Answer to Question 2:

(a) Explain the serialization concept in Transaction. "Are all view serializable schedules conflict serializable" - Justify your answer.

Serialization Concept in Transactions:

Serialization in the context of database transactions refers to the process of ensuring that the concurrent execution of multiple transactions yields the same result as if they were executed in some serial order (one after the other). The goal is to maintain database consistency despite interleaving operations from different transactions. A schedule of operations from concurrent transactions is considered **serializable** if its effect on the database is equivalent to that of some serial execution of the same transactions.

There are two main types of serializability:

- **Conflict Serializability:** A schedule is conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting[1] operations. Two operations[2] conflict if they belong to different transactions, access the same data item, and at least one of them is a write operation.[3] Non-conflicting operations can be swapped without affecting the final outcome.
- **View Serializability:** A schedule is view serializable if it is view equivalent to some serial schedule. Two schedules are view equivalent if they satisfy the[4] following conditions:
  1. For each data item, if a transaction reads the initial value in one schedule, it also reads the initial value in the other schedule.
  2. For each data item, if a transaction reads a value written by another transaction in one schedule, it also reads the value written by the same transaction in the other schedule.
  3. For each data item, if a transaction performs the final write operation in one schedule, it also performs the final write operation in the other schedule.

"Are all view serializable schedules conflict serializable?" - Justify your answer.

No, not all view serializable schedules are conflict serializable.

Justification:

Conflict serializability is a stricter condition than view serializability. Every conflict serializable schedule is also view serializable, but the converse is not necessarily true.

Consider the following non-conflict serializable schedule involving three transactions (T1, T2, T3) and a data item A, initially with value 10:

Time	**T1**	**T2**	**T3**
1	Read(A)		
2		Read(A)	
3	Write(A, 15)		
4		Write(A, 20)	
5			Read(A)

This schedule is **not conflict serializable** because:

- The write operation of T1 on A conflicts with the read operation of T2 on A (if T1 precedes T2) and the write operation of T2 on A (if T2 precedes T1).
- The write operation of T2 on A conflicts with the read operation of T1 on A (if T2 precedes T1) and the read operation of T3 on A (if T2 precedes T3).

However, this schedule **is view serializable** to the serial schedule T1→T2→T3:

- **Initial Read:** T1 reads the initial value of A in both schedules.
- **Intermediate Reads:** T2 reads the value written by T1 in the serial schedule (if we consider a slight modification where T2 reads after T1's write). In the concurrent schedule, T2 reads the initial value, but we can find a view equivalent serial order. T3 reads the final value of A (20) written by T2 in both the concurrent and the serial order T1→T2→T3.
- **Final Write:** T2 performs the final write to A in both the concurrent schedule and the serial order T1→T2→T3.

This example demonstrates a schedule that is view serializable but not conflict serializable. View serializability considers the final outcome and the values read by transactions, while conflict serializability focuses solely on the order of conflicting operations.

(b) What are the three data anomalies that are likely to occur as a result of data redundancy? Can data redundancy be completely eliminated in the database

approach? Why or why not?[5]

Three Data Anomalies due to Data Redundancy:

Data redundancy, the duplication of data within a database, can lead to several problems known as data anomalies. The three main types are:

1. **Insertion Anomaly:** This anomaly occurs when it becomes impossible to insert a new tuple into a relation because the primary key value is not yet associated with some of the non-key attributes. For example, if we have a table storing employee information and their department, and department details are only recorded when an employee is assigned to it, we cannot add a new department to the database if no employee is currently assigned to that department.
2. **Deletion Anomaly:** This anomaly occurs when deleting a tuple results in the unintentional loss of information about other entities. For instance, if we have a table storing employee information along with their department, and the last employee of a particular department is deleted, we might lose all information about that department if it's only stored in the employee record.
3. **Update Anomaly:** This anomaly arises when updating a data item requires updating multiple rows due to data duplication. If these updates are not performed consistently across all redundant copies, it leads to data inconsistency. For example, if an employee's address is stored in multiple records, and we update the address in only some of the records, the database will contain inconsistent information about that employee's address.

Can data redundancy be completely eliminated in the database approach? Why or why not?

No, data redundancy cannot always be completely eliminated in the database approach, and sometimes it is even intentionally introduced.

Reasons why complete elimination is not always possible or desirable:

- **Normalization Trade-offs:** While database normalization aims to minimize redundancy, achieving the highest normal forms (like 5NF) can sometimes lead to a large number of tables and complex join operations, which can negatively impact query performance. Database designers often make trade-offs between minimizing redundancy and maintaining acceptable performance levels.
- **Denormalization for Performance:** In some cases, controlled redundancy is intentionally introduced (denormalization) to improve query performance. By duplicating certain data across tables, the need for expensive join operations can

be reduced, leading to faster data retrieval for frequently accessed information. This is common in data warehousing and OLAP (Online Analytical Processing) systems.

- **Physical Storage Limitations:** While the logical design can aim for minimal redundancy, physical storage constraints or the way different database systems handle data might introduce some level of physical redundancy.
- **Derived Data:** Some redundancy is inherent in storing derived data (data calculated from other data). While it could theoretically be recalculated every time, storing it can significantly improve query performance. For example, storing the total sales amount in an order table, even though it can be calculated from the individual item prices and quantities.

In conclusion, while the relational database approach and normalization techniques strive to minimize data redundancy to avoid anomalies and ensure data integrity, complete elimination is not always practical or beneficial. Database design often involves finding a balance between minimizing redundancy and optimizing performance based on the specific requirements of the application.

Got it! Here's the **question along with its 1-line answer** for each:

---

1. a) Define the term physical schema of a database with an example.
Answer: Physical schema defines how data is stored in the database (e.g., tables stored using B-trees on disk).

---

b) What is the need of the constraint in the database system?
Answer: Constraints ensure data accuracy, consistency, and integrity.

---

c) What is the rule to find an entity is a weak entity?
Answer: An entity is weak if it lacks a primary key and depends on a strong entity.

---

d) All primary keys are super keys — justify the statement.
Answer: A primary key uniquely identifies tuples, which is the definition of a super key.

---

e) What is the use of log book in transaction?
Answer: Log book helps in recovery by recording all transaction activities.

---

f) What is union compatibility?
Answer: Two relations are union compatible if they have the same number of attributes with matching data types.

---

g) Sparse indexing is better than dense indexing — justify the statement.
Answer: Sparse indexing is more space-efficient and faster for large datasets.

---

h) Justify that a relation which is in 3NF is not necessarily in BCNF.
Answer: 3NF allows non-prime attribute dependencies, while BCNF requires all dependencies to be on a super key.

---

i) What is the need of partially committed state in the transaction?
Answer: It ensures all operations are successfully executed before final commit.

---

j) Why certain functional dependencies are called trivial functional dependencies? Explain with an example.
Answer: A dependency is trivial if the RHS is a subset of LHS, e.g., $\{A, B\} \rightarrow A$.

Let me know if you want these in a PDF or formatted for notes!

Great! Here's the complete Q&A (1-mark answers, 1–2 lines each) from all 5 screenshots, numbered sequentially:

Screenshot 1: KIIT-U/2014/SOT

1. Advantages of having an index structure?
Answer: It speeds up data retrieval operations on a database table.

2. When does deadlock occur in transaction processing?
Answer: Deadlock occurs when two or more transactions wait indefinitely for each other's resources.

3. Define two-phase locking protocol.
Answer: It ensures serializability by acquiring all locks before releasing any (growing and shrinking phase).

4. What are spurious tuples and how can we prevent them?
Answer: Spurious tuples are incorrect rows from bad joins; avoided using lossless decomposition.

5. Candidate key for R=ABCDEG, F={AB→C, AC→B, AD→E, B→D, BC→A, E→G}?
Answer: Candidate key is **AD**.

6. Two commands for DCL and DML?
Answer: DCL: `GRANT`, `REVOKE`; DML: `INSERT`, `UPDATE`.

7. ACID properties of a transaction?
Answer: Atomicity, Consistency, Isolation, Durability — ensure reliable transactions.

8. Strong vs Weak entity set?
Answer: Strong entity has its own key; weak depends on a strong entity and has partial key.

9. Physical vs Logical data independence?
Answer: Physical: change in storage without affecting schema; Logical: change in schema without affecting apps.

10. Specialization and generalization in EER?
Answer: Specialization splits an entity; generalization combines entities into a

superclass.

Screenshot 2: KIIT-U/2015/SOT

11. Define Schema, instance, arity, cardinality with example.
Answer: Schema: structure; Instance: data; Arity: no. of columns; Cardinality: no. of rows.

12. Will `select ename, dno from EMP group by dno` work?
Answer: No, because `ename` is not in `GROUP BY` or an aggregate function.

13. Infer FD X⟶YZ from {X⟶Y, Y⟶Z}?
Answer: Yes, using transitivity: X⟶Y and Y⟶Z ⇒ X⟶Z, so X⟶YZ.

14. Referential and self-referential integrity with example?
Answer: Referential: FK must match PK (e.g., `dept_id`); self: FK refers to same table (`mgr_id` in `employee`).

15. Outer join vs Natural join?
Answer: Outer join includes unmatched rows; natural join matches common attributes only.

16. Candidate key of R(A,B,C,D,E), F = {AB⟶E, C⟶B, D⟶A, B⟶C, A⟶D}?
Answer: Candidate key is **AC**.

17. Define transaction and concurrency?
Answer: A transaction is a unit of work; concurrency allows simultaneous transactions maintaining consistency.

18. What is Primary Index?
Answer: Index built on a sorted column, typically the primary key.

19. When is a transaction conflict-serializable?
Answer: If it can be transformed into a serial schedule by swapping non-conflicting operations.

20. Relation algebra expression for ∃(P(X))?
Answer: Use projection: πX(P(X)) to represent existence over X.

Screenshot 3: KIIT-U/2016/SOT

21. Dependency preservation in 2NF/3NF?
Answer: Decomposition preserves dependency if all FDs are derivable from decomposed relations.

22. Importance of NULL values?
Answer: They represent unknown or missing data in attributes.

23. DB Analyst vs Administrator?
Answer: Analyst designs and models data; Admin manages users, backups, and performance.

24. Redundancy and NULL issues in tuples?
Answer: Leads to storage waste and confusion in query results.

25. Every candidate key is super key, but vice versa?
Answer: True; all super keys aren't minimal, so not all are candidate keys.

26. Closure of {A,B}, F={A→BC, B→CF, B→E, CD→EF}?
Answer: {A, B, C, F, E}.

27. Join size of A(m) and B(n)?
Answer: Max = m×n, Min = 0.

28. Partial to committed state?
Answer: Ensures all operations are successful before transaction commits.

29. What is a multi-valued attribute?
Answer: An attribute with multiple values; converted into a separate relation.

30. Serial vs Concurrent schedule?
Answer: Serial runs one transaction at a time; concurrent allows overlap with isolation.

Screenshot 4: KIIT-U/2017/SOT

31. Primary vs Candidate Key with example?

Answer: Candidate key is minimal super key; one becomes primary (e.g., `A` and `B` are candidate keys; choose `A`).

32. Outer Join Operation?
Answer: Joins including unmatched rows using NULL.

33. Closure of {A,B}, FDs: A→C, AB→D?
Answer: Closure = {A, B, C, D}.

34. Referential Integrity Example?
Answer: FK `dept_id` in `Employee` must match PK in `Department`.

35. DBMS physical vs logical structure?
Answer: Physical is storage-level; logical is table/attribute view.

36. Lost update and dirty read?
Answer: Lost update: one update overwrites another; dirty read: reading uncommitted data.

37. Aggregation in ER model?
Answer: Represents relationship between a relationship and an entity.

38. Roles of DBA?
Answer: Manages access, security, backups, and performance tuning.

39. Serializable schedule?
Answer: A schedule with same result as some serial schedule.

40. Lossless Join Decomposition?
Answer: Decomposed tables can be joined to recover original table without spurious tuples.

Screenshot 5: KIIT-U/2018/SOT

41. Draw entity with multivalued and composite attribute.
Answer: (Diagram not possible here) E.g., Student(Name{First, Last}, Phone{1..n}).

42. When is self join used?
Answer: To compare rows in same table, e.g., employee-manager hierarchy.

43. What is partial dependency?
Answer: Non-prime attribute depends on part of composite key.

44. Lossless join condition?
Answer: Decomposition R → R1, R2 is lossless if common attributes form a key in one of them.

45. Clustering Indexing?
Answer: Data and index are stored in same order; improves range query performance.

46. What is timestamp in DB?
Answer: Timestamp orders transactions to ensure serializability.

47. Importance of view?
Answer: Provides security, abstraction, and simplified query interface.

48. When is schedule recoverable?
Answer: If no transaction commits until all transactions it read from commit.

49. Define 4NF with example?
Answer: A relation is in 4NF if it has no multi-valued dependencies other than a candidate key.

50. What is a safe expression?
Answer: Ensures the result is finite and domain-independent.

Let me know if you'd like all this in PDF, table format, or as a printable doc!