BIT QUESTIONS

Unit I & II

Bits & short answers

1. Other name of SQL is sequel
2. Null is the value which is Unknown
3. To arrange rows in an order, order by clause is used
4. Rows of first relation but not in second relation - difference
5. Each row of first relation gets connected to all rows of second relation- Cartesian product

6. The join condition is that all pairs of attributes from two relations having a common name are equated, and there are no other conditions- Natural join

7. 'Distinct' is used with select statement

8. View is logical and it do not exist. It is created on base table

9. Variable length character strings are stored with endmarker

10. If records are longer than blocks, then records of this type occurs - Spanned records

11. Tombstone is placed in the place of deleted record which warns the system that the record is no longer exists.

12. These will have all paths from root to a leaf of same length- B Tree

13. Two tasks of transaction management are recoverability and correctness

14. This method logs the old value, each time a database element is changed. - Undo Logging

15. A sequence of actions from one or more transactions is called Schedule

16. A schedule is conflict serializable if and only if the precedence graph is acyclic

17. Transactions place Intension lock on large elements to warn other transactions that they plan to access.

18. Two optimistic concurrency control are time stamp based and Validation based scheduling

19. Update locks can be changed from read locks to write locks

20. The three sets of validation based scheduler - START, VAL, FIN

21. Syntax on Insert statement

22. Default constraint is used to set default values to the fields of tuples. It is used at the time of table cration.

23. Foreign key constraint is used for referential integrity

24. Primary key consideres NOT NULL and UNIQUE

25. DESC is used to describe the table

26. VIEW is logical window on base table

27. ANY set operator is used to check any one value satisfies the condition from SET of values

28. set inclusion is achieved by IN operator of SELECT statement

29. Unique allows Null values but not duplicates

30. Alter, create, drop are DDL commands

31. Insert, delete, update are DML commands

32. Commit is used to commit the transactions permanently

33. Rollback will cancel the previous DML transactions

34. **Wait-Die Scheme:**

(a) If T is older than U (i.e., the timestamp of T is smaller than U's timestamp) , then T is allowed to wait for the lock(s) held by U .
(b) If U i s older than T , then T "dies" ; it is rolled back.

35. **The Wound- Wait Scheme**

(a) If T is older than U, it "wounds" U. Usually, the "wound" is fatal: U must roll back and relinquish to T the lock(s) that T needs from U. There is an exception if, by the time the "wound" takes effect , Uhas already finished and released its locks. In that case, U survives and need not be rolled back.
(b) If U is older than T, then T waits for the lock(s) held by U.

36. **Consistent Database States:** Database states that obey whatever implied or declared constraints the designers intended are called consistent. It is essential that operations on the database preserve consistency, that is, they turn one consistent database state into another.
37.**Consistency of Concurrent Transactions:** It is normal for several transactions to have access to a database at the same time. Transactions, run in isolation, are assumed to preserve consistency of the database. It is the job of the scheduler to assure that concurrently operating transactions also preserve the consistency of the database.
38. **Schedules:** Transactions are broken into actions, mainly reading and writing from the database. A sequence of these actions from one or more transactions is called a schedule.
39. **Serial Schedules :** If transactions execute one at a time, the schedule is said to be serial.
40. **Serializable Schedules:** A schedule that is equivalent in its effect on the database to some serial schedule is said to be serializable. Interleaving of actions from several transactions is possible in a serializable schedule that is not itself serial, but we must be very careful what sequences of actions we allow, or an interleaving will leave the database in an inconsistent state.
41. **Confiict-Serializability:** A simple-to-test, sufficient condition for serializability is that the schedule can be made serial by a sequence of swaps of adjacent actions without conflicts. Such a schedule is called conflictserializable. A conflict occurs if we try to swap two actions of the same transaction, or to swap two actions that access the same database element, at least one of which actions is a write.

42. **Precedence Graphs :** An easy test for conflict-serializability is to construct a precedence graph for the schedule. Nodes correspond to transactions, and there is an arc T --+ U if some action of T in the schedule conflicts with a later action of U. A schedule is conflict-serializable if and only if the precedence graph is acyclic.

43.**Locking:** The most common approach to assuring serializable schedules is to lock database elements before accessing them, and to release the lock after finishing access to the element. Locks on an element prevent other transactions from accessing the element.

44. **Two-Phase Locking:** Locking by itself does not assure serializability. However, two-phase locking, in which all transactions first enter a phase where they only acquire locks, and then enter a phase where they only release locks, will guarantee serializability.

45. **Lock Modes:** To avoid locking out transactions unnecessarily, systems usually use several lock modes, with different rules for each mode about when a lock can be granted. Most common is the system with shared locks for read-only access and exclusive locks for accesses that include writing.

46. Compatibility Matrices: A compatibility matrix is a useful summary of when it is legal to grant a lock in a certain lock mode, given that there may be other locks, in the same or other modes, on the same element.

47. Update Locks : A scheduler can allow a transaction that plans to read and then write an element first to take an update lock, and later to upgrade the lock to exclusive. Update locks can be granted when there are already shared locks on the element, but once there, an update lock prevents other locks from being granted on that element.

48. **Increment Locks:** For the common case where a transaction wants only to add or subtract a constant from an element, an increment lock is suitable. Increment locks on the same element do not conflict with each other, although they conflict with shared and exclusive locks.

49. **Locking Elements With a Granularity Hierarchy:** When both large and small elements - relations, disk blocks, and tuples, perhaps - may need to be locked, a warning system of locks enforces serializability. Transactions place intention locks on large elements to warn other transactions that they plan to access one or more of its subelements.

50. **Locking Elements Arranged in a Tree:** If database elements are only accessed by moving down a tree, as in a B-tree index, then a non-two-phase locking strategy can enforce serializability. The rules require a lock to be held on the parent while obtaining a lock on the child, although the lock on on the parent can then be released and additional locks taken later.

51. **Optimistic Concurrency Control:** Instead of locking, a scheduler can assume transactions will be serializable, and abort a transaction if some potentially nonserializable behavior is seen. This approach, called optimistic, is divided into timestamp-based, and validation-based scheduling.

52.Sequential Files : Several simple file organizations begin by sorting the data file according to some search key and placing an index on this file.

53.Dense Indexes: These indexes have a key-pointer pair for every record in the data file. The pairs are kept in sorted order of their key values.

54.Sparse Indexes: These indexes have one key-pointer pair for each block of the data file. The key associated with a pointer to a block is the first key found on that block.

55.Multilevel Indexes: It is sometimes useful to put an index on the index file itself, an index file on that, and so on. Higher levels of index must be sparse.

56.Expanding Files : As a data file and its index file(s) grow, some provision for adding additional blocks to the file must be made. Adding overflow blocks to the original blocks is one possibility. Inserting additional blocks in the sequence for the data or index file may be possible, unless the file itself is required to be in sequential blocks of the disk.

57.Secondary Indexes : An index on a search key K can be created even if the data file is not sorted by K. Such an index must be dense.

58.Inverted Indexes : The relation between documents and the words they contain is often represented by an index structure with word-pointer pairs. The pointer goes to a place in a "bucket" file where is found a list of pointers to places where that word occurs.

59.B-trees : These structures are essentially multilevel indexes, with graceful growth capabilities. Blocks with n keys and $n + 1$ pointers are organized in a tree, with the leaves pointing to records. All blocks are between half-full and completely full at all times.

60.Range Queries: Queries in which we ask for all records whose search-key value lies in a given range are facilitated by indexed sequential files and B-tree indexes, although not by hash-table indexes.

61.Hash Tables : We can create hash tables out of blocks in secondary memory, much as we can create main-memory hash tables. A hash function maps search-key values to buckets, effectively partitioning the records of a data file into many small groups (the buckets). Buckets are represented by a block and possible overflow blocks.

62.Dynamic Hashing: Since performance of a hash table degrades if there are too many records in one bucket, the number of buckets may need to grow as time goes on. Two important methods of allowing graceful growth are extensible and linear hashing. Both begin by hashing search-key values to long bit-strings and use a varying number of those bits to determine the bucket for records.

63.Extensible Hashing: This method allows the number of buckets to double whenever any bucket has too many records. It uses an array of pointers to blocks that represent the buckets. To avoid having too many blocks, several buckets can be represented by the same block.

64.Linear Hashing: This method grows the number of buckets by 1 each time the ratio of records to buckets exceeds a threshold. Since the population of a single bucket cannot cause the table to expand, overflow blocks for buckets are needed in some situations.

65.Embedded SQL: Instead of using a generic query interface to express SQL queries and modifications, it is often more effective to write programs that embed SQL queries in a conventional host language. A preprocessor converts the embedded SQL statements into suitable function calls of the host language.

66.SQL: The language SQL is the principal query language for relational database systems. The current standard is called SQL-99 or SQL3. Commercial systems generally vary from this standard.

67.Select-From- Where Queries: The most common form of SQL query has the form select-from-where. It allows us to take the product of several relations (the FROM clause) , apply a condition to the tuples of the result (the WHERE clause) , and produce desired components (the SELECT clause) .

68.Subqueries : Select-from-where queries can also be used as subqueries within a WHERE clause or FROM clause of another query. The operators EXISTS, IN, ALL, and ANY may be used to express boolean-valued conditions about the relations that are the result of a subquery in a WHERE clause.

69.Set Operations on Relations : We can take the union, intersection, or difference of relations by connecting the relations, or connecting queries defining the relations, with the keywords UNION, INTERSECT, and EXCEPT, respectively.

70.Join Expressions: SQL has operators such as NATURAL JOIN that may be applied to relations, either as queries by themselves or to define relations in a FROM clause.

71.Null Values: SQL provides a special value NULL that appears in components of tuples for which no concrete value is available. The arithmetic and logic of NULL is unusual. Comparison of any value to NULL, even another NULL, gives the truth value UNKNOWN. That truth value, in turn, behaves in boolean-valued expressions as if it were halfway between TRUE and FALSE.

UNIT - III

1.Redundant Storage: Some information is stored repeatedly.

2.Update Anomalies: If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

3.Insertion Anomalies: It may not be possible to store certain information unless some other, unrelated, information is stored as well.

4.Deletion Anomalies: It may not be possible to delete certain information without losing some other, unrelated, information as well.

5. The *dependency-preservation* property enables us to enforce any constraint on the original relation by simply enforcing constraints on each of the smaller relations.

6. Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

7. The following three rules, called *Armstrong's Axioms*, can be applied repeatedly to infer all FDs implied by a set of FDs. We use X, Y, and Z to denote setsof attributes over a relation schema R:

- Reflexivity: If $X \supseteq Y$, then $X \rightarrow Y$.
- Augn1.entation: If $)(\rightarrow Y$, then $XZ \rightarrow YZ$ for any Z.
- Transitivity: If $)(\rightarrow Y$ and $Y \quad Z$, then $X \rightarrow Z$.

8. In a trivial FD, the right side contains only attributes that also appear on the left side

9. Decomposing relation into smaller relations to avoid redundancy is called Normalization

10. Normalization is used for mainly to Eliminating redundant data and to Ensuring data dependencies.
11. **first normal form:** an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.
12. As per the Second Normal Form there must ***not be any partial dependency*** of any column on primary key
13. In third normal form this ***transitive functional dependency should be removed*** from the table.
14. A relation is said to undergo BCNF normal form if
- The relation will have multiple composite keys
- Further the composite keys will have common attribute
- And the key of first composite key is functionally dependent on key of other composite key.

15. The multi valued dependency X →→ Y holds in a relation R if for each value of X, there exists a definite set of values of Y

UNIT - IV

16. Collections of operations that form a single logical unit of work are called transactions.
17. A transaction is delimited by statements (or function calls) of the form ***begin transaction*** and ***end transaction***.
18. This ***"all-or-none"*** property is referred to as ***atomicity***.
19. Execution of a transaction in isolation preserves the consistency of the database.
20. Once the changes caused by an aborted transaction have been undone, we say that the transaction has been ***rolled back***.
21. A transaction that completes its execution successfully is said to be ***committed***.
22. **Active**, the initial state; the transaction stays in this statewhile it is executing.
23. **Partially committed**, after the final statement has been executed.
24. **Failed**, after the discovery that normal execution can no longer proceed.
25. **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
26. **Committed**, transaction after successful completion.
27. A transaction enters the ***failed state*** after the system determines that the transaction can no longer proceed with its normal execution.
28. Failed transactions must be ***rolled back***
29. **Through put**: The number of transactions executed in a given amount of time
30. The average time for a transaction is to be completed after it has been submitted.
31. The execution sequences of transactions is called ***schedules***.
32. ***Serial*** **Schedules**: Each serial schedule consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.
33. concurrent execution of transactions is conducted by ***concurrency-control*** component of the database system.
34. ***Serializable schedules***: The schedule should, in some sense, should be equivalent to a serial schedule.
35. Graphical representation of transactions is shown by **precedence graph**

36. A **recoverable schedule** is one where, for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti , the commit operation of Ti appears before the commit operation of Tj .
37. We may have to *roll back several transactions which is called* ***cascading rollback***
38. Two-phase locking requires a transaction to have two phases, one where it acquires locks but does not release any, and a second phase where the transaction releases locks but does not acquire any.
39. Shared locks are used for data that the transaction ***reads***
40. ***Exclusive*** locks are used for those it writes.
41. The ***read timestamp*** of a data item holds the largest timestamp of those transactions that read the data item.
42. The ***write timestamp*** of a data item holds the timestamp of the transaction that wrote the current value of the data item.
43. **Multi version of a data item**, it is possible to allow a transaction to read an old version of a data item rather than a newer version written by an uncommitted transaction.
44. In **snapshot isolation**, we can imagine that each transaction is given its own version, or snapshot, of the database when it begins.
45. Stable storage that must be accessible online is approximated with mirrored disks, or other forms of ***RAID***, which provide redundant data storage. Off line, or archival, stable storage may consist of multiple tape copies of data stored in physically secure locations.
46. Cascadelessness is ensured by allowing transactions to only read committed data.
47. The transaction T1 may never make progress till T2 releases the required data items , and is said to be ***starvation.***
48. **Growing phase.** A transaction may obtain locks, but may not release any lock in 2phase locking
49.**Shrinking phase.** A transaction may release locks, but may not obtain any new locks in 2phase locking

50. *Cascading rollbacks* can be avoided by a modification of two-phase locking called the **strict two-phase locking protocol**
51. Mechanism for ***upgrading*** a shared lock to an exclusive lock, and ***downgrading*** an exclusive lock to a shared lock is called lock conversion.
52. While locking the data items, A ***lock manager*** can be implemented as a process that receives messages from transactions and sends messages in reply.
53. A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.
54. One way to prevent deadlock is to use an ***ordering of data items***.
55. ***wait–die*** **scheme:** When transaction Ti requests a data item currently held by Tj , Ti is allowed to wait only if it has a timestamp smaller than that of Tj (that is, Ti is older than Tj). Otherwise, Ti is rolled back (dies).
56. ***wound–wait*** **scheme** : When transaction Ti requests a data item currently held by Tj , Ti is allowed to wait only if it has a timestamp larger than that of Tj (that is, Ti is younger than Tj). Otherwise, Tj is rolled back (Tj is wounded by Ti).
57. Deadlocks can be described precisely in terms of a directed graph called a ***waitfor graph***.
58. There are circumstances where it would be advantageous to group several data items, and to treat them as one aggregate data item for purposes of working, resulting in **multiple levels of granularity**.
59. If a node is locked in ***intention-shared (IS) mode***, explicit locking is being done at a lower level of the tree, but with only shared-mode locks.
60. **Timestamps** are designed by system clocks or logical counters.
61. A modified version of the timestamp-ordering protocol with increased concurrency control is called **Thomas' Write Rule**

62. **Validation phase:** If a transaction fails the validation test, the system aborts the transaction.
63. If a transaction does not modify the database until it has committed, it is said to use the ***deferred-modification technique***.
64. If database modifications occur while the transaction is still active, the transaction is said to use the ***immediate-modification technique***.
65. A ***fuzzy checkpoint*** is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out.
66. Transaction processing is based on a storage model in which main memory holds a *log buffer*, a *database buffer*, and a *system buffer*.
67. Before a block of data in main memory can be output to the database, all log records pertaining to data in that block must have been output to stable storage. This rule is called the ***write-ahead logging (WAL)*** rule.
68. ***Force policy***: Transactions would force-output all modified blocks to disk when they commit.
69. **No-Steal policy:** Blocks modified by a transaction that is still active should not be written to disk.
70. ***Steal policy***, allows the system to write modified blocks to disk even if the transactions that made those modifications have not all committed.
71. Locks that are held for a short duration, are often referred to as **latches**.
72. ***fuzzy checkpoint***: To avoid interruptions, the check pointing technique can be modified to permit updates to start once the checkpoint record has been written, but before the modified buffer blocks are written to disk.
73. A dump of the database contents is also referred to as an ***archival dump***.

74. ***Fuzzy dump*** schemes have been developed that allow transactions to be active while the dump is in progress.
75. The recovery algoritham ARIES recovers from a system crash in three passes, Analysis pass, Redo pass, Undo pass
76. **One-safe.** A transaction commits as soon as its commit log record is written to stable storage at the primary site.
77. **Two-very-safe.** A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site.
78. **Two-safe.** This scheme is the same as two-very-safe if both primary and backup sites are active.

UNIT – V

79. Records in a heap file are stored in ***random order*** across the pages of the file.
80. An ***index*** is a data structure that organizes data records on disk to optimize certain kinds of retrieval operations.
81. An index that can be a clustered only if the data records are ***sorted on the search key field***.
82. A ***primary index*** is guaranteed ***not to contain duplicates***, but an index on other fields can contain duplicates.
83. The bucket to which a record belongs can be determined by applying a special function, called ***a hash function***, to the search key.
84. The **B+ tree** is an index structure that ensures that all paths from the root to a leaf in a given tree are of the same length, that is, the structure is always balanced in height.
85. The **height** of a balanced tree is the length of a path from root to leaf.
86. The data entries of the ISAM index are in the ***leaf pages*** of the tree and additional overflow pages chained to some leaf page.
87. The ISAM structure is *completely static* and facilitates low-level optimizations.

88. In ISAM, additional pages are needed because the index structure is static and these are called ***overflow pages.***

89. The tree which grows and shrinks dynamically is B+ TREE

90. When there is a *collection of data records*, systems provide a *bulk-loading utility* for creating a B+ tree.

91. The *Extendible Hashing* scheme uses a *directory* to support inserts and deletes efficiently with no overflow pages.

92. The *Linear Hashing* scheme uses a clever policy for creating *new buckets* and supports inserts and deletes efficiently without the use of a directory.