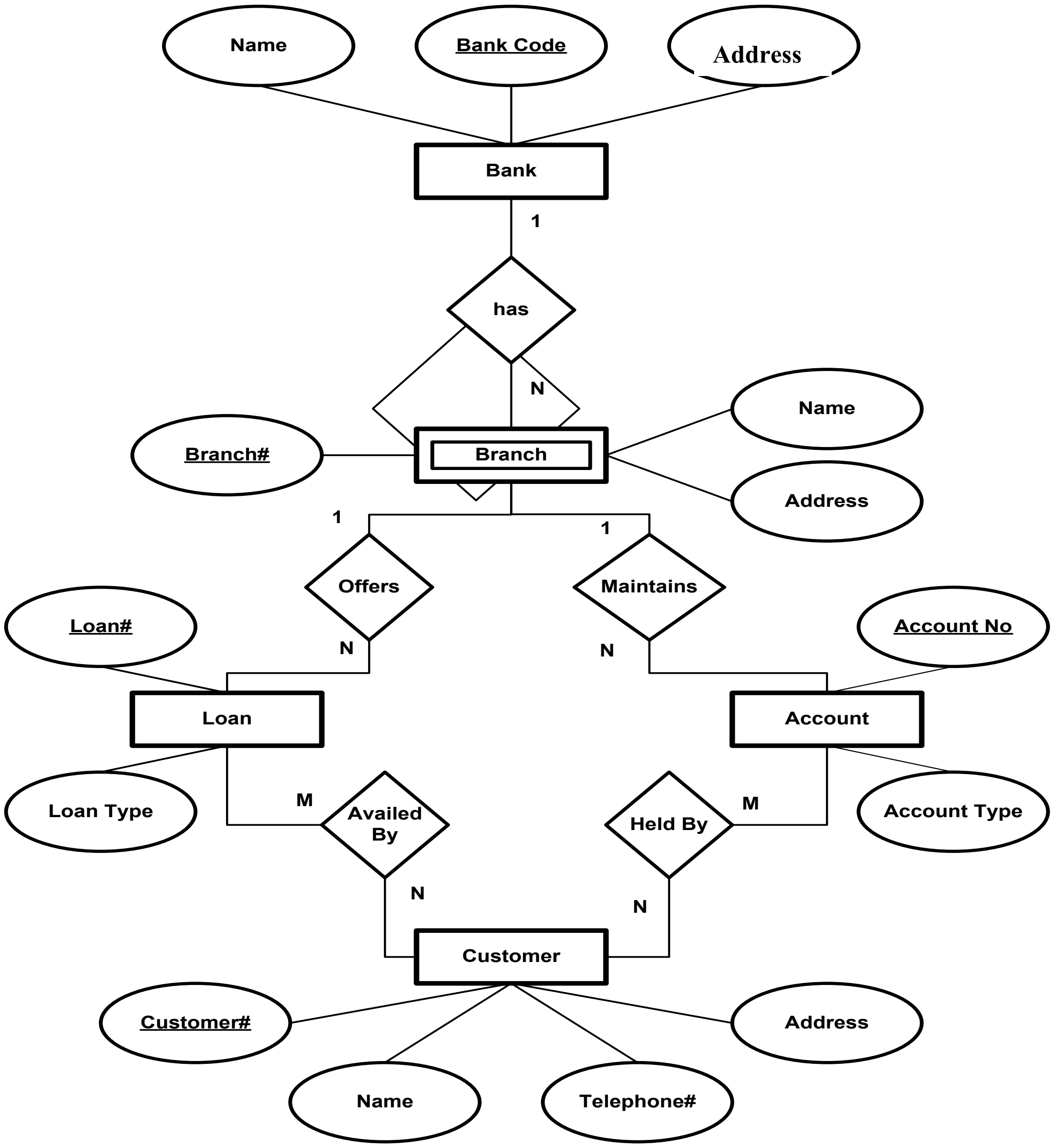
Q.2 a. A database is being constructed to keep track of the employees, customers and other entities of the banking system. Design an E-R schema diagram for this application and also list corresponding relation, attributes, primary keys using your own assumption. (10)

Answer: E-R Diagram for Bank [(8 marks for ER Diagram, below) + (2 marks for explanation)]



b. List out the reasons when not to use DBMS. Also state under what circumstances, regular files are more desirable to use. (6)

Answer: Refer article 1.8, page 26 of Text Book-I

Q.3 a. Explain about primary key, super key, candidate key, alternate key using suitable example. (8)

Answer:

**** Alternate key*** - An alternate key is any candidate key which is not selected to be the primary key

**** Candidate key*** - A candidate key is a field or combination of fields that can act as a primary key field for that table to uniquely identify each record in that table. For Eg: The table: Emloyee(Name,Address,Ssn,Employee_Id$_{\text{primary\_key}}$,Phone_ext) In the above example Ssn no. and employee identity are candidate keys.

**** Compound key*** - compound key (also called a composite key or concatenated key) is a key that consists of 2 or more attributes.

**** Primary key*** - a primary key is a value that can be used to identify a unique row in a table. Attributes are associated with it. Examples of primary keys are Social Security numbers (associated to a specific person) or ISBNs (associated to a specific book). In the relational model of data, a primary key is a candidate key chosen as the main method of uniquely identifying a tuple in a relation. For Eg: Emloyee(Name,Address,Ssn,Employee_Id$_{\text{primary\_key}}$,Phone_ext)

**** Superkey*** - A superkey is defined in the relational model as a set of attributes of a relation variable (relvar) for which it holds that in all relations assigned to that variable there are no two distinct tuples (rows) that have the same values for the attributes in this set. Equivalently a superkey can also be defined as a set of attributes of a relvar upon which all attributes of the relvar are functionally dependent. For Eg: Emloyee(Name,Address,Ssn,Employee_Id$_{\text{primary\_key}}$,Phone_ext) <Ssn,Name,Address> <Ssn,Name> <Ssn> All the above are super keys.

**** Foreign key*** - a foreign key (FK) is a field or group of fields in a database record that points to a key field or group of fields forming a key of another database record in some (usually different) table. Usually a foreign key in one table refers to the primary key (PK) of another table. This way references can be made to link information together and it is an essential part of database normalization. For Eg: For a Student.... School(Name,Address,Phone,School_Reg_no_primary_key)

b. List four significant differences between a file-processing system and a DBMS.(4)

Answer:

Some main differences between a database management system and a file-processing system are:

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.
- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, where as data written by one program in a file processing system may not be readable by another program.
- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).
- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read only access to the file.

c. Which of the following plays an important role in *representing* information about the real world in a database? Explain briefly. (4)
(i) The data definition language
(ii) The data manipulation language

Answer:
Let us discuss the choices in turn.
The data definition language is important in representing information because it is used to describe external and logical schemas.
The data manipulation language is used to access and update data; it is not important for representing the data. (Of course, the data manipulation language must be aware of how data is represented, and reflects this in the constructs that it supports.)

Q.4 a. Consider the following two sets of functional dependencies F= {A→C, AC→D, E→AD, E→H} and G = {A→CD, E→AH}. Check whether or not they are equivalent. (6)

Answer:
To show equivalence, we prove that G is covered by F and F is covered by G.
Proof that G is covered by F:
{A} + = {A, C, D} (with respect to F), which covers A→CD in G
{E} + = {E, A, D, H, C} (with respect to F), which covers E→AH in G
Proof that F is covered by G:
{A} + = {A, C, D} (with respect to G), which covers A→C in F
{A, C} + = {A, C, D} (with respect to G), which covers AC→D in F
{E} + = {E, A, H, C, D} (with respect to G), which covers E→AD and E→H in F

b. What is a minimal set of functional dependencies? Does every set of dependencies have a minimal equivalent set? Give an algorithm for finding a minimal cover G for F. (6)

Answer:
A set of functional dependencies F is minimal if it satisfies the following conditions:
1. Every dependencies in F has a single attribute for its right –hand side.
2. We cannot replace any dependency X→A in F with a dependency Y→A, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.
3. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

We can think of a minimal set of dependencies as being a set of dependencies in a standard or canonical form and with no redundancies. Condition 1 ensures that every dependency is in canonical form with a single attribute on the right-hand side. Conditions 2 and 3 ensures that there is no redundancies in the dependencies either by having redundant attributes on the left-hand side of a dependency, or by having dependency that can be inferred from the remaining FDs in F. A minimal cover of a set of functional dependencies that is equivalent to F. unfortunately, there can be several minimal covers for a set of functional dependencies. Algorithm: Finding a minimal cover G for F

1. Set G: = F.

2. Replace each functional dependency $X \rightarrow \{A_1, A_2 \ldots\ldots A_n\}$ in G by the n functional dependencies

   $X \rightarrow A_1, X \rightarrow A_2 \ldots\ldots X \rightarrow A_n.$
3. For each functional dependency $X \rightarrow A$ in G for each attribute B that is an element of X

   if $((G-\{X \rightarrow A\} \cup \{X - \{B\}) \rightarrow A\})$ is equivalent to G.
4. For each remaining functional dependency $X \rightarrow A$ in G

   if $(G-\{X \rightarrow A\})$ is equivalent to G, then remove $X \rightarrow A$ from G.

c. What is meant by a safe expression in relational calculus? (4)

Answer:

Whenever we use universal quantifiers, existential quantifiers, or negation of predicates in a calculus expression, we must make sure that the resulting expression makes sense. A safe expression in relational calculus is one that is guaranteed to yield a *finite number of tuples* as its result; otherwise, the expression is called unsafe. For example, the expression {t | not (EMPLOYEE(t))} is *unsafe* because it yields all tuples in the universe that are *not* EMPLOYEE tuples, which are infinitely numerous. If we follow the rules, we will get a safe expression when using universal quantifiers. We can define safe expressions more precisely by introducing the concept of the *domain of a tuple relational calculus expression:* This is the set of all values that either appear as constant values in the expression or exist in any tuple of the relations referenced in the expression. The domain of {t | not(EMPLOYEE(t))} is the set of all attribute values appearing in some tuple of the EMPLOYEE relation (for any attribute). An expression is said to be safe if all values in its result are from the domain of the expression. Notice that the result of {t | not(EMPLOYEE(t))} is unsafe, since it will, in general, include tuples (and hence values) from outside the EMPLOYEE relation; such values are not in the domain of the expression.

Q.5 a. Employee (employee_name, street, city) (6)

Works (employee_name, company_name, salary)

Company (company_name, city)

Manages (employee_name, manager_name)

Consider the given relational database. Give an expression in SQL for each of the following queries:

(i) Give all employees of First Bank Corporation a 10 percent raise.

(ii) Give all managers of First Bank Corporation a 10 percent raise.

(iii) Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer:

i. Give all employees of First Bank Corporation a 10-percent raise. (The solution assumes that each person works for at most one company.)

update *works* ***(2 marks)***

set *salary = salary * 1.1*

where *company name* = 'First Bank Corporation'

ii. Give all managers of First Bank Corporation a 10-percent raise. **(2 marks)**

update *works*

set *salary = salary * 1.1*

where *employee name* **in** (**select** *manager name* **from** *manages*)

and *company name* = 'First Bank Corporation'

iii. Delete all tuples in the *works* relation for employees of Small Bank Corporation. **(2 marks)**

delete *works*
where *company name* = 'Small Bank Corporation'

b. Give a relational-algebra expression for each of the following queries: (6)
(i) Find the company with the most employees.
(ii) Find the company with the smallest payroll.
(iii) Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

(i) $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\text{count-distinct } person\text{-}name}(works)$

$t_2 \leftarrow \max_{num\text{-}employees}(\rho_{company\text{-}strength(company\text{-}name, num\text{-}employees)}(t_1))$

$\Pi_{company\text{-}name}(\rho_{t_3(company\text{-}name, num\text{-}employees)}(t_1) \bowtie \rho_{t_4(num\text{-}employees)}(t_2))$

(ii) $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\text{sum } salary}(works)$

$t_2 \leftarrow \min_{payroll}(\rho_{company\text{-}payroll(company\text{-}name, payroll)}(t_1))$

$\Pi_{company\text{-}name}(\rho_{t_3(company\text{-}name, payroll)}(t_1) \bowtie \rho_{t_4(payroll)}(t_2))$

(iii) $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\text{avg } salary}(works)$

$t_2 \leftarrow \sigma_{company\text{-}name = \text{“First Bank Corporation”}}(t_1)$

$\Pi_{t_3.company\text{-}name}((\rho_{t_3(company\text{-}name, avg\text{-}salary)}(t_1))$

$\bowtie_{t_3.avg\text{-}salary > first\text{-}bank.avg\text{-}salary} (\rho_{first\text{-}bank(company\text{-}name, avg\text{-}salary)}(t_2)))$

c. List any four Codd's rule for relational database. (4)

Answer:

The Codd's 12 rules are based on following
Rule 0 by which a database could be evaluated to see how relational it is. The twelve rules of Codd are as follows;
1. All information in a relational database is represented explicitly at the logical level and in exactly one way.
2. Each and every atomic value in a relational database is guaranteed to be logically accessible by resorting to a combination of table name,primary key value and column value.
3. Null values are supported in a fully relational DBMS for representing missing information in a systematic way, independent of data type.
4. Users and programmers need only know one data language to operate the entire database.
5. A relational database must support at least language which should have: data definition,view definition,data manipulation,integrity constraints,authorization,transaction boundaries.
All views that are theoretically updatable are also updatable by the system.

Q.6 a. Explain Write-Ahead Logging (WAL). (8)

Answer:

The default method by which SQLite implements atomic commit and rollback is a rollback journal. Beginning with version 3.7.0, a new "Write-Ahead Log" option (hereafter referred to as "WAL") is available.
There are advantages and disadvantages to using WAL instead of a rollback journal. Advantages include:

1. WAL is significantly faster in most scenarios.
2. WAL provides more concurrency as readers do not block writers and a writer does not block readers. Reading and writing can proceed concurrently.
3. Disk I/O operations tends to be more sequential using WAL.
4. WAL uses many fewer fsync() operations and is thus less vulnerable to problems on systems where the fsync() system call is broken.

But there are also disadvantages:

1. WAL normally requires that the VFS support shared-memory primitives. (Exception: WAL without shared memory) The built-in unix and windows VFSes support this but third-party extension VFSes for custom operating systems might not.
2. All processes using a database must be on the same host computer; WAL does not work over a network filesystem.
3. Transactions that involve changes against multiple ATTACHed databases are atomic for each individual database, but are not atomic across all databases as a set.
4. It is not possible to change the database page size after entering WAL mode, either on an empty database or by using VACUUM or by restoring from a backup using the backup API. You must be in a rollback journal mode to change the page size.
5. It is not possible to open read-only WAL databases. The opening process must have write privileges for "-shm" wal-index shared memory file associated with the database, if that file exists, or else write access on the directory containing the database file if the "-shm" file does not exist.
6. WAL might be very slightly slower (perhaps 1% or 2% slower) than the traditional rollback-journal approach in applications that do mostly reads and seldom write.
7. There is an additional quasi-persistent "-wal" file and "-shm" shared memory file associated with each database, which can make SQLite less appealing for use as an application file-format.
8. There is the extra operation of checkpointing which, though automatic by default, is still something that application developers need to be mindful of.
9. WAL works best with smaller transactions. WAL does not work well for very large transactions. For transactions larger than about 100 megabytes, traditional rollback journal modes will likely be faster. For transactions in excess of a gigabyte, WAL mode may fail with an I/O or disk-full error. It is recommended that one of the rollback journal modes be used for transactions larger than a few dozen megabytes.

How WAL Works
The traditional rollback journal works by writing a copy of the original unchanged database content into a separate rollback journal file and then writing changes directly into the_database file. In the event of a crash or ROLLBACK, the original content contained in the rollback journal is played back into the database file to revert the database file to its original state. The COMMIT occurs when the rollback journal is deleted.

Checkpointing
Of course, one wants to eventually transfer all the transactions that are appended in the WAL file back into the original database. Moving the WAL file transactions back into the database is called a "*checkpoint*".
Another way to think about the difference between rollback and write-ahead log is that in the rollback-journal approach, there are two primitive operations, reading and writing, whereas with a write-ahead log there are now three primitive operations: reading, writing, and checkpointing.
Concurrency
When a read operation begins on a WAL-mode database, it first remembers the location of the last valid commit record in the WAL. Call this point the "end mark". Because the WAL can be growing and adding new commit records while various readers connect to the database, each reader can potentially have its own end mark. But for any particular reader, the end mark is unchanged for the duration of the transaction, thus ensuring that a single read transaction only sees the database content as it existed at a single point in time.
When a reader needs a page of content, it first checks the WAL to see if that page appears there, and if so it pulls in the last copy of the page that occurs in the WAL prior to the reader's end mark. If no copy of the page exists in the WAL prior to the reader's end mark, then the page is read from the original database file. Readers can exist in separate processes, so to avoid forcing every reader to scan the entire WAL looking for pages (the WAL file can grow to multiple megabytes, depending on how often checkpoints are run), a data structure called the "wal-index" is maintained in shared memory which helps readers locate pages in the WAL quickly and with a minimum of I/O. The wal-index greatly improves the performance of readers, but the use of shared memory means that all readers must exist on the same machine. This is why the write-ahead log implementation will not work on a network filesystem.
Performance Considerations
Write transactions are very fast since they only involve writing the content once (versus twice for rollback-journal transactions) and because the writes are all sequential. Further, syncing the content to the disk is not required, as long as the application is willing to sacrifice durability following a power loss or hard reboot. (Writers sync the WAL on every transaction commit if PRAGMA synchronous is set to FULL but omit this sync if PRAGMA synchronous is set to NORMAL.)
On the other hand, read performance deteriorates as the WAL file grows in size since each reader must check the WAL file for the content and the time needed to check the WAL file is proportional to the size of the WAL file. The wal-index helps find content in the WAL file much faster, but performance still falls off with increasing WAL file size. Hence, to maintain good read performance it is important to keep the WAL file size down by running checkpoints at regular intervals.
The checkpointer makes an effort to do as many sequential page writes to the database as it can (the pages are transferred from WAL to database in ascending order) but even then there will typically be many seek operations interspersed among the page writes. These factors combine to make checkpoints slower than write transactions.

b. Describe the optimistic concurrency control techniques. (4)

Answer:
In **optimistic concurrency control techniques**, also known as validation or certification techniques, no checking is done while the transaction is executing. Several proposed concurrency control methods use the validation technique. In this scheme, updates in the transaction are not

applied directly to the database items until the transaction reaches its end. During transaction execution, all updates are applied to local copies of the data items that are kept for the transaction. At the end of transaction execution, a validation phase checks whether any of the transaction's updates violate serializability.

Certain information needed by the validation phase must be kept by the system. If serializability is not violated, the transaction is committed and the database is updated from the local copies; otherwise, the transaction is aborted and then restarted later.

There are three phases for this concurrency control protocol:
1. Read phase: A transaction can read values of committed data items from the database.
However, updates are applied only to local copies (versions) of the data items kept in the transaction workspace.
2. Validation phase: Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.
3. Write phase: If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the transaction is restarted.

c. During the execution of a transaction, it passes through several states, until it finally commits or aborts. List any two possible sequences of states through which a transaction may pass. Explain why each state transition may occur. (4)

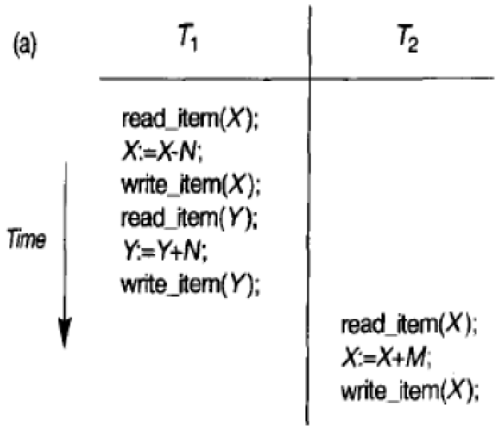
Answer:
a. active –partially committed-committed. This is the normal sequence a successful transaction will follow. After executing all its statement it enters the partially committed state.
b. **active –partially committed –aborted**. After executing the last statement of the transaction it enter the partially committed state. But before enough recovery information is written to disk, a hardware failure may occur destroying the memory contents. In this case the change which to disk, a hardware failure may occur destroying the memory contents. In this case the change which it made to the database are undone, and transaction enters the aborted state.
c. **Active – failure –aborted**. After the transaction start. if it is discovered at some points that normal execution cannot continue (either due to internet program errors or external errors) , it enters the failed state .it is then rolled back , after which it enters the enters the aborted state.

Q.7 a. How the schedules are characterized based on serializability? (8)

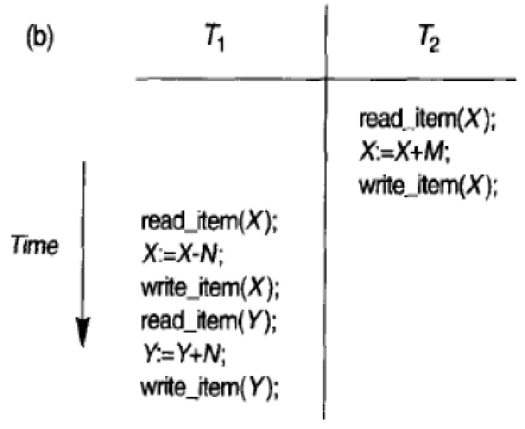
Answer:
Suppose that two users-two airline reservation clerks submit to the DBMS transactions T1 and T2 at approximately the same time.
If no interleaving of operations is permitted, there are only two possible outcomes:
1. Execute all the operations of transaction T1 (in sequence) followed by all the operations of transaction T2 (in sequence).
2. Execute all the operations of transaction T2 (in sequence) followed by all the operations oftransaction T1(in sequence).

These alternatives are shown in Figure a and b, respectively. If interleaving of operations is allowed, there will be many possible orders in which the system can execute the individual operations of the transactions. Two possible schedules are shown in Figure c. The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. This section defines serializability and discusses how it may be used in practice.

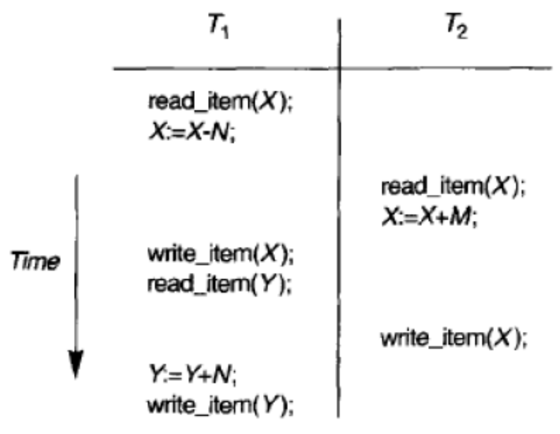
Serial, Nonserial, and Conflict-Serializable Schedules
Schedules A and B in Figure a and b are called **serial** because the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction. In a serial schedule, entire transactions are performed in serial order: T1 and then T2 in Figure a, and T2 and then T1 in Figure b. Schedules C and D in Figure c are called **nonserial** because each sequence interleaves operations from the two transactions. Formally, a schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called **nonserial.** Hence, in a serial schedule, only one transaction at a time is active-the commit (or abort) of the active transaction initiates execution of the next transaction. No interleaving occurs in a serial schedule.



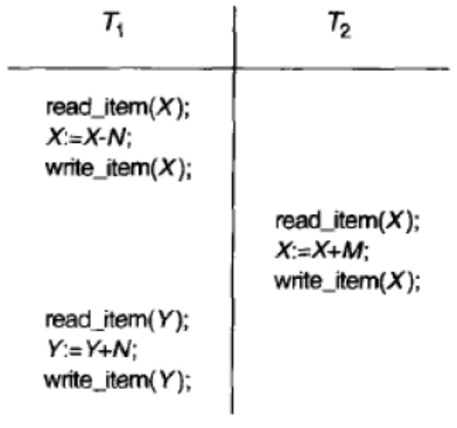
Schedule *A*



Schedule *B*



Schedule *C*



Schedule *D*

The problem with serial schedules is that they limit concurrency or interleaving of operations. In a serial schedule, if a transaction waits for an [/0 operation to complete, we cannot switch the CPU processor to another transaction, thus wasting valuable CPU processing time. In addition, if some transaction T is quite long, the other transactions must wait for T to complete all its operations before commencing. Hence, serial schedules are generally considered unacceptable in practice. A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions. Two schedules are called result equivalent if they produce the same final state of the database. The safest and most general approach to defining schedule equivalence is not to make any assumption about the types of operations included in the transactions. Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules. If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, and hence the schedules are not conflict equivalent.

b. Explain how a System Crash can be recovered using ARIES algorithm? (8)

Answer:
In computer science, **Algorithms for Recovery and Isolation Exploiting Semantics**, or **ARIES** is a recovery algorithm designed to work with a no-force, steal database approach; it is used by IBM DB2, Microsoft SQL Server and many other database systems.
Three main principles lie behind ARIES

- Write ahead logging: Any change to an object is first recorded in the log, and the log must be written to stable storage before changes to the object are written to disk.
- Repeating history during Redo: On restart after a crash, ARIES retraces the actions of a database before the crash and brings the system back to the exact state that it was in before the crash. Then it undoes the transactions still active at crash time.
- Logging changes during Undo: Changes made to the database while undoing transactions are logged to ensure such an action isn't repeated in the event of repeated restarts.

Logging

- For the ARIES algorithm to work a number of log records have to be created during the operation of the database. Log entries are sequentially numbered with Sequence Numbers.
- Usually the resulting logfile is stored on so-called "stable storage", that is a storage medium that is assumed to survive crashes and hardware failures. To gather the necessary information for the logging two data structures have to be maintained: the dirty page table (DPT) and the transaction table (TT).
- The dirty page table keeps record of all the pages that have been modified and not yet written back to disc and the first Sequence Number that caused that page to become dirty. The transaction table contains all transactions that are currently running and the Sequence Number of the last log entry they caused.
- We create log records of the form (Sequence Number, Transaction ID, Page ID, Redo, Undo, and Previous Sequence Number). The Redo and Undo fields keep information about the changes this log record saves and how to undo them. The Previous Sequence Number is a reference to the previous log record that was created for this transaction. In the case of an aborted transaction, it's possible to traverse the log file in reverse order using the Previous Sequence Numbers, undoing all actions taken within the specific transaction.
- Every time a transaction begins or commits we write a "Begin Transaction" entry or an "End Of Log" entry for that transaction respectively.

- During a recovery or while undoing the actions of an aborted transaction a special kind of log record is written, the Compensation Log Record (CLR), to record that the action has already been undone. CLRs are of the form (Sequence Number, Transaction ID, Page ID, Redo, Previous Sequence Number, and Next Undo Sequence Number). The Undo field is omitted because that information is already stored in the original log record for that action.

Recovery

- The recovery works in three phases. The first phase, Analysis, computes all the necessary information from the logfile. The Redo phase restores the database to the exact state at the crash, including all the changes of uncommited transactions that were running at that point in time. The Undo phase then undoes all uncommited changes, leaving the database in a consistent state.

Analysis

- During the Analysis phase we restore the DPT and the TT as they were at the time of the crash.
- We run through the logfile (from the beginning or the last checkpoint) and add all transactions for which we encounter Begin Transaction entries to the TT. Whenever an End Log entry is found, the corresponding transaction is removed. The last Sequence Number for each transaction is of course also maintained.
- During the same run we also fill the dirty page table by adding a new entry whenever we encounter a page that is modified and not yet in the DPT. This however only computes a superset of all dirty pages at the time of the crash, since we don't check the actual database file whether the page was written back to the storage.

Redo

- From the DPT we can compute the minimal Sequence Number of a dirty page. From there we have to start redoing the actions until the crash, in case they weren't persisted already.
- Running through the log file we check for each entry whether the modified page is in the DPT table and whether the Sequence Number in the DPT is smaller than the Sequence Number of the record (i.e. whether the change in the log is newer than the last version that was persisted). If it is we fetch the page from the database storage and check the Sequence Number on the actual if it is smaller than the Sequence Number on the log record. That check is necessary because the recovered DPT is only a conservative superset of the pages that really need changes to be reapplied. Lastly we reapply the redo action and store the new Sequence Number on the page. It is also important for recovery from a crash during the Redo phase, as the redo isn't applied twice to the same page.

Undo

- After the Redo phase the database reflects the exact state at the crash. However the changes of uncommited transactions have to be undone to restore the database to a consistent state.
- For that we run backwards through the log for each transaction in the TT table (those runs can of course be combined into one) using the Previous Sequence Number fields in the records. For each record we undo the changes (using the information in the Undo field) and write a compensation log record to the log file. If we encounter a Begin Transaction record we write an End Log record for that transaction.
- The compensation log records make it possible to recover during a crash that occurs during the recovery phase. That isn't as uncommon as one might think, as it is possible for the recovery phase to take quite long. CLRs are read during the Analysis phase and redone during the Redo phase.

Checkpoints

- To avoid rescanning the whole logfile during the analysis phase it is advisable to save the DPT and the TT regularly to the logfile, forming a checkpoint. Instead of having to run through the whole file it is just necessary to run backwards until a checkpoint is found. From that point it is possible to restore the DPT and the TT as they were at the time of the crash by reading the logfile forward again. Then it is possible to proceed as usual with Redo and Undo.
- The naive way for checkpointing involves locking the whole database to avoid changes to the DPT and the TT during the creation of the checkpoint. Fuzzy logging circumvents that by writing two log records. One Fuzzy Log Starts Here record and, after preparing the checkpoint data, the actual checkpoint. Between the two records other logrecords can be created. During recovery it is necessary to find both records to obtain a valid checkpoint.

Q.8 (For current scheme students i.e., AC61/AT61)

a. What do you understand by the term INDEX? Briefly describe various types of Indexes used for records in tables. (6)

Answer:
An index is usually defined on a single field of a file, called an indexing Field. The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain a record with that field value. The values in the index are ordered so that we can do a binary search on the index. The index file is much smaller than the data file, so searching the index using binary search is reasonably efficient. Multilevel indexing does away with the need for binary search at the expense of creating indexes to the index itself! There are several types of indexes. A primary index is an index specified on the ordering key field of an ordered file of records. An ordering key field is used to physically order the file records on disk, and every record has a unique value for that field. If the ordering field is not a key field that is, several records in the file can have the same value for the ordering field another type of index, called a clustering index, can be used. A file can have at most one physical ordering field, so it can have at most one primary index or one clustering index, but not both.

A third type of index, called a secondary index, can be specified on any non-ordering field of a file. A file can have several secondary indexes in addition to its primary access method.

b. What is Partitioned Hashing? What are its advantage and disadvantage? (6)

Answer:
Partitioned hashing is an extension of static external hashing (Section 5.9.2) that allows access on multiple keys. It is suitable only for equality comparisons; range queries are not supported. In partitioned hashing, for a key consisting of n components, the hash function is designed to produce a result with n separate hash addresses. The bucket address is a concatenation of these n addresses. It is then possible to search for the required composite search key by looking up the appropriate buckets that match the parts of the address.

An **advantage** of partitioned hashing is that it can be easily extended to any number of attributes. The bucket addresses can be designed so that high order bits in the addresses correspond to more frequently accessed attributes. Additionally, no separate access structure needs to be maintained for the individual attributes.

The main **drawback** of partitioned hashing is that it cannot handle range queries on any of the component attributes.

c. What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurrence of bucket overflows? (4)

Answer: The causes of bucket overflow are :-

a. Our estimate of the number of records that the relation will have was too low, and hence the number of buckets allotted was not sufficient.

b. Skew in the distribution of records to buckets. This may happen either because there are many records with the same search key value, or because the the hash function chosen did not have the desirable properties of uniformity and randomness.

To reduce the occurrence of overflows, we can :-

a. Choose the hash function more carefully, and make better estimates of the relation size.

b. If the estimated size of the relation is n_r and number of records per block is f_r, allocate $(n_r/f_r) * (1 + d)$ buckets instead of (n_r/f_r) buckets. Here d is a fudge factor, typically around 0.2. Some space is wasted: About 20 percent of the space in the buckets will be empty. But the benefit is that some of the skew is handled and the probability of overflow is reduced.

Q.8 (For New scheme students i.e., AC112/AT112)

a. Explain specialization, generalization and constraint on spcialization and generalization. (8)

Answer: Refer page 106 of Text Book-I

b. What is distributed database? Explain different types of distributed database systems in brief. (2+6)

Answer: Refer article 25.3, page 889 of Text Book-I

Q.9 (For current scheme students i.e., AC61/AT61)

a. Describe how to incrementally maintain the results of the following operations, on both insertions and deletions. (8)

(i) Union and set difference

(ii) Left outer join

Answer:

a. Given materialized view $v = r \cup s$, when a tuple is inserted in r, we check if it is present in v, and if not we add it to v. When a tuple is deleted from r, we check if it is there in s, and if not, we delete it from v. Inserts and deletes in s are handled in symmetric fashion.

For set difference, given view $v = r - s$, when a tuple is inserted in r, we check if it is present in s, and if not we add it to v. When a tuple is deleted from r, we delete it from v if present. When a tuple is inserted in s, we delete it from v if present. When a tuple is deleted from s, we check if it is present in r, and if so we add it to v.

b. Given materialized view $v = r ⟕ s$, when a set of tuples i_r is inserted in r, we add the tuples $i_r ⟕ s$ to the view. When i_r is deleted from r, we delete $i_r ⟕ s$ from the view. When a set of tuples i_s is inserted in s, we compute $r \bowtie i_s$. We find all the tuples of r which previously did not match any tuple from s(i.e. those padded with *null* in $r ⟕ s$) but which match i_s. We remove all those *null* padded entries from the view and add the tuples $r \bowtie s$ to the view. When i_s is deleted from s, we delete the tuples $r \bowtie i_s$ from the view. Also we find all the tuples in r which match i_s but which do not match any other tuples in s. We add all those to the view, after padding them with *null* values.

b. Briefly explain the different methods for implementing joins. (8)

Answer:

Different methods for implementing joins are as follows:

- **Nested-loop join (brute force)**: For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B] (Note 11).

- **Single-loop join (using an access structure to retrieve the matching records):** If an index (or hash key) exists for one of the two join attributes—say, B of S—retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A].

- **Sort–merge join:** If the records of R and S are physically sorted (ordered) by value of the join attributes A and B, respectively, we can implement the join in the most efficient way possible. Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If the files are not sorted, they may be sorted first by using external sorting. In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file—unless both A and B are non-key attributes, in which case the method needs to be modified slightly. In external sorting method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file—unless both A and B are non-key attributes, in which case the method needs to be modified slightly. The indexes provide the ability to access (scan) the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks, so this method may be quite inefficient, as every record access may involve accessing a different disk block.

- **Hash-join**: The records of files R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash keys. First, a single pass through the file with fewer records (say, R) hashes its records to the hash file buckets; this is called the partitioning phase, since the records of R are partitioned into the hash buckets. In the second phase, called the probing phase, a single pass through the other file (S) then hashes each of its records to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket. This simplified description of hash-join assumes that the smaller of the two files fits entirely into memory buckets after the first phase.

Q.9 (For New scheme students i.e., AC112/AT112)

a. Explain different type of discretionary privileges. (4)

Answer: Refer article 23.2.1, page 802 of Text Book-I

b. Differentiate between discretionary and mandatory access control. (4)

Answer: Refer article 23.3.1, page 809 of Text Book-I

c. What is public key infrastructure scheme? How does it provide security? (4+4)

Answer: Refer article 23.6.2, page 816 of Text Book-I

TEXT BOOK

I. Fundamentals of Database Systems, Elmasri, Navathe, Somayajulu, Gupta, Pearson Education, 2006 (TB-I)