W4111 - Introduction to Databases
Section 02, Fall 2019

Name: Wenjie Chen; uni: wc2685

Question 1: Suppose you have data that should not be lost on disk failure, and the application is write-intensive. How would you store the data? (Note: Your answer should consider differences between primarily sequential writes and primarily random writes).

Answer:

Cited: professor's lecture video

I would store the data with RAID (Redundant Arrays of Independent Disks). It is disk organization techniques that manage a large numbers of disks, providing a view of a single disk of high capacity, high speed, and high reliability. And the redundancy can store extra information that can be used to rebuild information lost in a disk failure.

For write-intensive data with mostly sequential writes, RAID 1 and RAID 5 will both perform well, but with less storage overhead for RAID 5. If writes are random, RAID 1 is better. If protecting from two-disk failure is needed, using RAID 1 but with three-way replication instead of mirroring is good.

Question 2: Both database management systems (DBMS) and operating systems (OS) provide access to files and implement optimizations for file data access. A DBMS provides significantly better performance for data in databases. How does the DBMS do this? What information does it use? How does it optimize access? Provide some examples.

Answer:

Cited: professor's OH video

DBMS is primarily a software system that can be considered as a management console or an interface to interact with and manage databases. It contains operational data, access to database records and metadata as a resource to perform the necessary functionality.

And DBMS has access to two sources of information that enable predicting future access patterns: 1. Predict future due to scans in SELECT and JOIN; 2. Statistical information about data and historical queries.

Some techniques that DBMS can use:

1. Prefetch data by predicting future block access due to scans.
2. Optimize placement of blocks of data that is sequentially accessed.
3. Proper indexing.
4. Retrieve the relevant data only.
5. Avoid coding loops.
6. Execution plan tools.

Question 3: Briefly explain CHS addressing and LBA for disks. Which approach is more common and why? Is it possible to convert a CHS address to an LBA. If yes, how?

Answer:

Cited: *https://www.thomas-krenn.com/en/wiki/CHS_and_LBA_Hard_Disk_Addresses*

CHS is Cylinder-header-sector, which is an early method for giving address to each physical block of data on a hard disk drive; It is a 3D-coordinate system made out of a vertical coordinate head, a horizontal (or radial) coordinate cylinder, and an angular coordinate sector. LBA is Logical block addressing, which is a common scheme used for specifying the location of blocks of data stored on disks; It is a linear addressing scheme; blocks are located by an integer index, with the first block being LBA 0.

LBA is more common now. Because CHS schems expose the physical details of the storage device to the software of the operating system. However LBA does not need to do that. And using LBA, the hard disk is simply addressed as a single, large device, which simply counts the existing blocks starting at 0.

Yes. An LBA block corresponds to a sector using CHS addressing. When mapped to CHS tuples, LBA numbering starts with the first cylinder, first head, and track's first sector. Then the second head, and so on. The logical block address can be computed. $LBA=(C*HPC+H)*SPT+(S-1)$ where C, H, S refers to

the cylinder number, the head number, and the sector number. HPC refers to the maximum number of the heads per cylinder, and SPT refers to the maximum number of sectors per track.

Question 4: Explain why the allocation of records to blocks affects database-system performance significantly.

Answer:

Disk accesses tend to be the bottlenecks in databases. For example, when records are related to blocks, most of the requested records in a query can be retrived with a single disk access. If we allocate the records with a allocation strategy, it reduce the number of disk accesses for a given request, thus significantly improves performance.

Question 5: Give benefits and disadvantages of variable length record management versus fixed length record management

Answer:

Variable length record management:

Benefits: less storage space since it uses only as much as storage as is needed; able to accomodate unusual data not originally planned; more intuitive for humans to think about. **Disadvantages:** difficult to insert or delete; hard to search through.

Fixed length records:

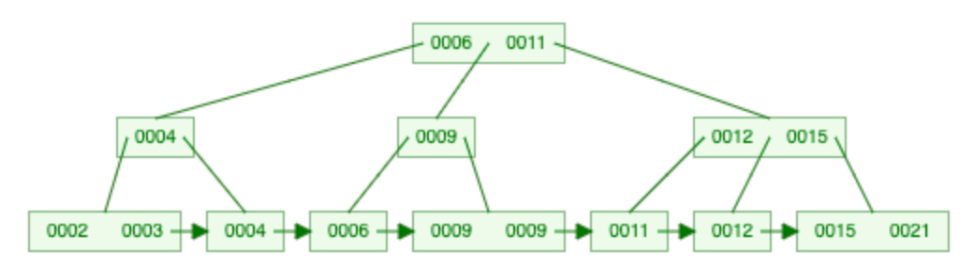
Benefits: easy allocating and easy searching. **Disadvantage:** not flexible and hard to change its length.

Question 6: Build and draw a B+ tree after inserting the following values. Assume the maximum degree of the B+ tree is 3.

Values: 3, 11, 12, 9, 4, 6, 21, 9, 15, 2

Answer:

Cited: https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html



Question 7: Perform the same insertions in the same order for a hash index. Assume that:

1. The size of the hash table is 13.
2. The hash function is simple modulo.
3. The size of a bucket is one entry.
4. The size of each bucket is one value.
5. The index algorithm uses linear probing to resolve conflicts/duplicates.

Answer:

Cited: http://iswsa.acm.org/mphf/openDSAPerfectHashAnimation/perfectHashAV.html

0	1	2	3	4	5	6	7	8	9	10	11	12
		15	3	4	2	6		21	9	9	11	12

Question 8: When is it preferable to use a dense index rather than a sparse index? Explain your answer.

Answer:

When the file is not sorted on the indexed field. The indexed filed refers to the field being considered for the new index, for example, a sparse index relies on the records being sorted in the same order as the indexed field.

Or when the index file is small compared to the size of memory. If there is still enough space for index, then database engine is able to find each record in the index and go directly to the data.

Question 9: Since indexes improve search/lookup performance, why not create an index on every combination of columns?

Answer:

Cited: Database system book

Although indexes improves search performance, creating index for every combination of columns is unnecessary. Because each index is system managed, so every modification (such as update, delete) to the data in a user table potentially involves updating the indexes, thus slowing down the performance of data updates. So when columns are frequently updated, index affects its spead. What's more, if number of columns is small or columns are rarely used, index would cost extra memory space. And indexes requires more computation power.

Question 10: Consider the table below. Add indexes that you think are appropriate for the table and explain your choices. You may use MySQL Workbench to add the indexes. Paste the resulting create statement in the answer section.

...

Answer:

Use Case 1: A user wants to be able find a customer(s) by country_region; country_region and state_province; country_region, state_province, city; just by city.

```
Create index id1 on customers (country_region,
state_province, city)
Create index id2 on customers (city)
```

Use Case 2: A user wants to be able find a customer(s) by email_address.

```
Create Unique index email_addr on customers
(email_address)
```

Use Case 3: A user wants to ensure the mobile-phone is unique.

```
Create Unique index phone on customers (mobile_phone)
```

Use Case 4: A user wants to find customers by company and job_title.

```
Create index company_title on customers
(company,job_title)
```

Use Case 5: A user wants to find customers by city and address.

```
Create index city_address on customers (city,address)
```

Question 11: Assume that:

1. The query processing engine can use four blocks in the buffer pool to hold disk blocks.
2. The records are fixed size, and each block contains 3 records.
3. There are two relations are R and S. Their formats on disk are:

...

Explain how a partition hash join would perform a natural join. You should illustrate your explanation using diagrams of the form below for various steps in the processing:

...

The notation (n,X) means the record in file/relation X with value n for the key. Ti is a temporary file/relation that is created during the processing. You will likely have to create more than one temporary files.

Answer:

First we use hash (mod 3) to separate R and S into 3 indexes in each bucket.

The example of the first step for R:

Buffer pool

(3,T1),(6,T1),(9,T1)
(1,R),(2,R),(3,R)
(4,R),(5,R),(6,R)
(7,R),(8,R),(9,R)

Files

R	**(1,R),(2,R),(3,R)**	**(4,R),(5,R),(6,R)**	**(7,R),(8,R),(9,R)**
S	(11,S),(4,S),(21,S)	(3,S),(31,S),(13,S)	(5,S),(27,S),(23,S)
T1	(3,T1),(6,T1),(9,T1)		
T2			

Repeat for all and get:

Files

R	**(1,R),(2,R),(3,R)**	**(4,R),(5,R),(6,R)**	**(7,R),(8,R),(9,R)**
S	(11,S),(4,S),(21,S)	(3,S),(31,S),(13,S)	(5,S),(27,S),(23,S)
T1	(3,T1),(6,T1),(9,T1)	(1,T1),(4,T1),(7,T1)	(2,T1),(5,T1),(8,T1)
T2	(21,T2),(3,T2),(27,T2)	(4,T2),(31,T2),(13,T2)	(11,T2),(5,T2),(23,T2)

Result:

T1	**(3,T1),(6,T1),(9,T1)**	**(1,T1),(4,T1),(7,T1)**	**(2,T1),(5,T1),(8,T1)**
T2	(21,T2),(3,T2),(27,T2)	(4,T2),(31,T2),(13,T2)	(11,T2),(5,T2),(23,T2)

We load T1[0] into buffer and perform hash to the records. And match the records by the hash. Output the result.

OUTPUT: (3,T1,T2)

(21,T2),(3,T2),(27,T2)
(3,T1),(6,T1),(9,T1)

Repeat for T1[1], T1[2] and T2[1], T2[2].

OUTPUT: (4,T1,T2)

(4,T2),(31,T2),(13,T2)
(1,T1),(4,t1),(7,t1)

OUTPUT: (5,T1,T2)

(11,T2),(5,T2),(23,T2)
(2,T1),(5,T1),(8,T1)

Finally we get (3,F),(4,F),(5,F), which is the join result.

F	(3,F)	(4,F)	(5,F)

Question 12: Give three reasons why a query processing engine might use a sort-merge join instead of a hash join? What are the key differences sort-merge and hash join?

Answer:

1.When the join condition columns are sorted. If sorting is requred due to other attribute, sort-merge join is cheaper than hash join. Because the cost of the sort-merge join can be lowered if one of the data sets can be accessed in sorted order via an index. 2. Sort-merge join perform better when the join condition is an inequality condition (such as <, >), because hash join cannot be used for inquality conditions. 3. If hash table grows too big to fit into the memory space so that part of it are written to temporary segments on disk. The cost of hash table will be higher than that of using sort-merge join. 4. The number of unnecessary comparisons is relatively low in any situation.

The key differences between them is that sort merge join is performed by sorting the two data sets to be joined according to the join keys and then merging them together, while hash join is performed by hashing one data set into memory based on join columns and reading the other one and probing the hash table for matches, and it's used only in equi-join.

Question 13: Let r and s be relations with no indices, and assume that the relations are not sorted. Assuming infinite memory, what is the lowest-cost way (in terms of I/O operations) to compute ...

Answer:

Cited: https://dev.mysql.com/doc/refman/5.5/en/select-optimization.html

First we can store the entire small relation in memory, read the larger relation block by block. Then we implemente nested loop join using the larger one as the outer relation. Then memory requirement should be min(operation of r, operation of s)+2, which is $min(b_r,b_s)+2$ pages.

Question 14

Question:

Rewrite/transform the following query into an equivalent query that would be significantly more efficient.

...

...

Answer:

```
SELECT
a.playerid, a.nameLast, a.throws, b.teamid, b.yearid,
b.ab, b.h, b.rbi
FROM
(SELECT playerid, nameLast, throws FROM people) as a
JOIN
(SELECT teamid, yearid, ab, h, rbi, playerid FROM
batting WHERE teamid='BOS' and yearID='1960') as b
USING (playerid)
```

Question 15: Suppose that a B+-tree index on (dept_name, building) is available on relation department. (Note: This data comes from the database at https://www.db-book.com/db7/index.html)

What would be the best way to handle the following selection?

...

Answer:

By using B+-tree index on (dept_name, building), we locate the first tuple having dept_name=music and building<Watson. From the tuples got, select the ones that satisfies the condition (budget<55000).

Question 16: Consider the following relational algebra expression

...

This is a project on the result of a natural join on R and S. Assume that column *a* comes from R, column *b* comes from S and that *c* is the join column. Also assume that both R and S have many large columns. Write an equivalent query that will be more efficient, and explain why.

Answer:

$$(\pi_{a,c}R) \bowtie (\pi_{b,c}S)$$

The natural join of R and S require a lot more computation power. We can see that the original query require the query processor to scan all the data in R and S to find the requered data. But in the query above, first we only need to make a projection on a,c from R, projection on b,c from S, then join them together, which is more efficient than the previous one.

Question 17: For each of the following isolation levels, give an example of schedule that respects the specified level of isolation but is not serializable:

a. Read uncommitted

b. Read committed

c. Repeatable read

Answer:

Cited: https://www.geeksforgeeks.org/transaction-isolation-levels-dbms/

a. Read Uncommitted:

In the following schedule, before T1 commits, T2 READS(A). This schedule is read uncommitted since T1 also READ(A) written by T2.

T1	T2
READ(A)	
WRITE(A)	
	READ(A)
	WRITE(A)
READ(A)	

b. Read Committed:

In the following schedule, T1 READ(A) and then T2 WRITE(A). Then the value written by T2 is committed. The second READ(A) shows that T2 precedes T1. Thus it is read commiteed.

T1	T2
LOCK-S(A)	
READ(A)	
UNLOCK(A)	
	LOCK-X(A)
	WRITE(A)
	UNLOCK(A)
	COMMIT
LOCK-S(A)	
READ(A)	
UNLOCK-S(A)	
COMMIT	

c. Repeatable read:

In the following schedule, to satisfy repeatable read, it must also share-lock these tuples in a two-phase manner.

T1	T2
PRED READ(r, p)	
	INSERT(t)
	WRITE(A)
	COMMIT
READ(A)	
COMMIT	

Question 18: Explain the difference between a *serial schedule* and a *serializable schedule*.

Answer:

A serial schedule is a schedule in which the transactions are executed non-interleaved and all the operations of one transactions appear together, while a serializable schedule is a weaker term. For the serializable schedule, the operations of different transactions may be mixed together on it, so long as they are conflict-equivalent to some serial schedule.

Question 19: What are the benefits and disadvantages of strict two phase locking?

Answer:

The benefit is that recovery is easy, because it provides only cascadeless schedules. The disadvantage is that concurrency is reduced, because the set of schedules obtainable is a subset of those obtainable from plain two phase locking.

Question 20: Outline the no-steal and force buffer management policies.

Answer:

No-steal means all pages are retained in the buffer pool until the the transaction commits. Force buffer management policies mean after the transaction's commit, the buffer manager locates the modified pages and writes them into disk.

Put it together, it means that the buffer page updated by a transaction cannot be written to disk before the transaction commits, and updated pages are written to disk when a transaction commits.