**Name: Justin Senia**

**EID: E00851822**

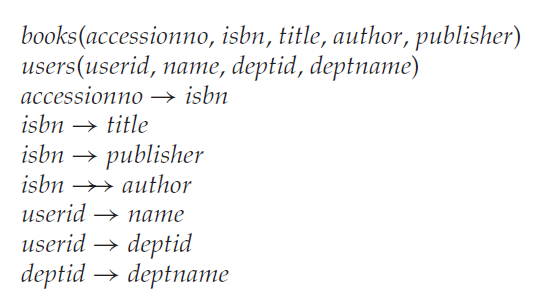
**Date: 12/9/2017**

**Class: COSC 471 T 10:00am-11:50am**

**Assignment: Homework\_4**

**Chapter 7**

**8.21 (4 pts.) Use the schema given in this exercise. Ignore the double arrow and assume that it is a single arrow that represents a functional dependency. Take each of the two relations and specify which normal form they are in. Then normalize them to 3NF. Hint: AccessionNo and userid are the primary keys.**

****

books

(accessionno, isbn, title, author, publisher)

is in 2NF because every non prime-attribute is fully functionally dependent on the primary key “accessionno”

users

(userid, name, deptid, deptname)

Is in 1NF because it follows the rules of 1NF but fails to meet the requirements of 2NF because “name” could be removed and it wouldn’t affect “deptid” or “deptname”

books (2NF)

(accessionno, isbn, title, author, publisher)

\/

accession (3NF)

(accessionno, isbn)

book (3NF)

(isbn, title, author, publisher)

users (1NF)

(userid, name, deptid, deptname)

\/

users (3NF)

(userid, name, deptid)

dept (3NF)

(deptid, deptname)

**8.35 (5 pts.) – This question is not included in the textbook.**

**Consider the relation R(A,B,C,D,E,F,G,H) in which ABC is the primary key. If the following dependencies hold in this relation, is this relation in 3NF? If not, which normal form does it satisfy? Reduce it to 3NF.**

**AB -> D**

**C -> H**

**EF-> G**

No, it’s not in 3NF.

It’s in 1NF.

\*Converting to 3NF\*

R1 (3NF)

(A,B,C,D)

R2 (3NF)

(C,H)

R3 (3NF)

(E,F,G)

**Chapter 11**

**11.15 (2 pts.) When is it preferable to use a dense index rather than a sparse index? Explain your answer**:

It’s preferable to use a dense index instead of a sparse one, when the file is not sorted based on the index field or when the index file is super tiny in relation to the memory size that’s available. Because if the index file is super tiny, even if dense indexes are more generally expensive (size wise) it still won’t take up very much space and will be faster as a result with minimal negative side effects. In the case of the file being unsorted based on the index field, you would need to have a link to every field in order to find the data your looking for effectively, as you wouldn’t be able to interpolate data locations based upon it’s location relative to other data items.

**11.16 (2 pts.) What is the difference between a clustering index and a secondary index?**

Secondary Index: search key specifies an order that differs from the sequential order of the file (quite literally, the data doesn’t have to be “clustered” together), also called a non-clustering index.

Clustering Index: is defined on a sequentially ordered data file. The search key is most of the time but not always) the primary key. Its the index that’s search key specifies the sequential ordering of the file. (The data literally “clustered” together via the ordered data), also known as the primary index.

**11.19 (3 pts.) Explain the distinction between closed and open hashing. Discuss the relative merits of each technique in database applications**:

Open Hashing: May place keys with the same hash function value in different buckets. Deletion is hard in open hashing because all buckets must be inspected to ensure that key value deletion has occurred. As deletions are difficult, any sort of data structure that would just be used to look up information (that would involve little or no deletions) would benefit from this method (like a library database).

Closed Hashing: Always places keys with the same hash function value in the same bucket. Deletion is easy because only the bucket with the address obtained by hashing the key value needs to be inspected to find where you need to delete from. Any structure that will require deletions to happen (like in store inventory database that has alot of rotating stock) would benefit from this method.

**11.21 (2 pts.) Why is a hash structure not the best choice for a search key on which range queries are likely**?

Hashing distributes search key values in a uniform/random manner across the set of buckets available. Because key values don’t occupy consecutive buckets, it’s highly likely that you’ll have to search a large number of buckets to get the values you need (and in the worst case scenario you may have to access literally every bucket in order to find what you’re searching for). As such, virtually any sort of ordered structure would be a significantly more sound choice for range queries.

**11.29 (5 pts.) – This question is not included in the textbook. This material is covered in the lectures.**

**Consider a disk with the following parameters:**

**B = 512 bytes**

**P = 6 bytes**

**Pr = 7 bytes**

**and an Employee data file with the following parameters:**

**r = 30000**

**R = 115 bytes**

**bfr = 4 records/block**

**Suppose the file is not ordered by the key field SSN (9 bytes) and we want to construct a *secondary index* with record pointers on SSN. Calculate the following:**

1. **The index blocking factor, bfri:**

FLOOR(512/16) = 32 records/block

1. **The number of first-level index entries and the number of number of first-level index blocks.**

bi = CEILING(30000/32) = 938 blocks

1. **The number of levels needed if we make it into a multi-level index**

**3**

1. **The total number of blocks required by the multi-level index**

bi = CEILING(30000/32) = 938 blocks

bi2 = CEILING(938/32) = 30 blocks

bi3 = CEILING(30/32) = 1 block

biTotal = 938+30+1 = 969 blocks

1. **The number of block accesses needed to search for and retrieve a record from the file--given its SSN value--using the primary index.**

Number of black accesses= 1 + 1 + 1 + 1 = 4

\*Misc work/data below\*

Disk: B = 512 bytes, P = 6 bytes, Pr = 7 bytes

dataFile: r = 30000, R = 115 bytes, bfr = 4 records/block

Ri = 7 + 9 = 16 bytes

B = 512 bytes

P = 6 bytes

Pr = 7 bytes

r = 30000

R = 115 bytes

bfr = 4 records/block

Secondary Index - unordered key field (SSN 9 bytes)

Pr =7 bytes

Ri = 7 + 9 = 16 bytes

ri = 30000

Bi = 512 bytes

bfri = FLOOR(512/16) = 32 records/block

bi = CEILING(30000/32) = 938 blocks

bi2 = CEILING(938/32) = 30 blocks

bi3 = CEILING(30/32) = 1 block

biTotal = 938+30+1 = 969 blocks

#Block accesses = log2(938) + 1

**Chapter 14**

**14.12 (5 pts.) List the ACID properties. Explain the usefulness of each**:

A.C.I.D. (Atomicity, Consistency, Isolation, Durability): A number of properties that ensure database transactions are processed reliably, accurately and efficiently.

-Atomicity: Identifies the transaction to be atomic, i.e. it is either fully completed or not carried out at all.

-Consistency: ensuring that the system state remains consistent by making sure the end of any transaction is in a valid state.

-Isolation: when a transaction is run in isolation, it means that it appears to be the only action the system is carrying out at that point in time.

-Durability: A transaction is considered durable if once it has completed, all the changes that it made to the system in question are permanent.

**14.13 (2 pts.) During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which a transaction may pass**:

Active > Partially Committed > Committed

Active > Partially Committed > Aborted

Active > Failed > Aborted

**14.14 (2 pts.) Explain the distinction between the terms serial schedule and serializable schedule**:

Serial Schedule: A schedule that completes operation transactions sequentially, only starting and finishing one transaction at a time & making sure that transaction is complete before the next begins. As such, locking is unnecessary as an individual transaction doesn’t rely on other transactions (which has the potential to interfere with it’s own process) for itself to complete. Therefore this is a very reliable/stable method as the worst case scenario is concerned, if a transaction fails, it won’t have a cascading effect on other transactions in the grand scheme of things due to no transaction overlapping.

Serializable Schedule: This term refers to when a non-serial schedule produces the same result as a serial schedule. This schedule requires a locking mechanism because there might be other processes that are occurring and accessing the variables of the data that needs to be accessed by the current transaction.

**Chapters 15 & 16**

**15.40 (8 pts.) – This question is not included in the textbook.**

**Define or explain briefly the following terms.**

**2PL**: Stands for “Two-Phase Locking”, it’s a concurrency control method that guarantees serializability, and it’s also the name of the resulting set of database transaction schedules (histories). The protocol utilizes locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction’s life. These locks are applied and removed in two phases, 1) Expanding phase: where locks are acquired and no locks are released. And 2) Shrinking phase: where locks are released and no locks are required. Two major types of locks are utilized in this method, 1) The write-lock (exclusive lock): which is associated with a database object by a transaction, and 2) The Read-lock (shared lock) which is associated with a database object by a transaction before reading.

**Deadlock**: Mutual blocking between transactions results in a deadlock, where execution of these transactions is stalled, and no completion can be reached. Thus deadlocks need to be resolved to complete these transactions' executions and release related computing resources. A deadlock is a reflection of a potential cycle in the precedence graph, that would occur without the blocking. A deadlock is resolved by aborting a transaction involved with such potential cycle, and breaking the cycle. As such this is a very important concept to be aware of during the process of locking block data-access operations as even though there are many protocols and methodologies for dealing with these issues when they happen (many of which are done automatically by a number of different systems), it’s a relatively deep subject that has had much study done to get where it is today, and still has many professionals continuing to do research on the topic.

**Deferred database modification**: It’s a concept that ensures transaction atomicity by recording all database modifications in the log but deferring the execution of all write operations of a transaction until the transaction partially commits. A transaction is said to be partially committed once the final action of the transaction has been executed. When a transaction has performed all the actions, then the information in the log associated with the transaction is used in executing the deferred writes. In other words, at partial commits, time logged updates are “replayed” into database item. Therefore this is essentially a method that checks to make sure that a particular series of actions taken in a given scenario are going to work before committing to the task at hand and executing it’s given series of tasks on/with the data provided in it’s various forms.

**Checkpoint**: Is essentially an optimal point decided upon/created to log the current state of different data configurations and processes that can be applied after an unexpected shutdown or crash, which greatly improves reliability, durability and dependability for the systems in question. For performance reasons, the Database Engine performs modifications to database pages in memory (in the buffer cache) and does not write these pages to disk after every change. Rather, the Database Engine periodically issues a checkpoint on each database. A *checkpoint* writes the current in-memory modified pages (known as *dirty pages*) and transaction log information from memory to disk and, also, records information about the transaction log. There are actually quite a few different types of Checkpoints (Automatic, indirect, internal and manual) but they all function to ensure that data loss is either eliminated or significantly reduced due to unexpected events.