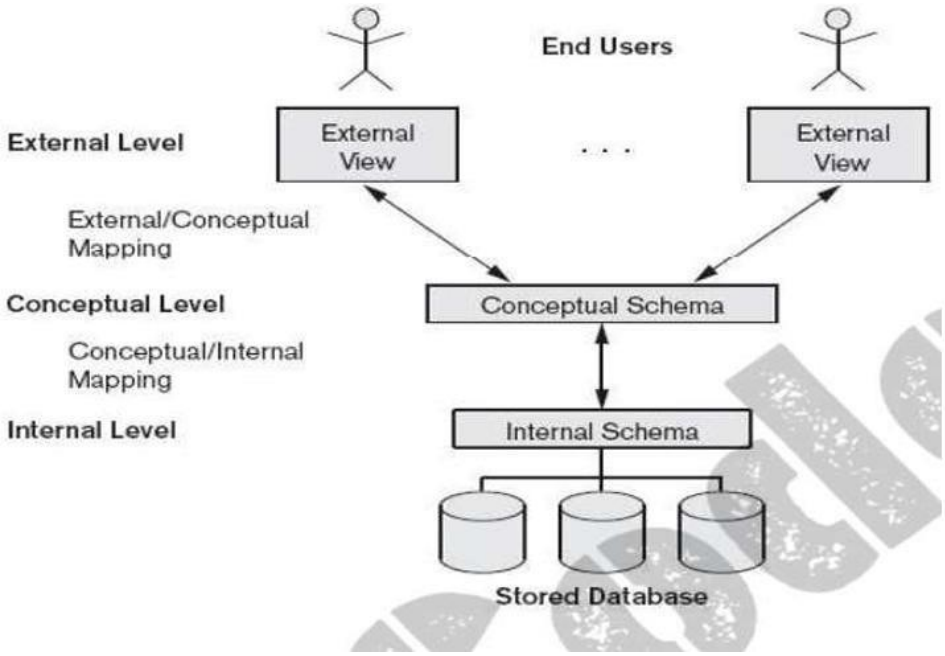
Sub:	VTU QP Solutions DATABASE MANAGEMENT SYSTEMS						
Sub code	MMC103	Duration:	3 hrs	Max Marks:	100	Sem:	I
Note:AnswerFIVEFULLQuestions,choosingONEfullquestionfromeachModule							

MODULE-1

Q1.a.ExplaintheThreeschemaarchitectureofaDBMSwithneatdiagram.Discussthe importance of each schema layer in managing database systems with suitable example.

Threeschemaarchitectureof DBMS:

The goal of the three-schema architecture is to separate the user applications from the physicaldatabase.Inthisarchitecture,schemascanbedefinedatthefollowingthreelevels:



a. Internal Level:

The internal level has an internal schema, which describes the physical storage struct the database.Theinternalschemausesaphysicaldatamodelanddescribesthecompletedetails of data storage and access paths for the database.

Itisimportantbecauseitmanageshowdataisstoredphysically,improvesperformanceand efficiency, and ensures fast data retrieval and storage optimization.

Example: Anadministratormightchangethestorageenginefrom InnoDBtoMyISAMor implement B-tree indexing for faster retrieval without the end-user ever noticing.

b. ConceptualLevel

The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe has a conceptual schema when a database system is implemented.

It is important because it defines the overall structure of the database, maintains data consistency and integrity, and acts as a bridge between user views and physical storage.

Example:Auniversitydatabase conceptualschemadefinestableslike STUDENT, COURSE, and ENROLLMENT along with their relationships, such as "a student enrolls in many courses".

c. External or View Level

The external or view level includes a number of external schemas or user views. Each external schemadescribesthepartofthedatabasethataparticularusergroupisinterested inandhidestherestofthedatabasefromthatusergroup.Eachexternalschemaistypically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

It is important because it provides data security and privacy by showing only the required data to each user, supports multiple customized views, and makes the system easy to use for different users.

Example:In abanking system, a customer sees only their personal account balance (ExternalView1),whileabanktellercanaccesstransactionhistoryandcustomerprofiles (External View 2).

Q1b.Definetheconceptofkeysinarelationaldatabase.Explainthedifferencesbetween candidate keys, primary keys, and foreign keys with examples.

Keys are fundamental elements of the relational database model that ensure uniqueness, data integrity, and efficient data access.

LetconsiderStudenttableasExample:

USN	NAME	AGE	PHONE
1	Tom	33	1234567890
2	Jerry	22	9087654321
3	John	55	8907654321

1. SUPERKEY

- Asuperkeyisagroupofsingleormultiplekeysthatuniquelyidentifies rowsinatable. It supports NULL values in rows.
- Asuperkeycancontainextraattributesthataren’tnecessaryforuniqueness.
- **For example,** if the "USN" column can uniquely identify a student, adding "NAME" to it will still form a valid super key, though it's unnecessary.

2. CANDIDATEKEY

- Acandidate key is a minimal super key, meaning it can uniquely identify a record but contains no extra attributes.
- Itisasuperkeywithno repeateddataiscalledacandidatekey.
- Theminimal setof attributesthat canuniquely identifya record.

- Acandidate key must contain uniquevalues, ensuringthat no two rows havethesame value in the candidate key's columns.
- Everytablemusthave at leastasinglecandidatekey.
- Atablecan havemultiplecandidatekeysbutonly oneprimary key.

Example: FortheSTUDENTtablebelow,USNcanbeacandidatekey,asituniquely identifies each record.

3. **PRIMARYKEY**
   - Ituniquelyidentifieseverytuple(row)anddoesnotallowduplicatevalues.
   - Itcannot be NULL, aseachrecord must haveavalid identifier.
   - Itmaybesingle-column orcomposite(madeofmultiplecolumns).
   - Databasesoftenorganizedatausingtheprimarykeytoallowfasteraccessandsearching.

   Example: The STUDENT table has the structure Student(USN, NAME,AGE, PHONE), where USN is the primary key.
4. **ALTERNATIVEKEY**
   - Analternatekeyisalsoreferredtoasasecondarykeybecauseitcanuniquelyidentify records in a table, just like the primary key.
   - Analternatekeycanconsistofoneormorecolumns(fields)thatcanuniquelyidentify a record, but it is not the primary key

   Example:In the STUDENT table, both USN and PHONE are candidate keys. If USN is chosen as the primary key, then PHONE would be considered an alternate key.

5. **FOREIGNKEY**
   - A foreignkeyinone table pointsto theprimarykeyinanother table,establishinga relationship between them.
   - Ithelpsconnecttwoormoretables,enablingyoutocreaterelationshipsbetweenthem. This is important for maintaining data integrity and preventing data redundancy.
   - Theyactasacross-referencebetweenthetables.

   Example: Student(usn,name),

   COURSE(course_id,cname),

   ENROLLMENT(usn,course_id)

   InEnrollment:

   - ID → Foreign key referencing Student(ID)

   CourseID→ForeignkeyreferencingCourse(CourseID)
6. **COMPOSITEKEY**
   - Itactsasaprimarykeyifthereisnoprimary keyina table
   - Twoormoreattributesareusedtogethertomakeacompositekey.
   - Differentcombinationsofattributesmaygivedifferentaccuracyintermsofidentifying the rows uniquely.

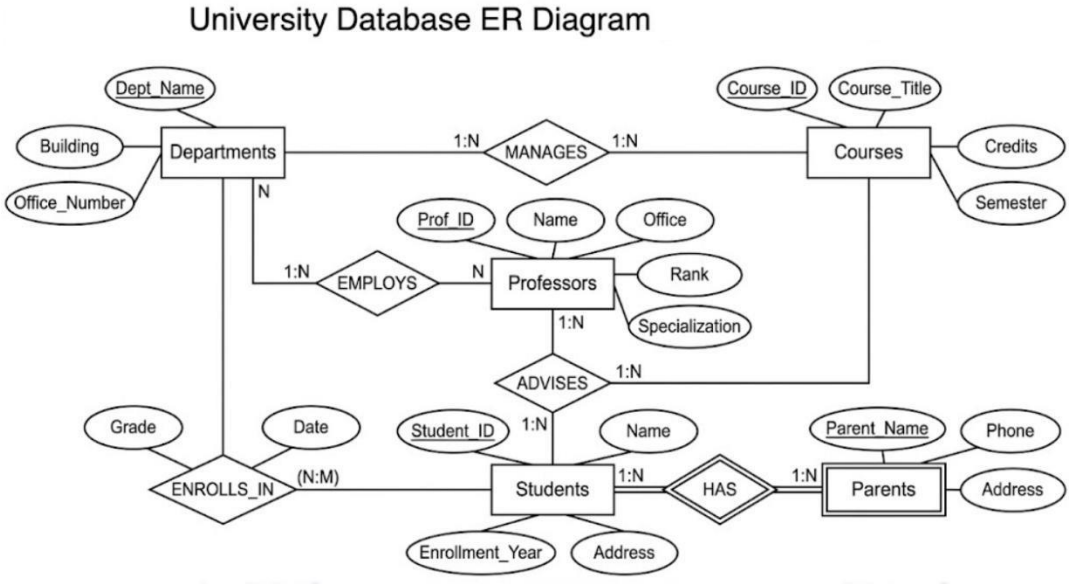
Example: In the STUDENT_COURSE table, {STUD_NO, COURSE_NO} can form acomposite key to uniquely identify each record.

Features	**Candidatekey**	**Primarykey**	**Foreignkey**
Definition	Possible unique identifier	Selected unique identifier	Referencetoanothertable
Number	Multiple allowed	Only one	Multiple allowed
NULL allowed	No	No	Yes(sometimesallowed)
Purpose	Identify records	Mainidentifier	Maintain relationships

OR

Q2 ConsideraUniversitydatabasewiththefollowingentities:Students,Courses, Professors,ParentsandDepartments

a. DrawanERdiagramrepresentingtherelationshipsbetweentheseentities. Clearlyspecifyentitytypes,attributes,relationships,keysandcardinalities



b. Convert the ER diagram into relational tables, specifying primary keys and foreign keys

1. Department

- deptID(PK)
- deptName
- office

2. Student

- studentID(PK)
- firstName
- lastName
- age
- deptID(FK→Department.deptID)

3. Parent

- parentID(PK)
- parentName
- phone
- studentID(FK→Student.studentID)

4. Professor

- profID(PK)
- firstName
- lastName
- specialization
- deptID(FK→Department.deptID)

5. Course

- courseID(PK)
- courseName
- credits
- profID(FK→Professor.profID)

6. Enrollment(forM:Nrelationship)

- studentID(PK,FK→Student.studentID)
- courseID(PK,FK→ Course.courseID)

- PK(PrimaryKey)→uniquelyidentifieseachrecord
- FK(ForeignKey)→ creates relationships
- M:Nrelationship→convertedintoseparatetable(Enrollment)

MODULE-2

Q3.Considerthefollowingrelationalschemaforalibrarydatabase

GivenSchema:

- Books(BookID,Title,Author,Genre,PublisherID)
- Publishers(PublisherID,PublisherName,Location)
- BorrowedBooks(MemberID,BookID,BorrowDate)

a.Retrieveallbookswrittenby"J.K.Rowling"

$$\sigma_{Author='J.K.Rowling'}(Books)$$

b. Findnamesofpublisherslocatedin"NewYork"

$$\pi_{PublisherName}(\sigma_{Location='NewYork'}(Publishers))$$

c. ListallBookIDsthathavebeenborrowedatleastonce

$$\pi_{BookID}(BorrowedBooks)$$

d. RetrievetitlesofbooksthathaveNOTbeen borrowed

$$\pi_{Title}(Books) - \pi_{Title}(Books \bowtie BorrowedBooks)$$

e. Displaypublishernamesalongwithtitlesofbooksthey published

$$\pi_{PublisherName,Title}(Books \bowtie Publishers)$$

OR

Q4Considerthefollowingdatabaseschemaforanemployeemanagementsystem: Employee

(EmpID, Name, Department, Salary, ManagerID)

Department(DeptID,DeptName,Location) Write

SQL queries for the following:

a.CreatetheEmployeeandDepartmenttableswithappropriatedatatypesand constraints.

b.InsertanewemployeerecordintotheEmployeetable.

c.Increasethesalaryofemployeesinthe"IT"departmentby10%.

d.Deleteemployeesearninglessthan30,000.

e.RetrievethenamesofemployeesandtheirdepartmentnamesusingaJOIN.

GivenSchema:

- Employee(EmpID,Name,Department,Salary,ManagerID)

- Department(DeptID,DeptName,Location)

a) CreateTables

```
CREATETABLE Department (
   DeptIDINTPRIMARYKEY,
   DeptNameVARCHAR(50),
   Location VARCHAR(50)
);

CREATE TABLE Employee (
   EmpIDINTPRIMARYKEY,
   Name VARCHAR(50),
   DepartmentVARCHAR(50),
   Salary DECIMAL(10,2),
   ManagerIDINT,
   FOREIGNKEY(ManagerID)REFERENCESEmployee(EmpID)
);
```

b) InsertaNew Employee

```
INSERTINTOEmployee(EmpID,Name,Department,Salary,ManagerID) VALUES
(101, 'Rahul', 'IT', 50000, NULL);
```

c) IncreaseSalaryofITEmployeesby10%

```
UPDATEEmployee
SETSalary = Salary * 1.10
WHEREDepartment ='IT';
```

d) DeleteEmployeesEarningLessThan 30,000

```
DELETEFROMEmployee
WHERE Salary < 30000;
```

e) **RetrieveEmployeeNameswithDepartmentNames(JOIN)**

```
SELECTE.Name,D.DeptName
FROM Employee E
JOINDepartmentD
ONE.Department=D.DeptName;
```

MODULE-3

Q5a.Explaintheconceptof Normalizationanditsimportanceindatabasedesign.

Normalization is a process in database design that organizes data in a way that reduces redundancy and ensures data integrity. The goal of normalization is to decompose a large, complextableintosmaller,moremanageabletableswhileminimizingdataanomaliessuchas update anomalies, insert anomalies, and delete anomalies.

Normalization is a systematic process of organizing data in a database to reduce redundancy and improve data integrity. It divides large tables into smaller, well-structured tables and establishes relationships between them.

Importanceof Normalization

1. EliminatesData Redundancy

Normalization ensures that each pieceofdatais stored only oncein thedatabase.This reduces unnecessary duplication and saves storage space.

2. ImprovesData Integrity

Itpreventsanomaliessuchas:

- Insertion anomaly
- Update anomaly
- Deletion anomaly
  Thus,ensuringaccurateandreliabledataacrossalltables.

3. EnhancesData Consistency

Whendataisupdatedinoneplace,itisautomaticallyconsistentthroughoutthe database, avoiding conflicting or mismatched information.

4. BetterDataOrganization

Normalizationorganizesdata intologicaltablesbasedonrelationships,makingthe database structure clear and easy to understand.

5. EasierMaintenance

Anormalized database is easier to maintain, as changes or updates need to be made in only one place.

6. ImprovesQueryEfficiency

Well-structuredtablesmakeiteasiertowriteefficientqueriesandretrievedataquickly.

7. SupportsScalability

Normalized databases can handle growth more effectively, making them suitable for large systems and OLTP (Online Transaction Processing) applications.

Q5b.DefineFunctionalDependencywithexample?

Functional Dependency in DBMS

Functional Dependency(FD) is a key concept in Database Management Systems (DBMS), particularlyinNormalization.Itdescribesarelationshipbetweenattributesinarelation(table), ensuring data consistency and eliminating redundancy.

DEFINITION:

Afunctionaldependencyisaconstraintbetweentwosetsofattributesinarelation.Itisdenoted as:

X→Y

This means:

- Iftwotuples(rows)havethesamevaluesforX, theymusthavethesame valuesforY.
- Xiscalledthedeterminant(left-handside).
- Yiscalledthedependent(right-hand side).

Example

ConsiderarelationSTUDENT(Roll_No,Name, Course,Age)withthefollowing data:

Roll_No	Name	Course	Age
101	Alex	CS	20
102	Bob	IT	22
103	Charlie	CS	21
101	Alex	CS	20

Here,weobservethat Roll_No→ Name,Course,Age because:

- If two students have the same Roll_No, they must have the same Name, Course, andAge.

Thus, Roll_No functionally determines Name, Course, and Age.

TypesofFunctionalDependencies

1. TrivialFunctionalDependency

- Adependency istrivialif theright-handside(Y) isasubset ofthe left-hand side(X).
- Example:

{A,B}→AistrivialbecauseAisalreadyintheleft-handside.

2. Non-TrivialFunctionalDependency

- AdependencyisnotrivialifYisnota subsetof X.
- Example:

Roll_No→Nameisnon-trivialbecause NameisnotpartofRoll_No.

3. FullyFunctionalDependency

- Afunctional dependency X→Yis fully dependent ifremoving any attributefrom Xbreaks the dependency.

- Example:

{Roll_No,Course}→Instructoro

If Instructor depends on both Roll_No and Course, removing Course will break the dependency.

4. PartialFunctionalDependency

- IfYdependsonapartofX,not thewholeset, itispartial.

- Example:

{Roll_No,Course}→Instructor,butInstructordependsonlyonCourse,notonRoll_No. This

leads to 2NF (Second Normal Form) violation in Normalization.

5. TransitiveFunctionalDependency

- IfX→YX→YX→YandY→ZY→ZY→Z,thenX→ZX→ ZX→Z.

- Example:

Roll_No→CourseandCourse→Instructor,thenRoll_No→Instructor. This

leads to 3NF (Third Normal Form) violation.

Q5c.DiscussNon-LossDecompositionwithexample.

In database normalization**,** non-loss decomposition (or lossless decomposition) refers to the process of breaking down a relation (table) into two or more smaller relations without losing any information.This ensures that the original relation can be reconstructed through a natural join operation without generating spurious tuples

Lossless(Non-Loss) Decomposition:

- AdecompositionofrelationRRRintosub-relationsR1,R2,…,Rnislosslessifthenaturaljoin of these sub-relations results in exactly the original relation R.

- Formally,R1⋈R2⋈⋯⋈Rn=R.

- Noextra(spurious)tuplesshouldappear, andno informationshouldbe lost.

LosslessJoinCondition(Armstrong'sTheorem)

decomposition R into R1 and R2 is lossless if:

(R1∩R2)→R1or (R1∩R2) →R2

Thismeansthatthecommonattributesbetweenthedecomposedrelationsmustactasakeyfor at least one of the decomposed relations.

ExampleofNon-LossDecomposition Original

Relation:

ConsiderarelationR(Eno,Ename,Dept,Dname)withthefollowingfunctionaldependencies (FDs):

Eno→Ename,Dept andDept→Dname

DecomposingRintoTwoRelations:

1. R1(Eno,Ename,Dept)

2. R2(Dept,Dname)

CheckingforLossless Decomposition

- ThecommonattributebetweenR1andR2isDept.

- SinceDept→Dnameholds,DeptisakeyforR2,ensuringthatnoinformationislost.Thus, the decomposition is lossless.

Example:BeforeDecomposition

OriginalEmployeeTable(BeforeDecomposition):

Eno	**Ename**	**Dept**	**Dname**
101	John	D1	HR
102	Alice	D2	Finance
103	Bob	D1	HR
104	Carol	D3	IT

AfterDecompositionintoSmallerTables

R1(EmployeeInformation)

Eno	**Ename**	**Dept**
101	John	D1
102	Alice	D2
103	Bob	D1
104	Carol	D3

R2(DepartmentInformation)

Dept	Dname
D1	HR
D2	Finance
D3	IT

QueryingtheDecomposedTables

Now,wewilljointhedecomposedtablesR1andR2toretrievethesamedata. SELECT

R1.Eno, R1.Ename, R1.Dept, R2.Dname

FROMR1

JOINR2ONR1.Dept=R2.Dept;

ResultAfterJOIN:

Eno	Ename	Dept	Dname
101	John	D1	HR
102	Alice	D2	Finance
103	Bob	D1	HR
104	Carol	D3	IT

ImportantofNon-Loss Decomposition

- Preventsredundancyandanomaliesin databases.
- Ensuresdataintegritywhilemaintainingefficient storage.
- Helpsinachievinghighernormalforms(3NF,BCNF,etc.)withoutlosinginformation.

Q5d.Whataretheadvantagesofdependencypreservationindatabasenormalization?

Dependency Preservation

Dependency Preservation is a crucial concept in database normalization. It refers to the property of a decomposition of a relational database schema that ensures all the functional dependencies (FDs) that existed in the original schema are preserved after the decomposition intosmallerrelations(tables).Inotherwords,thedecompositionshouldbedoneinsuchaway thatwecanstillenforcealltheoriginalfunctionaldependenciesbyusingonlythedecomposed relations (i.e., without having to join the relations back together).

AdvantagesofDependency Preservation

1. PreservesDataConstraints

It ensures that all functional dependencies are maintained even after decomposition. This meanstheoriginalrulesandrelationshipsdefinedinthedatabasearenotlost,helpingmaintain correctness of data.

2. SimplifiesConstraintEnforcement

Constraintscanbecheckeddirectlyonindividualtableswithoutperformingjoins.Thismakes iteasierforthedatabase systemtoenforcerules andreducestheeffortrequiredforvalidation.

3. ImprovesQueryPerformance
Sincethereisnoneedtojoinmultipletablestoverifydependencies,queryexecutionbecomes faster. This improves the overall performance of the database system.

4. ReducesComplexity
Avoids complexoperationsrequiredto enforce constraintsacross multipletables.This makes the database design simpler and easier to understand.

5. MaintainsDataConsistency
Ensuresthatthedataremainsaccurateandconsistentacrossalltables.Ithelpsprevent anomalies during insertion, deletion, and updating of records.

6. Easy Maintenance
Database maintenance becomes easier because constraints are handled at the table level.Any changes can be made without affecting multiple relations.

WhyisDependencyPreservationImportant?

Dependencypreservationisimportantbecause:

- Itensuresthatconstraintsonthedataarepreserved,evenafterthedatabaseisdecomposed into multiple smaller tables.
- It simplifies query optimization and ensures that data integrity constraints (like foreign keys and functional dependencies) can be checked directly on the smaller relations without needing to join them.
- Without dependency preservation, enforcing the original constraints may require complex joins, which can lead to performance issues or even loss of consistency.

OR

Q6. Consider the following table:

Order ID	Customer Name	Product	Quantity	Price	Total Amount	Customer Address
101	Alice	Laptop	1	800	800	NY,USA
102	Bob	Phone	2	500	1000	CA,USA
103	Alice	Mouse	1	50	50	NY,USA

a. Identify the functional dependencies in the table

Functional Dependency - A functional dependency (FD) describes the relationship between attributes. A functional dependency is denoted by X--->Y which means attribute Y is functionally dependent on attribute X.

Functional Dependencies identified in the given table are as follows:

- **OrderID → CustomerNamei.e**Each order belongs to one customer
- **CustomerName → CustomerAddressi.e**Each customer has a fixed address

- **Product → Price i.e**Each product has a fixed price
- **(OrderID, Product) → Quantity i.e**Composite key determines quantity
- **Quantity, Price → TotalAmount**I.e*Derived attribute*

b. Convert the 1NF table to Second Normal Form(2NF) and justify your changes.
1NF - A relation is said to be in 1NF if all attributes contains atomic or indivisible value and no repeating groups.

The given table satisifes 1NF.
The attribute CustomerAddress (e.g., "NY, USA") does not violate 1NF because it is treated as a single atomic value. 1NF only requires that each field contains indivisible values and does not contain repeating groups. However, for better database design, the address can be further decomposed into smaller attributes like City and Country.

2NF - A relation is said to be in 2NF if it is in 1NF and has no partial dependencies.

Converting 1NF to 2NF :
For the given table:
Composite key is (OrderId,Product)
Partial Functional Dependencies are
1) OrderID -->CustomerName
2) Product -->Price
Table Decomposition is:
Orders Table(OrderID,CustomerName,CustomerAddress)

OrderID	CustomerName	CustomerAddress
101	Alice	NY, USA
102	Bob	CA, USA
103	Alice	NY, USA

Products Table(Product,Price)

Product	Price
Laptop	800
Phone	500
Mouse	50

OrderDetailTable(OrderID, Product,Quantity,TotalAmount)

OrderID	Product	Quantity	TotalAmount
101	Laptop	1	800
102	Phone	2	1000
103	Mouse	1	50

Justification :
1) Orders Table - All attributes depend fully on primary key OrderID. There is no composite key & partial dependency.So Orders table satisifies 2NF.
2) Products Table - Price attributes depend fully on primary key Product. There is no composite key & partial dependency.So Orders table satisifies 2NF.

3) OrderDetails Table - Quantity and TotalAmount depend on (OrderID, Product) together. There is no attribute depends on only OrderID or only Product, so no partial dependency.SoOrderDetails table satisifies 2NF.

c. Convert the 2NF table to Third Normal Form(3NF) and explain its advantages.
3NF - A relation is said to be in 3NF if it is in 2NF & has no transitive dependency.

For the decomposed Oderstable:
Transitive dependency is:
OrderID -->CustomerName
CustomerName -->CustomerAddress
So OrderID -->CustomerAddress

Orders table decompositionis :

Customers Table(CustomerName,CustomerAddress)

CustomerName	CustomerAddress
Alice	NY, USA
Bob	CA, USA

Orders Table(OrderID,CustomerName)

OrderID	CustomerName
101	Alice
102	Bob
103	Alice

Products Table(Product,Price)

Product	Price
Laptop	800
Phone	500
Mouse	50

OrderDetailsTable(OrderID,Product,Quantity,TotalAmount)

OrderID	Product	Quantity	TotalAmount
101	Laptop	1	800
102	Phone	2	1000
103	Mouse	1	50

Justification :
All relations are in 2NF and there are no transitive dependencies present. Every non-key attribute is directly dependent on the primary key only. Hence, the decomposition satisfies Third Normal Form (3NF).

Advantages of 3NF :
1) Elimination of Redundancy : Data is stored only once (e.g., customer address stored in **Customers table only**).Avoids repeated storage of same information.
2) Removal of Update Anomalies :Changes need to be made in only one place.
Example: If Alice's address changes, update only in **Customers table.**
3) Prevention of Insertion Anomalies :New data can be inserted without unnecessary dependencies.
Example: New product can be added without creating an order.
4) Prevention of Deletion Anomalies :Deleting one record does not remove important data.
Example: Deleting an order does not remove customer details.
5) Improved Data Integrity :Ensures consistency and correctness of data and which ensures dependencies are properly maintained .
6) Better Database Design: Tables are logically structured which makes database easier to maintain and extend.

d. Differentiate BCNF from 3NF
3NF - A relation is in Third Normal Form (3NF)if:It is in 2NF, and For every functional dependency $X \rightarrow Y$, at least one of the following holdstrue : X is a super key, OR Y is a prime attribute (part of some candidate key)

BCNF - A relation is in Boyce-Codd Normal Form (BCNF)if:For every functional dependency $X \rightarrow Y$,X must be a super key

Difference between 3NF &BCNF :

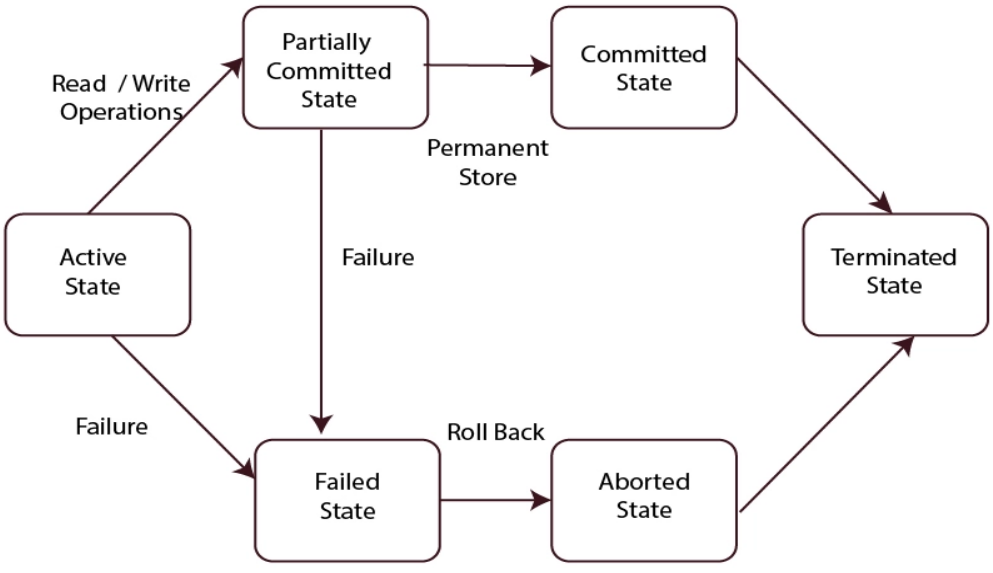
Basis	Third Normal Form (3NF)	Boyce-Codd Normal Form (BCNF)
Definition	A relation is in 3NF if for every FD $X \rightarrow Y$, X is a super key or Y is a prime attribute	A relation is in BCNF if for every FD $X \rightarrow Y$, X must be a super key
Strictness	Less strict	More strict than 3NF
Dependency Rule	Allows some non-key determinants	Does not allow any non-key determinant
Redundancy	May still have some redundancy	Removes redundancy more completely
Anomalies	Some anomalies may still exist	Eliminates all anomalies due to FDs
Super Key Requirement	Not always required	Always required
Handling of Prime Attributes	Allows dependency if RHS is prime	No such relaxation

Basis	Third Normal Form (3NF)	Boyce-Codd Normal Form (BCNF)
Normalization Level	Lower than BCNF	Higher than 3NF
Dependency Preservation	Usually preserved	May not be preserved
Complexity	Easier to achieve and implement	More complex to achieve
Decomposition	Less decomposition required	May require more decomposition
Practical Usage	Widely used in real-world databases	Used in high-integrity systems

MODULE-4

Q7 a. Define a transaction in DBMS and explain different states of a transaction with a neat diagram

Transaction states in a DBMS represent the various stages a transaction passes through from execution to completion, crucial for ensuring ACID properties. The primary states are Active, Partially Committed, Committed, Failed, and Aborted. These states manage operations, ensure data integrity, and handle failures by tracking execution.



1. Active

This is the initial state of every transaction. The transaction begins execution and is actively performing read and/or write operations on the database. It stays in this state until either all its operations are completed, or an error is encountered.

2. Partially Committed

Once the final operation of the transaction has been executed, it moves to the partially

committed state. The changes exist in the system's memory (buffer), but have not yet been permanently written to the disk. The DBMS now checks whether it is safe to commit.

3. Committed
If the system verifies that all operations completed successfully and permanently writes the changes to the database, the transaction enters the committed state. At this point, the changes are durable and cannot be undone. This is the successful terminal state.

4. Failed
If an error occurs either during the active state (e.g., hardware failure, deadlock, constraint violation) or during the partially committed state (e.g., disk write failure), the transaction moves to the failed state. Normal execution can no longer proceed.

5. Aborted
After failure, the DBMS performs a rollback — it undoes all changes that the failed transaction had made, restoring the database to the state it was in before the transaction began. From the aborted state, the system has two choices: restart the transaction (if the failure was due to a non-logical cause like hardware) or terminate it (if the error was logical/unrecoverable).

6. Terminated
This is the final state for any transaction — reached from either committed or aborted. The transaction is completely finished and removed from the system. No further processing occurs.

Q7 b. How are atomicity and durability implemented in DBMS?
Atomicity is a database property guaranteeing that all operations in a transaction succeed or none do. If any part fails, the entire transaction is aborted and the database restores its previous state, following the all-or-nothing principle.
Durability ensures that once a transaction is committed, its changes are permanent and survive system failures like power outages. The data is saved to non-volatile storage, making it available even after a system restart.

Implementation of Atomicity and Durability
The recovery manager uses specific techniques to implement these properties and maintain system reliability.

1. Transaction Logs (Undo and Redo Logs)
The system uses a transaction log on stable storage to record every change. To ensure atomicity, an undo log stores original values to roll back failed transactions. To ensure durability, a redo log stores new values, allowing the system to re-apply changes if a crash occurs before memory data is written to the disk.

2. Write-Ahead Logging (WAL) Protocol
The WAL protocol requires that log records describing a change must be moved to stable storage before the actual data is updated on the disk. This rule ensures the system always has a permanent record available to either undo or redo operations during recovery.

3. Shadow Paging
Shadow paging maintains two page tables: a current version and a shadow version. Updates are made on new pages while the shadow table preserves the original state. For atomicity, a failed transaction simply discards the current table; for durability, the shadow table is replaced by the current one upon a successful commit.

4. Checkpointing
Checkpoints are periodic operations where the DBMS flushes all committed data from volatile memory to the permanent disk. This limits the amount of log processing required during a restart, as the system only needs to review records created after the last successful checkpoint.

Q7 c. Provide an example where atomicity is violated. How can this issue be resolved?
Example of Atomicity Violation
An atomicity violation occurs when a transaction consisting of multiple steps is interrupted, leaving the database in a partial or inconsistent state.

Scenario: Bank Fund Transfer Suppose Person A wants to transfer 500 units of currency to Person B. This transaction involves two distinct operations:

1. Debit: Subtract 500 from Account A.
2. Credit: Add 500 to Account B.

The Violation: If the system crashes or a network error occurs immediately after the debit operation but before the credit operation, the 500 units are removed from Account A but never arrive in Account B. Because the transaction was not treated as a single, indivisible unit, the money effectively vanishes from the system. This partial execution leaves the database inconsistent.

- How the Issue is Resolved

To resolve atomicity violations, a DBMS uses a Recovery Manager and specific logging protocols to ensure the "all-or-nothing" rule.

1. Transaction Commit and Rollback The DBMS groups the debit and credit operations into a single logical unit. The changes are only made permanent (Committed) if both operations succeed. If any part fails, the system triggers a Rollback, which cancels every action taken since the start of that specific transaction.

2. Use of Undo Logs Before modifying any data on the disk, the DBMS records the original values in an Undo Log. In the banking example, the system records that Account A originally had a certain balance. If the crash occurs after the debit, the recovery manager reads the Undo Log upon restart and restores Account A to its original balance, ensuring no partial data remains.

3. Write-Ahead Logging (WAL) The system follows the WAL protocol, ensuring that the log entry describing the transfer is written to stable storage before the actual database records are changed. This ensures that even if the power cuts out, the system has a "memory" of the failed attempt and can automatically fix the inconsistency during the next boot-up

OR

Q8 a. What are the two types of lock? Explain lock compatibility matrix

In database management systems, locking is a mechanism used to manage concurrent access to data, ensuring that multiple transactions do not interfere with each other and maintain data consistency.

The Two Types of Locks

There are two primary types of locks used to control access to database items:

1. **Shared Lock (S-Lock):** This lock is used when a transaction only needs to read a data item. Multiple transactions can hold a shared lock on the same data item simultaneously. It allows concurrent reads but prevents any transaction from modifying the data.
2. **Exclusive Lock (X-Lock):** This lock is used when a transaction needs to update or delete a data item. Only one transaction can hold an exclusive lock on a data item at any given time. While an exclusive lock is held, no other transaction can obtain any type of lock (Shared or Exclusive) on that same item.

Lock Compatibility Matrix

The lock compatibility matrix is a table used to determine whether a lock request from one transaction can be granted if another transaction already holds a lock on the same data item. It defines the rules for concurrency: "Compatible" means the second lock can be granted, while "Incompatible" means the requesting transaction must wait until the first lock is released.

Lock Held \ Lock Requested	Shared (S)	Exclusive (X)
Shared (S)	Compatible	Incompatible
Exclusive (X)	Incompatible	Incompatible

- Shared vs. Shared (S, S): If Transaction 1 holds a Shared lock, and Transaction 2 requests a Shared lock, the request is Compatible. This allows multiple users to read the same record at the same time.

- Shared vs. Exclusive (S, X): If Transaction 1 holds a Shared lock and Transaction 2 wants to write (Exclusive), the request is Incompatible. The writer must wait until all readers have finished to ensure it doesn't change data while others are looking at it.

- Exclusive vs. Shared (X, S): If Transaction 1 holds an Exclusive lock, Transaction 2 cannot even read the data. This is Incompatible because the data might be in the middle of being changed, which would lead to an inconsistent read.

- Exclusive vs. Exclusive (X, X): If one transaction is writing to a record, no other transaction can write to it. This is Incompatible to prevent lost updates or data corruption.

Q8 b. Explain how Timestamp-Based Protocols work. How do these locks prevent conflicts?

The Timestamp-Based Protocol is a non-locking concurrency control mechanism that uses time values to determine the execution order of transactions. Unlike locking protocols that make transactions wait, this protocol uses timestamps to decide if an operation should proceed or be aborted.

How Timestamp-Based Protocols Work

Every transaction is assigned a unique fixed identifier called a Timestamp. This is typically assigned by the system using the system clock or a logical counter at the time the transaction starts.

For every data item Q in the database, the protocol maintains two timestamp values:

1. W-timestamp(Q): The largest (most recent) timestamp of any transaction that successfully executed a write(Q).
2. R-timestamp(Q): The largest (most recent) timestamp of any transaction that successfully executed a read(Q).

1. Read Operation (Timestamp)

A read is allowed only if the transaction's timestamp is greater than or equal to the data item's last write timestamp. This ensures a transaction doesn't read "old" data that a newer transaction has already updated; if it's too old, the transaction is rolled back.

2. Write Operation (Timestamp)

A write is allowed only if the transaction's timestamp is greater than or equal to both the last read and the last write timestamps of the data item. If a newer transaction has already read or written the data, the current "older" transaction is rejected and rolled back to prevent it from overwriting more recent information.

How These Protocols Prevent Conflicts

While the user asked how "locks" prevent conflicts, it is important to clarify that Timestamp

protocols are lock-free. They prevent conflicts through Ordering and Rollbacks rather than holding resources.

1. Eliminating Deadlocks: Because transactions never "wait" for each other (they are either executed or immediately aborted), circular waiting cannot occur. This completely eliminates the possibility of deadlocks.

2. Ensuring Serializability: By using the start time as a priority, the protocol forces the execution to be equivalent to a serial schedule where transactions run one after another in the order of their timestamps.

3. Conflict Resolution:
    - Read-Write Conflict: If a transaction tries to write a value that a younger transaction has already read, the older transaction is killed to protect the integrity of the younger one's view.
    - Write-Write Conflict: If two transactions try to write to the same item, the protocol ensures only the "latest" one survives or that they occur in the correct chronological order.

Q8 c. Explain two phase locking protocol with a diagram. How is it different from pre-claiming lock protocol?

The Two-Phase Locking (2PL) protocol is a concurrency control method that ensures serializability by requiring that all locks are acquired before any are released.

Two-Phase Locking (2PL) Protocol

As the name suggests, this protocol divides the execution of a transaction into two distinct phases:

- Growing Phase: In this phase, a transaction can obtain any number of locks (Shared or Exclusive) but cannot release any. The point where the transaction has acquired all its required locks is called the Lock Point.

- Shrinking Phase: Once the first lock is released, the transaction enters this phase. It can release existing locks but is strictly prohibited from acquiring any new ones.

Feature	Two-Phase Locking (2PL)	Pre-claiming Lock Protocol
Lock Acquisition	Locks are acquired one by one as they are needed during the growing phase.	All required locks must be requested and granted before the transaction starts.
Transaction Start	Starts immediately; locks are requested during execution.	Delayed until the system can grant every single required lock at once.

Feature	Two-Phase Locking (2PL)	Pre-claiming Lock Protocol
Deadlocks	High risk of deadlocks because two transactions can hold locks the other needs.	Zero risk of deadlocks because a transaction never holds a partial set of locks while waiting.
Resource Utility	Higher efficiency as locks are held only when necessary.	Lower efficiency as resources are locked even if they aren't used until the end of the transaction.

MODULE-5

Q9 a. Explain the Log-Based Recovery mechanism in database systems.

Log-based recovery is a fundamental mechanism that uses a sequence of records stored on stable storage to restore a database to a consistent state after a crash. This log file acts as a history of all modifications, ensuring that no committed data is lost and no partial changes remain.

The Mechanism of Log-Based Recovery

The system records every operation in a log entry before any actual data is modified on the disk. This is known as Write-Ahead Logging (WAL). Each log record typically contains:

- Transaction ID: To identify which transaction performed the change.
-
- Data Item: The specific record or block being modified.
- Old Value (Before Image): Used for Undo operations to revert changes.
- New Value (After Image): Used for Redo operations to re-apply changes.

The Two Recovery Operations

When a system restarts after a failure, the recovery manager scans the log to perform two critical actions:

1. **Redo (Forward Recovery):** The system re-performs all operations of transactions that were successfully committed before the crash but whose changes might not have reached the physical disk. This ensures Durability.

2. **Undo (Backward Recovery):** The system reverses all operations of transactions that were active (uncommitted) at the time of the crash. This ensures Atomicity by removing partial updates that could leave the database inconsistent.

Log Update Techniques

There are two primary ways the log-based recovery is managed during normal operation:

- Deferred Database Modification: Changes are only recorded in the log while the transaction is active. The actual database on the disk is updated only *after* the transaction commits. If a crash occurs mid-transaction, no "Undo" is needed because the disk was never touched; only a "Redo" is needed for committed items.

- Immediate Database Modification: Database updates can happen while the transaction is still active. Because the disk might contain uncommitted data, the recovery manager must be able to perform both Undo (to remove uncommitted changes) and Redo (to ensure committed changes stick) after a crash.

Checkpoints in Log Recovery
To prevent the recovery process from scanning the entire history of the database, the system uses Checkpoints.
At regular intervals, the DBMS flushes all modified data from volatile memory to the disk and marks a "Checkpoint" in the log. During recovery, the system only needs to process the log records that occurred after the most recent checkpoint, significantly reducing downtime.

Q9 b. Discuss how different log records help ensure atomicity and durability in case of system failures.
A Log Record is a small entry stored in a sequential file on a stable disk. It acts as a "diary" of every change a transaction makes before that change is actually written to the main database. Each record follows a specific structure: Transaction_ID, Data_Item, Old_Value, New_Value.

1) Transaction ID: The unique ID of the task.
2) Data Item: The specific record being modified.
3) Old Value: The data before the change (the "Before Image").
4) New Value: The data after the change (the "After Image").
5) Status Marks: Specific entries like <Ti, Start>, <Ti, Commit>, or <Ti, Abort>.

•Atomicity: The "All or Nothing" rule—either the entire transaction completes, or no part of it is saved.

- Durability: The "Permanent" rule—once a transaction is committed, its changes must survive even if the system crashes.

Ensuring Atomicity using Log Records:

Atomicity is maintained by the Undo process. If a system fails, the recovery manager scans the log for any transaction that has a Start record but no Commit record.

The system performs an Undo by:
1. Reading the log records of that failed transaction in reverse order.
2. Using the Old Value from the log to overwrite any partial changes on the disk.
3. This effectively "erases" the incomplete work, returning the database to its original state.

Ensuring Durability using Log Records:

Durability is maintained by the Redo process. The system ensures that if a user was told "Success," the data is actually there, even if the power cut out before the RAM could save it to the disk.

The system performs a Redo by:

1. Scanning the log for any transaction that has both a Start and a Commit record.
2. Reading the log records forward and using the New Value to write the data into the database.
3. This ensures that every committed change is physically present on the hard drive, fulfilling the durability promise.

OR

Q10 a. List the different types of checkpoints and explain the need for checkpoints in recovery mechanisms.

Core Definitions

- Atomicity: The rule that a transaction is an "all or nothing" unit; if it fails, every partial change is undone.
- Durability: The guarantee that once a transaction is committed, its changes are permanent and survive system failures.

The Need for Checkpoints in Recovery

In a standard log-based recovery, the system must scan the entire log from the very beginning of time to identify which transactions to Undo (for atomicity) and which to Redo (for durability).

The Need for Checkpoints arises from three main issues:

1. Recovery Time: Scanning a log that could be days or weeks old takes too long, keeping the database offline.
2. Log Size: Storing every single transaction ever performed consumes massive amounts of disk space.
3. Redundant Work: Most older transactions in the log have already been physically written to the disk, so re-applying them is a waste of processing power.

Checkpointing is the process where the DBMS periodically "flushes" all modified data from the volatile RAM to the permanent disk and marks a "safe point" in the log. During recovery, the system only needs to look at the log records that occurred after the last successful checkpoint.

- **Types of Checkpoints:**

Depending on how the database handles active transactions during the flush, there are three main types:

1. Sharp Checkpoint (Blocking)

In this type, the DBMS briefly stops all new transactions. It forces all modified data in the memory (dirty blocks) to be written to the disk and records the checkpoint entry.

- Pros: Very simple and ensures the disk is perfectly up to date.
- Cons: Causes a "freeze" in database performance, which is noticeable to users.

2. Fuzzy Checkpoint (Non-blocking)
The DBMS does not stop transactions. Instead, it records the start of a checkpoint and begins writing dirty blocks to the disk in the background while other transactions continue to run.

- Pros: High system availability; users don't notice any lag.
- Cons: The recovery process is slightly more complex because transactions might have changed data *while* the checkpoint was being recorded.

3. Indirect Checkpoint
Instead of waiting for a specific "checkpoint command," the system continuously flushes dirty pages to the disk at a steady rate based on a target recovery time (e.g., "ensure recovery takes less than 60 seconds").

- Pros: Smoother I/O performance and predictable recovery times.
- Cons: Requires constant monitoring of system resources.

Q10 b. Describe how checkpointing helps in reducing recovery time and explain the steps involved in implementing checkpoints.
A Checkpoint is a synchronization point between the database log and the physical database files. It is a snapshot in the transaction log that guarantees all modified data (dirty pages) and log records up to that specific time have been successfully saved to the permanent disk.

- Log Record: A sequential file on a stable disk containing the Old Value and New Value of every modified data item.
- Atomicity: The "all-or-nothing" property where failed transactions are rolled back using the Old Value.
- Durability: The guarantee that committed data is permanent, ensured by re-applying the New Value after a crash.

The Need for Checkpoints:

Without checkpoints, the system must scan the entire log from the beginning of time during a recovery. Checkpoints are necessary for the following reasons:

- Reduction in Recovery Time: It limits the log scan to only the records created after the last checkpoint, significantly speeding up system restart.
- Resource Efficiency: It prevents the system from performing redundant "Redo" operations for transactions that are already physically saved on the disk.
- Log Space Management: It allows the DBMS to safely delete or archive older log records that are no longer needed for recovery, saving disk space.
- Memory Management: It forces the clearing of "dirty pages" from the volatile RAM, ensuring the physical database stays updated.

Steps to Implement a Checkpoint:

The implementation of a checkpoint follows a strict protocol to maintain data integrity:

1. Suspend Execution: The DBMS temporarily pauses all active transactions to ensure the database state remains static during the process.
2. Flush Log Buffer: All log records currently in the RAM are written to the stable disk to satisfy the Write-Ahead Logging (WAL) rule.
3. Flush Data Buffer: All modified data blocks (dirty pages) in the main memory are physically written to the database files on the disk.
4. Create Checkpoint Record: A special record, <Checkpoint, L>, is written to the log. L is a list of all transactions that were active at the moment of the checkpoint.
5. Resume Execution: The system releases the pause, and transactions continue their normal operations.

Recovery Process Using Checkpoints

When a system failure occurs, the recovery manager performs the following steps:

- Step A: Locate the most recent <Checkpoint, L> record in the log.
- Step B: For every transaction in list L or those started after the checkpoint:
    - If the log contains a <Commit> record, the transaction is added to the Redo List.
    - If the log contains no <Commit> record, the transaction is added to the Undo List.
- Step C: Perform Redo for all committed transactions and Undo for all uncommitted ones to restore consistency.