Total No. of Questions – [04] Total No. of Printed Pages 02

G.R. No.	

NOVEMBER 2019/INSEM (T2)
T. Y. B. TECH. (COMPUTER ENGINEERING/INFORMATION TECHNOLOGY)

(SEMESTER -I)

COURSE NAME: DATABASE MANAGEMENT SYSTEM

COURSE CODE: CSUA31173/ITUA31173

SOLUTION (PATTERN 2017)

Time: [**1Hour**] [Max. Marks: **30**]

Q.1)a)Explain Second Normal form with suitable example? [6]

Second Normal form -A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table. i.e. no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

- **Example 1 –** Consider table-1 as given below:

The second step in Normalization is 2NF. A table is in 2NF, only if a relation is in 1NF and meet all the rules, and every non-key attribute is fully dependent on primary key.The Second Normal Form eliminates partial dependencies on primary keys.

Example

<StudentProject>

StudentID	ProjectID	StudentName	ProjectName
S89	P09	Olivia	Geo Location
S76	P07	Jacob	Cluster Exploration
S56	P03	Ava	IoT Devices
S92	P05	Alexandra	Cloud Deployment

In the above table, we have partial dependency; let us see how:

The prime key attributes are **StudentID** and **ProjectID**. As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent. The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent. The **ProjectName** can be determined by **ProjectID**, which makes the relation Partial Dependent.

Therefore, the <**StudentProject**> relation violates the 2NF in Normalization and is considered a bad database design.

Example (Table converted to 2NF)

To remove Partial Dependency and violation on 2NF, decompose the above tables:

<StudentInfo>

StudentID	ProjectID	StudentName
S89	P09	Olivia
S76	P07	Jacob
S56	P03	Ava
S92	P05	Alexandra

<ProjectInfo>

ProjectID	ProjectName
P09	Geo Location
P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

Q.1.b)Consider following set of Functional dependencies(F) [6]

A→BC,B→C,A→B,AB→C

Calculate Canonical cover for above given F.

Ans:

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$.

Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β.

/* Note: the test for extraneous attributes is done using F_c, not F */

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c.

until (F_c does not change)

Let us compute the canonical cover for F.

- There are two functional dependencies with the same set of attributes on the left side of the arrow:

$$A \rightarrow BC$$
$$A \rightarrow B$$

We combine these functional dependencies into $A \rightarrow BC$.

- A is extraneous in $AB \rightarrow C$ because F logically implies $(F - \{AB \rightarrow C\}) \cup \{B \rightarrow C\}$. This assertion is true because $B \rightarrow C$ is already in our set of functional dependencies.
- C is extraneous in $A \rightarrow BC$, since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow C$.

Thus, our canonical cover is:

$$A \rightarrow B$$
$$B \rightarrow C$$

Q.1)c)Define the following terms: [4]

a)Trivial functional dependency

A functional dependency is trivial if it is satisfied by all instances of a relation. In general, $\alpha \rightarrow \beta$is trivial if$\beta \subseteq \alpha$.

E.g.

customer-name, loan-number →customer-name

customer-name →customer-name

b) Multivalued dependency

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation.

Example:

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

In this example, Maf_year and Color are independent of each other but dependent on Car_model. In this example, these two columns are said to be multivalue dependent on car_model.

OR

Q.2) a. Explain Third Normal form with suitable example? [6]

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute. In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Q.2)b.Explain Armstrong's Axioms for functional dependencies. [6]

Reflexivity rule. If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

Augmentation rule. If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

Transitivity rule. If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

Union rule. If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.

Decomposition rule. If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

Pseudotransitivity rule. If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.

Q.2)c. Write short note on: Codd's Rule (any 4) [4]

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one of the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use

the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to return sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Q.3) a. Explain two-phase locking protocol with example. [6]

Two-Phase Locking Protocol:-This protocol requires that each transaction issue lock and unlock requests in two phases:

1.**Growingphase**.Atransactionmayobtainlocks,but may not release any lock.

2. **Shrinking phase**. A transaction may release locks, but may not obtain any new locks.

Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests. For example, transactions T3 andT4 are two phase. On the other hand, transactions T1 and T2 are not two phase. The unlock instructions do not need to appear at the end of the transaction. For example, in the case of transaction T3, we could move the unlock(B) instruction to just after the lock-X(A) instruction, and still retain the two-phase locking property.

```
T1: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    unlock(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(A).
```

Figure 16.2 Transaction T_1.

```
T2: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A + B).
```

Figure 16.3 Transaction T_2.

```
T3: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(B);
    unlock(A).
```

Figure 16.5 Transaction T_3.

```
T4: lock-S(A);
    read(A);
    lock-S(B);
    read(B);
    display(A + B);
    unlock(A);
    unlock(B).
```

Figure 16.6 Transaction T_4.

Q.3)b. Explain different properties of transaction. [4]

A transaction is a unit of program execution that accesses and possibly updates various data items. The transaction consists of all operations executed between the begin transaction and end transaction. To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity**-Eitheralloperationsofthetransactionarereflectedproperlyinthe database, or none are.
- **Consistency**- Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
- **Isolation**-Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj, it appears to Ti that either Tj finished execution before Ti started, or Tj started execution after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
- **Durability-A**fter a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Q.3) c. Consider the transaction (T3), transaction (T4) and transaction(T6) are transactions wing on data item Q. Schedule explaining the execution of T3,T4 and T6 are given below. Decide whether following schedule is view serializable or not. [4]

T3	T4	T6
Read(Q)		
	Write(Q)	
Write(Q)		
		Write(Q)

Ans:-Let us consider a schedule S in which there are two consecutive instructions Ii and Ij, of transactions Ti and Tj, respectively (i = j).
If Ii and Ij refer to different data items, then we can swap Ii and Ij without affecting the results of any instruction in the schedule. However, if Ii and Ij refer to the same data item Q, then the order of the two steps may matter.
1. Ii = read(Q), Ij = read(Q). The order of Ii and Ij does not matter,since the same value of Q is read by Ti and Tj, regardless of the order. 2. Ii =read(Q),Ij =write(Q).If Ii comes before Ij then Ti does not read the value of Q that is written by Tj in instruction Ij. If Ij comes before Ii, then Ti reads the value of Q that is written by Tj. Thus, the order of Ii and Ij matters.
3. Ii = write(Q), Ij = read(Q). The order of Ii and Ij matters for reasons similar to those of the previous case.
4. Ii = write(Q), Ij = write(Q). Since both instructions are write operations, the order of these instructions does not affect either Ti or Tj. However, the value obtained by the next read(Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other write(Q) instruction after Ii and Ij in S, then the order of Ii and Ij directly affects the final value of Q in the database state that results from schedule S.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Figure 15.12 Schedule 9—a view-serializable schedule.

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule. As an illustration, suppose that we augment schedule 7 with transaction T6, and obtain schedule 9 in Figure 15.12. **Schedule 9 is view serializable.** Indeed, it is view equivalent to the serial schedule<T3,T4,T6>,since the one read(Q) instruction reads the initial value of Q in both schedules, and T6 performs the final write of Q in both schedules. Hence **this schedule is view serializable.**

OR

Q.4)a. Explain different states of a transaction. [6]

A transaction must be in one of the following states:

• **Active**, the initial state; the transaction stays in this state while it is executing.
• **Partially committed**, after the final statement has been executed.
• **Failed**, after the discovery that normal execution can no longer proceed.
• **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
• **Committed**, after successful completion.

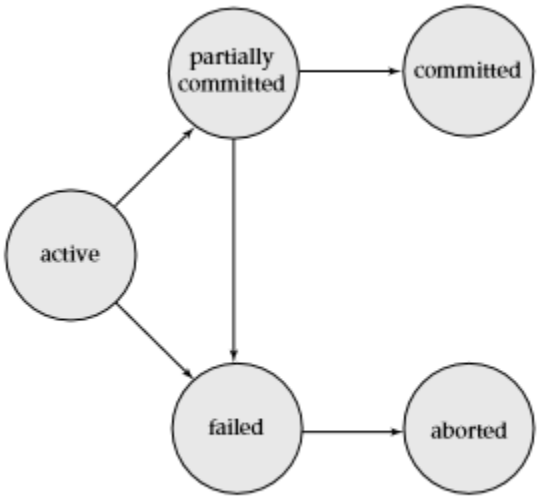

Figure 15.1 State diagram of a transaction.

It can restart the transaction, but only if the transaction was aborted as a result of some hardware or software error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction. It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

Q.4) b. Explain Lock-based protocol with example. [4]

Lock-Based Protocols:-There are various modes in which a data item may be locked. In this section, we restrict our attention to two modes:
1. Shared. If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write Q.
2. Exclusive. If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

	S	X
S	true	false
X	false	false

Figure 16.1 Lock-compatibility matrix comp.

We require that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. The transaction makes the request to the concurrency-control manager. The transaction can proceed with the operation only after the concurrency-control manager grants the lock to the transaction.

A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction. Similarly, a transaction requests an exclusive lock through the lock-X(Q) instruction. A transaction can unlock a data item Q by the unlock(Q) instruction.

To access a data item, transaction Ti must first lock that item. If the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. Thus, Ti is made to wait until all incompatible locks held by other transactions have been released.

Q.4) c. Consider the transactions (T3), and (T4) working on data items [4] A,B. Schedule explaining the execution of T3,T4 given below. Decide whether following schedule is conflict serializable or not? [4]

T3	**T4**
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)

Ans:- We say that *Ii* and *Ij* **conflict** if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.
The write(*A*) instruction of *T*1 conflicts with the read(*A*) instruction of *T*2.However, the write(*A*) instruction of *T*2 does not conflict with the read(*B*) instruction of *T*1, because the two instructions access different data items. Let *Ii* and *Ij* be consecutive instructions of a schedule *S*. If *Ii* and *Ij* are instructions of different transactions and *Ii* and *Ij* do not conflict, then we can swap the order of *Ii* and *Ij* to produce a new schedule *S'*. We expect *S* to be equivalent to *S'*, since all instructions appear in the same order in both schedules except for *Ii* and *Ij*, whose order does not matter. Since the write(*A*) instruction of *T*2 in schedule 3 of Figure 15.7 does not conflict with the read(*B*) instruction of *T*1, we can swap these instructions to generate an equivalent schedule.We continue to swap nonconflicting instructions:

- Swap the read(*B*) instruction of *T*1 with the read(*A*) instruction of *T*2.
- Swap the write(*B*) instruction of *T*1 with the write(*A*) instruction of *T*2.
- Swap the write(*B*) instruction of *T*1 with the read(*A*) instruction of *T*2.

The final result of these swaps, is a serial schedule. If a schedule *S* can be transformed into a schedule *S'* by a series of swaps of non-conflicting instructions, we say that *S* and *S'* are **conflict equivalent**.