

Database Management System (313302)

Model Question Paper — Set C · Model Answers

About the SQL and PL/SQL in these answers: All code is written in **Oracle SQL / PL/SQL** (as used in the MSBTE syllabus and practicals). Every query and program was actually executed before printing — on PostgreSQL 16, with Oracle data types and PL/SQL syntax converted automatically — and the result tables and outputs shown are from those runs, displayed in SQL*Plus style (headings in capitals, Oracle feedback messages such as “1 row updated.”). Sample data is assumed wherever a question says “assume suitable data”.

Q.1

a) Data and data warehouse.

2 M

i) Data: Data means raw, unprocessed facts and figures (numbers, text, images) that can be recorded and stored, and that have no meaning until processed. Example: 101, ‘Asha’, 78.

ii) Data warehouse: A data warehouse is a large, central, subject-oriented, integrated, time-variant and non-volatile collection of data, gathered from many operational sources, used for analysis, reporting and decision making (OLAP, data mining).

(Each definition 1 M)

b) Hierarchical vs network model.

2 M

Point	Hierarchical model	Network model
Structure	Tree (parent–child)	Graph (owner–member sets)
Parents of a child	Only one parent	Many parents allowed
Relationships	1:1 and 1:N	1:1, 1:N and M:N
Access path	Only from root downwards	Many paths to a record

(Any two points, 1 M each)

c) Candidate key with example.

2 M

Candidate key: A candidate key is a minimal set of attributes that uniquely identifies each tuple in a relation; no proper subset of it can identify tuples uniquely. A relation may have several candidate keys and one of them is chosen as the primary key.

Example: Student (roll_no, enroll_no, aadhaar_no, name) — roll_no, enroll_no and aadhaar_no are each candidate keys; name is not, because two students can have the same name.

(Definition 1 M, example 1 M)

d) Purpose of each section of a PL/SQL block.

2 M

- DECLARE (declaration section, optional):** declares variables, constants, cursors and user-defined exceptions used in the block.
- BEGIN (executable section, mandatory):** contains the SQL and procedural statements (assignments, IF, loops) that perform the work.

3. **EXCEPTION (exception-handling section, optional):** contains handlers that execute when an error (exception) occurs in the executable section.
4. **END;** marks the end of the block.

(2 M)

e) PL/SQL data types.

2 M

1. **NUMBER(p, s)** — numeric values (integer or decimal), e.g. salary **NUMBER(8,2)**.
2. **CHAR(n) / VARCHAR2(n)** — fixed / variable length strings.
3. **DATE** — date and time values.
4. **BOOLEAN** — TRUE, FALSE or NULL (only in PL/SQL).

Also: **INTEGER**, **BINARY_INTEGER / PLS_INTEGER**, **LONG**, **CLOB / BLOB** (LOB types), and anchored types **%TYPE** and **%ROWTYPE**.

(Any four, ½ M each)

f) Tuple and attribute.

2 M

i) Tuple: A single row of a relation (table); it represents one record / entity. Example: (101, 'Asha', 'CO') in Student.

ii) Attribute: A named column of a relation that represents a property of the entity. Example: roll_no, name, dept of Student. The set of allowed values of an attribute is its domain.

(Each 1 M)

g) Applications of triggers.

2 M

1. Enforcing complex business rules / integrity constraints, e.g. salary must not be less than 5000, no transaction on Sunday.
2. Auditing — automatically recording who changed which data and when, in an audit (log) table.
3. Automatically generating derived column values, e.g. primary key from a sequence, total = qty × rate.
4. Maintaining replicated / summary tables and preventing invalid transactions.

(Any two, 1 M each)

Q.2

a) Any four Codd's rules.

4 M

1. **Rule 6 — View updating rule:** All views that are theoretically updatable must also be updatable by the system.
2. **Rule 7 — High-level insert, update and delete:** The DBMS must support insertion, updation and deletion of a set of rows in a single operation (set-at-a-time), not only retrieval.
3. **Rule 10 — Integrity independence:** Integrity constraints (primary key, foreign key, check) must be defined in the relational language and stored in the catalog, not in application programs; changing them must not affect applications.
4. **Rule 12 — Non-subversion rule:** If the system provides a low-level (record-at-a-time) interface, it must not be possible to use it to bypass the integrity rules and security constraints.

(Other rules: 0 Foundation, 1 Information, 2 Guaranteed access, 3 Systematic NULL treatment, 4 Dynamic online catalog, 5 Comprehensive data sub-language, 8 Physical data independence, 9 Logical data independence, 11 Distribution independence.)

(Any four rules, 1 M each)

b) Physical, logical and view levels.

4 M

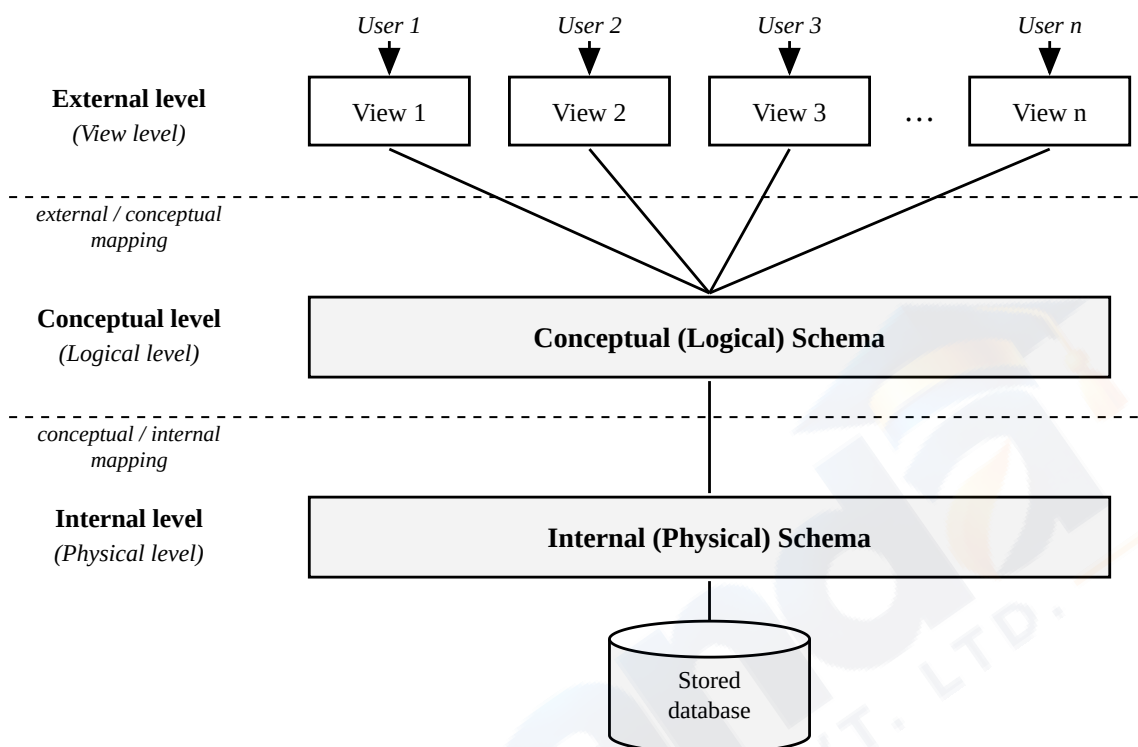


Fig.: Three-level architecture of DBMS (levels of data abstraction)

- Physical level:** The lowest level of abstraction; describes how the data is actually stored — data structures, storage blocks, indexes, hashing, compression. Example (bank): customer records stored as blocks in a data file with a B-tree index on account number.
- Logical (conceptual) level:** Describes what data is stored in the whole database and the relationships among them, in terms of tables, columns, data types and constraints. Example: Customer(cust_id, name, city), Account(acc_no, cust_id, balance).
- View level:** The highest level; each user sees only the part of the database he needs, and other details are hidden. Example: a cashier views only account number and balance, not the customer's loan details.

Mappings between levels provide physical and logical data independence.

(Diagram 2 M, explanation 2 M)

c) GROUP BY and HAVING.

4 M

GROUP BY: divides the rows of a table into groups having the same value in the given column(s); an aggregate function (SUM, AVG, COUNT, MAX, MIN) is then applied to each group, giving one result row per group.

HAVING: applies a condition to the groups formed by GROUP BY (WHERE filters rows before grouping; HAVING filters groups after grouping, and can use aggregate functions).

```
SELECT column, aggregate_function(column) FROM table
[WHERE condition]
GROUP BY column
[HAVING group_condition];
```

Sample data (assumed) — table **EMP**:

EMPNO	ENAME	JOB	SAL	DEPTNO
1	Amit	Clerk	22000	10
2	Bina	Manager	55000	10
3	Chetan	Clerk	20000	20
4	Divya	Analyst	38000	20
5	Esha	Analyst	41000	20
6	Farhan	Clerk	24000	30

Example 1 — department-wise total salary and number of employees:

```
SELECT deptno, SUM(sal), COUNT(*) FROM emp GROUP BY deptno ORDER BY deptno;
```

Output:

DEPTNO	SUM(SAL)	COUNT(*)
10	77000	2
20	99000	3
30	24000	1

Example 2 — departments whose total salary is more than 60000:

```
SELECT deptno, SUM(sal) FROM emp GROUP BY deptno HAVING SUM(sal) > 60000 ORDER BY deptno;
```

Output:

DEPTNO	SUM(SAL)
10	77000
20	99000

(GROUP BY 2 M, HAVING 2 M)

d) PL/SQL — prime number.

4 M

```
SET SERVEROUTPUT ON;
DECLARE
    n    NUMBER := &n;
    flag NUMBER := 1;
BEGIN
    IF n < 2 THEN
        flag := 0;
    ELSE
        FOR i IN 2..TRUNC(SQRT(n)) LOOP
            IF MOD(n, i) = 0 THEN
                flag := 0;
                EXIT;
            END IF;
        END LOOP;
    END IF;
END IF;
```

```

IF flag = 1 THEN
    DBMS_OUTPUT.PUT_LINE(n || ' is a prime number');
ELSE
    DBMS_OUTPUT.PUT_LINE(n || ' is not a prime number');
END IF;
END;
/

```

Output (run 1):

```

Enter value for n: 29
29 is a prime number
PL/SQL procedure successfully completed.

```

Output (run 2):

```

Enter value for n: 91
91 is not a prime number
PL/SQL procedure successfully completed.

```

The number is divided by every i from 2 to $\sqrt{n}$; if any remainder is 0 the number is not prime.

(Logic 3 M, syntax 1 M)

Q.3

a) Advantages of DBMS over file processing system.

4 M

- Controls redundancy:** data is stored once in an integrated database and shared, instead of separate copies in many files of different departments.
- Data integrity:** constraints such as primary key, NOT NULL, CHECK and foreign key are enforced by the DBMS for every application, so invalid data is not stored.
- Data independence:** the storage structure can be changed without changing application programs (three-level architecture); in a file system the programs depend on the file format.
- Concurrency control:** many users can safely update data at the same time; the DBMS prevents anomalies such as lost updates.
- Security:** access is controlled through users, passwords, privileges and views.
- Backup and recovery:** the DBMS restores the database to a consistent state after failures using logs and backups.

(Any four, 1 M each)

b) Normalization; 1NF and 2NF.

4 M

Normalization: the process of decomposing a relation into smaller relations using functional dependencies, to minimize redundancy and avoid insertion, deletion and update anomalies.

Unnormalized relation — table **STUD_UN**:

ROLL_NO	NAME	PHONES	COURSE_ID	COURSE_NAME	MARKS
1	Asha	98220, 94220	C1	DBMS	78
1	Asha	98220, 94220	C2	Java	81
2	Ravi	97300	C1	DBMS	69

1NF: A relation is in First Normal Form if every attribute contains only atomic (single, indivisible) values — no multivalued or repeating groups. Above, *phones* holds two values, so it is not in 1NF. Making each value a separate row gives the 1NF relation (key: roll_no, course_id, phone):

ROLL_NO	NAME	PHONE	COURSE_ID	COURSE_NAME	MARKS
1	Asha	94220	C1	DBMS	78
1	Asha	98220	C1	DBMS	78
1	Asha	94220	C2	Java	81
1	Asha	98220	C2	Java	81
2	Ravi	97300	C1	DBMS	69

2NF: A relation is in Second Normal Form if it is in 1NF and every non-key attribute is **fully** functionally dependent on the whole primary key (no partial dependency). Here name depends only on roll_no, course_name only on course_id, and marks on (roll_no, course_id); the multivalued phone goes to its own table. Decomposition into 2NF:

2NF — table **STUDENT**: 2NF — table **STUD_PHONE**: 2NF — table **COURSE**:

ROLL_NO	NAME
1	Asha
2	Ravi

ROLL_NO	PHONE
1	94220
1	98220
2	97300

COURSE_ID	COURSE_NAME
C1	DBMS
C2	Java

2NF — table **RESULT**:

ROLL_NO	COURSE_ID	MARKS
1	C1	78
1	C2	81
2	C1	69

(Definition 1 M, 1NF 1½ M, 2NF 1½ M)

c) Function vs procedure.

4 M

Function	Procedure
Must return exactly one value using the RETURN statement.	Does not return a value through RETURN; may return values through OUT / IN OUT parameters.
Header contains a RETURN clause: CREATE FUNCTION f(...) RETURN NUMBER IS	No RETURN clause: CREATE PROCEDURE p(...) IS
Called as part of an expression, e.g. x := f(5); or in SELECT f(5) FROM dual;	Called as a standalone statement: EXEC p(5); or p(5); inside a block.
Can be used in SQL statements (SELECT, WHERE) if it does no DML.	Cannot be called from a SELECT statement.
Mainly used to compute and return a value.	Mainly used to perform an action / business process (e.g. DML).

(Any four points, 1 M each)

d) ACID — railway seat booking.

4 M

Transaction T (book one seat for a passenger): read available seats of the train; decrease seats by 1; insert a booking record with the passenger name; deduct the fare from the passenger's wallet; COMMIT.

1. **Atomicity:** All steps happen or none. If the payment step fails, the seat count decrement and booking record are rolled back, so no seat is lost.
2. **Consistency:** Before and after T, (seats available + seats booked) equals the total seats of the coach, and no seat count becomes negative.
3. **Isolation:** If two passengers try to book the last seat at the same time, the transactions are isolated (locking), so the seat is given to only one of them; the result is as if they ran one after the other.
4. **Durability:** Once the booking is committed and the ticket (PNR) is generated, the booking survives any later system crash, because committed changes are written to the log / disk.

(Each property 1 M)

Q.4

a) Index — types with commands.

4 M

Index: An index is a database object (a sorted structure of key values with row addresses, like the index of a book) created on one or more columns of a table to speed up searching and sorting of rows. It is used automatically by the DBMS; too many indexes slow down INSERT / UPDATE.

1) Simple index — on a single column:

```
CREATE INDEX idx_name ON student(name);
```

Index created.

2) Composite index — on two or more columns (useful when they are searched together):

```
CREATE INDEX idx_dept_roll ON student(dept, rollno);
```

Index created.

3) Unique index — does not allow duplicate values in the indexed column:

```
CREATE UNIQUE INDEX idx_email ON student(email);
```

Index created.

Effect of the unique index:

```
INSERT INTO student VALUES (3, 'Neha', 'CO', 'asha@mail.com');
```

ERROR at line 1:
ORA-00001: unique constraint (IDX_EMAIL) violated

Removing an index:

```
DROP INDEX idx_name;
```

Index dropped.

(Oracle also supports bitmap and function-based indexes; a unique index is created automatically for PRIMARY KEY and UNIQUE constraints.)

(Definition 1 M, three types with commands 3 M)

b) Predefined and user-defined exceptions.

4 M

1) Predefined exceptions are named by Oracle and raised automatically when a run-time error occurs, e.g. NO_DATA_FOUND, TOO_MANY_ROWS, ZERO_DIVIDE, VALUE_ERROR, INVALID_NUMBER, DUP_VAL_ON_INDEX, CURSOR_ALREADY_OPEN. Example (Emp table has three employees in dept 20):

```
DECLARE
    v_name emp.ename%TYPE;
BEGIN
    SELECT ename INTO v_name FROM emp WHERE deptno = 20;
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Query returned more than one employee');
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee found');
END;
/
```

Output:

Query returned more than one employee
PL/SQL procedure successfully completed.

2) User-defined exceptions are declared by the programmer (name EXCEPTION;), raised explicitly with RAISE name; when a business rule is violated, and handled in the EXCEPTION section.

```
DECLARE
    age NUMBER := &age;
    under_age EXCEPTION;
BEGIN
    IF age < 18 THEN
        RAISE under_age;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Eligible to vote');
EXCEPTION
    WHEN under_age THEN
        DBMS_OUTPUT.PUT_LINE('Not eligible: age is below 18');
END;
/
```

Output (run 1):

Enter value for age: 16
Not eligible: age is below 18
PL/SQL procedure successfully completed.

Output (run 2):

```
Enter value for age: 21
Eligible to vote
PL/SQL procedure successfully completed.
```

(Each type with example 2 M)

c) SQL queries on Course table.

4 M

Sample data (assumed) — table **COURSE**:

CID	CNAME	DURATION	FEES	FACULTY
1	Python	3	4500	Patil
2	Full Stack	9	12000	Kulkarni
3	Oracle SQL	4	3800	Patil
4	Data Science	8	7500	Shaikh
5	Tally	2	2500	Joshi

i)

```
SELECT * FROM course WHERE faculty = 'Patil';
```

Output:

CID	CNAME	DURATION	FEES	FACULTY
1	Python	3	4500	Patil
3	Oracle SQL	4	3800	Patil

ii)

```
SELECT cname FROM course WHERE fees < 5000;
```

Output:

CNAME
Python
Oracle SQL
Tally

iii)

```
UPDATE course SET fees = fees + 500 WHERE duration > 6;
```

```
2 rows updated.
```

iv)

```
SELECT * FROM course WHERE fees BETWEEN 3000 AND 8000 ORDER BY cid;
```

Output:

CID	CNAME	DURATION	FEES	FACULTY
1	Python	3	4500	Patil
3	Oracle SQL	4	3800	Patil
4	Data Science	8	8000	Shaikh

(In iv) Data Science now has fees 8000 after the update in iii.)

(Each query 1 M)

d) User, GRANT and REVOKE.

4 M

i)

```
CREATE USER lab_user IDENTIFIED BY lab2026;
```

User created.

ii)

```
GRANT CREATE SESSION TO lab_user;
```

Grant succeeded.

iii)

```
GRANT SELECT, UPDATE, DELETE ON student TO lab_user;
```

Grant succeeded.

iv)

```
REVOKE DELETE ON student FROM lab_user;
```

Revoke succeeded.

(Each command 1 M)

e) Subquery with examples.

4 M

A **subquery** (inner / nested query) is a SELECT statement written inside another SQL statement (in WHERE, HAVING or FROM clause). The inner query executes first and its result is used by the outer query.

Types: **single-row subquery** (returns one row; used with =, >, <), **multiple-row subquery** (returns many rows; used with IN, ANY, ALL), and **correlated subquery** (inner query refers to a column of the outer query and runs once per outer row).

1) Single-row — employees earning more than the average salary:

```
SELECT ename, sal FROM emp WHERE sal > (SELECT AVG(sal) FROM emp);
```

Output:

ENAME	SAL
Bina	55000
Divya	38000
Esha	41000

2) Multiple-row (IN) — employees working in the departments where some Analyst works:

```
SELECT ename, deptno FROM emp WHERE deptno IN (SELECT deptno FROM emp WHERE job = 'Analyst');
```

Output:

ENAME	DEPTNO
Chetan	20
Divya	20
Esha	20

3) Correlated — highest-paid employee of each department:

```
SELECT ename, sal, deptno FROM emp e WHERE sal = (SELECT MAX(sal) FROM emp WHERE deptno = e.deptno);
```

Output:

ENAME	SAL	DEPTNO
Bina	55000	10
Esha	41000	20
Farhan	24000	30

(Emp data as in Q.2 c; average salary = 33333.33.)

(Definition 1 M, examples 3 M)

Q.5

a) Overall structure of DBMS.

6 M

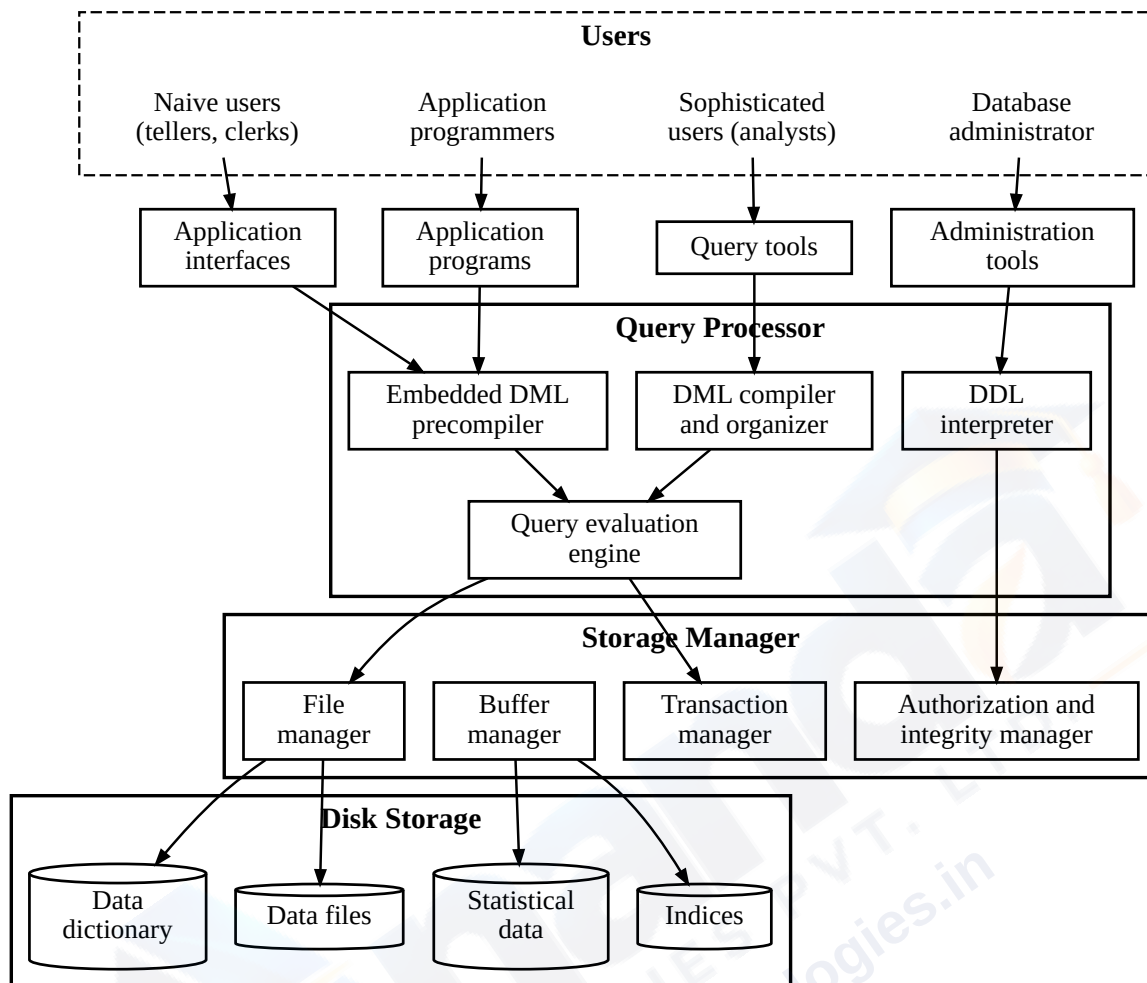


Fig.: Overall structure of DBMS

1) Query processor: converts user requests into low-level instructions that the storage manager understands.

- **DDL interpreter:** interprets DDL statements and records the definitions (metadata) in the data dictionary.
- **DML compiler:** translates DML statements of a query language into an evaluation plan of low-level instructions; it also performs *query optimization* — choosing the lowest-cost plan.
- **Embedded DML pre-compiler:** converts DML statements embedded in an application program into normal procedure calls of the host language.
- **Query evaluation engine:** executes the low-level instructions generated by the DML compiler.

2) Storage manager: the interface between the low-level data stored on disk and the application programs / queries; responsible for storing, retrieving and updating data.

- **Authorization and integrity manager:** checks the authority of users to access data and tests integrity constraints.
- **Transaction manager:** keeps the database consistent despite system failures, and ensures that concurrent transactions run without conflicts.

- **File manager:** manages allocation of disk space and the data structures used to represent data on disk.
- **Buffer manager:** fetches data from disk into main memory and decides what data to cache.

Disk storage holds data files, the data dictionary (metadata), indices and statistical data.

(Diagram 2 M, query processor 2 M, storage manager 2 M)

b) E-R diagram — Banking System.

6 M

Assumptions: a customer may hold many accounts and an account may be joint (M : N); a branch maintains many accounts and every account belongs to a branch (1 : N, total); a transaction (deposit / withdrawal) cannot exist without its account, so Transaction is a weak entity identified by acc_no + txn_no.

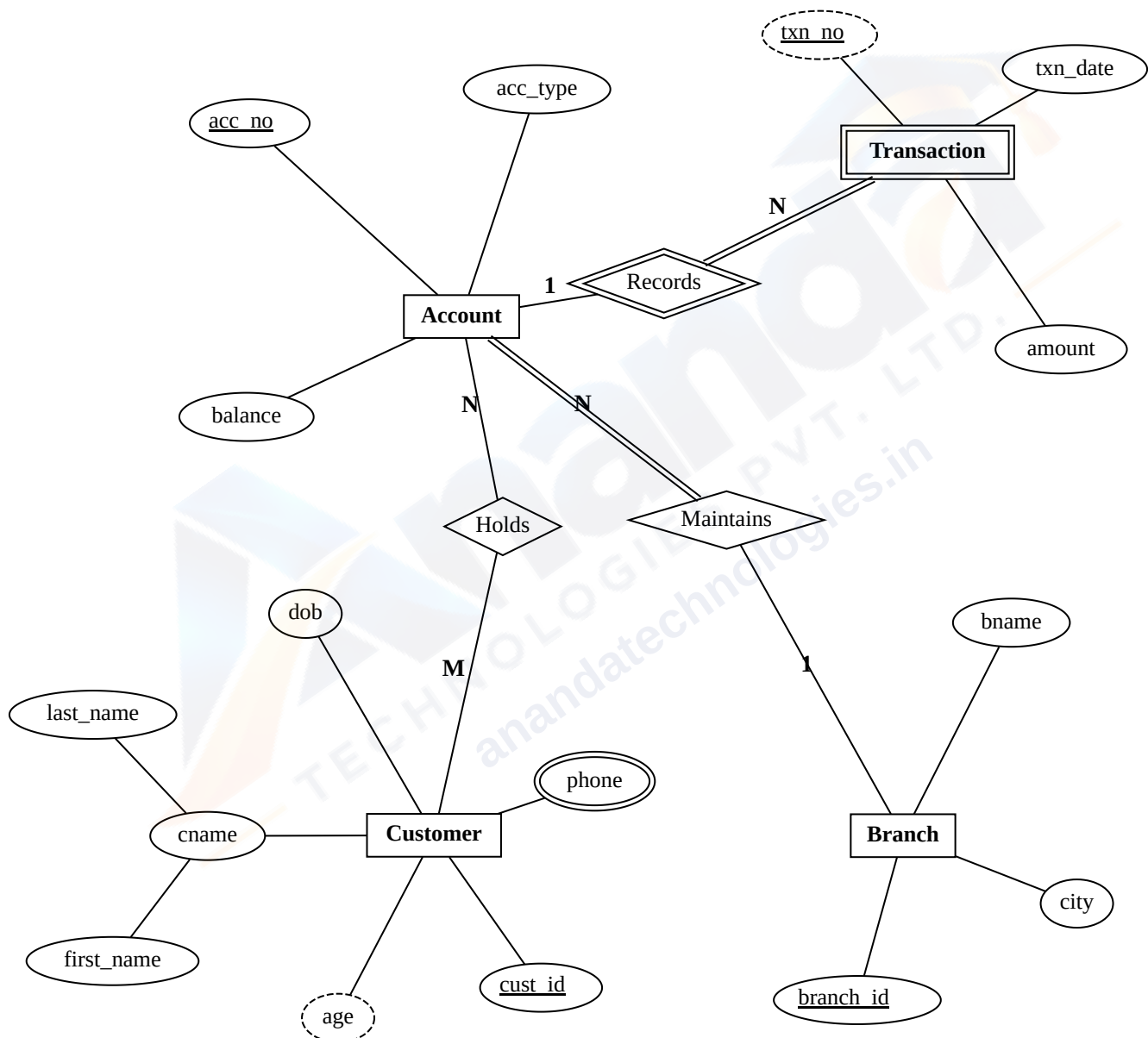


Fig.: E-R diagram for Banking System

Notation: double rectangle — weak entity (Transaction); double diamond — identifying relationship (Records); dashed-underlined txn_no — partial key; cname — composite; phone — multivalued; age — derived from dob; double line — total participation; 1, M, N — cardinality ratios.

(Entities 2 M, relationships with cardinality 2 M, attribute types and weak entity 2 M)

c) Explicit cursor — employees of department 20.

6 M

Emp table data is the same as in Q.2 c.

```
SET SERVEROUTPUT ON;
DECLARE
    CURSOR c20 IS
        SELECT empno, ename, sal FROM emp WHERE deptno = 20;
    v_no    emp.empno%TYPE;
    v_name  emp.ename%TYPE;
    v_sal   emp.sal%TYPE;
    v_total NUMBER := 0;
BEGIN
    OPEN c20;
    DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME      SALARY');
    LOOP
        FETCH c20 INTO v_no, v_name, v_sal;
        EXIT WHEN c20%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(RPAD(v_no, 7) || RPAD(v_name, 10) || v_sal);
        v_total := v_total + v_sal;
    END LOOP;
    CLOSE c20;
    DBMS_OUTPUT.PUT_LINE('Total salary of dept 20 = ' || v_total);
END;
/
```

Output:

```
EMPNO  ENAME      SALARY
3      Chetan    20000
4      Divya     38000
5      Esha      41000
Total salary of dept 20 = 99000
PL/SQL procedure successfully completed.
```

Steps: the cursor c20 is declared for the query; OPEN executes the query; FETCH reads one row at a time into variables inside a loop, which ends when c20%NOTFOUND is TRUE; the salary is added to v_total; CLOSE releases the cursor memory.

(Cursor declaration 1 M, open/fetch loop 3 M, total 1 M, close 1 M)

Q.6

a) Sequence roll_seq.

6 M

Create the sequence:

```
CREATE SEQUENCE roll_seq
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 60
NOCYCLE;
```

Sequence created.

Insert three records:

```
INSERT INTO student VALUES (roll_seq.NEXTVAL, 'Aarti');
```

1 row created.

```
INSERT INTO student VALUES (roll_seq.NEXTVAL, 'Bhushan');
```

1 row created.

```
INSERT INTO student VALUES (roll_seq.NEXTVAL, 'Chaitali');
```

1 row created.

```
SELECT * FROM student;
```

Output:

ROLLNO	NAME
1	Aarti
2	Bhushan
3	Chaitali

Display the current value:

```
SELECT roll_seq.CURRVAL FROM dual;
```

Output:

CURRVAL
3

Change maximum value to 120:

```
ALTER SEQUENCE roll_seq MAXVALUE 120;
```

Sequence altered.

Remove the sequence:

```
DROP SEQUENCE roll_seq;
```

Sequence dropped.

NEXTVAL increments the sequence and returns the new value; CURRVAL returns the value last generated in the current session. NOCYCLE means that after 60 (now 120) no more numbers are generated.

(Create 2 M, insert 2 M, currval 1 M, alter / drop 1 M)

b) Roll forward and roll backward recovery.

6 M

Recovery brings the database back to a consistent state after a failure. The DBMS keeps a **log (journal)** on stable storage that records every change: <T start>, <T, item, old value, new value>, <T commit>. A **checkpoint** is taken periodically, when all modified buffers are written to disk.

1) Roll forward (REDO): The changes of transactions that had **committed** (their <T commit> record is in the log) but whose updates may not have reached the disk are re-applied using the *new*

values from the log. It is also used after media failure: the last backup is restored and the archived logs are applied forward to bring it up to date.

2) Roll backward (UNDO): The changes made by transactions that were **active (not committed)** at the time of failure are undone using the *old values* from the log, moving backward through the log, so that no partial transaction remains in the database.

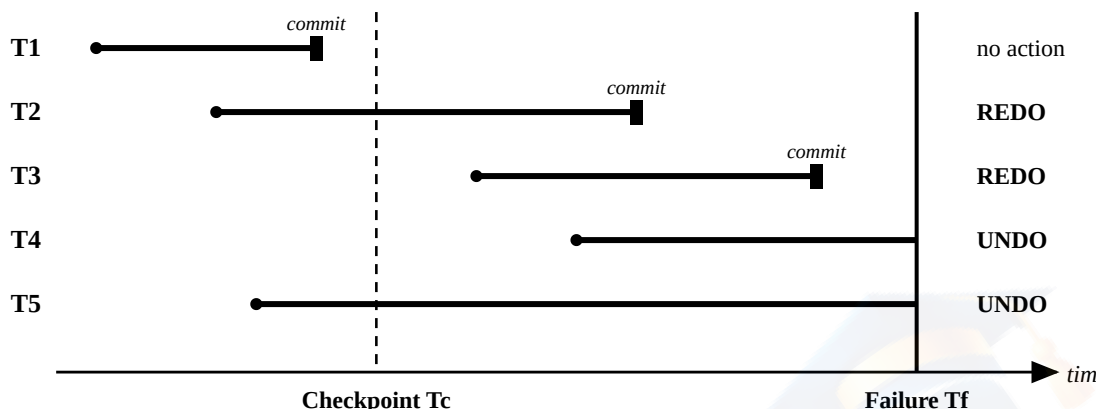


Fig.: Transactions at checkpoint (Tc) and failure (Tf)

Example: T1 committed before the checkpoint — its changes are already on disk, no action. T2 and T3 committed after the checkpoint but before the failure — **roll forward (REDO)**. T4 and T5 were still running at failure — **roll backward (UNDO)**.

Log: <T2 start> <T2, A, 5000, 4000> <T2 commit> ... <T4 start> <T4, B, 3000, 3500> --- crash ---
Recovery: REDO T2 → A = 4000 (new value); UNDO T4 → B = 3000 (old value)

(Log / checkpoint 1 M, roll forward 2 M, roll backward 2 M, example 1 M)

c) Customer and Orders — FK, total and HAVING.

6 M

i) Create tables:

```
CREATE TABLE customer (
  cid  NUMBER(4) CONSTRAINT pk_cust PRIMARY KEY,
  cname VARCHAR2(15),
  city  VARCHAR2(12) );
```

Table created.

```
CREATE TABLE orders (
  oid  NUMBER(5) CONSTRAINT pk_orders PRIMARY KEY,
  cid  NUMBER(4) CONSTRAINT fk_cust REFERENCES customer(cid),
  amount NUMBER(8) );
```

Table created.

Sample data inserted:

Data — table **CUSTOMER**:

CID	CNAME	CITY
1	Ganesh	Pune
2	Shilpa	Nashik
3	Irfan	Aurangabad

Data — table **ORDERS**:

OID	CID	AMOUNT
501	1	4500
502	2	12500
503	1	7200
504	3	3100
505	3	2600

ii) Total order amount of each customer:

```
SELECT c.cid, c.cname, SUM(o.amount) AS total_amount
FROM customer c JOIN orders o ON c.cid = o.cid
GROUP BY c.cid, c.cname
ORDER BY c.cid;
```

Output:

CID	CNAME	TOTAL_AMOUNT
1	Ganesh	11700
2	Shilpa	12500
3	Irfan	5700

iii) Customers whose total order amount is more than 10000:

```
SELECT c.cname
FROM customer c JOIN orders o ON c.cid = o.cid
GROUP BY c.cname
HAVING SUM(o.amount) > 10000;
```

Output:

CNAME
Ganesh
Shilpa

(Tables 2 M, query ii 2 M, query iii 2 M)

Note: This is a **model question paper** prepared by **Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in