(Affiliated to University of Mumbai)	***End Semester Examination (R-24/R-25) SH 2026 Answer Key with marking scheme***
Branch: IT	Course: Database Management Systems
Year/ Semester: SE / III	Course code: ITC303
Time: 03 hours	Marks: 80

		Marks
Q. 1	All questions are compulsory.	
A.	**Explain Set Theoretic Relational Algebra operations: Union, Intersection, Set Difference and Cartesian Product.** 1. **Union (R ∪ S)** – all tuples in R or S or both, duplicates removed. 1M 2. **Intersection (R ∩ S)** – tuples present in both R and S. 1M 3. **Set Difference (R − S)** – tuples present in R but not in S. 1M 4. **Cartesian Product (R × S)** – every tuple of R concatenated with every tuple of S; degree = n + m, tuples = |R| × |S|. 1M 5. Union, Intersection and Difference need union-compatible relations (same degree and domains); Cartesian Product does not. Example: R = {1,2,3}, S = {2,3,4} → R ∪ S = {1,2,3,4}, R ∩ S = {2,3}, R − S = {1}. 1M	5M
B.	**R(A,B) = {(1,10),(2,20),(3,30)}, S(B,C) = {(10,'X'),(20,'Y'),(40,'Z')}. Compute (a) R ⋈ S, (b) R Left Outer Join S, (c) R Right Outer Join S.** (a) **Natural Join R ⋈ S** (join on B) – 1M Result (A,B,C): {(1,10,'X'), (2,20,'Y')} (b) **R ⟕S (Left Outer Join)** – 2M Result (A,B,C): {(1,10,'X'), (2,20,'Y'), (3,30,NULL)} (c) **R ⟖S (Right Outer Join)** – 2M Result (A,B,C): {(1,10,'X'), (2,20,'Y'), (NULL,40,'Z')}	5M

C.	**Discuss any five SQL aggregate functions with examples.**
	Each function with description and example – 1M × 5 = 5M (Employee table: Eid, Ename, Salary, Dept)
	1. **COUNT()** – returns number of rows. 1M
	`SELECT COUNT(*) FROM Employee;`
	2. **SUM()** – returns total of a numeric column. 1M
	`SELECT SUM(Salary) FROM Employee;`
	3. **AVG()** – returns average value of a numeric column. 1M
	`SELECT AVG(Salary) FROM Employee;`
	4. **MIN()** – returns smallest value. 1M
	`SELECT MIN(Salary) FROM Employee;`
	5. **MAX()** – returns largest value. 1M
	`SELECT MAX(Salary) FROM Employee;`
		5M

D.	**Classify DML commands with suitable examples.** Classification of DML 1M; SELECT 1M; INSERT 1M; UPDATE 1M; DELETE 1M = 5M **Classification (1M):** DML commands are used to retrieve and modify data in tables. DML is classified as **procedural DML** (user specifies what data and how to get it) and **non-procedural/declarative DML** (user specifies only what data is needed, e.g. SQL). SQL DML commands: SELECT, INSERT, UPDATE, DELETE. 1. **SELECT** – retrieves data from table(s). 1M	5M

```
SELECT Ename, Salary FROM Employee WHERE Salary > 30000;
```

2. **INSERT** – adds new rows. 1M

```
INSERT INTO Employee VALUES (101, 'Asha', 40000, 'IT');
```

3. **UPDATE** – modifies existing rows. 1M

```
UPDATE Employee SET Salary = 45000 WHERE Eid = 101;
```

4. **DELETE** – removes rows. 1M

```
DELETE FROM Employee WHERE Eid = 101;
```

E. **Describe the different data models.** 5M

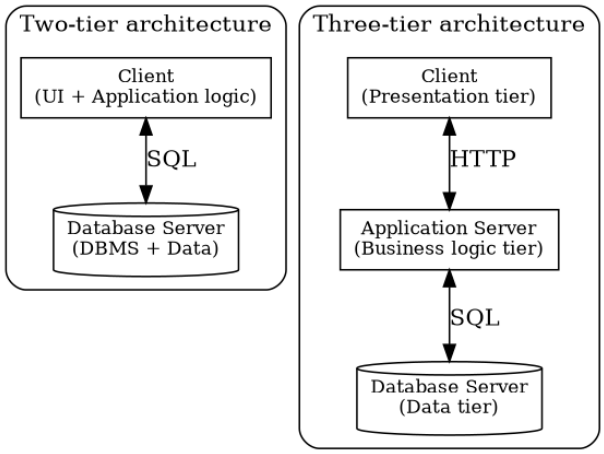
Introduction/definition of data model 1M; four models 1M each = 5M

Data model (1M): a collection of concepts used to describe the structure of a database (data types, relationships, constraints).

1. **Relational model** – data stored as tables (relations) of rows and columns; keys link tables; queried using SQL/relational algebra. 1M
2. **Entity–Relationship (ER) model** – conceptual model using entities, attributes and relationships, shown in ER diagrams. 1M
3. **Object-based model** – data represented as objects with attributes, methods, classes and inheritance (object-oriented / object-relational). 1M
4. **Semi-structured model** – data without a rigid schema, self-describing using tags/key–value pairs (XML, JSON). 1M

F. **Differentiate two and three tier DBMS architecture with diagram.** 5M

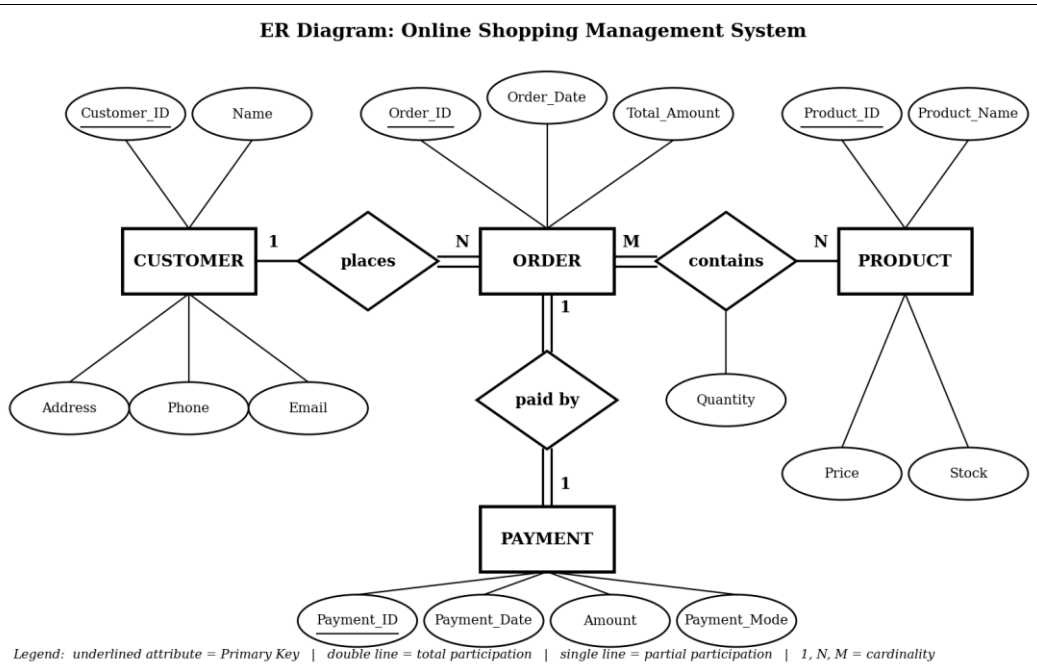
Diagram 2M; differences (any three points) 3M



Point	Two-tier	Three-tier
Layers	Client and Database server	Client, Application server and Database server
Business logic	In the client (or in DB)	In the separate application server
Scalability	Poor, limited users	High, tiers can scale separately
Security	Client talks directly to DB (less secure)	DB hidden behind application tier (more secure)
Maintenance	Difficult, client updates needed	Easier, changes at application server
Example	Desktop apps using ODBC/JDBC	Web applications

Q.2A **Design an ER diagram for an Online Shopping Management System with entities Customer, Product, Order and Payment. Show relationships, cardinalities and key attributes.** 10M

Entities with key attributes 4M; relationships 3M; cardinalities 2M; notation/neatness 1M



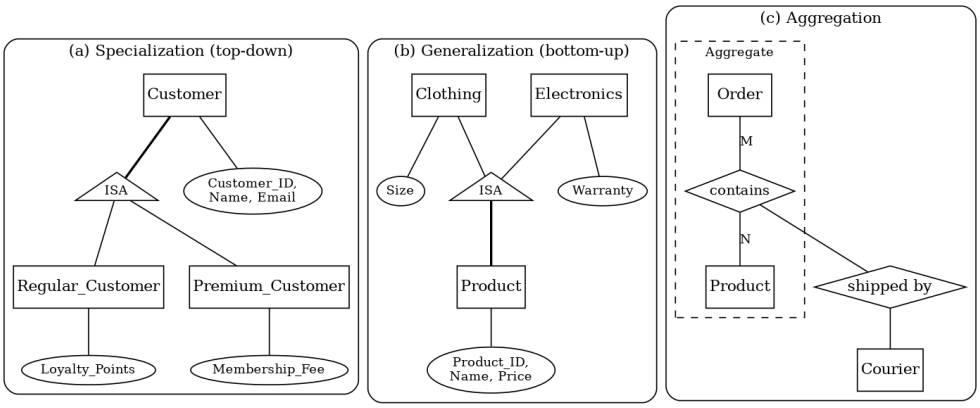
Entities: **Customer**(Customer_ID, Name, Email, Phone, Address); **Product**(Product_ID, Name, Price, Stock); **Order**(Order_ID, Order_Date, Status, Total_Amount); **Payment**(Payment_ID, Amount, Pay_Date, Mode). Underlined attribute = key.

Relationships: Customer **places** Order (1 : N); Order **contains** Product (M : N, with relationship attribute Quantity); Order **paid by** Payment (1 : 1).

OR

Q.2B. **Using the Online Shopping Management System case study given in Q.2A, illustrate Specialization, Generalization and Aggregation in the EER model with suitable examples.** 10M

Specialization 3M; Generalization 3M; Aggregation 3M; diagram/neatness 1M



1. Specialization (top-down): process of defining sub-classes of an entity based on distinguishing characteristics; sub-classes inherit all attributes and relationships of the super-class and add their own. Example: CUSTOMER specialised into REGULAR_CUSTOMER (Loyalty_Points) and PREMIUM_CUSTOMER (Membership_Fee) – disjoint, partial. 3M

2. Generalization (bottom-up): process of combining entities that share common attributes into a higher-level super-class. Example: ELECTRONICS (Warranty) and CLOTHING (Size) share Product_ID, Name, Price, so they are generalised into PRODUCT. (Same ISA hierarchy read bottom-up.) 3M

	3. Aggregation: a relationship (with its participating entities) is treated as a single higher-level entity so that it can take part in another relationship; used when a relationship must relate to a relationship. Example: the relationship ORDER **contains** PRODUCT is aggregated and related to COURIER through **shipped by**, i.e. a courier ships a particular ordered product item. 3M	
Q.3A	**Employee(EmpID, Name, Salary, DeptID); Department(DeptID, DeptName, Location). Write Relational Algebra queries.** Each query 2M i. Employees with salary greater than 50000 $\sigma_{Salary > 50000}(Employee)$ ii. Natural join between Employee and Department Employee ⋈ Department iii. Names and departments of all employees $\pi_{Name, DeptName}(Employee \bowtie Department)$ iv. Names of employees working in the IT department $\pi_{Name}(\sigma_{DeptName = 'IT'}(Employee \bowtie Department))$ v. Names and salaries of all employees $\pi_{Name, Salary}(Employee)$	10M
	OR	

Q.3B	***CO3 | BL: Apply (L3)***
	1. Strong entity: becomes a relation; simple attributes become columns; composite attribute → its simple components; entity key → primary key. **[2]**
	2. Multivalued attribute: separate relation with the entity key as FK; PK = (FK, attribute). Weak entity: relation with owner key as FK; PK = (owner key, partial key). **[2]**
	3. Binary relationships: 1:1 → add FK to either side (prefer total side) [1]; 1:N → add FK on N side [1]; M:N → new relation with both keys as FKs (PK = both) plus relationship attributes. **[1]**
	4. EER specialization/generalization (any two options): (a) relation for superclass and each subclass (subclass PK = superclass key); (b) relation only for subclasses (total, disjoint); (c) single relation with type attribute; (d) single relation with boolean flags (overlapping). **[2]**
	Example (1 mark ER, 1 mark mapping): STUDENT(Roll_No, Name, Phone*) and COURSE(Course_ID, Title) with M:N ENROLLS(Grade); PERSON → STUDENT, TEACHER. **[2]**
	`STUDENT(Roll_No PK, Name)  STUDENT_PHONE(Roll_No FK, Phone)`
	`COURSE(Course_ID PK, Title)  ENROLLS(Roll_No FK, Course_ID FK, Grade)`
	`PERSON(P_ID PK, Name)  STUDENT(P_ID PK/FK, Branch)  TEACHER(P_ID PK/FK, Dept)`
		10M

Q.4A	**Course(Cid, Course_Name, Staff_name, Duration, fees) and Student(Sid, name, Location, Cid).**
	Each part 2M
	(i) Create Course table and insert values
	`CREATE TABLE Course (`
	`  Cid INT PRIMARY KEY,`
	`  Course_Name VARCHAR(20),`
	`  Staff_name VARCHAR(20),`
	`  Duration INT,`
	`  fees INT);`
	`INSERT INTO Course VALUES (1,'DBMS','Menon',6,45000);`
	`INSERT INTO Course VALUES (2,'PCPF','Rai',4,28000);`
	`INSERT INTO Course VALUES (3,'JAVA','Rajput',2,16000);`
		10M

```
INSERT INTO Course VALUES (4,'DSA','Govilkar',5,32000);
```

(ii) Create Student table with Cid as foreign key

```
CREATE TABLE Student (
  Sid INT PRIMARY KEY,
  name VARCHAR(20),
  Location VARCHAR(20),
  Cid INT,
  FOREIGN KEY (Cid) REFERENCES Course(Cid));
INSERT INTO Student VALUES (1,'Anaya','Thane',1);
-- remaining 4 rows inserted similarly
```

(iii) Courses in descending order of fees

```
SELECT * FROM Course ORDER BY fees DESC;
```

(iv) Course enrolled by student 'Rajiv'

```
SELECT c.Course_Name FROM Course c, Student s
WHERE c.Cid = s.Cid AND s.name = 'Rajiv';
```

Output: DSA

(v) Students whose course duration is more than 3 weeks

```
SELECT s.name FROM Student s, Course c
WHERE s.Cid = c.Cid AND c.Duration > 3;
```

Output: Anaya, Rajiv, Suyog, Dhariya

OR

Q.4B — 10M

CO4 | BL: Apply (L3) (2 marks per part)

(i) **[2]**

```
SELECT Dept_Name, SUM(Budget) AS Total_Budget FROM PROJECT GROUP BY
Dept_Name;
```

(ii) (Duration assumed in months) **[2]**

```
UPDATE PROJECT SET Dept_Name='Development' WHERE Duration < 6;
```

(iii) **[2]**

```
CREATE VIEW project_record AS SELECT Project_ID, Project_Name, Budget
FROM PROJECT;
```

(iv) **[2]**

```
SELECT * FROM PROJECT ORDER BY Budget DESC;
```

(v) GRANT 1 + REVOKE 1 **[2]**

```
GRANT SELECT ON PROJECT TO user1;
REVOKE SELECT ON PROJECT FROM user1;
```

Q.5A — 10M

Define normalization. Explain 1NF and 2NF in detail with an example.

Definition 2M; 1NF (definition 1M + example 2M) 3M; functional dependencies 1M; 2NF (definition 1M + decomposition 3M) 4M

Normalization: process of organising relations by decomposing them using functional dependencies to reduce redundancy and eliminate insert, delete and update anomalies.

Relation (unnormalized): STUDENT_COURSE(Student_ID, Student_Name, Course_ID, Course_Name, Grade)

Student_ID	Student_Name	Course_ID	Course_Name	Grade
S1	Amit	C1, C2	DBMS, OS	A, B
S2	Riya	C1	DBMS	B

1NF: every attribute holds only atomic (single) values. Split multivalued rows; PK = (Student_ID, Course_ID).

Student_ID	Student_Name	Course_ID	Course_Name	Grade
S1	Amit	C1	DBMS	A
S1	Amit	C2	OS	B

S2	Riya	C1	DBMS	B

FDs: Student_ID → Student_Name; Course_ID → Course_Name; (Student_ID, Course_ID) → Grade. Student_Name and Course_Name depend on part of the key (partial dependency), so the relation is in 1NF but not 2NF.

2NF: relation is in 1NF and no non-prime attribute is partially dependent on the candidate key. Decompose:

```
STUDENT(Student_ID PK, Student_Name)
COURSE(Course_ID PK, Course_Name)
ENROLL(Student_ID FK, Course_ID FK, Grade)   PK(Student_ID, Course_ID)
```

OR

Q.5B **Illustrate the three relational anomalies (Insertion, Deletion and Updation) in the EMP_DEPT(Emp_ID, Emp_Name, Dept_ID, Dept_Name, Dept_Location) table using suitable examples, and suggest how normalization can help eliminate these anomalies.** 10M

Sample table 1M; each anomaly with example 2M; normalization solution 3M

Emp_ID	Emp_Name	Dept_ID	Dept_Name	Dept_Location
E1	Ravi	D1	Sales	Mumbai
E2	Sita	D1	Sales	Mumbai
E3	Amit	D2	HR	Pune

1. **Insertion anomaly:** a new department D3 (Accounts, Delhi) cannot be inserted until an employee is assigned, as Emp_ID (primary key) cannot be NULL.

2. **Deletion anomaly:** deleting employee E3 also deletes all information about department D2 (HR, Pune).

3. **Updation anomaly:** changing the location of D1 needs updates in every row of D1 (E1 and E2); missing one gives inconsistent data.

Normalization: the FD Dept_ID → Dept_Name, Dept_Location (transitive via Dept_ID) causes the redundancy. Decompose into 3NF relations (lossless, dependency preserving):

```
EMPLOYEE(Emp_ID PK, Emp_Name, Dept_ID FK)
DEPARTMENT(Dept_ID PK, Dept_Name, Dept_Location)
```

Now a department can exist without employees, deleting an employee keeps department data, and a location change is made in one row only.

Q.6A **Explain deadlock with respect to database transactions. Also discuss deadlock handling.** 10M

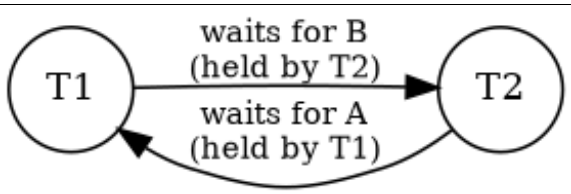
Deadlock definition and conditions 2M; example 2M; wait-for graph 2M; handling techniques 4M

Deadlock: a situation where two or more transactions wait indefinitely, each holding a lock needed by another. Necessary conditions: mutual exclusion, hold and wait, no preemption, circular wait.

Example: (A and B are data items)

Time	T1	T2
t1	Lock-X(A)	
t2		Lock-X(B)
t3	Lock-X(B) → waits for T2	
t4		Lock-X(A) → waits for T1

Wait-for graph: nodes = transactions; edge Ti → Tj if Ti waits for a lock held by Tj. A cycle (T1 → T2 → T1) means deadlock.



Handling techniques:

1. **Prevention:** wait-die and wound-wait (timestamp based); conservative 2PL (lock all items in advance); timeout.
2. **Detection:** DBMS periodically builds the wait-for graph and searches for a cycle.
3. **Recovery:** select a victim (minimum cost/youngest), roll it back (total or partial), release its locks; avoid starvation by considering rollback count. Here T2 is rolled back, T1 gets B and completes, then T2 restarts.

OR

Q.6B **Explain about Lock based protocol for concurrency control.** 10M

Need/lock types 2M; compatibility 1M; schedule 2M; 2PL 3M; variants and drawbacks 2M

Lock-based protocol: a transaction must obtain a lock on a data item before accessing it, so concurrent transactions do not interfere. Lock modes: **Shared (S)** – read only, many transactions may hold it; **Exclusive (X)** – read and write, only one transaction may hold it. If the lock is not compatible, the requester waits.

Requested \ Held	**S**	**X**
S	Granted	Wait
X	Wait	Wait

Schedule using locks (T1 transfers 100 from A to B; T2 reads A and B):

Step	**T1**	**T2**
1	Lock-X(A); read(A); A = A − 100; write(A)	
2		Lock-S(A) → blocked (A held in X mode)
3	Lock-X(B); read(B); B = B + 100; write(B) [lock point]	
4	Unlock(A); Unlock(B)	
5		Lock-S(A); read(A); Lock-S(B); read(B) [lock point]
6		Unlock(A); Unlock(B)

Two-Phase Locking (2PL): Growing phase – transaction may only obtain locks, none released; Shrinking phase – may only release locks, none obtained. The point where the last lock is obtained is the lock point. 2PL guarantees conflict-serializable schedules (serial order = order of lock points).

Variants: Basic 2PL,Strict 2PL – X locks held until commit/abort (avoids cascading rollbacks); Rigorous 2PL – all locks held until commit; Conservative 2PL – all locks acquired before execution (no deadlock).

Drawbacks: basic 2PL may cause deadlock and cascading rollbacks; locking overhead reduces concurrency.

*********** All the Best***********