

# Database Management (4331603) - Summer 2024 Solution

Milav Dabgar

June 12, 2024

## Question 1(a) [3 marks]

Define Following Terms: 1. Data 2. Information 3. Metadata

### Solution

**Table 1.** Data vs Information vs Metadata

Term	Definition	Example
<b>Data</b>	Raw facts and figures without context	"25", "John", "Mumbai"
<b>Information</b>	Processed data with meaning and context	"John is 25 years old and lives in Mumbai"
<b>Metadata</b>	Data about data describing structure and properties	"Age field: Integer, Max length: 3"

- **Data:** Basic building blocks of information systems
- **Information:** Result of data processing for decision making
- **Metadata:** Essential for database design and management

### Mnemonic

"DIM - Data gives Information using Metadata"

## Question 1(b) [4 marks]

Compare File System vs Database System

### Solution

**Table 2.** File System vs Database System Comparison

Aspect	File System	Database System
<b>Data Storage</b>	Separate files for each application	Centralized storage
<b>Data Redundancy</b>	High redundancy	Minimal redundancy
<b>Data Consistency</b>	Poor consistency	High consistency
<b>Data Security</b>	Limited security	Advanced security features
<b>Concurrent Access</b>	Limited support	Full concurrent support
<b>Data Independence</b>	No independence	Physical and logical independence

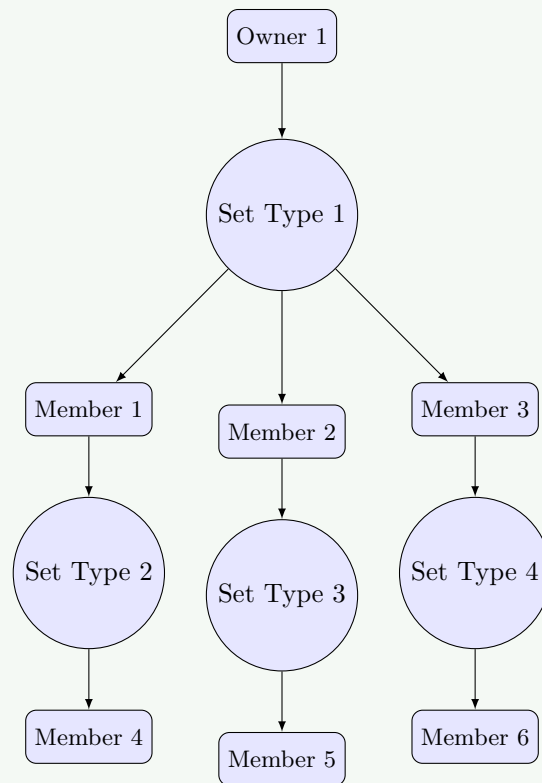
- **File System:** Simple but with data duplication issues
- **Database System:** Complex but efficient data management
- **Main Advantage:** DBMS eliminates data redundancy and inconsistency

**Mnemonic**

“DBMS = Data Better Managed Systematically”

**Question 1(c) [7 marks]**

Draw and Explain Network Data Model

**Solution**

**Figure 1.** Network Data Model Structure

**Table 3.** Network Model Components

Component	Description	Example
<b>Record Type</b>	Entity representation	Employee, Department
<b>Set Type</b>	Relationship between records	Works-In, Manages
<b>Owner</b>	Parent record in relationship	Department (owner)
<b>Member</b>	Child record in relationship	Employee (member)

- **Owner Record:** Controls the set and can have multiple members
- **Member Record:** Belongs to one or more sets
- **Set Occurrence:** Instance of set type linking owner to members
- **Navigation:** Uses pointers for record access

**Mnemonic**

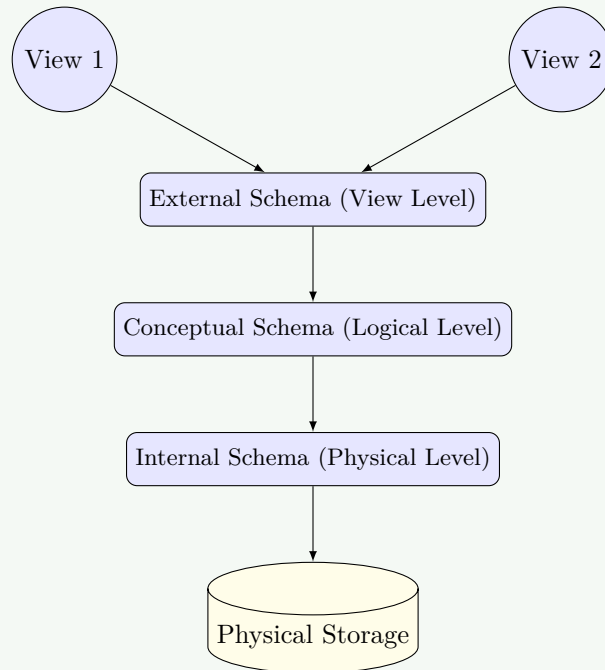
“Network = Nodes with Multiple Connections”

## Question 1(c) OR [7 marks]

What is Schema? Explain different types of Schema with example

### Solution

**Definition:** Schema is the logical structure or blueprint of a database that defines how data is organized.



**Figure 2.** Three-Schema Architecture

**Table 4.** Types of Schema

Schema Type	Level	Description	Example
External Schema	View Level	User-specific view of database	Student grades view for teachers
Conceptual Schema	Logical Level	Complete logical structure	All tables, relationships, constraints
Internal Schema	Physical Level	Physical storage structure	Index files, storage allocation

- **External Schema:** Provides data independence for users
- **Conceptual Schema:** Database designer's complete view
- **Internal Schema:** Database administrator's physical view

### Mnemonic

"ECI - External Conceptual Internal"

## Question 2(a) [3 marks]

Define Following Terms: 1. Entity 2. Attributes 3. Relationship

## Solution

Table 5. ER Model Basic Concepts

Term	Definition	Example
<b>Entity</b>	Real-world object with independent existence	Student, Course, Teacher
<b>Attributes</b>	Properties that describe an entity	Student: ID, Name, Age
<b>Relationship</b>	Association between two or more entities	Student ENROLLS IN Course

- **Entity:** Represented by rectangles in ER diagrams
- **Attributes:** Represented by ovals connected to entities
- **Relationship:** Represented by diamonds connecting entities

## Mnemonic

“EAR - Entity has Attributes and Relationships”

## Question 2(b) [4 marks]

Describe Weak Entity Sets with example

## Solution

**Definition:** Weak entity is an entity that cannot be uniquely identified by its own attributes and depends on a strong entity.

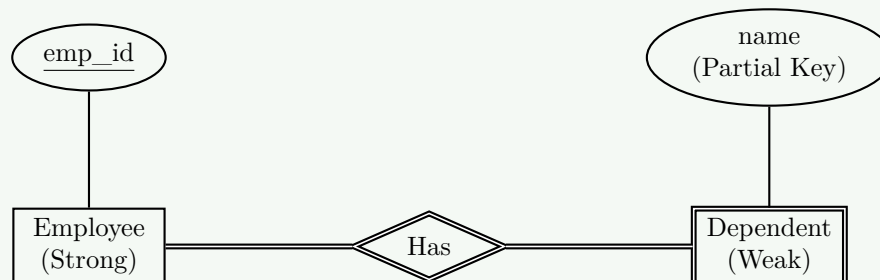


Figure 3. Strong vs Weak Entity

Table 6. Weak vs Strong Entity

Aspect	Strong Entity	Weak Entity
<b>Primary Key</b>	Has its own primary key	No primary key
<b>Existence</b>	Independent existence	Depends on strong entity
<b>Representation</b>	Single rectangle	Double rectangle
<b>Example</b>	Employee	Dependent of Employee

- **Partial Key:** Attribute that partially identifies weak entity
- **Identifying Relationship:** Connects weak entity to strong entity
- **Total Participation:** Weak entity must participate in relationship

## Mnemonic

“Weak entities are DEPENDent”

## Question 2(c) [7 marks]

Draw ER Diagram for University Management System

### Solution

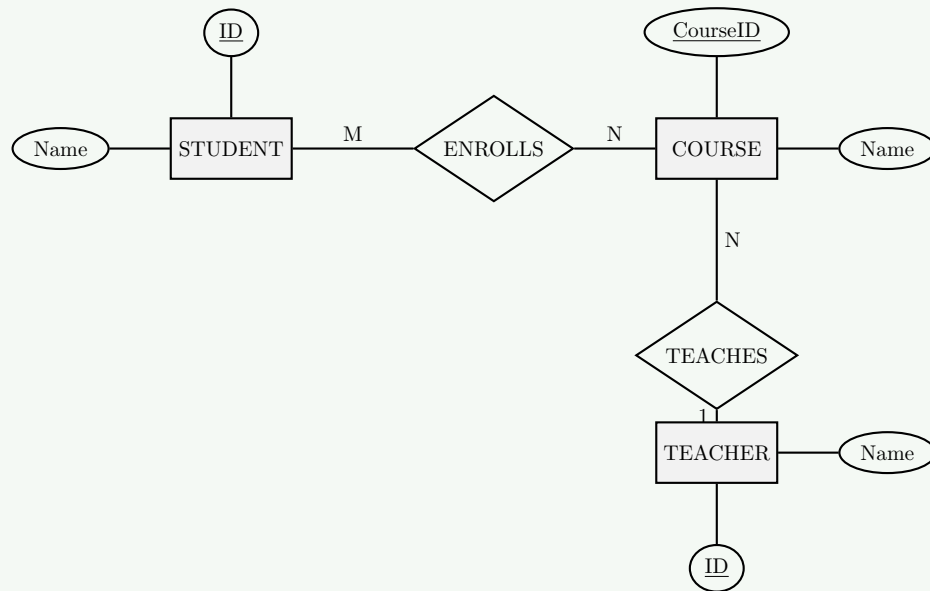


Figure 4. University ER Diagram

Table 7. Entity Relationships

Relationship	Cardinality	Description
Student ENROLLS Course	M:N	Many students can enroll in many courses
Teacher TEACHES Course	1:N	One teacher teaches multiple courses
Course HAS Enrollment	1:N	One course has multiple enrollments

- **Primary Entities:** Student, Course, Teacher
- **Associative Entity:** Enrollment (resolves M:N relationship)
- **Key Attributes:** All entities have unique identifier

### Mnemonic

“University = Students Take Courses from Teachers”

## Question 2(a) OR [3 marks]

Define Following Terms: 1. Primary Key 2. Foreign Key 3. Candidate Key

### Solution

Table 8. Database Keys

Key Type	Definition	Example
<b>Primary Key</b>	Unique identifier for each record	Student_ID in Student table
<b>Foreign Key</b>	References primary key of another table	Student_ID in Enrollment table
<b>Candidate Key</b>	Potential primary key attribute	Email, Phone in Student table

- **Primary Key:** Cannot be NULL and must be unique
- **Foreign Key:** Maintains referential integrity
- **Candidate Key:** Alternative unique identifiers

### Mnemonic

“PFC - Primary Foreign Candidate”

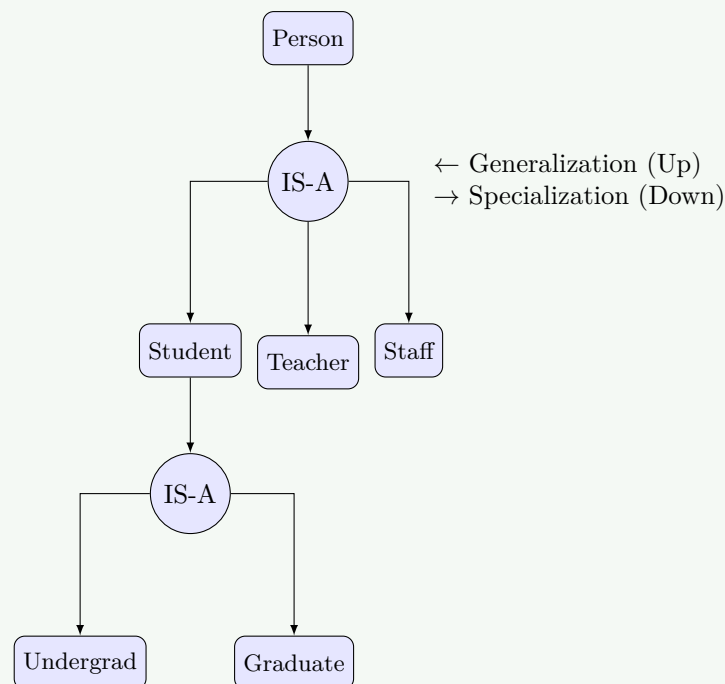
## Question 2(b) OR [4 marks]

Write a Short note on Generalization and Specialization

### Solution

**Generalization:** Process of extracting common attributes from multiple entities to create a general entity.

**Specialization:** Process of defining subclasses of an entity based on distinguishing characteristics.



**Figure 5.** Generalization and Specialization Hierarchy

**Table 9.** Generalization vs Specialization

Aspect	Generalization	Specialization
Direction	Bottom-up approach	Top-down approach
Purpose	Remove redundancy	Add specific attributes
Result	Superclass creation	Subclass creation

- **ISA Relationship:** “Is-A” relationship between superclass and subclass
- **Inheritance:** Subclasses inherit attributes from superclass

### Mnemonic

“General goes UP, Special goes DOWN”

## Question 2(c) OR [7 marks]

Explain different Relational Algebra operation with example

### Solution

Table 10. Relational Algebra Operations

Operation	Symbol	Description	Example
Select	$\sigma$	Selects rows based on condition	$\sigma_{age>20}(Student)$
Project	$\pi$	Selects specific columns	$\pi_{name,age}(Student)$
Union	$\cup$	Combines two relations	$R \cup S$
Intersection	$\cap$	Common tuples from relations	$R \cap S$
Difference	$-$	Tuples in R but not in S	$R - S$
Join	$\bowtie$	Combines related tuples	$Student \bowtie Enrollment$

**Example Relations:** Student: (ID=1, Name=John, Age=20)

Course: (CID=101, CName=DBMS, Credits=3)

- **Selection:**  $\sigma_{Age>18}(Student)$  returns students above 18
- **Projection:**  $\pi_{Name}(Student)$  returns only names
- **Join:**  $Student \bowtie Enrollment$  combines student and enrollment data

### Mnemonic

“SPUDIJ - Select Project Union Difference Intersection Join”

## Question 3(a) [3 marks]

List out Numeric Functions in SQL. Explain any Two

### Solution

Table 11. SQL Numeric Functions

Function	Purpose	Example
<b>ABS()</b>	Absolute value	$ABS(-15) = 15$
<b>CEIL()</b>	Smallest integer $\geq$ value	$CEIL(4.3) = 5$
<b>FLOOR()</b>	Largest integer $\leq$ value	$FLOOR(4.7) = 4$
<b>ROUND()</b>	Round to specified places	$ROUND(15.76, 1) = 15.8$
<b>SQRT()</b>	Square root	$SQRT(16) = 4$
<b>POWER()</b>	Raise to power	$POWER(2, 3) = 8$

**Detailed Examples:**

- **ABS(number):** Returns absolute value, removing negative sign
- **ROUND(number, decimal\_places):** Rounds number to specified decimal places

### Mnemonic

“Math functions make Numbers Nice”

## Question 3(b) [4 marks]

Describe Having and Order by Clause with example

### Solution

**HAVING Clause:** Used with GROUP BY to filter groups based on aggregate conditions.

**ORDER BY Clause:** Used to sort result set in ascending or descending order.

Table 12. HAVING vs WHERE

Aspect	WHERE	HAVING
Usage	Filters individual rows	Filters grouped results
With Aggregates	Cannot use	Can use aggregate functions
Position	Before GROUP BY	After GROUP BY

### Example:

```

1 SELECT department, COUNT(*) as emp_count
2 FROM employees
3 WHERE salary > 30000
4 GROUP BY department
5 HAVING COUNT(*) > 5
6 ORDER BY emp_count DESC;
```

- **WHERE:** Filters employees with salary > 30000
- **HAVING:** Shows only departments with more than 5 employees
- **ORDER BY:** Sorts by employee count in descending order

### Mnemonic

“WHERE filters rows, HAVING filters groups, ORDER BY sorts results”

## Question 3(c) [7 marks]

Perform the following Query on the table student having the fields Student\_ID, Stu\_Name, Stu\_Subject\_ID, Stu\_Marks, Stu\_Age in SQL

### Solution

#### 1. Create student table:

```

1 CREATE TABLE student (
2     Student_ID INT PRIMARY KEY,
3     Stu_Name VARCHAR(50),
4     Stu_Subject_ID INT,
5     Stu_Marks INT,
6     Stu_Age INT
7 );
```

#### 2. Insert record in student table:

```

1 INSERT INTO student VALUES
2 (1, "John", 101, 85, 22),
3 (2, "Mary", 102, 90, 21);
```

#### 3. Find minimum and maximum marks:

```

1 SELECT MIN(Stu_Marks) as Min_Marks,
```



```

2      MAX(Stu_Marks) as Max_Marks
3 FROM student;

```

4. Students with marks > 82 and age = 22:

```

1 SELECT * FROM student
2 WHERE Stu_Marks > 82 AND Stu_Age = 22;

```

5. Students whose name begins with "m":

```

1 SELECT * FROM student
2 WHERE Stu_Name LIKE "m%";

```

6. Find average marks:

```

1 SELECT AVG(Stu_Marks) as Average_Marks
2 FROM student;

```

7. Add Stu\_address column:

```

1 ALTER TABLE student
2 ADD Stu_address VARCHAR(100);

```

#### Mnemonic

“CRUD + Analytics = Complete Database Operations”

## Question 3(a) OR [3 marks]

Describe different date function in SQL with example

#### Solution

Table 13. SQL Date Functions

Function	Purpose	Example
<b>SYSDATE</b>	Current system date	SYSDATE returns '2024-06-12'
<b>ADD_MONTHS()</b>	Add months to date	ADD_MONTHS('2024-01-15', 3)
<b>MONTHS_BETWEEN()</b>	Months between dates	MONTHS_BETWEEN('2024-06-12', '2024-01-12')
<b>LAST_DAY()</b>	Last day of month	LAST_DAY('2024-02-15') = '2024-02-29'
<b>NEXT_DAY()</b>	Next occurrence of day	NEXT_DAY('2024-06-12', 'FRIDAY')

#### Examples:

- **SYSDATE**: Returns current system date and time
- **ADD\_MONTHS**: Useful for calculating future dates like loan due dates

#### Mnemonic

“Date functions help with Time Management”

## Question 3(b) OR [4 marks]

List out Constraints in SQL. Explain any two with example

## Solution

Table 14. SQL Constraints

Constraint	Purpose	Example
<b>PRIMARY KEY</b>	Unique identifier	Student_ID INT PRIMARY KEY
<b>FOREIGN KEY</b>	References another table	REFERENCES Student(Student_ID)
<b>NOT NULL</b>	Prevents null values	Name VARCHAR(50) NOT NULL
<b>UNIQUE</b>	Ensures uniqueness	Email VARCHAR(100) UNIQUE
<b>CHECK</b>	Validates data	Age INT CHECK (Age >= 18)
<b>DEFAULT</b>	Default value	Status VARCHAR(10) DEFAULT 'Active'

## Detailed Examples:

**PRIMARY KEY Constraint:**

```

1 CREATE TABLE Student (
2     Student_ID INT PRIMARY KEY,
3     Name VARCHAR(50)
4 );

```

**CHECK Constraint:**

```

1 CREATE TABLE Employee (
2     Emp_ID INT,
3     Salary INT CHECK (Salary > 0)
4 );

```

- **PRIMARY KEY:** Ensures each record has unique identifier
- **CHECK:** Validates business rules during data entry

## Mnemonic

“Constraints Control Data Quality”

## Question 3(c) OR [7 marks]

Explain different types of joins with example in SQL

## Solution

Table 15. Types of SQL Joins

Join Type	Description	Syntax
<b>INNER JOIN</b>	Returns matching records from both tables	Table1 INNER JOIN Table2 ON condition
<b>LEFT JOIN</b>	All records from left table + matching from right	Table1 LEFT JOIN Table2 ON condition
<b>RIGHT JOIN</b>	All records from right table + matching from left	Table1 RIGHT JOIN Table2 ON condition
<b>FULL OUTER JOIN</b>	All records from both tables	Table1 FULL OUTER JOIN Table2 ON condition

**Example Tables:** Students: (ID=1, Name=John), (ID=2, Name=Mary)  
 Enrollments: (StudentID=1, Course=DBMS), (StudentID=3, Course=Java)  
**INNER JOIN Example:**

```

1 SELECT s.Name, e.Course
2 FROM Students s
3 INNER JOIN Enrollments e ON s.ID = e.StudentID;

```

Result: Only John with DBMS course

**LEFT JOIN Example:**

```

1 SELECT s.Name, e.Course
2 FROM Students s
3 LEFT JOIN Enrollments e ON s.ID = e.StudentID;

```

Result: John-DBMS, Mary-NULL

### Mnemonic

“JOIN connects Related Tables”

## Question 4(a) [3 marks]

Give an example of Grant and Revoke command in SQL

### Solution

**GRANT Command:** Provides specific privileges to users on database objects.

**REVOKE Command:** Removes previously granted privileges from users.

Table 16. Common Privileges

Privilege	Description	Example
<b>SELECT</b>	Read data	GRANT SELECT ON Student TO user1
<b>INSERT</b>	Add new records	GRANT INSERT ON Student TO user1
<b>UPDATE</b>	Modify existing records	GRANT UPDATE ON Student TO user1
<b>DELETE</b>	Remove records	GRANT DELETE ON Student TO user1
<b>ALL</b>	All privileges	GRANT ALL ON Student TO user1

**Examples:**

```

1 -- Grant SELECT privilege
2 GRANT SELECT ON Student TO john;
3
4 -- Revoke INSERT privilege
5 REVOKE INSERT ON Student FROM john;

```

- **WITH GRANT OPTION:** Allows user to grant privileges to others
- **CASCADE:** Revokes privileges from all users who received them

### Mnemonic

“GRANT gives rights, REVOKE removes rights”

## Question 4(b) [4 marks]

Write a short note on SQL Views

### Solution

**Definition:** A view is a virtual table based on the result of an SQL statement containing rows and columns like a real table.

**Table 17.** View Characteristics

Aspect	Description	Example
<b>Virtual Table</b>	Does not store data physically	CREATE VIEW student_view AS...
<b>Security</b>	Hides sensitive columns	Hide salary column from employees
<b>Simplification</b>	Simplifies complex queries	Join multiple tables in single view
<b>Data Independence</b>	Changes in base tables don't affect users	Modify table structure without affecting applications

**Example:**

```

1 CREATE VIEW active_students AS
2 SELECT Student_ID, Name, Age
3 FROM Student
4 WHERE Status = 'Active';
5
6 -- Using the view
7 SELECT * FROM active_students;
```

- **Security:** Restrict access to sensitive data
- **Simplicity:** Hide complex joins from end users
- **Consistency:** Standardized data access

### Mnemonic

“Views are Virtual Windows to Data”

## Question 4(c) [7 marks]

What is Normalization? Explain 2NF with example

### Solution

**Normalization:** Process of organizing database to reduce redundancy and improve data integrity by dividing large tables into smaller related tables.

**2NF (Second Normal Form):**

- Must be in 1NF
- Remove partial functional dependencies
- Non-key attributes must depend on entire primary key

**Example - Unnormalized Table:**

Student_ID	Course_ID	Student_Name	Course_Name	Instructor
101	C1	John	DBMS	Dr. Smith
101	C2	John	Java	Dr. Jones
102	C1	Mary	DBMS	Dr. Smith

**Problems:**

- Student\_Name depends only on Student\_ID (partial dependency)
- Course\_Name and Instructor depend only on Course\_ID

**After 2NF:**

**Student Table:**

Student_ID	Student_Name
101	John
102	Mary

Course Table:

Course_ID	Course_Name	Instructor
C1	DBMS	Dr. Smith
C2	Java	Dr. Jones

Enrollment Table:

Student_ID	Course_ID
101	C1
101	C2
102	C1

- **Eliminates Redundancy:** Student names not repeated
- **Reduces Storage:** Less duplicate data
- **Improves Consistency:** Update student name in one place

### Mnemonic

“2NF = No Partial Dependencies”

## Question 4(a) OR [3 marks]

Give an example of Group By Clause in SQL

### Solution

**GROUP BY Clause:** Groups rows with same values in specified columns and allows aggregate functions on each group.

Table 18. GROUP BY Usage

Purpose	Function	Example
Counting	COUNT()	Count students per department
Summing	SUM()	Total salary per department
Averaging	AVG()	Average marks per course
Finding Min/Max	MIN()/MAX()	Highest salary per department

Example:

```
1 SELECT Department, COUNT(*) as Total_Students, AVG(Marks) as Avg_Marks
2 FROM Student
3 GROUP BY Department;
```

Result:

Department	Total_Students	Avg_Marks
IT	25	78.5
CS	30	82.1

- **Groups:** Creates separate groups for each department
- **Aggregates:** Calculates count and average for each group

**Mnemonic**

“GROUP BY creates Summary Reports”

**Question 4(b) OR [4 marks]**

Describe Set Operators in SQL with example

**Solution**

**Set Operators:** Combine results from two or more SELECT statements.

**Table 19.** SQL Set Operators

Operator	Description	Requirement	Example
UNION	Combines results, removes duplicates	Same column structure	SELECT name FROM students UNION SELECT name FROM teachers
UNION ALL	Combines results, keeps duplicates	Same column structure	SELECT name FROM students UNION ALL SELECT name FROM alumni
INTERSECT	Returns common records	Same column structure	SELECT course FROM current_courses INTERSECT SELECT course FROM popular_courses
MINUS	Records in first query but not second	Same column structure	SELECT student_id FROM enrolled MINUS SELECT student_id FROM graduated

**Example:**

```

1  -- Students who are also teachers
2  SELECT name FROM students
3  INTERSECT
4  SELECT name FROM teachers;
5
6  -- All people in university
7  SELECT name, 'Student' as type FROM students
8  UNION
9  SELECT name, 'Teacher' as type FROM teachers;
```

- **Column Count:** Must be same in all queries
- **Data Types:** Corresponding columns must have compatible types
- **Order:** ORDER BY can only be used at the end

**Mnemonic**

“Set operators Unite, Intersect, and Subtract data”

**Question 4(c) OR [7 marks]**

Justify the importance of Normalization. Explain 1NF with example

**Solution**

**Importance of Normalization:**

**Table 20.** Benefits of Normalization

Benefit	Description	Impact
<b>Eliminates Redundancy</b>	Reduces duplicate data storage	Saves storage space
<b>Prevents Anomalies</b>	Avoids insertion, deletion, update problems	Maintains data consistency
<b>Improves Integrity</b>	Ensures data accuracy	Reliable information system
<b>Flexible Design</b>	Easy to modify and extend	Adaptable to business changes

**1NF (First Normal Form):**

- Eliminate duplicate columns from same table
- Create separate tables for related data
- Each cell contains single value (atomic values)

**Example - Unnormalized Table:**

Student_ID	Name	Subjects
101	John	Math, Science, English
102	Mary	Science, History

**Problems:**

- Subjects column contains multiple values
- Difficult to query specific subjects
- Update anomalies when adding/removing subjects

**After 1NF:****Student Table:**

Student_ID	Name
101	John
102	Mary

**Student\_Subject Table:**

Student_ID	Subject
101	Math
101	Science
101	English
102	Science
102	History

- **Atomic Values:** Each cell contains single value
- **Flexible Queries:** Easy to find students studying specific subjects
- **Easy Updates:** Add/remove subjects without affecting other data

**Mnemonic**

“1NF = One value per cell, No repeating groups”

**Question 5(a) [3 marks]****Explain Serializability in Transaction Management****Solution**

**Serializability:** Property that ensures concurrent execution of transactions produces same result as some serial execution of those transactions.

**Table 21.** Types of Serializability

Type	Description	Method
<b>Conflict Serializability</b>	Based on conflicting operations	Precedence graph
<b>View Serializability</b>	Based on read-write patterns	View equivalence

**Example:** Transaction T1: R(A), W(A), R(B), W(B)

Transaction T2: R(A), W(A), R(B), W(B)

**Serial Schedule:** T1 → T2 or T2 → T1

**Concurrent Schedule:** Interleaved operations

- **Conflict Operations:** Operations on same data item where at least one is write
- **Serializable Schedule:** Equivalent to some serial schedule
- **Non-serializable:** May lead to inconsistent database state

#### Mnemonic

“Serializability ensures Transaction Consistency”

## Question 5(b) [4 marks]

Describe Partial Functional Dependency with example

#### Solution

**Partial Functional Dependency:** When a non-key attribute is functionally dependent on only part of a composite primary key.

**Table 22.** Functional Dependency Types

Type	Definition	Example
<b>Full Dependency</b>	Depends on entire primary key	(Student_ID, Course_ID) → Grade
<b>Partial Dependency</b>	Depends on part of primary key	(Student_ID, Course_ID) → Student_Name

**Example:**

**Enrollment Table:**

Primary Key: (Student\_ID, Course\_ID)

Student_ID	Course_ID	Student_Name	Course_Name	Grade
101	C1	John	DBMS	A
101	C2	John	Java	B

**Partial Dependencies:**

- Student\_ID → Student\_Name (Student\_Name depends only on Student\_ID)
- Course\_ID → Course\_Name (Course\_Name depends only on Course\_ID)

**Problems:**

- **Update Anomaly:** Changing student name requires multiple updates
- **Insertion Anomaly:** Cannot add student without enrolling in course
- **Deletion Anomaly:** Deleting enrollment may lose student information

**Solution:** Normalize to 2NF by removing partial dependencies

#### Mnemonic

“Partial dependency = Part of key determines attribute”



## Question 5(c) [7 marks]

Write a Short note on Locking Mechanism with example in Transaction Management

### Solution

**Locking Mechanism:** Concurrency control technique that prevents simultaneous access to data items during transaction execution.

**Table 23.** Types of Locks

Lock Type	Description	Usage
<b>Shared Lock (S)</b>	Multiple transactions can read	Read operations
<b>Exclusive Lock (X)</b>	Only one transaction can access	Write operations
<b>Intention Lock</b>	Indicates intent to lock at lower level	Hierarchical locking

### Two-Phase Locking (2PL) Protocol:

- **Growing Phase:** Acquire locks, cannot release any lock
- **Shrinking Phase:** Release locks, cannot acquire new locks

### Example:

```

1 Transaction T1: Read(A), Write(A), Read(B), Write(B)
2 Transaction T2: Read(A), Write(A), Read(C), Write(C)
3
4 T1: S-lock(A), Read(A), X-lock(A), Write(A), S-lock(B),
5     Read(B), X-lock(B), Write(B), Unlock(A), Unlock(B)
6
7 T2: Wait for A, S-lock(A), Read(A), X-lock(A), Write(A),
8     S-lock(C), Read(C), X-lock(C), Write(C), Unlock(A), Unlock(C)

```

### Lock Compatibility Matrix:

Current/Requested	S	X
S	✓	✗
X	✗	✗

- **Deadlock:** Two transactions waiting for each other's locks
- **Starvation:** Transaction waits indefinitely for lock

### Solutions:

- **Deadlock Detection:** Use wait-for graph
- **Deadlock Prevention:** Timestamp-based protocols

### Mnemonic

"Locking prevents Concurrent Conflicts"

## Question 5(a) OR [3 marks]

Explain Deadlock in Transaction Management

### Solution

**Deadlock:** Situation where two or more transactions are waiting indefinitely for each other to release locks, creating a circular wait condition.

**Table 24.** Deadlock Components

Component	Description	Example
<b>Mutual Exclusion</b>	Resources cannot be shared	Exclusive locks
<b>Hold and Wait</b>	Process holds resources while waiting	T1 holds A, waits for B
<b>No Preemption</b>	Resources cannot be forcibly taken	Locks cannot be revoked
<b>Circular Wait</b>	Circular chain of waiting processes	T1→T2→T1

**Example:**

```

1 Transaction T1: Lock(A), Lock(B)
2 Transaction T2: Lock(B), Lock(A)
3
4 Time 1: T1 gets Lock(A)
5 Time 2: T2 gets Lock(B)
6 Time 3: T1 waits for Lock(B) - held by T2
7 Time 4: T2 waits for Lock(A) - held by T1
8 Result: DEADLOCK!

```

- **Detection:** Use wait-for graph to identify cycles
- **Prevention:** Use timestamp ordering or wound-wait protocols

**Mnemonic**

“Deadlock = Circular Waiting for Resources”

**Question 5(b) OR [4 marks]**

Describe Full Functional Dependency with example

**Solution**

**Full Functional Dependency:** A non-key attribute is functionally dependent on the entire primary key (not just part of it).

**Table 25.** Dependency Comparison

Type	Definition	Example
<b>Full Dependency</b>	Depends on complete primary key	(Student_ID, Course_ID) → Grade
<b>Partial Dependency</b>	Depends on part of primary key	(Student_ID, Course_ID) → Student_Name

**Example:****Enrollment Table:**

Primary Key: (Student\_ID, Course\_ID)

Student_ID	Course_ID	Grade	Hours
101	C1	A	4
101	C2	B	3
102	C1	B	4

**Full Functional Dependencies:**

- (Student\_ID, Course\_ID) → Grade ✓
- (Student\_ID, Course\_ID) → Hours ✓

**Explanation:**

- **Grade** depends on both Student\_ID AND Course\_ID (specific student in specific course)
- **Hours** also depends on both (student's hours in specific course)
- Cannot determine Grade from Student\_ID alone

- Cannot determine Grade from Course\_ID alone

**Benefits:**

- **No Update Anomalies:** Changes affect only relevant records
- **Proper Normalization:** Supports 2NF requirements
- **Data Integrity:** Ensures accurate relationships

**Mnemonic**

“Full dependency needs Complete Key”

**Question 5(c) OR [7 marks]****Explain ACID Properties of Transaction with example****Solution**

**ACID Properties:** Four fundamental properties that guarantee database transaction reliability.

**Table 26.** ACID Properties

Property	Description	Example
<b>Atomicity</b>	All or nothing execution	Bank transfer: both debit and credit must happen
<b>Consistency</b>	Database remains in valid state	Account balance cannot be negative
<b>Isolation</b>	Transactions don't interfere	Concurrent transactions appear sequential
<b>Durability</b>	Committed changes are permanent	Data survives system crashes

**Detailed Examples:****Atomicity Example:**

```

1 BEGIN TRANSACTION;
2 UPDATE Account SET Balance = Balance - 1000 WHERE AccNo = 'A001';
3 UPDATE Account SET Balance = Balance + 1000 WHERE AccNo = 'A002';
4 COMMIT;

```

*If either update fails, entire transaction is rolled back*

**Consistency Example:**

```

1 -- Before: A001 = 5000, A002 = 3000, Total = 8000
2 -- Transfer 1000 from A001 to A002
3 -- After: A001 = 4000, A002 = 4000, Total = 8000
4 -- Total money in system remains constant

```

**Isolation Example:**

```

1 T1: Read(A=100), A=A+50, Write(A=150)
2 T2: Read(A=100), A=A*2, Write(A=200)
3 Serial Result: A=300 or A=250
4 Isolated execution must produce one of these results

```

**Durability Example:**

```

1 After COMMIT is executed, even if system crashes,
2 the transferred amount remains in destination account

```

**Implementation:**

- **Atomicity:** Using transaction logs and rollback
- **Consistency:** Using constraints and triggers
- **Isolation:** Using locking mechanisms
- **Durability:** Using write-ahead logging

**Mnemonic**

“ACID keeps Transactions Reliable”