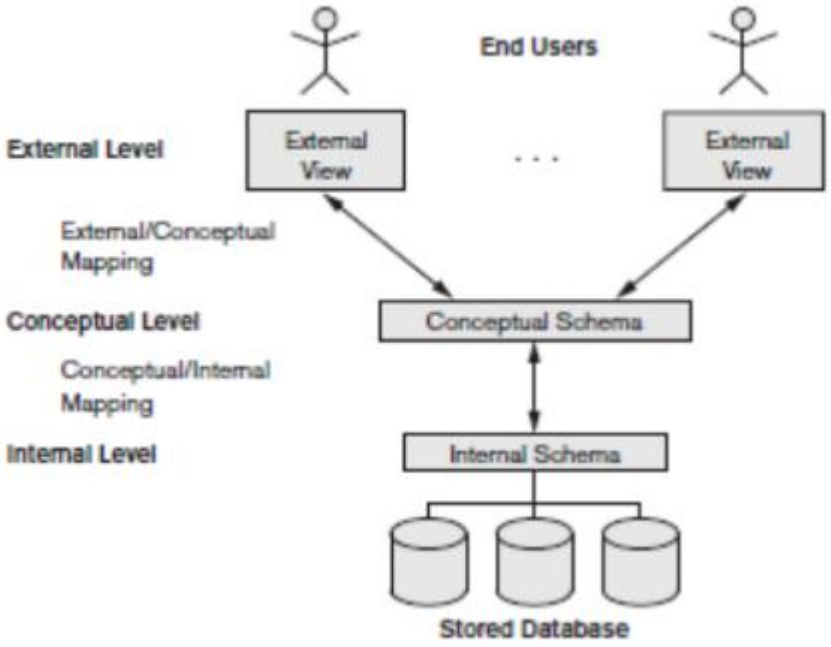
USN | | | | | | | | | | |

Internal Assessment Test 1 – March 2025

Sub:	**Database Management System**					Sub Code:	BCS403	Branch:	AIML	
Date:	**27/3/2025**	Duration:	90 min	Max Marks:	50	Sem/Sec:	IV /A, B & C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Describe Main characteristics of DBMS approach and how it is different from the traditional file system. 1. Self-Description: A database system includes —in addition to the data stored that is of relevance to the organization— a complete definition/description of the database's structure and constraints. This meta-data (i.e., data about data) is stored in the so-called system catalog, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy). The system catalog is used not only by users (e.g., who need to know the names of tables and attributes, and sometimes data type information and other things), but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure. 2. Insulation between Programs and Data; Data Abstraction: Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. (E.g., Consider a file descriptor in a COBOL program: it gives a detailed description of the layout of the records in a file by describing, for each field, how many bytes it occupies.) In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described (in the system catalog) separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data 3. Data Abstraction: A data model is used to hide storage details and present the users with a conceptual view of the database. Programs refer to the data model constructs rather than data storage details Example by which to illustrate this concept: Suppose that you are given the task of	10	CO1	L1

	developing a program that displays the contents of a particular data file. Specifically, each record should be displayed as follows: Record #i: value of first field value of second field ... ... value of last field To keep things very simple, suppose that the file in question has fixed-length records of 57 bytes with six fixed-length fields of lengths 12, 4, 17, 2, 15, and 7 bytes, respectively, all of which are ASCII strings. Developing such a program would not be difficult. However, the obvious solution would be tailored specifically for a file having the particular structure described here and would be of no use for a file with a different structure. 4. Multiple Views of Data: Different users (e.g., in different departments of an organization) have different "views" or perspectives on the database. For example, from the point of view of a Bursar's Office employee, student data does not include anything about which courses were taken or which grades were earned. (This is an example of a subset view.) A good DBMS has facilities for defining multiple views. This is not only convenient for users, but also addresses security issues of data access. 5. Data Sharing and Multi-user Transaction Processing: As you learned about (or will) in the OS course, the simultaneous access of computer resources by multiple users/processes is a major source of complexity. The same is true for multi-user DBMS's. Arising from this is the need for concurrency control, which is supposed to ensure that several users trying to update the same data do so in a "controlled" manner so that the results of the updates are as though they were done in some sequential order This gives rise to the concept of a transaction, which is a process that makes one or more accesses to a database and which must have the appearance of executing in isolation from all other transactions (even ones that access the same data at the "same time") and of being atomic (in the sense that, if the system crashes in the middle of its execution, the database contents must be as though it did not execute at all). Applications such as airline reservation systems are known as online transaction processing applications.			
2 a.	Explain the three-schema architecture with neat diagram. Why do we need mapping among the schema levels?	6	CO1	L2



The goal of the three-schema architecture, illustrated in the above Figure , is to separate the user
applications and the physical database. In this architecture, schemas can be defined at the following
three levels:
1. The internal level has an internal schema, which describes the physical storage structure of the
database. The internal schema uses a physical data model and describes the complete details of data
storage and access paths for the database.
2. 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

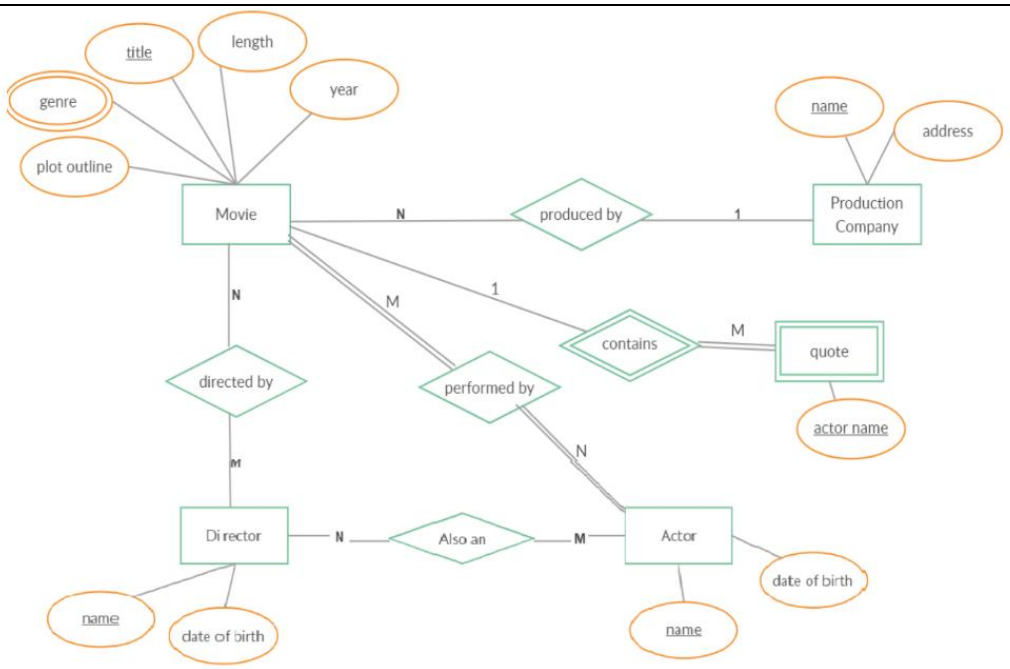
concentrates on describing entities, data types, relationships, user operations, and constraints. A high-
level data model or an implementation data model can be used at this level.

3. The external or view level includes a number of external schemas or user views. Each external
schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model
can be used at this level
The three-schema architecture can be used to explain the concept of data independence, which can be
defined as the capacity to change the schema at one level of a database system without having to
change the schema at the next higher level. We can define two types of data independence:
1. Logical data independence is the capacity to change the conceptual schema without having to
change external schemas or application programs. We may change the conceptual schema to expand
the database (by adding a record type or data item), or to reduce the database (by

	removing a record type or data item). In the latter case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence. Application programs that reference the external schema constructs must work as before, after the conceptual schema undergoes a logical reorganization. Changes to constraints can be applied also to the conceptual schema without affecting the external schemas or application programs. 2. Physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.			
2 b.	Explain data models and its types with the help of examples. **Data Models and Their Types (With Examples)** A **data model** defines how data is organized, stored, and manipulated within a database system. It provides a **structured representation** of the data and relationships between different elements. **Types of Data Models:** **1. Hierarchical Data Model** • **Structure:** Data is organized in a tree-like structure with **parent-child relationships**. • **Example:** o Consider an **organization structure** where the CEO is at the top, followed by managers, then employees. o **Example Representation:** css CopyEdit CEO ├── Manager 1 │ ├── Employee A │ ├── Employee B ├── Manager 2 ├── Employee C ├── Employee D **2. Network Data Model** • **Structure:** Similar to the hierarchical model but allows a **many-to-many** relationship using pointers. • **Example:** o **University Database:** A student can enroll in multiple courses, and a course can have multiple students. o **Representation:** css CopyEdit Student A <--> Course X Student A <--> Course Y	4	CO1	L2

	Student B <--> Course X
	3. Relational Data Model
	• **Structure:** Data is stored in **tables (relations)** with **rows (records) and columns (fields)**.
	• **Example:**
	○ **Customer Database:**
	Customer_ID | **Name** | **Email**
	101 | Alice | alice@email.com
	102 | Bob | bob@email.com
	○ SQL (Structured Query Language) is used to manage relational databases.
	4. Object-Oriented Data Model
	• **Structure:** Data is stored as **objects**, similar to object-oriented programming (OOP) concepts.
	• **Example:**
	○ **Online Shopping System:** A "Product" can have multiple attributes like name, price, and category.
	arduino
	CopyEdit
	class Product {
	String name;
	double price;
	String category;
	}
	5. Entity-Relationship (E-R) Model
	• **Structure:** Uses **entities, attributes, and relationships** to design the database structure.
	• **Example:**
	○ **Hospital Management System:**
	▪ **Entities:** Patient, Doctor, Appointment
	▪ **Relationships:** A patient can have multiple appointments, and each appointment is assigned to a doctor.			

3 a.	Construct an Entity-Relationship (E-R) diagram for a Movie Database considering the following entities, attributes, and relationships. Represent the entities, attributes, relationships, and cardinalities accurately using standard notations. **Entities and Attributes:** **Movie** (Movie_ID, Title, Release_Year, Genre, Language, Duration) **Production House** (Production_ID, Name, Established_Year, Country) **Director** (Director_ID, Name, Date_of_Birth, Nationality) **Actor** (Actor_ID, Name, Date_of_Birth, Gender, Nationality) **Role** (Role_ID, Character_Name, Actor_ID, Movie_ID) **Award** (Award_ID, Name, Category, Year, Winner_ID)	6	CO2	L3



3 b.	For the Movie database given in question number 3a, Write relational algebra queries for the following. 1. Retrieve all **movies** that belong to the **"Action"** genre. 2. Retrieve the **names** of all **production houses**. 3. Retrieve the **title and release year** of all movies that were released in **2020**. **1. Retrieve all movies that belong to the "Action" genre.** **Relational Algebra Query:** σGenre=′Action′(Movie)\sigma_{Genre = 'Action'}(Movie)σGenre=′Action′(Movie) **Explanation:** • σ (Selection) is used to filter movies where the Genre is "Action". • This will return all **movies that belong to the Action genre**.	4	CO2	L3
	2. Retrieve the names of all production houses. **Relational Algebra Query:** πName(Production_House)\pi_{Name}(Production_House)πName(Production_House) **Explanation:** • π (Projection) is used to select only the Name attribute from the Production_House table. • This will return **all unique production house names**.			
	3. Retrieve the title and release year of all movies that were released in 2020. **Relational Algebra Query:** πTitle,Release_Year(σRelease_Year=2020(Movie))\pi_{Title, Release_Year}(\sigma_{Release_Year = 2020}(Movie))πTitle,Release_Year(σRelease_Year=2020(Movie)) **Explanation:** • σ_{Release_Year = 2020}(Movie): Selects movies where Release_Year is 2020. • π_{Title, Release_Year}: Extracts only the Title and Release_Year attributes from the filtered movies.			
4.	Explain the following. i) Database Schema and Database State	10	CO2	L2

ii) Participation Constraints
iii) Recursive Relationships and Role names.
iv) Cardinality Ratio
v) Primary Key
vi) Candidate Key
vii) Foreign Key

i) Database Schema and Database State

- **Database Schema**:
  - The *logical structure/design* of the database.
  - Defines tables, attributes, data types, constraints, relationships, etc.
  - **Example**:

sql
Copy
```
CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(50),
   Dept VARCHAR(20)
);
```

  - Schema remains *constant* unless altered.
- **Database State (Instance)**:
  - The *actual data* stored in the database at a given time.
  - Changes with insertions, updates, and deletions.
  - **Example**:

EmpID	**Name**	**Dept**
101	Alice	HR
102	Bob	IT

ii) Participation Constraints

- Specifies whether an entity's participation in a relationship is **mandatory (total)** or **optional (partial)**.
- **Total Participation (Double Line in ER Diagram)**:
  - Every entity must participate in the relationship.
  - **Example**: Every Student *must* enroll in at least one Course.
- **Partial Participation (Single Line)**:
  - Some entities may not participate.
  - **Example**: Not every Employee manages a project.

iii) Recursive Relationships and Role Names

- **Recursive Relationship**:
  - An entity relates *to itself* in a relationship.
  - **Example**: An Employee supervises other Employees.

mermaid
Copy
```
erDiagram
   EMPLOYEE ||--o{ EMPLOYEE : "supervises"
```

- **Role Name**:
  - Clarifies the purpose of an entity in a relationship.
  - **Example**: In "supervises", roles are **Supervisor** (one side) and **Subordinate** (many side).

iv) Cardinality Ratio

- Defines the *numeric relationship* between entities in a relationship:
  - **1:1 (One-to-One)**:
    - **Example**: One Employee has one CompanyCar.

- **1:N (One-to-Many)**:
  - **Example**: One Department has many Employees.
- **M:N (Many-to-Many)**:
  - **Example**: Students enroll in many Courses, and Courses have many Students.

v) Primary Key

- A *minimal set of attributes* that **uniquely identifies** a row in a table.
- **Properties**:
  - Unique
  - Not NULL
  - Immutable (should not change)
- **Example**:

sql
Copy

```
CREATE TABLE Student (
   StudentID INT PRIMARY KEY,  -- Primary Key
   Name VARCHAR(50)
);
```

vi) Candidate Key

- A *super key* (set of attributes that uniquely identifies a row) *without unnecessary attributes*.
- A table can have multiple candidate keys, but only one becomes the primary key.
- **Example**:
  - In Employee(EmpID, Email, SSN), both EmpID and SSN are candidate keys.

vii) Foreign Key

- An attribute that *references the primary key* of another table to enforce referential integrity.
- **Example**:

sql
Copy

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   ProductID INT,
   CustomerID INT,
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID),
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

- Ensures that ProductID in Orders must exist in Products.

Summary Table

Concept	Definition	Example
Database Schema	Blueprint/structure of the database.	CREATE TABLE E
Database State	Actual data stored at a given time.	Rows in the Employ
Participation	Whether an entity must participate in a relationship (total/partial).	All students must en
Recursive Relationship	An entity relates to itself.	Employee supervise
Cardinality	Numerical relationship between entities (1:1, 1:N, M:N).	One department has employees.
Primary Key	Uniquely identifies a row (unique, NOT NULL).	StudentID in Studen

	Candidate Key	A possible primary key (minimal super key).	SSN or Email in Employee.		
	Foreign Key	References a primary key in another table.	CustomerID in Orders.		
5	Explain the select, project, union, intersection, set difference, Cartesian product and join operations in relational algebra with suitable example. Selection Operator(σ) Selection and Projection are unary operators. The selection operator is sigma: σ The selection operation acts like a filter on a relation by returning only a certain number of tuples. σC(R) Returns only those tuples in R that satisfy condition C A condition C can be made up of any combination of comparison or logical operators that operate on the attributes of R. Comparison operators: >,<,<=,>=, ≠,== Logical operators ┐ - not, v - or Example Select only those Employees in the CS department: σ Dept= 'CS'(EMP) Projection(π) Projection is also a Unary operator. The Projection operator is pi: π Projection limits the attributes that will be returned from the original relation. The general syntax is: π attributes R Where attributes is the list of attributes to be displayed and R is the relation. The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced). The degree of the resulting relation may be equal to or less than that of the original relation Project only the names and departments of the employees: πname, dept(EMP) THETA JOIN: Similar to a CARTESIAN PRODUCT followed by a SELECT. The condition c is called a join condition. R(A1, A2, ..., Am, B1, B2, ..., Bn) R1(A1, A2, ..., Am) c R2 (B1, B2, ..., Bn) EQUIJOIN: The join condition c includes one or more equality comparisons involving attributes from R1 and R2. That is, c is of the form: (Ai=Bj) AND ... AND (Ah=Bk); 1<i,h<m, 1<j,k<n In the above EQUIJOIN operation: Ai, ..., Ah are called the join attributes of R1 Bj, ..., Bk are called the join attributes of R2 Example of using EQUIJOIN: Retrieve each DEPARTMENT's name and its manager's name: T DEPARTMENT MGRSSN = SSN EMPLOYEE RESULT DNAME,FNAME,LNAME (T) NATURAL JOIN (*): In an EQUIJOIN R R1 c R2, the join attribute of R2 appear redundantly in the	10	CO2	L3

	result relation R. In a NATURAL JOIN, the redundant join attributes of R2 are eliminated from R. The equality condition is implied and need not be specified. R R1 *(join attributes of R1),(join attributes of R2) R Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for: T EMPLOYEE *(DNO),(DNUMBER) DEPARTMENT RESULT FNAME,LNAME,DNAME (T) If the join attributes have the same names in both relations, they need not be specified and we can write R R1 * R2. Example: Retrieve each EMPLOYEE's name and the name of his/her SUPERVISOR:			
6 a.	Demonstrate the usage of the following SQL commands by writing and executing appropriate queries on a sample database: i) INSERT, ii) DELETE, iii) UPDATE, iv) ALTER, v) SELECT insert INSERT INTO target [(field1[, field2[, ...]])] VALUES (value1[, value2[, ...]); So, to add a User record for user Jim Jones, we would issue the following INSERT query: INSERT INTO User (FirstName, LastName, UserID, Dept, EmpNo, PCType) 6 VALUES ("Jim", "Jones", "Jjones","Finance", 9, "DellDimR450"); Obviously populating a database by issuing such a series of SQL commands is both tedious and prone to error, which is another reason why database applications have front-ends. Even without a specifically designed front-end, many database systems - including MS Access - allow data entry direct into tables via a spreadsheet-like interface. ii) Delete Now that we know how to add new records and to update existing records it only remains to learn how to delete records before we move on to look at how we search through and collate data. As you would expect SQL provides a simple command to delete complete records. The syntax of the command is: DELETE [table.*] FROM table WHERE criteria; iii) drop If you have already executed the original CREATE TABLE command your database will already contain a table called User, so let's get rid of that using the DROP command:	6	CO3	L3

	DROP TABLE User; iv) alter Once a table is created it's structure is not necessarily fixed in stone. In time requirements change and the structure of the database is likely to evolve to match your wishes. SQL can be used to change the structure of a table, so, for example, if we need to add a new field to our User table to tell us if the user has Internet access, then we can execute an SQL ALTER TABLE command as shown below: ALTER TABLE User ADD COLUMN Internet BOOLEAN; To delete a column the ADD keyword is replaced with DROP, so to delete the field we have just added the SQL is: ALTER TABLE User DROP COLUMN Internet; v) update the UPDATE command, with syntax: UPDATE table SET newvalue WHERE criteria; For example, let's assume that we want to move user Jim Jones from the Finance department to Marketing. Our SQL statement would then be: UPDATE User SET Dept="Marketing" WHERE EmpNo=9;			
6 b.	For the Movie database given in question number 3a, Write SQL query for the following. 1. Retrieve the **Movie Title and Release Year** of all movies released after **2018**. 2. Retrieve the **names** of all **directors** from the database. 3. Retrieve all details of movies that belong to the **"Comedy"** genre. **1. Retrieve the Movie Title and Release Year of all movies released after 2018** sql Copy SELECT title AS "Movie Title", release_year AS "Release Year" FROM movies WHERE release_year > 2018; **2. Retrieve the names of all directors from the database** sql Copy SELECT DISTINCT director_name AS "Director Name" FROM directors; **3. Retrieve all details of movies that belong to the "Comedy" genre** SELECT * FROM movies	4	CO3	L3

	WHERE genre = 'Comedy';			

HOD Signature

Faculty Signature CCI Signature