

Database Management System (313302)

Model Question Paper — Set A · 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) Define: Database, DBMS.

2 M

i) Database: A database is an organized collection of logically related data (facts that can be recorded) stored together so that it can be easily accessed, managed and updated. Example: a college database storing students, courses and results.

ii) DBMS (Database Management System): A DBMS is a collection of programs (software) that enables users to create, store, retrieve, update and manage the data in a database in a secure and efficient way. Examples: Oracle, MySQL, MS SQL Server, PostgreSQL.

(Each definition 1 M)

b) Hierarchical vs network data model.

2 M

Hierarchical data model	Network data model
Data is organized as a tree; each child record has only one parent.	Data is organized as a graph; a child record can have many parents (owners).
Represents one-to-many (1:N) relationships only.	Represents one-to-many and many-to-many (M:N) relationships.
Example: IBM IMS.	Example: IDMS (CODASYL DBTG model).

(Any two points, 1 M each)

c) PL/SQL block structure.

2 M

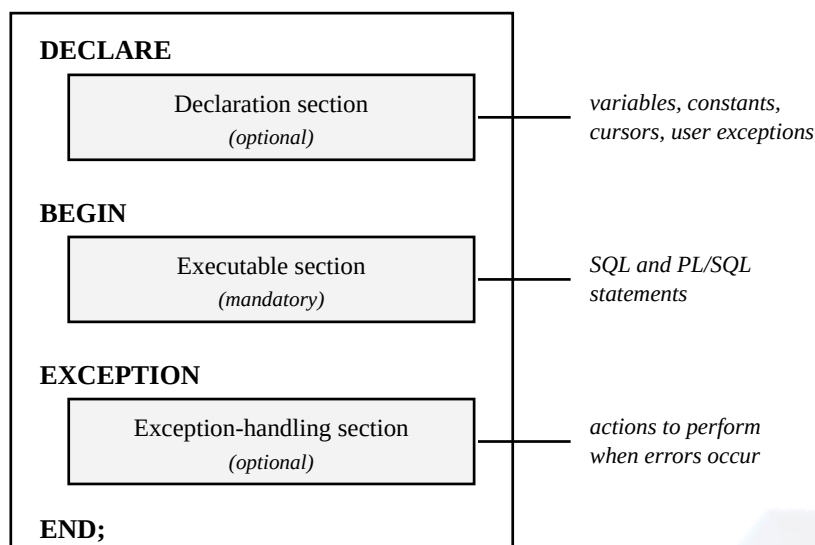


Fig.: PL/SQL block structure

```

DECLARE
    -- declarations (optional)
BEGIN
    -- executable statements (mandatory)
EXCEPTION
    -- exception handlers (optional)
END;
  
```

(Diagram / structure 2 M)

d) View — definition and syntax.

2 M

View: A view is a virtual (logical) table based on the result of a SELECT query on one or more tables. It does not store data itself; data is taken from the base tables whenever the view is used.

Syntax:

```

CREATE [OR REPLACE] VIEW view_name AS
    SELECT column1, column2, ...
    FROM table_name
    [WHERE condition];
  
```

(Definition 1 M, syntax 1 M)

e) Four DDL commands.

2 M

DDL (Data Definition Language) commands define or change the structure of database objects:

1. **CREATE** — creates a table, view, index, sequence etc.
2. **ALTER** — changes the structure of an existing table.
3. **DROP** — deletes the table (structure + data).
4. **TRUNCATE** — deletes all rows of a table, keeping its structure.

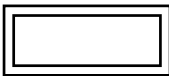
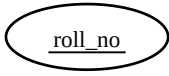
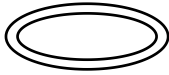
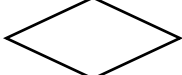
(Also: RENAME, COMMENT.)

(½ M each)

f) E-R diagram symbols.

2 M

Symbol	Name	Meaning
--------	------	---------

	Rectangle	Entity set (strong entity)
	Double rectangle	Weak entity set
	Ellipse	Attribute
	Ellipse with underlined name	Key (primary key) attribute
	Double ellipse	Multivalued attribute
	Diamond	Relationship set

(Any four symbols with meaning, ½ M each)

g) Characteristics of Big Data.

2 M

1. **Volume:** Huge amount of data (terabytes to petabytes) is generated and stored.
2. **Velocity:** Data is generated and must be processed at very high speed (e.g. social media, sensors).
3. **Variety:** Data comes in many forms — structured (tables), semi-structured (XML, JSON) and unstructured (images, videos, text).
4. **Veracity:** Uncertainty / quality of data; data may be inaccurate or incomplete.

(Also **Value** — useful information extracted from the data.)

(Any four, ½ M each)

Q.2

a) Any four Codd's rules.

4 M

Dr. E. F. Codd gave 13 rules (Rule 0 to Rule 12) which a DBMS must satisfy to be called a fully relational DBMS.

1. **Rule 1 — Information rule:** All information in a relational database is represented explicitly in only one way — as values in tables (rows and columns).
2. **Rule 2 — Guaranteed access rule:** Every single data value must be accessible logically by using the combination of *table name* + *primary key value* + *column name*.
3. **Rule 3 — Systematic treatment of NULL values:** NULL values (missing / not applicable information) must be supported and handled in a uniform way, independent of data type. NULL is different from zero or blank.
4. **Rule 4 — Dynamic online catalog:** The description of the database (data dictionary / catalog) must itself be stored as tables, so authorized users can query it using the same relational language.

(Other rules: Rule 0 Foundation rule, 5 Comprehensive data sub-language, 6 View updating, 7 High-level insert/update/delete, 8 Physical data independence, 9 Logical data independence, 10 Integrity independence, 11 Distribution independence, 12 Non-subversion.)

(Any four rules, 1 M each)

b) Three levels of data abstraction.

4 M

Data abstraction means hiding the complex details of how data is stored and showing users only what they need. DBMS provides three levels (ANSI-SPARC architecture):

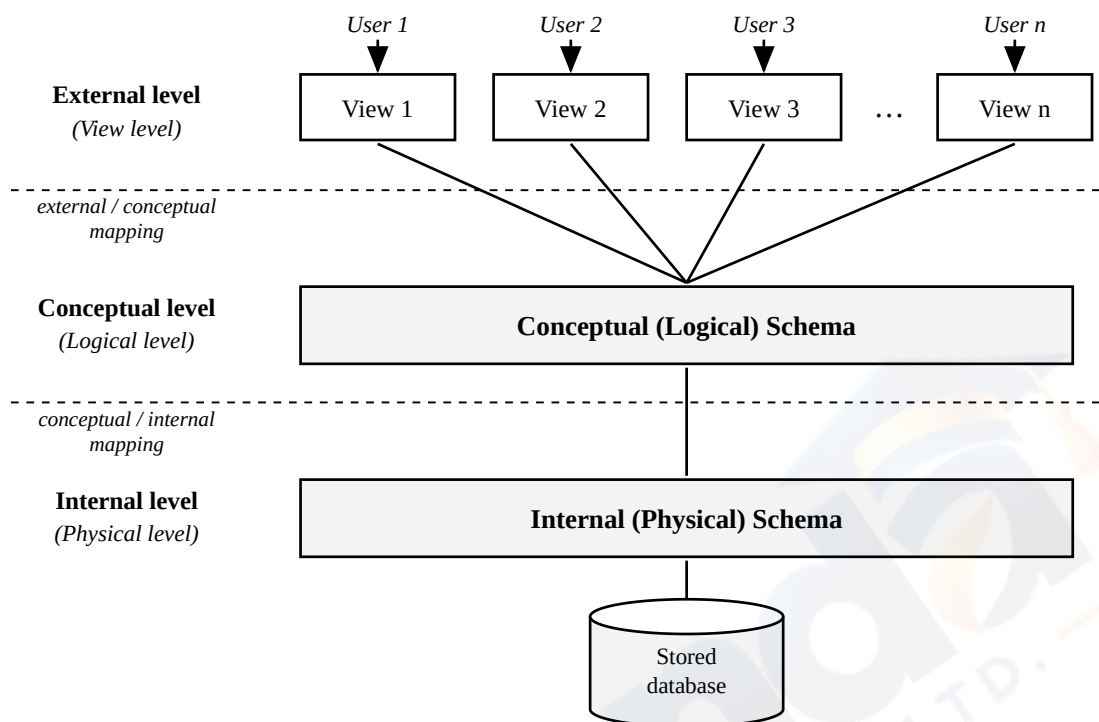


Fig.: Three-level architecture of DBMS (levels of data abstraction)

- Physical (internal) level:** Lowest level. Describes *how* data is actually stored on disk — file organization, records, indexes, blocks, access paths. Handled by DBA / system.
- Logical (conceptual) level:** Describes *what* data is stored and the relationships among the data — all tables, attributes, data types, constraints. Used by DBA and programmers. Example: Student(rollno, name, dept, marks).
- View (external) level:** Highest level. Describes only the part of the database that a particular user needs. Many views can exist for the same database. Example: an accounts clerk sees only fee details of students.

(Diagram 2 M, explanation 2 M)

c) PL/SQL — multiplication table.

4 M

```
SET SERVEROUTPUT ON;
DECLARE
    n NUMBER := &n;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Multiplication table of ' || n);
    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE(n || ' x ' || i || ' = ' || n * i);
    END LOOP;
END;
/
```

Output:

```
Enter value for n: 7
Multiplication table of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
PL/SQL procedure successfully completed.
```

(Correct logic 3 M, syntax 1 M)

d) SQL queries on Student table.

4 M

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

ROLLNO	NAME	DEPT	MARKS	CITY
1	Amit	CO	78	Nashik
2	Sneha	IT	85	Pune
3	Rahul	CO	91	Nagpur
4	Priya	ME	67	Nashik
5	Karan	CO	74	Mumbai
6	Pooja	IT	88	Nagpur
7	Vikas	EJ	59	Pune

i) Name and marks of CO students:

```
SELECT name, marks FROM student WHERE dept = 'CO';
```

Output:

NAME	MARKS
Amit	78
Rahul	91
Karan	74

ii) Students staying in Nashik or Nagpur:

```
SELECT * FROM student WHERE city IN ('Nashik', 'Nagpur');
```

Output:

ROLLNO	NAME	DEPT	MARKS	CITY
1	Amit	CO	78	Nashik
3	Rahul	CO	91	Nagpur
4	Priya	ME	67	Nashik
6	Pooja	IT	88	Nagpur

iii) Change marks of roll number 5 to 82:

```
UPDATE student SET marks = 82 WHERE rollno = 5;
```

```
1 row updated.
```

```
SELECT * FROM student WHERE rollno = 5;
```

Output:

ROLLNO	NAME	DEPT	MARKS	CITY
5	Karan	CO	82	Mumbai

iv) Students having marks greater than Sneha (sub-query):

```
SELECT name FROM student WHERE marks > (SELECT marks FROM student WHERE name = 'Sneha');
```

Output:

NAME
Rahul
Pooja

(Each query 1 M)

Q.3

a) Advantages of DBMS over file processing system.

4 M

- Reduced data redundancy:** In a file system the same data (e.g. student address) is repeated in many files. DBMS stores data in an integrated way, so duplication is controlled.
- Data consistency:** Because redundancy is reduced, an update is made at one place, so the problem of different copies having different values (inconsistency) is avoided.
- Data sharing and concurrent access:** Many users and applications can access the same data at the same time; DBMS controls concurrent updates so data remains correct.
- Data security:** DBMS allows only authorized users to access data, using user names, passwords and privileges (GRANT / REVOKE). File systems give little security.
- Integrity constraints:** Rules such as PRIMARY KEY, NOT NULL, CHECK and FOREIGN KEY are enforced automatically by the DBMS.
- Backup and recovery:** DBMS provides automatic backup and recovery from failures; in a file system it must be done manually.
- Easy data access:** Data can be retrieved with a query language (SQL) without writing a new program for every request.

(Any four advantages, 1 M each)

b) ACID properties with example.

4 M

A **transaction** is a logical unit of work (set of operations) that must be completed fully or not at all.

Example — transfer ₹1000 from account A (balance 5000) to account B (balance 3000):

```
T: read(A); A := A - 1000; write(A);
   read(B); B := B + 1000; write(B); COMMIT;
```

- Atomicity:** All operations of the transaction are done, or none is done. If the system fails after write(A) but before write(B), the changes to A are rolled back. (Managed by the transaction /

recovery manager.)

2. **Consistency:** A transaction takes the database from one consistent state to another. Here $A + B = 8000$ before and $4000 + 4000 = 8000$ after the transfer.
3. **Isolation:** When transactions run concurrently, each must behave as if it is running alone; intermediate results (A already debited, B not credited) are not visible to other transactions. (Concurrency-control manager.)
4. **Durability:** Once the transaction commits, its changes ($A = 4000$, $B = 4000$) are permanent even if the system crashes afterwards. (Recovery manager using logs.)

(Each property 1 M)

c) Normalize Orders up to 3NF.

4 M

Functional dependencies: $\text{order_no} \rightarrow \text{order_date}, \text{cust_id}$; $\text{cust_id} \rightarrow \text{cust_name}, \text{cust_city}$; $\text{item_code} \rightarrow \text{item_name}, \text{rate}$; $(\text{order_no}, \text{item_code}) \rightarrow \text{qty}$.

Given relation with sample data — table **ORDERS_UN**:

ORDER_NO	ORDER_DATE	CUST_ID	CUST_NAME	CUST_CITY	ITEM_CODE	ITEM_NAME	RATE	QTY
1001	02-AUG-26	11	Ravi	Pune	I1	Pen	10	5
1001	02-AUG-26	11	Ravi	Pune	I2	Notebook	40	2
1002	05-AUG-26	12	Meena	Nashik	I1	Pen	10	10
1003	06-AUG-26	11	Ravi	Pune	I3	Marker	25	4

1NF: All attributes are atomic (single-valued), so the relation is in 1NF. But data is repeated (Ravi, Pune and Pen, 10 appear many times).

2NF (remove partial dependencies on part of the composite key):

$\text{order_date}, \text{cust_id}, \text{cust_name}, \text{cust_city}$ depend only on order_no ; $\text{item_name}, \text{rate}$ depend only on item_code . So decompose into:

ORDERS (order_no, order_date, cust_id, cust_name, cust_city)

ITEM (item_code, item_name, rate)

ORDER_ITEM (order_no, item_code, qty)

3NF (remove transitive dependency): in ORDERS, $\text{order_no} \rightarrow \text{cust_id} \rightarrow \text{cust_name}, \text{cust_city}$. So split ORDERS into ORDERS (order_no, order_date, cust_id) and CUSTOMER (cust_id, cust_name, cust_city).

Final 3NF relations (sample data after decomposition):

3NF — table **CUSTOMER**:

CUST_ID	CUST_NAME	CUST_CITY
11	Ravi	Pune
12	Meena	Nashik

3NF — table **ORDERS**:

ORDER_NO	ORDER_DATE	CUST_ID
1001	02-AUG-26	11
1002	05-AUG-26	12
1003	06-AUG-26	11

3NF — table **ITEM**:

ITEM_CODE	ITEM_NAME	RATE
I1	Pen	10
I2	Notebook	40
I3	Marker	25

3NF — table **ORDER_ITEM**:

ORDER_NO	ITEM_CODE	QTY
1001	I1	5
1001	I2	2
1002	I1	10
1003	I3	4

Joining the four tables gives back exactly the original rows (checked with a MINUS query — no rows selected), so the decomposition is lossless.

(1NF/2NF 2 M, 3NF 2 M)

d) Cursor and steps of explicit cursor.

4 M

Cursor: A cursor is a pointer to a private (context) memory area where Oracle stores the result set of a SELECT statement, so that a PL/SQL program can process the rows one at a time. Types: **implicit cursor** (created automatically for DML and SELECT INTO) and **explicit cursor** (declared by the programmer for a query returning many rows).

Steps to use an explicit cursor:

1. **Declare** the cursor in the declaration section:

```
CURSOR cursor_name IS SELECT_statement;
```

2. **Open** the cursor — executes the query and sets the pointer before the first row:

```
OPEN cursor_name;
```

3. **Fetch** rows one by one into variables (in a loop):

```
FETCH cursor_name INTO variable1, variable2;  
EXIT WHEN cursor_name%NOTFOUND;
```

4. **Close** the cursor to release the memory:

```
CLOSE cursor_name;
```

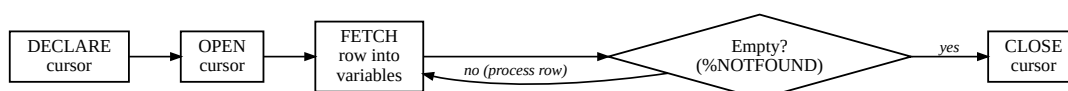


Fig.: Steps to use an explicit cursor

(Definition 1 M, steps with syntax 3 M)

Q.4

a) Two-tier and three-tier architecture.

4 M

Two-tier architecture: The client (tier 1) directly communicates with the database server (tier 2). The user interface and application programs run on the client machine; they send queries to the DBMS on the server using APIs such as ODBC / JDBC.

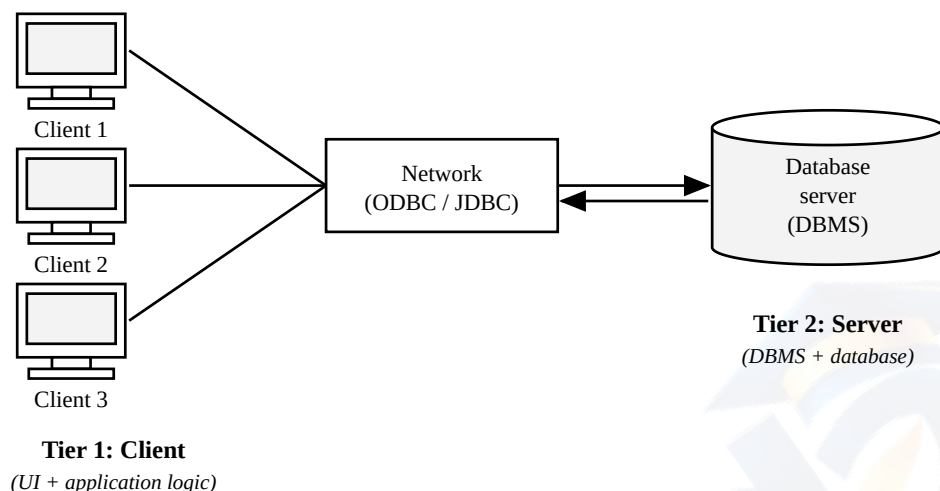


Fig. (a): Two-tier client-server architecture

- Simple and fast for a small number of users (e.g. a desktop application in an office LAN).
- Less scalable and less secure, because every client connects directly to the database.

Three-tier architecture: An application server (middle tier) is placed between the client and the database server. The client only has the user interface (e.g. a web browser); the business logic runs on the application server, which communicates with the database.

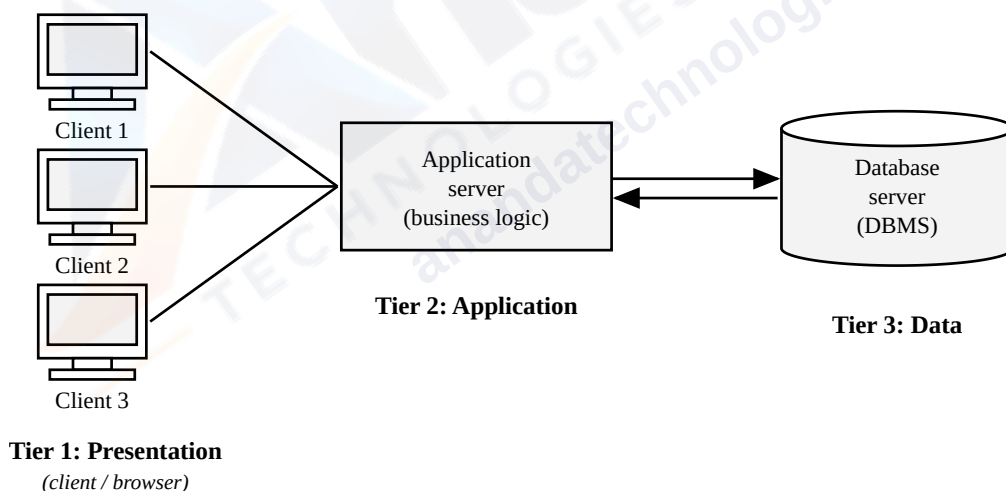


Fig. (b): Three-tier client-server architecture

- Better security (clients never access the database directly), scalability and easy maintenance.
- Used in web applications (browser → web/application server → database).

(Each architecture with diagram 2 M)

b) Types of exceptions in PL/SQL.

An **exception** is an error condition that occurs during execution of a PL/SQL block. When it occurs, control passes to the EXCEPTION section. Types:

1) Predefined (named system) exceptions: Raised automatically by Oracle; they already have names.

Exception	Raised when
NO_DATA_FOUND	SELECT INTO returns no row
TOO_MANY_ROWS	SELECT INTO returns more than one row
ZERO_DIVIDE	a number is divided by zero
VALUE_ERROR	size / conversion error in assignment
DUP_VAL_ON_INDEX	duplicate value inserted in a unique column
INVALID_CURSOR	illegal cursor operation

```

DECLARE
    a NUMBER := 50;
    b NUMBER := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Result = ' || a / b);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Error: division by zero');
END;
/

```

Output:

```

Error: division by zero
PL/SQL procedure successfully completed.

```

2) User-defined exceptions: Declared by the programmer in the DECLARE section, raised explicitly with RAISE and handled in the EXCEPTION section.

```

DECLARE
    marks NUMBER := 120;
    invalid_marks EXCEPTION;
BEGIN
    IF marks > 100 THEN
        RAISE invalid_marks;
    END IF;
    DBMS_OUTPUT.PUT_LINE('Marks = ' || marks);
EXCEPTION
    WHEN invalid_marks THEN
        DBMS_OUTPUT.PUT_LINE('Marks cannot be more than 100');
END;
/

```

Output:

```

Marks cannot be more than 100
PL/SQL procedure successfully completed.

```

(Unnamed system exceptions can be given a name using PRAGMA EXCEPTION_INIT, and RAISE_APPLICATION_ERROR raises a user error with a number from -20000 to -20999.)

(Each type with example 2 M)

c) Aggregate functions.

Aggregate (group) functions work on a set of rows and return a single value. NULL values are ignored (except in COUNT(*)).

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

EMPNO	ENAME	JOB	SAL	DEPTNO
101	Ajay	Clerk	25000	10
102	Seema	Manager	40000	20
103	Rohan	Analyst	32000	10
104	Kavita	Clerk	18000	30
105	Nikhil	Analyst	35000	20

1) COUNT() — counts rows / non-NULL values:

```
SELECT COUNT(*) FROM emp;
```

Output:

COUNT (*)
5

2) SUM() — total of a numeric column:

```
SELECT SUM(sal) FROM emp;
```

Output:

SUM(SAL)
150000

3) AVG() — average of a numeric column:

```
SELECT AVG(sal) FROM emp;
```

Output:

AVG(SAL)
30000

4) MAX() and MIN() — highest and lowest value:

```
SELECT MAX(sal), MIN(sal) FROM emp;
```

Output:

MAX(SAL)	MIN(SAL)
40000	18000

(Any four functions with example, 1 M each)

d) User creation, GRANT and REVOKE.

i) Create user clerk1:

```
CREATE USER clerk1 IDENTIFIED BY clerk123;
```

User created.

ii) Grant create session:

```
GRANT CREATE SESSION TO clerk1;
```

Grant succeeded.

iii) Grant select and insert on Product:

```
GRANT SELECT, INSERT ON product TO clerk1;
```

Grant succeeded.

iv) Revoke insert on Product:

```
REVOKE INSERT ON product FROM clerk1;
```

Revoke succeeded.

(Each command 1 M)

e) Physical backup vs logical backup.

4 M

Physical backup	Logical backup
Copy of the physical database files — data files, control files, redo / archived log files.	Copy of logical objects — tables, schemas, views, procedures, data.
Taken at operating-system level or with tools like Oracle RMAN (cold or hot backup).	Taken using export utilities such as Oracle exp / expdp (Data Pump), mysqldump.
Restores the whole database (or tablespace) to a point in time, using logs.	Can restore selected tables / schemas using import (imp / impdp).
Faster for large databases; backup files are in binary format.	Slower for large databases; output is a dump / SQL script.
Platform dependent — restored on the same type of system.	Portable across platforms and DBMS versions.
Main method for complete recovery after media failure.	Used for moving data, and as an extra protection.

(Any four points, 1 M each)

Q.5

a) E-R diagram — Library Management System.

6 M

Entities: Member, Book, Publisher, Staff. **Relationships:** Member *Borrows* Book (M : N, with issue_date and return_date), Publisher *Publishes* Book (1 : N, every book must have a publisher — total participation), Staff *Manages* Book (1 : N).

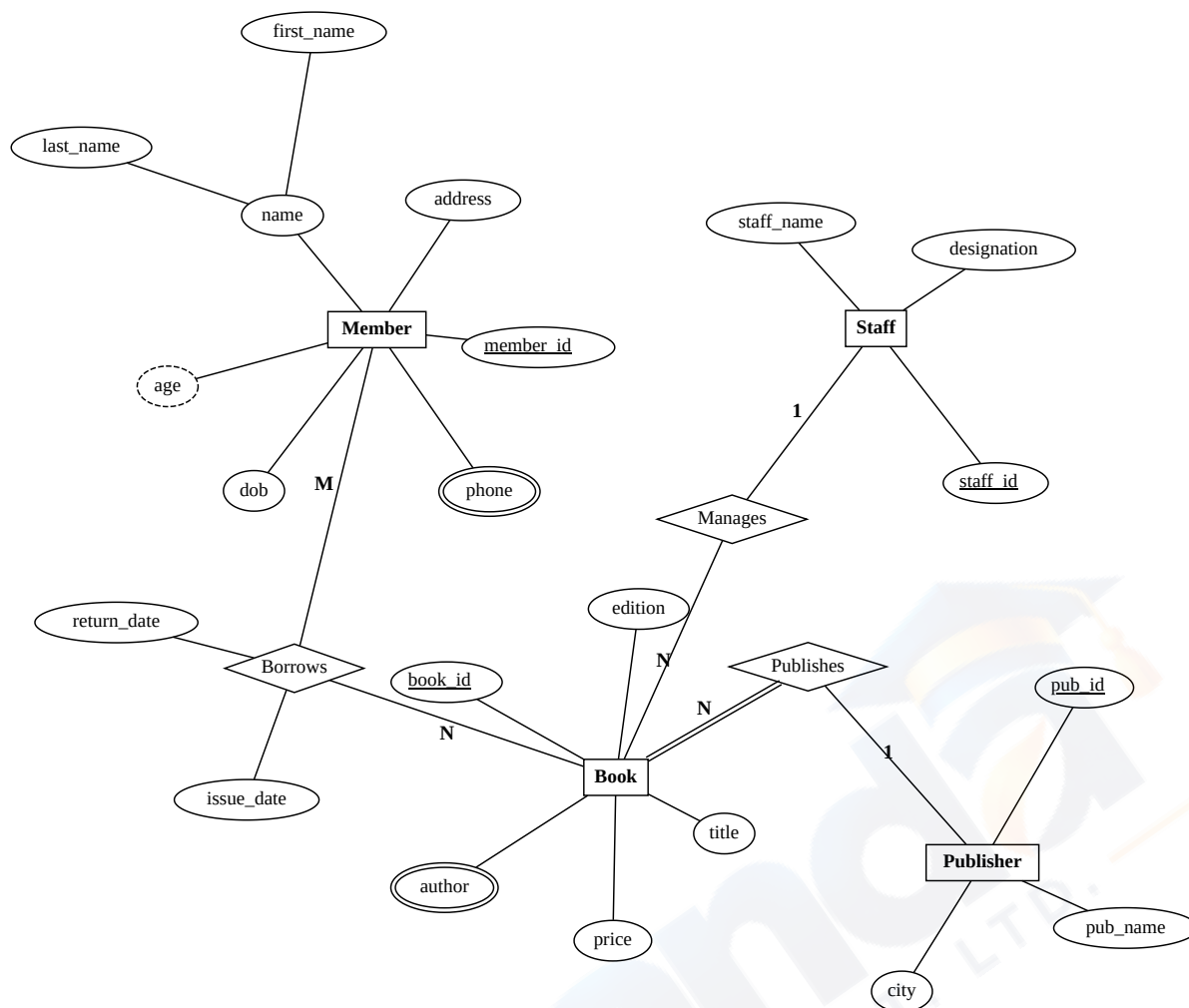


Fig.: E-R diagram for Library Management System

Notation: rectangles = entities, ellipses = attributes, underlined = key attributes (member_id, book_id, pub_id, staff_id), name = composite attribute (first_name, last_name), double ellipse = multivalued (phone, author), dashed ellipse = derived (age from dob), diamonds = relationships, double line = total participation, 1 / M / N = cardinality.

(Entities 2 M, relationships with cardinality 2 M, attribute types 2 M)

b) PL/SQL — employee details with exception.

6 M

Table used: Emp (empno, ename, job, sal, deptno) with the same sample data as in Q.4 c.

```
SET SERVEROUTPUT ON;
DECLARE
    v_no    emp.empno%TYPE := &v_no;
    v_name  emp.ename%TYPE;
    v_sal   emp.sal%TYPE;
BEGIN
    SELECT ename, sal INTO v_name, v_sal
    FROM emp WHERE empno = v_no;
    DBMS_OUTPUT.PUT_LINE('Employee name : ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Salary       : ' || v_sal);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || v_no || ' does not exist');
END;
/
```

Output (run 1):

```
Enter value for v_no: 103
Employee name : Rohan
Salary       : 32000
PL/SQL procedure successfully completed.
```

Output (run 2):

```
Enter value for v_no: 110
Employee 110 does not exist
PL/SQL procedure successfully completed.
```

Explanation: %TYPE declares variables with the same data type as the table column. SELECT INTO raises the predefined exception NO_DATA_FOUND when no row matches, which is handled in the EXCEPTION section instead of terminating the program with an error.

(Declarations 1 M, SELECT INTO 2 M, exception handling 2 M, output 1 M)

c) Sequence bill_seq.

6 M

Use of sequence: A sequence is a database object that automatically generates unique sequential numbers. It is mainly used to generate primary key values (bill number, roll number, order id) without conflicts between users. NEXTVAL gives the next number and CURRVAL gives the current number.

Syntax:

```
CREATE SEQUENCE seq_name
  [START WITH n] [INCREMENT BY n]
  [MINVALUE n | NOMINVALUE] [MAXVALUE n | NOMAXVALUE]
  [CYCLE | NOCYCLE] [CACHE n | NOCACHE];
```

Create the sequence:

```
CREATE SEQUENCE bill_seq
  START WITH 101
  INCREMENT BY 1
  MAXVALUE 999
  NOCYCLE;
```

Sequence created.

Insert two records using the sequence:

```
INSERT INTO bill VALUES (bill_seq.NEXTVAL, 'Ravi', 1250);
```

1 row created.

```
INSERT INTO bill VALUES (bill_seq.NEXTVAL, 'Anita', 840);
```

1 row created.

```
SELECT * FROM bill;
```

Output:

BILL_NO	CUST_NAME	AMOUNT
101	Ravi	1250
102	Anita	840

Alter the sequence to increment by 2 (next number = 102 + 2 = 104):

```
ALTER SEQUENCE bill_seq INCREMENT BY 2;
```

Sequence altered.

```
INSERT INTO bill VALUES (bill_seq.NEXTVAL, 'Sagar', 560);
```

1 row created.

```
SELECT * FROM bill;
```

Output:

BILL_NO	CUST_NAME	AMOUNT
101	Ravi	1250
102	Anita	840
104	Sagar	560

Note: START WITH cannot be changed by ALTER SEQUENCE; to restart, the sequence must be dropped and created again.

(Use 1 M, create 2 M, insert 2 M, alter 1 M)

Q.6

a) Create table Product with constraints.

6 M

The parent table Supplier (sup_id primary key) must exist first.

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

SUP_ID	SNAME
1	Shree Traders
2	Om Agencies

```
CREATE TABLE product (
  pid          NUMBER(4)    CONSTRAINT pk_product PRIMARY KEY,
  pname        VARCHAR2(20) CONSTRAINT nn_pname NOT NULL
                                CONSTRAINT uq_pname UNIQUE,
  category     VARCHAR2(15),
  price        NUMBER(8,2)  CONSTRAINT ck_price CHECK (price > 0),
  qty          NUMBER(5)    DEFAULT 0,
  sup_id       NUMBER(3)    CONSTRAINT fk_sup REFERENCES supplier(sup_id)
);
```

Table created.

Checking the constraints:

```
INSERT INTO product(pid, pname, category, price, sup_id) VALUES (1, 'Mouse',
'Computer', 450, 1);
```

1 row created.

```
SELECT * FROM product;
```

Output:

PID	PNAME	CATEGORY	PRICE	QTY	SUP_ID
1	Mouse	Computer	450	0	1

```
INSERT INTO product VALUES (1, 'Keyboard', 'Computer', 700, 5, 2);
```

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

```
INSERT INTO product VALUES (2, NULL, 'Computer', 700, 5, 2);
```

```
ERROR at line 1:
ORA-01400: cannot insert NULL into (PRODUCT.PNAME)
```

```
INSERT INTO product VALUES (3, 'Pen drive', 'Storage', 0, 5, 1);
```

```
ERROR at line 1:
ORA-02290: check constraint (CK_PRICE) violated
```

```
INSERT INTO product VALUES (4, 'Monitor', 'Computer', 7500, 2, 9);
```

```
ERROR at line 1:
ORA-02291: integrity constraint (FK_SUP) violated - parent key not found
```

```
INSERT INTO product VALUES (5, 'Mouse', 'Computer', 500, 3, 2);
```

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

Row 1 got qty = 0 from the DEFAULT. The next five inserts are rejected by the primary key, NOT NULL, CHECK, FOREIGN KEY and UNIQUE constraints respectively.

(Each constraint 1 M)

b) PL/SQL function — factorial.

6 M

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE FUNCTION fact(n IN NUMBER) RETURN NUMBER IS
    f NUMBER := 1;
BEGIN
    FOR i IN 1..n LOOP
        f := f * i;
    END LOOP;
    RETURN f;
END fact;
/
DECLARE
    num NUMBER := &num;
    res NUMBER;
BEGIN
    res := fact(num);
    DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is ' || res);
END;
/
```

Output:

```
Function created.
Enter value for num: 6
Factorial of 6 is 720
PL/SQL procedure successfully completed.
```

Explanation: The function *fact* takes *n* as an IN parameter, multiplies *f* by 1, 2, ..., *n* in a FOR loop and returns *f* using RETURN. The anonymous block accepts a number, calls *fact(num)* as part of an expression and prints the returned value. The function can also be called in SQL: `SELECT fact(5) FROM dual;` → 120.

(Function 4 M, calling block 2 M)

c) Super, candidate, primary and foreign key.

6 M

Example relations: STUD (roll_no, enroll_no, name, dept_id) and DEPARTMENT (dept_id, dname).

Example — table **STUD**:

ROLL_NO	ENROLL_NO	NAME	DEPT_ID
1	2610001	Asha	10
2	2610002	Rahul	20
3	2610003	Asha	10

Example — table **DEPARTMENT**:

DEPT_ID	DNAME
10	CO
20	IT

- Super key:** A set of one or more attributes that can uniquely identify each tuple (row) of a relation. It may contain extra attributes. Example: {roll_no}, {enroll_no}, {roll_no, name}, {roll_no, enroll_no, name}.
- Candidate key:** A minimal super key — no proper subset of it is a super key. A relation can have more than one candidate key. Example: {roll_no} and {enroll_no}. ({roll_no, name} is not candidate because roll_no alone is enough; name alone is not a key because 'Asha' repeats.)
- Primary key:** The candidate key chosen by the database designer to identify rows uniquely. It cannot be NULL and cannot repeat. Example: roll_no is the primary key of STUD (enroll_no becomes an alternate key).
- Foreign key:** An attribute of one relation (child) that refers to the primary key of another relation (parent); its value must match a value in the parent table or be NULL. Example: STUD.dept_id references DEPARTMENT.dept_id.

```
CREATE TABLE stud (
    roll_no    NUMBER(3) PRIMARY KEY,
    enroll_no  VARCHAR2(10) UNIQUE NOT NULL,
    name       VARCHAR2(15),
    dept_id    NUMBER(2) REFERENCES department(dept_id) );
```

(Each key with example 1½ 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