

Homework 3: Database Management

Question 1

Question: 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:

A RAID array can handle the failure of a single drive (two drives in the case of RAID 6) without data loss and is relatively inexpensive. There are several RAID alternatives, each with different performance and cost implications. 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 preferred, since a random block write requires multiple reads and writes in RAID 5. If you wish to protect from two-disk failure, using RAID 1 but with three-way replication instead of mirroring is an option.

Cited from Office hours 11/18/2019

Question 2

Question: 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:

DBMS provides better performance for data in databases by storing the data in a structured manner in tables (or references) and providing appropriate queries to access the required data. This is not the case in OS because in OS access to the files is done via command line or

via interactive features of the GUI format. Therefore accessing the exact required data via some specified queries is easier and efficient than the methods provided by the person OS. A DBMS can predict, with some errors, query access patterns. The DBMS can predict future record block access during scan for SELECT processing and can predict future block access for JOIN processing. The DBMS can gather statistical information about record/block access to infer future access patterns for tables, records and blocks.

The information used by DBMS includes the data to be accessed and the table from which it needs to be accessed along with some additional features which the user might want to provide to get the exact data.

Optimization of data is done by ensuring that each table consists of data relevant to the domain of that table and the database in which that table is present. Moreover, for access of data from one or more table, the facilities of JOIN and VIEWS are available. This optimizes the task of the DBMS, which is to manage the data and provide it to users in the most efficient manner.

Some techniques that the DBMS that the DBMS can apply are: Placing sequentially accessed blocks in order in the same cylinder and using adjacent cylinders if the number of blocks exceeds a cylinder's capacity. Prefetching data into the buffer pool when a scan is in progress to implement a SELECT query.

Cited from Office Hours 11/18/2019

Question 3

Question: 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:

The Cylinder-head-sector CHS addressing is an early method for giving addresses to each physical block of data on a hard disk drive. The Logical block addressing LBA is a common scheme used for specifying the location of blocks of data stored on computer storage devices, generally secondary storage systems such as hard disk drive. LBA is more common since

compared to CHS addressing, LBA is simpler, more flexible, and supports larger capacities. It's possible to convert.

The formula is $(C * HPC + H) * SPT + (S - 1) = LBA$

where C,H and S are the cylinder number, the head number and the sector number. LBA is the logical block address, HPC is the maximum number of heads per cylinder (reported by disk drive, typically 16 for 28-bit LBA), HPC is the maximum number of heads per cylinder (reported by disk drive, typically 16 for 28-bit LBA), SPT is the maximum number of sectors per track (reported by disk drive, typically 63 for 28-bit LBA)

Cited from https://en.wikipedia.org/wiki/Logical_block_addressing

Question 4

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

Answer:

If records are related to blocks, we can often retrieve most or all the requested records in a query with a single disk access (i.e. if the table is indexed). Disk accesses tend to be the bottlenecks in databases; since this allocation strategy reduces the number of disk accesses for a given operation, it significantly improves performance.

Cited from https://www2.cs.sfu.ca/CourseCentral/354/louie/Chap10_practice_key.pdf

Question 5

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

Answer:

Variable length records:

Benefits: 1. Use only as much storage as is needed. 2. Can accommodate unusual data not originally planned. 3. Most natural way for humans.

Disadvantages: 1. Difficult to insert or delete. 2. Hard to search through. 3. Cannot simply map a C struct to a record.

Fixed length records:

Benefits: Easy allocation, deallocation and searching, the space is pre-allocated and therefore always available to the database.

Disadvantages: Not flexible and hard to change a field length

Cited from http://www.eli.sdsu.edu/courses/spring95/cs596_3/notes/databases/lect10.html

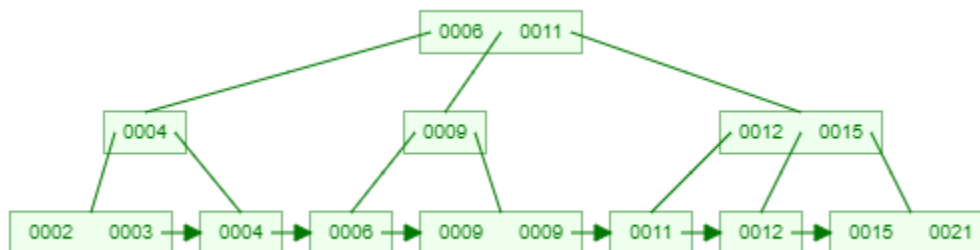
Question 6

Question: 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:



B+ Tree generated from <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

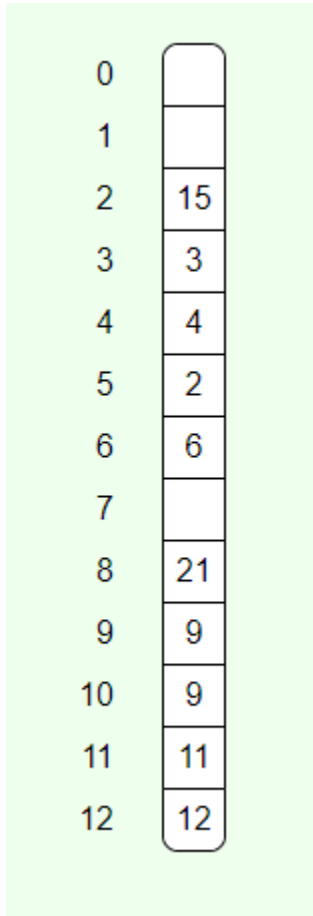
Question 7

Question: 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:



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

Hashing table generated from
<http://iswsa.acm.org/mpfh/openDSAPerfectHashAnimation/perfectHashAV.html>

Question 8

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

Answer:

It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file is small compared to the size of memory. Dense indices are faster in general, but sparse indices require less space and impose less maintenance for insertions and deletions.

Cited from <http://web.cs.ucla.edu/classes/fall04/cs143/solutions/chap12a.pdf>,
<https://stackoverflow.com/questions/36808877/difference-between-sparse-index-and-dense-index>

Question 9

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

Answer:

Index is a table or other data structure, which is used to determine the file from a specific location that satisfies some condition. Index plays an important role in physical database design. The reason why we cannot create an index on every combination of columns: The necessity of indexing is not based upon the costs. Indexing is used with the caution for databases that help substantial transaction handling necessities. Indexing every column in a table might lead to very costly for updating operations of databases. There is a swapping between the improved performance of retrieval operations using indexes and degraded performance. It takes up space in memory and they also increase insert and delete time.

Cited from <https://stackoverflow.com/questions/5446124/mysql-why-not-index-every-field>

Question 10

Question: 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. Choosing indexes is not possible without understanding use cases/access patterns. Define five use cases and the index you define to support the use case. See the answer section for an example.

```
CREATE TABLE IF NOT EXISTS `customers` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `company` VARCHAR(50) NULL DEFAULT NULL,  
  `last_name` VARCHAR(50) NULL DEFAULT NULL,
```

```
`first_name` VARCHAR(50) NULL DEFAULT NULL,  
`email_address` VARCHAR(50) NULL DEFAULT NULL,  
`job_title` VARCHAR(50) NULL DEFAULT NULL,  
`business_phone` VARCHAR(25) NULL DEFAULT NULL,  
`home_phone` VARCHAR(25) NULL DEFAULT NULL,  
`mobile_phone` VARCHAR(25) NULL DEFAULT NULL,  
`fax_number` VARCHAR(25) NULL DEFAULT NULL,  
`address` LONGTEXT NULL DEFAULT NULL,  
`city` VARCHAR(50) NULL DEFAULT NULL,  
`state_province` VARCHAR(50) NULL DEFAULT NULL,  
`zip_postal_code` VARCHAR(15) NULL DEFAULT NULL,  
`country_region` VARCHAR(50) NULL DEFAULT NULL)
```

Answer:

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

state_province; country_region, state_province, city; just by city.

```
CREATE INDEX idx_location_region ON customers(country_region);
```

```
CREATE INDEX idx_location_province ON customers(country_region, state_province);
```

```
CREATE INDEX idx_location_city ON customers(country_region, state_province, city);
```

```
CREATE INDEX idx_city ON customers(city);
```

The first two index could be ignored since the third index, it could support the first two.

Use Case 2: A user wants to ensure that email addresses are unique.

```
CREATE UNIQUE INDEX uidx_email ON customers(email_address);
```

Use Case 3: A user wants to be able to find a customer by email_address.

```
CREATE INDEX idx_email ON customers(email_address);
```

Use Case 4: A user wants to be able to find a customer(s) by last_name; last_name and first_name.

```
CREATE INDEX idx_lastname ON customers(last_name);
CREATE INDEX idx_name ON customers(last_name, first_name);
```

Use Case 5: A user wants to ensure that mobile_phone is unique.

```
CREATE UNIQUE INDEX uidx_phone ON customers(mobile_phone);
```

Question 11

Question: 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:

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)

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:

Step N:

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

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:

Step 1: Load R into Buffer Pool

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

Step 2: Find 0 Mod 3 for tuples in Buffer Pool and store into T1

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

Step 3: Find 1 Mod 3 for tuples in Buffer Pool and store into T1

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

Step 4: Find 2 Mod 3 for tuples in Buffer Pool and store into T1

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

Step 5: Do the same thing for S and store into T2, 0 Mod 3

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

Step 6: Do the same thing for S and store into T2, 1 Mod 3

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

Step 7: Do the same thing for S and store into T2, 2 Mod 3

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

Step 8: Load T1 and T2 and find intersection and store them into RS, starting from first column

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

Step 9: Second Column

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

Step 10: Third Column

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

Step 11: Thus we get (3,RS), (4,RS) and (5,RS)

Question 12

Question: 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:

Three reasons why a query processing engine might use a sort-merge join instead of a hash join:

Sort merge join is used to join two independent data sources. It performs better than hash join when the join condition columns are already sorted or there is no sorting required. Hash joins are used when the joining large tables. These sort-merge joins is used from the two independent resources to make the search efficient in the query processing engine.

Give efficient costs while joining. If we use the sort-merge join, then the unnecessary comparisons in that case will be gradually reduced so the efficiency can automatically increase.

The optimizer uses smaller of the 2 tables to build a hash table in memory and then scans the large tables and compares the hash value (of rows from large table) with this hash table to find the joined rows. Sort-merge join is mainly a robust, when only there are no indexes exist in their attributes.

Key differences: Sort-merge join is only performed between the two datasets while hash join is only performed in the one dataset. Operations through the sort-merge join are bit slower while operations through the hash join are very fast.

Cited from <https://stackoverflow.com/questions/1111707/what-is-the-difference-between-a-hash-join-and-a-merge-join-oracle-rdbms>

Question 13

Question:

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 $r \bowtie s$? What is the amount of memory required for this algorithm?

Answer:

We can store the entire smaller relation in memory, read the larger relation block by block and perform nested loop join using the larger one as the outer relation. The number of I/O operations is equal to $b_r + b_s$, and memory requirement is $\min(b_r, b_s) + 2$ pages.

Cited from <https://www.db-book.com/db6/practice-exer-dir/12s.pdf>

Question 14

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

Select

people.playerid, people.nameLast, peoplethrows, batting.teamid, batting.yearid, ab,
 h, rbi
 from
 (people join batting using(playerid))
 where teamid='BOS' and yearID='1960';

Answer:

SELECT *
 FROM (SELECT playerid, nameLast, throws FROM people) as a
 JOIN (SELECT playerid, teamid, yearid, ab, h, rbi FROM batting WHERE teamid='BOS'
 and yearid='1960') as b
 using(playerid);

Question 15

Question: 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](https://www.db-book.com/db7/index.html)) What would be the best way to handle the following selection?

$$\sigma_{(building < \text{"Watson"}) \wedge (budget < 55000) \wedge (dept_name = \text{"Music"})}(department)$$

Answer:

Using indexing, locate the first tuple having dept_name=music. Now retrieve successive
 tuples using pointers as long as building is less than Watson. From the tuples retrived, the
 ones not satisfying the condition (budget<55000) are rejected.

Cited from lecture slides and recorded lectures

Question 16

Question: Consider the following relational algebra expression

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

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:

$$\pi_{a,b,c} (\sigma_{a,c}(R) \bowtie_{R.c=S.c} \sigma_{b,c}(S))$$

```
SELECT *
FROM (SELECT a, c from R) as r
JOIN (SELECT b, c from S) as s
using(c)
```

If we just natural join R and S together then choose the column we want to see, it will result two large tables with all columns to be joined. There are a lot of unnecessary columns to be joined and dropped. Using this query, we will have two smaller table with less amount of columns to be joined together and this will be more efficient since we only choose the columns we need and then join these columns together.

Question 17

Question:

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

- Read uncommitted
- Read committed
- Repeatable read

Answer:

- Read uncommitted:

T1	T2
----	----

read(A) write(A)	
	read(A) write(A)
read(A)	

Transaction T2 reads the value of data item A written by transaction T1 even though T1 is not committed yet. Also this schedule is not serializable as there exists a cycle in the precedence graph of this schedule because T1 reads data written by T2 and T2 reads data item written by T1.

b. Read committed:

T1	T2
lock-S(A) read(A) unlock(A)	
	lock-X(A) write(A) unlock(A) commit
lock-S(A) read(A) unlock(A) commit	

Transaction T1 reads data item A, it sees a-value of A before it was written by T2. It results in T1 preceding T2. When T1 reads A the second time, it sees the value written by T2. It results in T2 preceding T1. Therefore, there is a cycle in the precedence graph and the schedule is not serializable.

c. Repeatable read:

T1	T2
pred_read(r, P)	
	Insert(t) Write(A) commit
Read(A) commit	

Insert record t by transaction T2 satisfy predicate P. Since T1 does not see t, so the insert by T2 causes that T2 to serialize T1. But the final read(A) by transaction T1 forces T1 to precede T2. Therefore, there is a cycle in the precedence graph and the schedule is not serializable.

Cited from <https://www.coursehero.com/file/38478299/C14pdf/?justUnlocked=1#/doc/qa>

Question 18

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

Answer:

A schedule in which all the instructions belonging to one single transaction appear together is called a serial schedule. A serial schedule is not inter-leaved, it executes each transaction in isolation. Different order of transactions might produce separate final values. The outcome depends on the order of execution of transactions. A serializable schedule has a weaker restriction that it should be equivalent to some serial schedule. They are inter-leaved and not executed in sequence. It does not depend on the initial values of database items and the outcome depends only on order of instructions. A schedule S is serializable schedule if and only if there is a serial schedule S' such that the effect of the execution of S and S' are identical for every DB state.

Cited from <https://secweb.cs.odu.edu/~cs450/450/pdf/view-on-line/serial/serial3.pdf>

Question 19

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

Answer:

In the strict two-phase locking protocol. Locks are obtained and released in two separate phases just like two-phase locking protocol. Additionally, it also requires that all the exclusive mode locks should be held by the transaction till it commits.

Benefits: It ensures serializability just like two-phase locking protocol. Also, it produces only cascade less schedules, recovery is very easy. Since the data written by an uncommitted transaction cannot be read until it commits. It is simpler to implement as compared to other locking protocols. The level of concurrency allowed by it is acceptable.

Disadvantages: it reduces concurrency as the set of resultant schedules is a subset of those obtained from the plain two-phase locking protocol and transactions end up waiting.

Cited from <http://web.cs.ucla.edu/classes/fall04/cs143/solutions/chap16a.pdf>

Question 20

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

Answer:

For no-steal policy, it does not allow buffer-pool frames with uncommitted updates to overwrite committed data on disk. It is useful for ensuring atomicity without UNDO logging but can cause poor performance. No-steal means that the cache (buffer) page updated by a transaction cannot be written to disk before the transaction commits. With No-steal, the checkpoint scheme that writes all modified main memory buffers to disk would not be able to write pages updated by uncommitted transactions.

For force policy, it makes sure that every update is on disk before commit. It provides durability without REDO logging but can cause poor performance as well. Force means that updated pages are written to disk when a transaction commit. With Force, once a transaction is done, its updates would be pushed to disk. If there is a failure during this, REDO is still needed; however, no UNDO is needed since uncommitted updates are never propagated to disk.

Cited from http://db.cs.berkeley.edu/jmh/cs186/f02/lecs/lec25_6up.pdf

