Assignment-3

(1) Explain the informal design guidelines used as measures to determine the quality of relation schema design.

Four informal guidelines that may be used as measures to determine the quality of relation schema design are:

- Making sure that the semantics of the attributes is clear in the schema.
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples.
- Disallowing the possibility of generating spurious tuples.

(.1)

EMPLOYEE

Ename	SSN	B.date	Address	Dnumber
	P.K			

DEPARTMENT

Dname	D number	Dmgr-ssn
	P.K	F.K

Guideline 1.

→ Design a relational schema so that it is easy to explain its meaning.

→ Do not combine attributes from multiple entity types and relationship types into a

Single relation.

→ If a relation schema corresponds to one entity type or one relationship type, it is straight forward to interpret and to explain its meaning.

Example of violating Guideline 1:

(a) EMP_DEPT

Ename	SSn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

(b) EMP_PROJ

SSn	P_number	Hours	Ename	Pname	Plocation

FD1
FD2
FD3

(2) Update nomalies can be classified into insertion anomalies, deletion anomalies, and modification.

Guideline 2:

→ Design the base relation schemas so that no insertion deletion or modification anomalies are the present in the relations

→ If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

(3) Guideline 3

→ Avoiding placing attributes in a base relation whose values may frequently be NULL. If NULL's are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

(4) Guidelines 4.

→ Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related pairs in a way that guarantees that no spurious tuples are generated.

→ Avoid relations that contain matching attributes that are not combinations because joining on such attributes may produce spurious.

(2) Define Functional Dependency. Explain with an example.

A Functional Dependency, denoted by $X \rightarrow Y$ between two sets of attributes X and Y that are subsets of R specifies a constraint on the tuples in a relation state X of R.

A functional dependency, means that the values of the Y are determined by the values of X

→ Consider the relation schema EMP PROJ from the semantics of the attributes and the relation, the following functional dependencies should hold

(a) SSn → Ename

b) Pnumber → {Pname, Plocation}

c) {SSn, Pnumber} → Hours.

These functional dependencies specifies that

a) the value of a project's number (Pnumber) uniquely determines the employee name (Ename)

b) the value of a project's number (Pnumber) uniquely determines the project name (Pname) and location (Plocation) and

c) Combination of SSn and Pnumber values uniquely determines the no of hours the employee currently works on the project per week (hours).

③ Define Normal Form. Explain INF, 2NF and 3NF with suitable examples of each.

The Normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

First Normal Form (1NF):

→ A relation will be 1NF if it contains an atomic value.

→ It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.

→ First normal form disallows the multi-valued attribute, and their combinations.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	9064738238 7272786385	UP
20	Hary	8574838321	Bihar
12	Sam	8589830302 7390372389	Punjab.

The decomposition of Employee table into 1NF has been shown below

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATUS
14	John	9064738238	UP
14	John	7272786385	UP
20	Hary	8574838321	Bihar
12	Sam	8589830302	Punjab
12	Sam	7390372389	Punjab

Second Normal Form (2NF):

→ In the 2NF, relational must be in 1NF

→ In the second normal form, all the non-key attributes are fully functional dependent on the primary key.

Example:- Let's assume a school can store the data of teacher and the subjects they teach. In a school, a teacher can teach more than one subject.

Teacher Table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

To convert the given table into 2NF, we decompose it into two table

TEACHER_DETAIL table

Teacher_id
25
47
83

TEACHER_AGE

30
35
38

TEACHER_SUBJECT table:

Teacher_id	Subject
25	Chemistry
25	Biology.
47	English
83	Math
83	Computer

Third Normal Form (3NF):

→ A relation will be in 3NF if it is 2NF and not contain any transitive partial dependency.

→ 3NF is used to reduce the data duplication it is also used to achieve the data integrity.

→ If there is no transitive dependency for non-prime attributes, then the relation must be in NF.

→ A relation is in 3rd normal form if it holds atleast one of the following conditions for every non trivial function dependency $X \rightarrow Y$

i) X is a super key

ii) Y is a prime attribute, i.e each element of Y is part of some candidate key.

④. Define multivalued Dependency. Explain 4NF and 5NF with suitable examples of each

Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.

Fourth Normal Form (4NF)

→ A relation will be in 4NF if it is in Boyce Codd normal form and has no

multivalued dependency.

→ For a dependency A → B, if for a single value of A → multiple values of B exists, then the relation will be a multi valued dependency.

Example:

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey.

So to make the above tables into 4NF we can decompose it into two tables

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hocket

Fifth Normal Form (5NF)

→ A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

→ 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

→ 5NF is also known as Project-join Normal form (PJ/NF)

Example:

SUBJECT	LECTURER	SEMESTER
Computer	Anushka.	Sem1
Computer	John	Sem1
Math	John	Sem1
Math	AKash	Sem2
Chemistry	Praveen	Sem1

P1

SEMESTER	SUBJECT
Sem1	Computer.
Sem1	Math
Sem1	Chemistry
Sem2	Math

P2:

SUBJECT	LECTURER
Computer	Anushka
Computer	John
Math	John
Math	AKash
chemistry	Praveen

P3:

SEMESTER	LECTURER
Sem1	Anushka
Sem1	John
Sem1	John
Sem2	AKash

(5) What are the inference rules on FD's? How they are useful? Explain with examples.

The set of functional dependencies are specified by F on relation schema R, other functional dependencies can be inferred or deduced from the FD's in F.

For example: Dept has one * manager, the dept_no uniquely determines Mgr_ssn, and manager uniquely determines phone number called Mgr_phone then these two dependencies together imply that Dept_no → Mgr_phone.

(Dept_no → Mgr_ssn), (Mgr_ssn → Mgr_phone)

(Dept_no → Mgr_phone)

→ The set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F, it is denoted by F+.

→ The closure F+ of E is the set of all functional that can be inferred from F.

To determine a systematic way to infer dependencies, the set of inference rules are used to infer new dependences from a given set of dependences.

→ The notation $F \models X \rightarrow Y$ to denote that the functional dependency $X \rightarrow Y$ is inferred from the set of FD F. The FD $\{X, Y\} \rightarrow Z$ is abbreviated to $XY \rightarrow Z$, and the FD $\{X, Y, Z\} \rightarrow \{U, V\}$ is abbreviated to $XYZ \rightarrow UV$.

The six inference rules IR1 through IR6 for functional dependencies:

① IR1 (Reflexive Rule):

If $X \supseteq Y$, then $X \rightarrow Y$. The reflexive rule states that a set of attributes always determines itself or any of its subsets

Ex: {fname, lname} → {fname}

② IR2 [Augmentation rule):

$\{X \rightarrow Y\} \models XZ \rightarrow YZ)$

The augmentation rule (IR2) says that adding the same set of attributes to both the left and right hand sides of a dependency results in another valid dependency.

Ex: If {SSN} → {fname} then: {SSN, DName} →

{fname, DName}

③ IR3 [Transitive Rule]:

$\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

This functional dependencies are transitive

Ex: If: {SSN} → {DNO} {DNO} → {DName}
then:
{SSN} → {DName}

④ IR4 [Decomposition or projective Rule]
{X → YZ} |= {X → Y}. The decomposition rule IR4 says that we can remove attributes from the right hand side of a dependency, applying this rule repeatedly can decompose the FD X → YZ to X → Y and X → Z

Ex: If {SSN} → {fname, DNO} then
{SSN} → {fname}
{SSN} → {DNO}

⑤ IR5 [Union or additive Rule]
{X → Y, X → Z} |= X → YZ allows to combine a set of dependencies {X → A1, X → A2 - - X → An} into the single FD X → {A1, A2 - - - An}

⑥ IR6 [Pseudotransitive rule]:
{X → Y, WY → Z} |= WX → Z.
Allows us to replace a set of attributes from the left hand side of a dependency with another set X that functionally determines Y and can be derived from IR2 and IR3.

MODULE 5

① Explain transition diagram of a Transaction.

A Transaction is an atomic unit of work that should either be completed in its entirely or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates and commits or aborts. Therefore the recovery manager of the DBMS needs to keep track of the following operations:

- BEGIN TRANSACTION:
This marks the beginning of transaction execution
- READ or WRITE
These specify read or write operations on the database items that are executed as part of a transaction.
- END TRANSACTION:
This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to

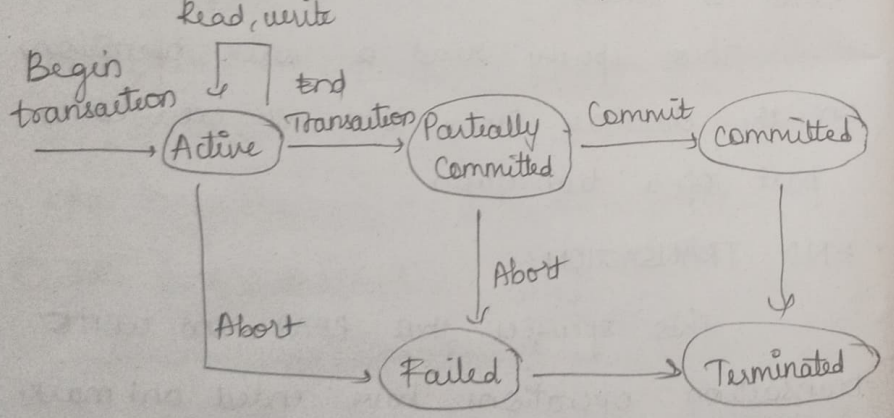
the database (committed) or whether the transaction has to be aborted because it violates Serializability or for some other reason

- COMMIT TRANSACTION:

This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

- ROLL BACK (or ABORT):

This signals that the transaction has ended unsuccessfully. so that any change or effects that the transaction may have applied to the database must be undone.



#) Why concurrency control is needed demonstrate with an example.

A. Several problems can occur when concurrent transactions execute in an uncontrolled manner. We illustrate some of these problems by referring to a much simplified airline reservations database in which a record is stored for each airlinescript.

→ Each record includes the no of reserved seats on that flight as a named data item, among other information.

→ Transaction T1 that transfers N reservations from one flight whose number of reserved seats are stored in the database item named X to another flight whose no of reserved seats is stored in the database item named Y.

T1	T2
read_item(X);	read_item(X);
X := X - N;	X := X + M;
write_item(X);	write_item(X);
read_item(Y);	
Y := Y + N;	
write_item(Y);	

The types of problems we may encounter with these two simple transactions if they run concurrently:

1) The Lost Update Problem
2) Temporary Update (or Dirty Read) Problem
3) The Incorrect summary Problem
4) The Unrepeatable Read Problem

1)

T1	T2
read_item(X);	
X := X-N;	read_item(X);
write_item(X);	X := X+M;
read_item(Y);	write_item(X);
Y := Y+N;	
write_item(Y);	

Time ↓

2)

T1	T2
read_item(X);	
X := X-N;	read_item(X);
write_item(X);	X := X+M;
read_item(Y);	write_item(X);

Time ↓

3)

T1	T2
	sum := 0;
read_item(X);	read_item(A);
X := X-N;	sum := sum+A;
write_item(X);	}
}	

read_item(Y);	sum := sum+X;
Y := Y+N;	read_item(Y);
write_item(Y);	sum := sum+Y;

The Unrepeatable Reead Problem: Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction between T between the two reads

⑧ Discuss the Desirable properties of Transactions.

Transactions should possess several properties often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS.

The following are the ACID properties:

- Atomicity

A transaction is an atomic unit of processing. It should either be performed in its entirely or not performed at all.

- Consistency preservation

A transaction should be consistency upresting, meaning that if it is completely executed from beginning to end without

interference from another transaction it should tell the database from one consistent state to another.

- Isolation

A Transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.

- Durability or permanency

The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

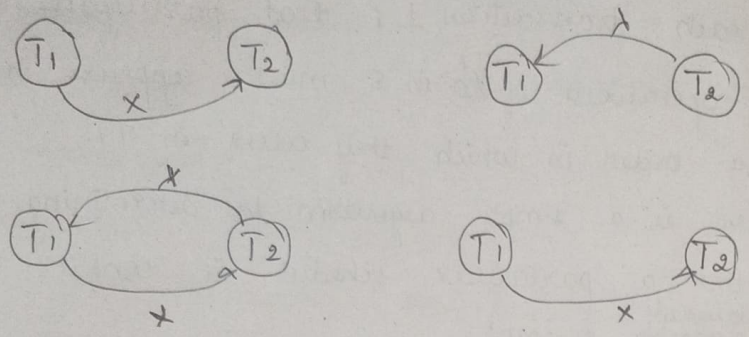
(9) What is schedule? Explain conflict Serializable schedule with example.

When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from all the various transactions is known as a schedule.

→ A schedule S of n transactions T1, T2 - Tn is an ordering of the operation of the transaction subject to the constraint that, for each transaction I_i that participates in S, the operations of T_i in S must appears in the same order in which they occur in T_i. There is a simple algorithm for determining whether a particular schedule is conflict serializable or not.

→ The Algorithm looks at only the read_item and write_item operations in a schedule to construct a precedence graph which is a directed graph $G = (N, E)$ that consists of a set of nodes $N = \{T_1, T_2, \ldots T_n\}$ and a set of directed edge $E = \{e_1, e_2 \ldots e_m\}$. There is one node in the graph for each Transaction T_i in the schedule. Each edge e_i in the graph is of the form $(T_j \rightarrow T_k)$, $1 \le j \le n$, $1 \le k \le n$. where T_j is the starting node of e_i and T_k is the ending node of e_i. Such an edge from node T_j to node T_k is created by the algorithm if one of operations in T_j appears in the Schedule before some conflicting operations in T_k.

The precedence graph is constructed - If there is a cycle in the precedence graph, schedule S is not (conflict) serializable; if there is no cycle, S is serializable.



(10) Briefly explain the recovery process.

Recovery concepts:

1) Recovery Outline and Categorization of recovery algorithms.

(2) Caching (Buffering) of Disk Blocks.

Two main strategies can be employed when flushing a modified buffer back to disk.

- in-place updating writes the buffer to the same original disk location, thus overwriting the old value of any changed data items on disk. Hence a single copy of each database disk block is maintained.
- Shadowing strategy writes an updated buffer at a different disk location, so multiple versions of data items can be maintained, but this approach is not typically used in practice.

(3) Write-Ahead logging, Steal / No-steal and Force / No-Force.

(4) Check points in the system log and fuzzy checkpointing

(5) Transaction Rollback and Cascading Rollback.

Recovery Techniques based on Immediate Update:

→ In these techniques, when a transaction issues an update command, the database on disk can be updated immediately, without any need to wait for the transaction to reach its commit point.

Two main categories of immediate update

(1) If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO ~~and~~ any operations of committed transaction.

(2) If the transaction is allowed to commit before all its changes are written to the database, we have the most general case, known as the UNDO / REDO recovery algorithm.