USN | | | | | | | | | |

Internal Assessment Test 1 – March 2025

Sub:	Database Management System	Sub Code:	BCS403	Branch:	CSE

Question Paper

		MARKS	CO	RBT
1	Differentiate the following concepts with relevant example: i. Database Schema vs Database State ii. Logical independence vs Physical independence iii. Composite attribute vs Multivalued attribute iv. Primary Key vs Super Key v. Strong entity vs Weak entity	[10M]	CO1	L2
2	Consider the database schemas as Employee (E_id, E_name, salary, age, address) and Write SQL statements to: i. Create the Employee table with primary key constraint and not null constraint. ii. Insert 5 records in the Employee table iii. Retrieve names of all employees whose age is greater than 25 iv. Delete record of the Employee named "Hari". v. Update salary of the Employee named "Sam" by 15%.	[10M]	CO2	L3
3 (a)	Describe the Three-schema architecture with a neat diagram. Relate the different data models with this architecture.	[5M]	CO1	L2
(b)	Discuss the main characteristics of the database approach.	[5M]	CO1	L2
4	Consider the following scenario of a Bank database: Bank have Customer. Banks are identified by a name, code, address of main office. Banks have branches. Branches are identified by a branch_no., branch_name, address. Customers are identified by name, cust-id, phone number, address. Customer can have one or more accounts. Accounts are identified by account_no., acc_type, balance. Customer can avail loans. Loans are identified by loan_id, loan_type and amount. Account and loans are related to bank's branch. For the above-mentioned scenario draw (show all types of constraints as applicable): (i) Schema diagram (ii) ER Diagram	[10M]	CO1	L3
5	Consider the database schema: Faculty (ID, Name, Dept, Sal, Address) Write SQL statements to: i. Find the maximum, minimum, total salary of Faculty members. ii. Find the count of faculty in each Dept. iii. Sort all the faculty members in descending order of their ID. iv. Find the highest salary of CSE dept faculty. v. Change the datatype of Sal attribute from int to float.	[10M]	CO2	L3
6	Suppose there is a banking database which comprises following tables : (i) Customer(Cust_name, Cust_street, Cust_city) (ii) Loan (Branch_name, Loan_number, Amount) (iii) Depositor(Cust_name, Account_number) & (iv) Borrower(Cust_name, Loan_number) Q1) Find the names of all the customers who have taken a loan from the bank and also have an account at the bank. Q2) Find the loan amount of all the borrowers who have taken a loan from the bank Q3) Rename the Loan relation to Loan-Details and also change the attribute Loan_number to Loan-no Q4) List only Customer names who stays in Bangalore City	[10M]	CO1	L2

Scheme & Solution

1	**Differentiate the following concepts with relevant example:** **i) Database Schema vs Database State** **ii) Logical independence vs Physical independence** **iii) Composite attribute vs Multivalued attribute** **iv) Primary Key vs Super Key** v) **Strong entity vs Weak entity**	[5Q*2M= 10M]	CO1	L2

(i) <u>Database Schema vs Database State</u>

Database Schema:

- The **schema** is the **structure** or blueprint of the database. It defines how the data is organized, including tables, columns, data types, relationships, constraints, and indexes.
- It is essentially a design or framework that tells you what kind of data can be stored in the database, how it can be stored, and the relationships between various pieces of data.

Example: Let's say we are designing a database for a library system. The schema could be as follows:

- **Tables**:
  - `Books` table with columns like `book_id`, `title`, `author`, `published_year`, etc.
  - `Members` table with columns like `member_id`, `name`, `email`, `phone`, etc.
  - `BorrowedBooks` table with columns like `borrow_id`, `member_id`, `book_id`, `borrow_date`, `return_date`, etc.

Database State:

- The **state** of the database refers to the **current content** of the database at a particular moment in time. It is the actual data that resides in the tables as per the schema.
- This is the **instance** or the **snapshot** of the database, showing all the actual records, values, and their current state in all the tables at any given point in time.

Example: Let's assume some data has already been inserted into the library system's database:

- `Books` table:
  - `(1, 'The Catcher in the Rye', 'J.D. Salinger', 1951)`
  - `(2, 'To Kill a Mockingbird', 'Harper Lee', 1960)`
- `Members` table:
  - `(1, 'Alice Johnson', 'alice@example.com', '555-1234')`
  - `(2, 'Bob Smith', 'bob@example.com', '555-5678')`

(ii) Logical independence vs Physical independence

- **Logical independence** refers to the **ability to change the logical schema** (the conceptual design) of the database without affecting the external schema (user views) or the application programs.

Example: Suppose we have a database system with the following structure for an **Employee** table:

- **Logical Schema** (Conceptual Level):
  - Table `Employee` with columns: `emp_id`, `emp_name`, `emp_salary`.

Now, you want to split the `Employee` table into two separate tables for better data organization:

- `EmployeePersonal` table with columns: `emp_id`, `emp_name`.
- `EmployeeSalary` table with columns: `emp_id`, `emp_salary`.

If the database has **logical independence**, you can change the logical schema without affecting the external schema (user views or application programs). Users and applications should still be able to access the data without knowing that the table has been split into two.

- **Physical independence** refers to the **ability to change the physical storage structure** (how the data is stored, indexed, or partitioned) without affecting the logical schema or the application programs.
- This means you can optimize or modify the storage mechanisms (such as indexing, file organization, or storage devices) for performance improvements without impacting how users access the data or the logical structure of the database.

Example: Let's say that you have an `Employee` table, and the data is stored in a particular format or on a specific set of disks. If you change the way the data is physically stored (for example, using a new indexing technique, partitioning the table, or moving the database to a new server), **physical independence** ensures that the logical schema (how the data is organized conceptually) and the application programs are unaffected by these physical changes.

Physical Independence Example:

- Initially, the `Employee` table is stored on a single disk.
- Later, you decide to store the table on multiple disks (partitioning), or use a **B-tree index** to optimize query performance for searching employees by their `emp_id`.

(iii) Composite attribute vs Multivalued attribute

- A **composite attribute** is an attribute that can be **divided into smaller sub-attributes**, which are more meaningful individually. These sub-attributes can collectively represent the information of the composite attribute.
- Essentially, it's an attribute that can be broken down into other attributes.

Example: Consider an entity **Person**. One of the attributes of this entity could be **Address**. An **Address** could be broken down into several sub-attributes, such as:

- `Street`
- `City`
- `State`
- `ZipCode`

In this case, the **Address** attribute is a **composite attribute** because it can be divided into smaller components (sub-attributes) that represent specific pieces of information about the address.

- A **multivalued attribute** is an attribute that can have **multiple values** for a single entity. This means that an entity can have more than one value for this attribute at the same time.
- For example, a person can have multiple phone numbers, multiple email addresses, or multiple skills.

Example: Consider an entity **Employee**. An **Employee** can have multiple **phone numbers**. This means the **PhoneNumbers** attribute is a **multivalued attribute** because it can have multiple values for a single employee.

(iv) Primary Key vs Super Key

- A **super key** is any combination of **attributes (columns)** that can uniquely identify a record (row) in a table.
- It can be a single attribute or a set of attributes.
- A table can have multiple super keys, as adding more attributes to an existing key (even if those extra attributes are not necessary for uniqueness) still qualifies it as a super key.
- A **primary key** is a **special type of super key** that is selected to uniquely identify records in a table.
- A **primary key** must satisfy two main conditions:

1. **Uniqueness**: Each value in the primary key must be unique for each record.
2. **Non-nullability**: No part of the primary key can have a null value.

- There can only be **one primary key** for a table, but it could be made up of one or more attributes (i.e., it could be a **composite key**).

	(v) <u>Strong entity vs Weak entity</u> • A **strong entity** (also called a **regular entity**) is an entity that **can exist independently** of any other entity. It has a **primary key** that uniquely identifies each instance of the entity. • Strong entities have their own unique identifiers (attributes that can uniquely identify an instance of the entity) and don't depend on any other entity for their identification. • A **weak entity** is an entity that **cannot exist independently**. It depends on a **strong entity** (called the **owner entity**) for its identification. • A weak entity does not have a unique primary key on its own. Instead, it uses a **partial key** (also called a **discriminator**) in combination with the primary key of the strong entity to form a **composite key** that uniquely identifies instances of the weak entity. • A weak entity is typically represented with a **double rectangle** in an ER diagram			
2	**Consider the database schemas as Employee (E_id, E_name, salary, age, address) and Write SQL statements to:** **i) Create the Employee table with primary key constraint and not null constraint.** **ii) Insert 5 records in the Employee table** **iii) Retrieve names of all employees whose age is greater than 25** **iv) Delete record of the Employee named "Hari".** **v) Update salary of the Employee named "Sam" by 15%.**	[5Q*2M= 10M]	CO2	L3
	i) CREATE TABLE Employee (E_id INT PRIMARY KEY, -- Primary key on E_id E_name VARCHAR(255) NOT NULL, -- NOT NULL constraint on E_name salary DECIMAL(10, 2) NOT NULL, -- NOT NULL constraint on salary age INT NOT NULL, -- NOT NULL constraint on age address VARCHAR(255) NOT NULL -- NOT NULL constraint on address); ii) INSERT INTO Employee (E_id, E_name, salary, age, address) VALUES (1, 'John', 50000.00, 30, '123 Main St'); INSERT INTO Employee (E_id, E_name, salary, age, address) VALUES (2, 'Sam', 60000.00, 27, '456 Oak St'); INSERT INTO Employee (E_id, E_name, salary, age, address) VALUES (3, 'Hari', 45000.00, 24, '789 Pine St'); INSERT INTO Employee (E_id, E_name, salary, age, address) VALUES (4, 'Alice', 55000.00, 28, '101 Maple St'); INSERT INTO Employee (E_id, E_name, salary, age, address) VALUES (5, 'Bob', 48000.00, 35, '202 Birch St');			

iii)

```
SELECT E_name
FROM Employee
WHERE age > 25;
```

iv)

```
DELETE FROM Employee
WHERE E_name = 'Hari';
```

v)

```
UPDATE Employee
SET salary = salary * 1.15
WHERE E_name = 'Sam';
```

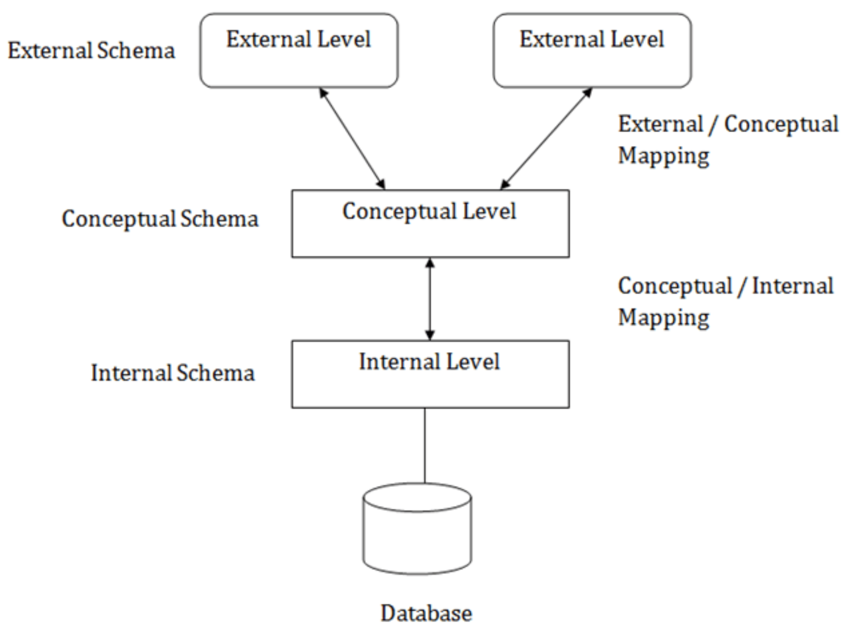
3 (a)	**Describe the Three-schema architecture with a neat diagram. Relate the different data models with this architecture.**	[5M]	CO1	L2

The **Three-Schema Architecture** is a framework used in Database Management Systems (DBMS) to separate user views, the logical structure of data, and the physical storage of data. This architecture helps achieve data independence, allowing changes in one level of the database schema without affecting the others.

The architecture consists of three levels:

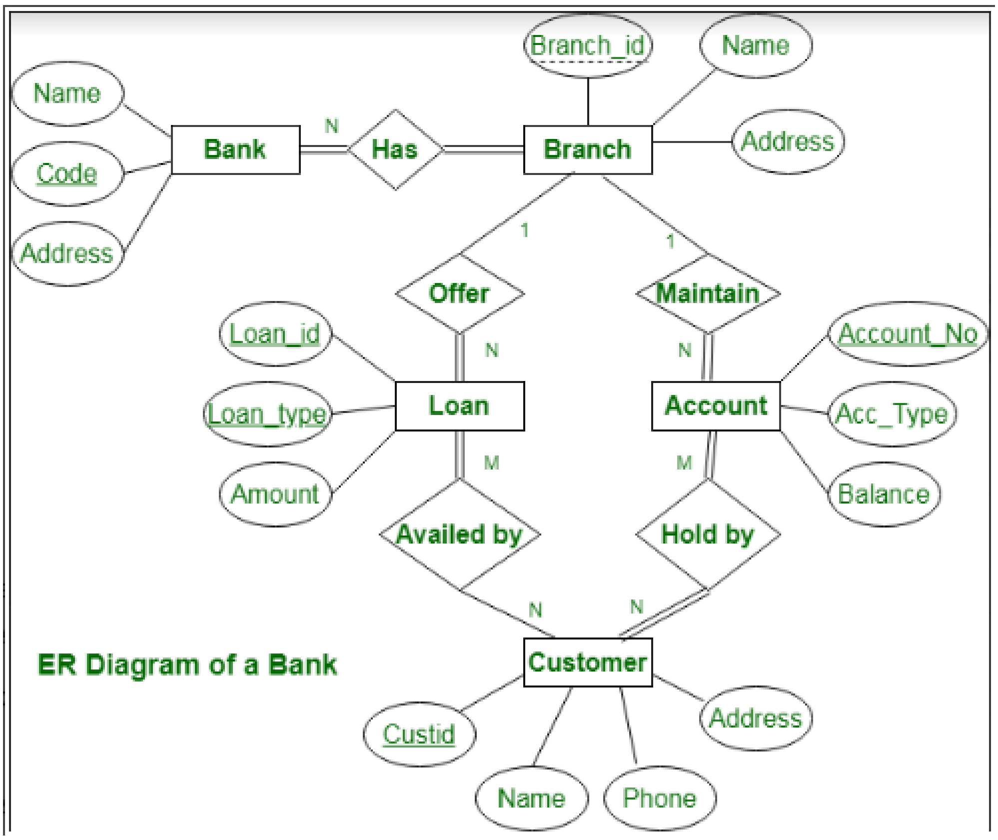
1. **External Schema (View Level)**
2. **Conceptual Schema (Logical Level)**
3. **Internal Schema (Physical Level)**

The three-schema architecture is as follows:



(b)	**Discuss the main characteristics of the database approach.**	[5M]	CO1	L2

1. Self-describing nature of a database system
2. Insulation between programs and data

	3. Data Abstraction: 4. Support of multiple views of the data: 5. Sharing of data and multi-user transaction processing			
4	Consider the following scenario of a Bank database: Bank have Customer. Banks are identified by a name, code, address of main office. Banks have branches. Branches are identified by a branch_no., branch_name, address. Customers are identified by name, cust-id, phone number, address. Customer can have one or more accounts. Accounts are identified by account_no., acc_type, balance. Customer can avail loans. Loans are identified by loan_id, loan_type and amount. Account and loans are related to bank's branch. For the above-mentioned scenario draw (show all types of constraints as applicable): (i) Schema diagram (ii) ER Diagram	[4M+6M]	CO1	L3
				
5	Consider the database schema: Faculty (ID, Name, Dept, Sal, Address) Write SQL statements to: i) Find the maximum, minimum, total salary of Faculty members. ii) Find the count of faculty in each Dept. iii) Sort all the faculty members in descending order of their ID. iv) Find the highest salary of CSE dept faculty. v) Change the datatype of Sal attribute from int to float.	[5Q*2M= 10M]	CO2	L3
	i) Find the maximum, minimum, total salary of Faculty members. SELECT MAX(Sal) AS Max_Salary, MIN(Sal) AS Min_Salary, SUM(Sal) AS Total_Salary FROM Faculty;			

	ii) Find the count of faculty in each Dept. SELECT Dept, COUNT(*) AS Faculty_Count FROM Faculty GROUP BY Dept; iii) Sort all the faculty members in descending order of their ID. SELECT * FROM Faculty ORDER BY ID DESC; iv) Find the highest salary of CSE dept faculty. SELECT MAX(Sal) AS Highest_Salary FROM Faculty WHERE Dept = 'CSE'; v) Change the datatype of Sal attribute from int to float. ALTER TABLE Faculty MODIFY COLUMN Sal FLOAT;			
6	Suppose there is a banking database which comprises following tables : (i) Customer(Cust_name, Cust_street, Cust_city) (ii) Loan (Branch_name, Loan_number, Amount) (iii) Depositor(Cust_name, Account_number) & (iv) Borrower(Cust_name, Loan_number) Q1) Find the names of all the customers who have taken a loan from the bank and also have an account at the bank. Q2) Find the loan amount of all the borrowers who have taken a loan from the bank Q3) Rename the Loan relation to Loan-Details and also change the attribute Loan_number to Loan-no Q4) List only Customer names who stays in Bangalore City	[4Q*2.5M= 10M]	CO1	L2
	Q1) Find the names of all the customers who have taken a loan from the bank and also have an account at the bank. $\pi Cust\_name(Borrower) \cap \pi Cust\_name(Depositor)$ **Explanation:** • **Projection (π)** is used to extract the `Cust_name` attribute from both the `Borrower` and `Depositor` relations. • **Intersection (∩)** retrieves customers present in both relations, ensuring they have both a loan and an account. Q2) Find the loan amount of all the borrowers who have taken a loan from the bank $\pi Cust\_name,Amount(\sigma Borrower.Loan\_number=Loan.Loan\_number (Borrower \bowtie Loan))$			

Explanation:

- **Selection (σ)** ensures that only matching `Loan_number` values from `Borrower` and `Loan` are considered.
- **Natural Join (⋈)** is performed between `Borrower` and `Loan` on `Loan_number` to combine borrower details with loan amounts.
- **Projection (π)** extracts only `Cust_name` and `Amount`.

Q3) Rename the Loan relation to Loan-Details and also change the attribute Loan_number to Loan-no

ρLoan-Details(Branch_name, Loan-no, Amount)(Loan)

Explanation:

- **Rename (ρ)** is used to change the table name from `Loan` to `Loan-Details`.
- **Rename (ρ)** is used again to rename the column `Loan_number` to `Loan-no`.

Q4) List only Customer names who stays in Bangalore City

πCust_name(σCust_city=′Bangalore′(Customer))

Explanation:

- **Selection (σ)** filters out customers who live in `Bangalore`.
- **Projection (π)** extracts only the `Cust_name` column.