction into two parts: the *growing phase* and the *shrinking phase*. In that case, a transaction can only acquire locks on items during the *growing phase* and release locks during the *shrinking phase*.

   Basic 2PL allows the given schedule S, and the reason is as follows:

   In process of $W_1(X)$; $R_2(Y)$; $R_1(Y)$; ..., the two transactions are all acquiring locks and T_2 does not request for operation on X, which is locked by T_1*'s* write lock (exclusive lock). After $R_1(Y)$, T_1 enters the shrinking phase and it is able to release its lock on X, thus $R_2(X)$ can be executed successfully.

2. Under the protocol of *Conservative 2PL*, a transaction must acquire locks on items that it will read or do operation on before its execution. After that, the transaction can gradually release its locks on items.

   Conservative 2PL does not allow the schedule S. The reason is as follows:

   At the beginning of the execution of the two transactions, T_1 acquires the exclusive write lock on the item X, while T_2 also requests to add a read lock on the same item. That causes a problem and is not allowed.

3. Under the protocol of *Strict 2PL*, a transaction is unable to release any lock until it ends, while the protocol does not require that the lock must be acquired at the same time.

   Strict 2PL does not allow the schedule S, and the reason is as follows:

   T_1*'s* write lock on item X will be held until T_1 commits. However, before T_1*'s* commitment, T_2 has requested for having the read lock on X. As the write lock is exclusive, such situation leads to a conflict between the transactions.

3.2 Answer of Question C (2)

1. The anomaly occurs is *Dirty Read* (Write-read Conflict), because after X being written by T_1 and before the change is confirmed (aborted), T_2 reads X and write X based on its reading. The read is inconsistent.

2. T_1 : *WL(X); R(X); W(X); WL(Y); R(Y); W(Y); UL(X); UL(Y)*

   T_2 : *WL(X); R(X); W(X); UL(X)*

3. $R_1(X)$; $W_1(X)$; $R_1(Y)$; $W_1(Y)$; A_1; $R_2(X)$; $W_2(X)$; C_2

   The new schedule can prevent *Dirty read* problem, since before T_2 begins to read and write on item X, all previous influence made by T_1 has been aborted and therefore, the two transactions are isolated.