

Advance Database Management (315324)

Model Question Paper — Set A · Model Answers

About the code in these answers: every query and command was actually executed before printing, and the outputs shown are copied from those runs. **SQL** — PostgreSQL 16 (object-relational features: composite/structured types, table inheritance, arrays); where the textbook SQL:1999 syntax (e.g. UNDER, MULTISSET) is not accepted by PostgreSQL, the standard syntax to write in the exam is given and the PostgreSQL equivalent is shown with its real output. **MongoDB** — shell commands were run on an in-memory MongoDB-compatible engine (mongomock 4.3) and printed in mongosh style; ObjectId values will be different on your computer. Plain JavaScript expressions (as typed in mongosh) were run on Node.js (the same V8 JavaScript engine used by mongosh), time zone IST. **XML** — XQuery / XPath on Saxon-HE 13; XML Schema / DTD validation with libxml2 (lxml). Commands are shown in **bold**, results below them. Sample data is assumed wherever needed.

Q.1

a) Concurrency control; two protocols.

2 M

Concurrency control: It is the mechanism of the DBMS that controls the interaction among concurrently executing transactions so that the database remains consistent and the result is the same as some serial execution (isolation is preserved).

Concurrency control protocols:

1. Lock-based protocols (e.g. Two-Phase Locking)
2. Timestamp-based protocols
3. Validation-based (optimistic) protocols
4. Multiversion schemes

(Definition 1 M, any two protocols 1 M)

b) Shared and exclusive lock.

2 M

A **lock** is a variable associated with a data item that controls access to it by concurrent transactions.

1. **Shared lock (S):** If a transaction T holds a shared lock on item Q, T can **read** Q but cannot write it. Many transactions can hold S locks on the same item at the same time. Requested by lock-S(Q).
2. **Exclusive lock (X):** If T holds an exclusive lock on Q, T can both **read and write** Q. No other transaction can hold any lock on Q at the same time. Requested by lock-X(Q).

(Each definition 1 M)

c) Four advantages of distributed database.

2 M

1. **Data sharing** — users at one site can use data of other sites.
2. **Reliability / availability** — failure of one site does not stop the whole system.
3. **Local autonomy** — every site manages its own data.
4. **Faster local response and modular growth** — local queries are fast; new sites can be added easily.

(Any four, ½ M each)

d) FLWOR expression.

2 M

FLWOR (pronounced “flower”) is the main expression of XQuery, named after its clauses **F**or, **L**et, **W**here, **O**rders by and **R**eturn. It works like SELECT–FROM–WHERE of SQL on XML data.

```
for    $var in path-expression      (iterate over nodes)
let    $v := expression             (bind a value)
where  condition                    (filter)
order  by expression [ascending|descending]
return expression                   (build the result)
```

(Definition 1 M, general form 1 M)

e) Syntax — subtable using table inheritance.

2 M

SQL:1999 (textbook) syntax:

```
CREATE TABLE super_table OF super_type;
CREATE TABLE sub_table OF sub_type UNDER super_table;

-- e.g.
CREATE TABLE people    OF Person;
CREATE TABLE students OF Student UNDER people;
```

PostgreSQL form: CREATE TABLE sub_table (extra columns) INHERITS (super_table);

(Correct syntax 2 M)

f) Four types of NoSQL databases.

2 M

Type	Example
Key–value store	Redis, Amazon DynamoDB
Document store	MongoDB, CouchDB
Column-family (wide-column) store	Apache Cassandra, HBase
Graph database	Neo4j

(½ M each)

g) Data warehouse — definition, characteristics.

2 M

Data warehouse: A data warehouse is a repository (archive) of information gathered from multiple sources, stored under a unified schema at a single site, and used for analysis and decision making. (W. H. Inmon: a *subject-oriented, integrated, time-variant and non-volatile* collection of data in support of management’s decisions.)

Characteristics: i) **Subject-oriented** — organised around subjects such as sales, customers. ii) **Non-volatile** — data is loaded and read; it is not updated or deleted by daily transactions. (Also integrated, time-variant.)

(Definition 1 M, any two characteristics 1 M)

Q.2

a) Centralized database system architecture.

4 M

A **centralized database system** is one that runs on a single computer system and does not interact with other computer systems. It may be a single-user system (desktop PC) or a multi-user system (server) that serves many users through terminals.

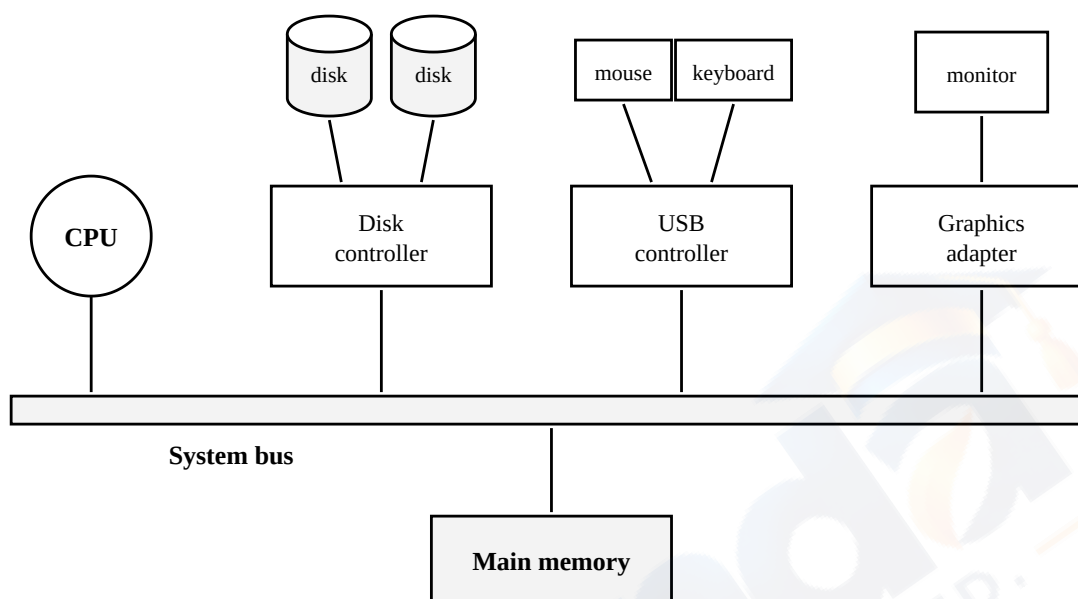


Fig.: Centralized database system (single computer system)

1. A general-purpose computer has one or a few **CPUs** and a number of **device controllers** connected through a common **system bus** that gives access to shared **main memory**.
2. Each device controller is in charge of a specific type of device — **disk controller** (disk drives), **USB controller** (mouse, keyboard, printer), **graphics adapter** (monitor).
3. The CPU(s) and the device controllers execute concurrently, competing for memory access; CPUs have local cache memories to reduce this contention.
4. The DBMS, all data, the transaction manager, recovery and concurrency control are on this one machine. Single-user DBMSs may not provide concurrency control and crash recovery; multi-user systems support many concurrent users.
5. Parallelism is **coarse-grained**: each transaction (query) runs on one processor; multiple queries run concurrently to increase throughput.

Advantages: simple to design and manage, easy security and integrity. **Disadvantages:** single point of failure, limited scalability, remote users face network delay.

(Diagram 2 M, explanation 2 M)

b) Shared-memory and shared-disk architectures.

4 M

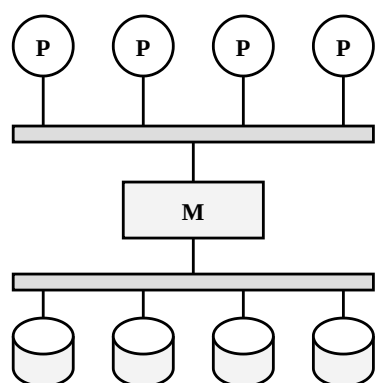
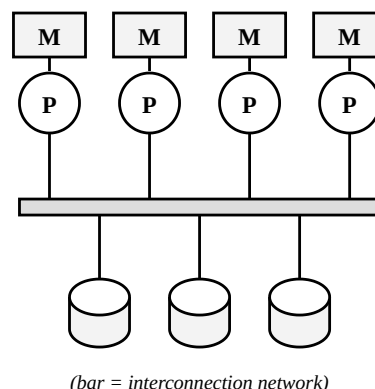


Fig. (a): Shared memory



(bar = interconnection network)

Fig. (b): Shared disk

P = processor, M = memory, cylinder = disk

1) Shared memory: All processors share a common main memory (and all disks) through a bus or interconnection network.

- Communication between processors is very fast — a processor can send data by writing into memory.
- **Limitation:** not scalable beyond 32–64 processors because the bus / memory becomes a bottleneck; processors spend time waiting for memory. Cache-coherency must be maintained.

2) Shared disk: Every processor has its own private memory, but all processors can access all disks directly through an interconnection network.

- Memory bus is not a bottleneck; offers **fault tolerance** — if a processor fails, other processors can take over its work because the database is on shared disks (e.g. Oracle RAC, DEC clusters).
- **Limitation:** the interconnection to the disk subsystem becomes a bottleneck; communication between processors is slower (through network).

(Each architecture: diagram 1 M + explanation 1 M)

c) Table inheritance in SQL.

4 M

Table inheritance allows a table (**subtable**) to be created as a specialisation of another table (**supertable**). The subtable inherits all columns of the supertable and adds its own columns. It corresponds to the E-R notion of specialisation/generalisation.

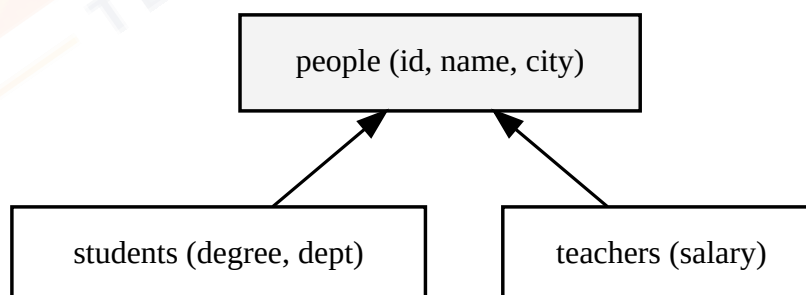


Fig.: Supertable people with subtables students and teachers

SQL:1999 syntax (types Person, Student, Teacher created first):

```
CREATE TABLE people OF Person;
CREATE TABLE students OF Student UNDER people;
CREATE TABLE teachers OF Teacher UNDER people;
```

Executed example (PostgreSQL uses INHERITS):

```

adm_mp=# CREATE TABLE people (id INT PRIMARY KEY, name VARCHAR(20), city
VARCHAR(15));
CREATE TABLE
adm_mp=# CREATE TABLE students (degree VARCHAR(10), dept VARCHAR(10)) INHERITS
(people);
CREATE TABLE
adm_mp=# CREATE TABLE teachers (salary INT) INHERITS (people);
CREATE TABLE
adm_mp=# INSERT INTO people VALUES (1, 'Kiran', 'Nashik');
INSERT 0 1
adm_mp=# INSERT INTO students VALUES (2, 'Asha', 'Pune', 'Diploma', 'IF');
INSERT 0 1
adm_mp=# INSERT INTO teachers VALUES (3, 'Mr. Joshi', 'Nanded', 55000);
INSERT 0 1
adm_mp=# SELECT * FROM people;
 id |   name   |  city
----+-----+-----
  1 |  Kiran   | Nashik
  2 |  Asha    |  Pune
  3 | Mr. Joshi | Nanded
(3 rows)
adm_mp=# SELECT * FROM ONLY people;
 id | name | city
----+-----+-----
  1 | Kiran | Nashik
(1 row)
adm_mp=# SELECT name, degree, dept FROM students;
 name | degree | dept
-----+-----+-----
 Asha | Diploma | IF
(1 row)

```

Points: every tuple of a subtable is implicitly present in the supertable (query on *people* returns 3 rows); **ONLY** restricts the query to the supertable's own rows. Inherited attributes need not be repeated, and a query on the supertable automatically covers all subtables.

(Explanation 2 M, example 2 M)

d) Basic CRUD operations in MongoDB shell.

4 M

CRUD = **C**reate, **R**ead, **U**ppdate, **D**elete. In the MongoDB shell (mongosh) they are methods of a collection object `db.<collection>`.

1. **Create:** `insertOne(doc)`, `insertMany([docs])` — add documents; collection is created automatically.
2. **Read:** `find(query, projection)`, `findOne()` — retrieve documents.
3. **Update:** `updateOne(filter, update)`, `updateMany()` with operators like `$set`, `$inc`.
4. **Delete:** `deleteOne(filter)`, `deleteMany(filter)` — remove documents.

```

test> use college
switched to db college
college> db.students.insertOne({ _id: 1, name: "Asha", branch: "IF", marks: 78
})
{ acknowledged: true, insertedId: 1 }
college> db.students.insertOne({ _id: 2, name: "Rohan", branch: "AN", marks: 64
})

```

```
{ acknowledged: true, insertedId: 2 }
college> db.students.find()
[
  { _id: 1, name: 'Asha', branch: 'IF', marks: 78 },
  { _id: 2, name: 'Rohan', branch: 'AN', marks: 64 }
]
college> db.students.updateOne({ _id: 2 }, { $set: { marks: 70 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
college> db.students.deleteOne({ _id: 1 })
{ acknowledged: true, deletedCount: 1 }
college> db.students.find()
[ { _id: 2, name: 'Rohan', branch: 'AN', marks: 70 } ]
```

(Each operation with example 1 M)

Q.3

a) Two-Phase Locking protocol with example.

4 M

The **Two-Phase Locking (2PL)** protocol ensures conflict-serializable schedules by requiring each transaction to issue its lock and unlock requests in two phases:

1. **Growing phase:** the transaction may obtain locks but may not release any lock.
2. **Shrinking phase:** the transaction may release locks but may not obtain any new lock.

Initially a transaction is in the growing phase; once it releases its first lock it enters the shrinking phase. The point where it obtains its final lock is the **lock point**; transactions are serializable in the order of their lock points.

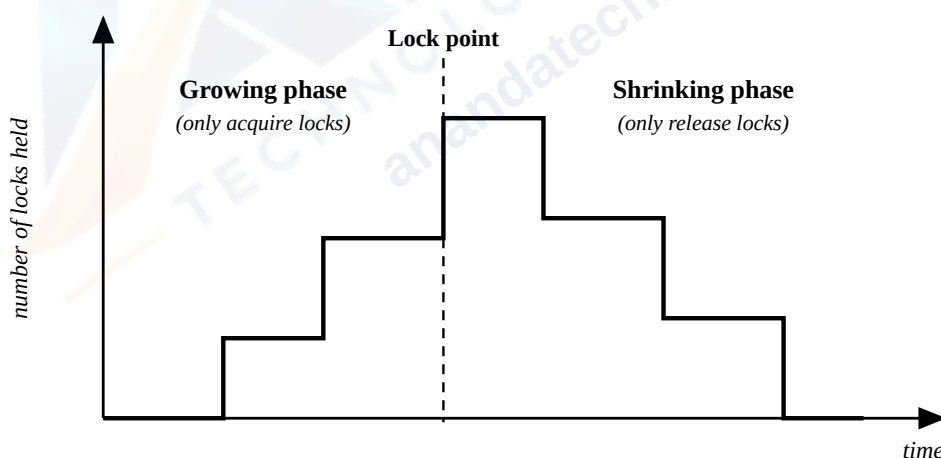


Fig.: Two-phase locking — growing and shrinking phases

Example (T1 transfers money from A to B; T2 displays A and B). The schedule below obeys 2PL — it was checked step by step with a small lock-manager program (S/X compatibility):

Step	Txn	Operation	Lock manager	Locks held after step	Phase of txn
1	T1	lock-X(A)	granted	T1:X(A)	growing
2	T1	read(A)	—	T1:X(A)	growing
3	T1	write(A)	—	T1:X(A)	growing
4	T1	lock-X(B)	granted	T1:X(A), T1:X(B)	growing
5	T1	unlock(A)	released	T1:X(B)	shrinking
6	T2	lock-S(A)	granted	T2:S(A), T1:X(B)	growing
7	T2	read(A)	—	T2:S(A), T1:X(B)	growing
8	T1	read(B)	—	T2:S(A), T1:X(B)	shrinking
9	T1	write(B)	—	T2:S(A), T1:X(B)	shrinking
10	T1	unlock(B)	released	T2:S(A)	shrinking
11	T2	lock-S(B)	granted	T2:S(A), T2:S(B)	growing
12	T2	read(B)	—	T2:S(A), T2:S(B)	growing
13	T2	unlock(A)	released	T2:S(B)	shrinking
14	T2	unlock(B)	released	—	shrinking

Schedule under 2PL (T1 lock point = step 4, T2 lock point = step 11) — equivalent to serial order T1 → T2

T1 acquires both locks before releasing A, and T2 acquires its locks only after T1 released them, so the schedule is equivalent to T1 followed by T2. Basic 2PL does not prevent deadlock or cascading rollback; **strict 2PL** (hold X-locks till commit) and **rigorous 2PL** (hold all locks till commit) are used in practice.

(Two phases 2 M, example 2 M)

b) SQL vs NoSQL.

4 M

Point	SQL (relational) database	NoSQL database
Data model	Relational — data stored in tables (rows and columns).	Non-relational — key-value, document, column-family or graph.
Schema	Fixed, predefined schema; must be altered to add columns.	Dynamic / flexible schema; each document can have different fields.
Scalability	Mainly vertical scaling (bigger server).	Horizontal scaling (add more commodity servers — sharding).
Transactions	Strong ACID properties.	Mostly BASE (basically available, soft state, eventual consistency); ACID support is limited / per document.
Examples	Oracle, MySQL, PostgreSQL, SQL Server.	MongoDB, Cassandra, Redis, Neo4j, HBase.

(Any four points, 1 M each)

c) Structure of XML data.

4 M

XML (Extensible Markup Language) stores data as a tree of **elements** marked with user-defined **tags**. It is self-describing and is used for data exchange.

1. **Element:** a pair of matching start and end tags with text/elements between them — `<city>Pune</city>`. Elements must be **properly nested**.
2. **Root element:** every document has exactly one top-level element enclosing all others.
3. **Attributes:** name = value pairs inside the start tag — `<item qty="2">`; an attribute can appear only once in a tag.
4. **Sub-elements / nesting** represent relationships (customer inside purchase_order); text content is called PCDATA. Empty element: `<br/>`.
5. **Prolog:** optional XML declaration `<?xml version="1.0"?>`, comments `<!-- -->`; namespaces avoid name clashes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- order details of a shop -->
<purchase_order id="PO-101">
  <customer>
    <name>Asha Patil</name>
    <city>Pune</city>
  </customer>
  <item qty="2">Keyboard</item>
  <item qty="1">Mouse</item>
</purchase_order>
```

(The document above was checked with an XML parser: well-formed = yes.)

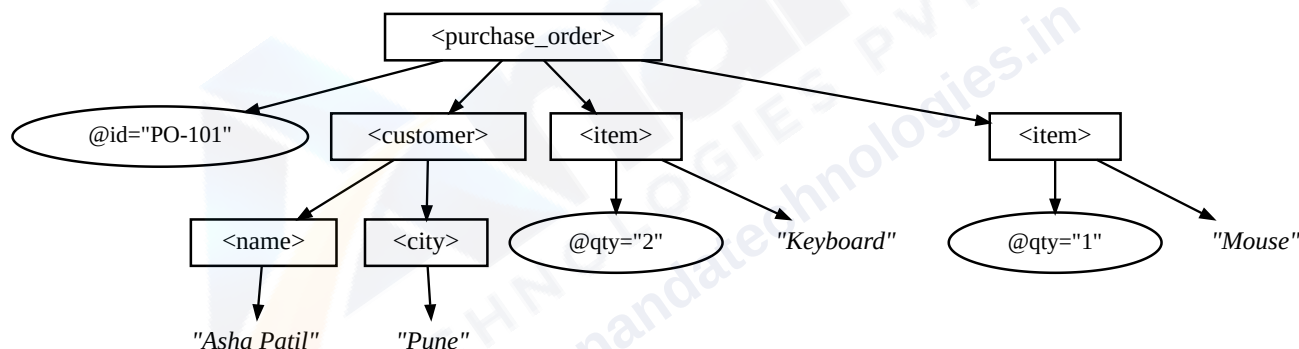


Fig.: Tree structure of the XML document

(Explanation 2 M, example 2 M)

d) Components of a data warehouse.

4 M

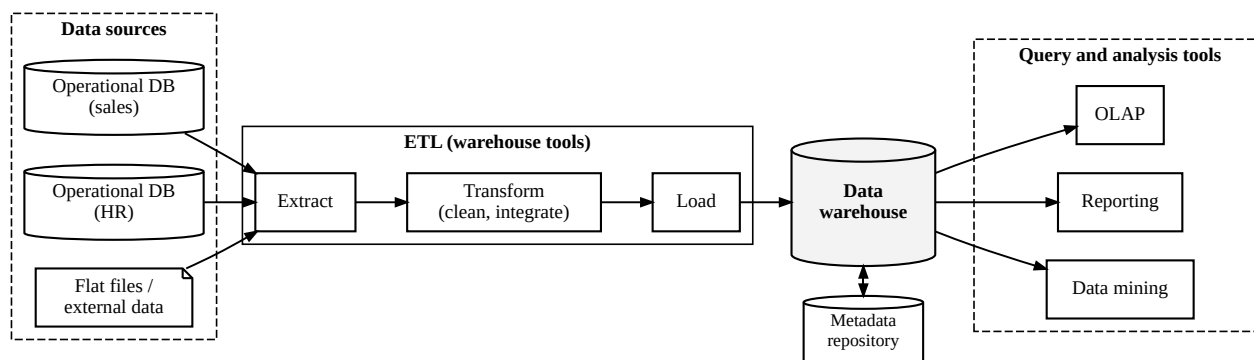


Fig.: Data-warehouse architecture (components)

1. **Data sources:** operational databases of different departments / branches, flat files and external data. They may use different schemas and formats.
2. **ETL tools (data loaders):** *Extract* data from sources, *Transform* it (cleansing, removing duplicates, converting units/formats, integrating to one schema) and *Load* it into the warehouse periodically (refresh).
3. **Data warehouse storage (DBMS):** stores integrated, historical, summarised data — usually in a star / snowflake schema (fact and dimension tables) and data marts.
4. **Metadata repository:** “data about data” — source of each item, transformation rules, schema, summary definitions.
5. **Query and analysis tools:** OLAP, reporting tools, dashboards and data-mining tools used by decision makers.

(Diagram 2 M, explanation 2 M)

Q.4

a) Speedup and scaleup; numerical.

4 M

Speedup: Running a *given task* in less time by increasing the degree of parallelism. If TS is the time on the small system and TL the time on a system N times larger, **Speedup** = TS / TL. Speedup is **linear** if it equals N, **sublinear** if it is less than N.

Scaleup: Handling an N-times *larger task* in the same time by an N-times larger system. If TS is the time for task Q on small system MS and TL is the time for task N×Q on large system ML, **Scaleup** = TS / TL; linear if = 1, sublinear if < 1.

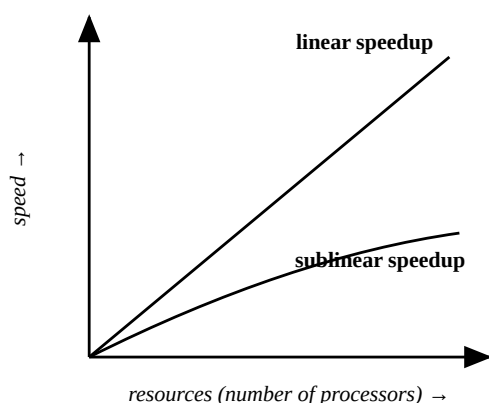


Fig. (a): Speedup

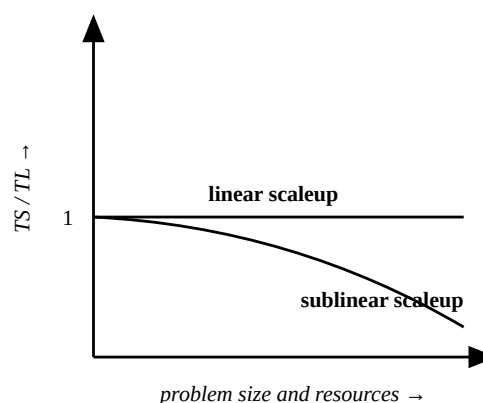


Fig. (b): Scaleup

Numerical: TS = 48 s, TL = 15 s, N = 4.

Speedup = TS / TL = 48 / 15 = **3.20**.

Since $3.20 < 4$ (number of processors), the speedup is **sublinear** (efficiency = $3.20/4 = 80\%$). The loss is due to start-up cost, interference and skew.

(Definitions 2 M, calculation 2 M)

b) XPath expressions on college.xml.

4 M

Assumed document *college.xml*:

```
<college>
  <student roll="1" branch="IF"><name>Asha</name><marks>82</marks>
<city>Pune</city></student>
  <student roll="2" branch="AN"><name>Rohan</name><marks>68</marks>
<city>Nanded</city></student>
  <student roll="3" branch="IF"><name>Sneha</name><marks>91</marks>
<city>Nashik</city></student>
  <student roll="4" branch="CO"><name>Imran</name><marks>74</marks>
<city>Pune</city></student>
  <student roll="5" branch="IF"><name>Kavya</name><marks>59</marks>
<city>Latur</city></student>
</college>
```

Requirement	XPath expression	Result (executed)
i) names of all students	/college/student/name	<name>Asha</name> <name>Rohan</name> <name>Sneha</name> <name>Imran</name> <name>Kavya</name>
ii) student whose roll is 3	/college/student[@roll="3"]	<student roll="3" branch="IF"> <name>Sneha</name> <marks>91</marks> <city>Nashik</city> </student>
iii) names of students with marks > 75	/college/student[marks > 75]/name	<name>Asha</name> <name>Sneha</name>
iv) count of IF students	count(/college/student[@branch="IF"])	3

/ selects children, @ selects an attribute, [] is a predicate (condition) and count() is an XPath function.

(Each expression 1 M)

c) find() queries on students collection.

4 M

Sample data (assumed) inserted first:

```
use college
db.students.insertMany([
  { name: "Asha", branch: "IF", marks: 82, city: "Pune" },
  { name: "Rohan", branch: "AN", marks: 68, city: "Nanded" },
  { name: "Sneha", branch: "CO", marks: 91, city: "Pune" },
  { name: "Imran", branch: "IF", marks: 55, city: "Pune" },
  { name: "Kavya", branch: "ME", marks: 47, city: "Latur" }
])
```

```
college> db.students.find({ marks: { $gt: 70 } }, { _id: 0 })
[
  { name: 'Asha', branch: 'IF', marks: 82, city: 'Pune' },
  { name: 'Sneha', branch: 'CO', marks: 91, city: 'Pune' }
]
college> db.students.find({ branch: { $in: ["IF", "AN"] } }, { _id: 0 })
[
  { name: 'Asha', branch: 'IF', marks: 82, city: 'Pune' },
  { name: 'Rohan', branch: 'AN', marks: 68, city: 'Nanded' },
  { name: 'Imran', branch: 'IF', marks: 55, city: 'Pune' }
]
college> db.students.find({}, { name: 1, marks: 1, _id: 0 })
[
  { name: 'Asha', marks: 82 },
  { name: 'Rohan', marks: 68 },
  { name: 'Sneha', marks: 91 },
  { name: 'Imran', marks: 55 },
  { name: 'Kavya', marks: 47 }
```

```
]
college> db.students.find({ city: "Pune", marks: { $lt: 60 } }, { _id: 0 })
[ { name: 'Imran', branch: 'IF', marks: 55, city: 'Pune' } ]
```

(i) \$gt = greater than. (ii) \$in matches any value in the list — same as {\$or: [{branch:"IF"}, {branch:"AN"}]}. (iii) The second argument of find() is the projection: 1 = show, 0 = hide. (iv) Two conditions in the same document are combined with AND. (_id: 0 is used only to keep the output short.)

(Each query 1 M)

d) Four applications of data mining.

4 M

1. **Market-basket analysis / retail:** finding items bought together (bread → butter) for shelf arrangement, offers and cross-selling.
2. **Banking and finance:** credit-risk prediction — deciding whether a loan applicant is a good or bad risk (classification); detecting fraudulent credit-card transactions (outlier detection).
3. **Customer relationship management:** customer segmentation (clustering) for targeted marketing; predicting customers likely to leave (churn) in telecom.
4. **Healthcare:** predicting diseases from symptoms and test results, finding effective treatments.
5. **Recommendation systems / web mining:** suggesting products, movies, courses based on the behaviour of similar users (e.g. online shopping sites).
6. **Education:** predicting student performance and identifying students at risk of failing.

(Any four applications, 1 M each)

e) Homogeneous and heterogeneous distributed databases.

4 M

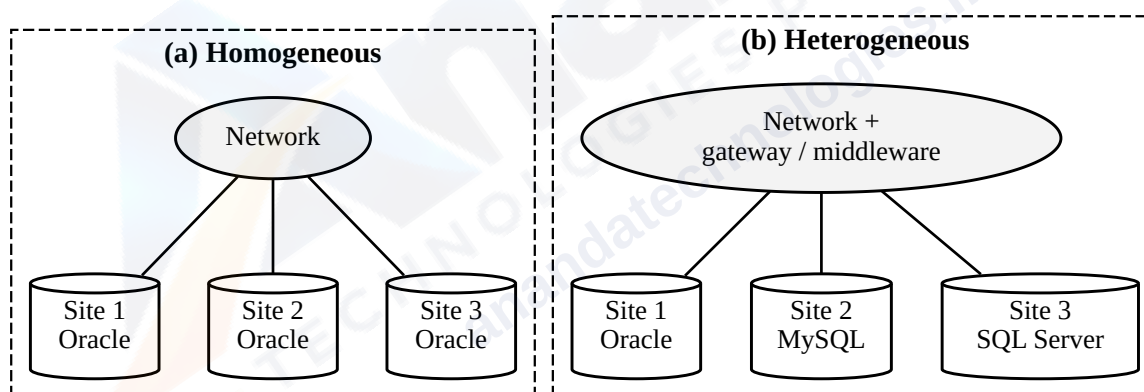


Fig.: Homogeneous vs heterogeneous distributed database

Homogeneous distributed database:

- All sites run the **same DBMS software** and have the same schema; they are aware of one another.
- Sites agree to cooperate in processing user requests; a site gives up part of its autonomy (cannot change schema or software alone).
- Appears to the user as a single system; easier to design and manage. Example: all branches of a bank using Oracle.

Heterogeneous distributed database:

- Different sites may use **different DBMSs, schemas and data models** (relational, object, files).
- Sites may not be aware of each other and give only limited facilities for cooperation; a gateway / middleware translates queries and data.
- Transaction processing and query processing are more difficult. Example: a company joining an Oracle site, a MySQL site and a SQL Server site.

(Each type 2 M)

Q.5

a) Array and multiset types in SQL.

6 M

SQL:1999 / SQL:2003 support two **collection types** so that one attribute can hold many values (non-1NF, complex data):

1. **Array:** an *ordered* collection of elements of the same type; elements are accessed by position (index starts at 1). Declared as type `ARRAY[n]`. Example: the authors of a book — the order of authors is important.
2. **Multiset:** an *unordered* collection in which an element *may occur many times* (a bag). Declared as type `MULTISET`. Example: the set of keywords of a book.

SQL:1999 syntax (as in textbook):

```
CREATE TYPE Publisher AS (name VARCHAR(20), branch VARCHAR(20));
CREATE TYPE Book AS (
    title          VARCHAR(20),
    author_array   VARCHAR(20) ARRAY [10],
    pub_date       DATE,
    publisher       Publisher,
    keyword_set    VARCHAR(20) MULTISET );
CREATE TABLE books OF Book;

INSERT INTO books VALUES ('Compilers', ARRAY['Smith', 'Jones'], DATE '2020-01-10',
    NEW Publisher('McGraw-Hill', 'New York'), MULTISET['parsing',
    'analysis']);

SELECT author_array[1] FROM books WHERE title = 'Compilers';
SELECT title FROM books WHERE 'parsing' IN (UNNEST(keyword_set));
```

Executed example — PostgreSQL supports arrays (type[]) but has no MULTISET type, so the keyword multiset is stored as an array and treated as a bag using `unnest` / `ANY`:

```
adm_mp=# CREATE TYPE publisher_t AS (name VARCHAR(20), branch VARCHAR(20));
CREATE TYPE
adm_mp=# CREATE TABLE books (
...     title          VARCHAR(30),
...     author_array   VARCHAR(20)[],
...     pub_date       DATE,
...     publisher       publisher_t,
...     keyword_set    VARCHAR(20)[]    -- multiset emulated by an array
... );
CREATE TABLE
adm_mp=# INSERT INTO books VALUES ('Database Concepts', ARRAY['Silberschatz',
    'Korth', 'Sudarshan'], '2019-02-01', ROW('McGraw-Hill', 'New York'),
    ARRAY['database', 'SQL', 'database']);
```

```

INSERT 0 1
adm_mp=# INSERT INTO books VALUES ('Learning MongoDB', ARRAY['Chodorow'], '2013-
05-10', ROW('OReilly', 'Mumbai'), ARRAY['NoSQL', 'MongoDB']);
INSERT 0 1
adm_mp=# SELECT title, author_array[1] AS first_author,
cardinality(author_array) AS no_of_authors FROM books;
      title      | first_author | no_of_authors
-----+-----+-----
 Database Concepts | Silberschatz |          3
 Learning MongoDB | Chodorow     |          1
(2 rows)
adm_mp=# SELECT title, unnest(author_array) AS author FROM books;
      title      | author
-----+-----
 Database Concepts | Silberschatz
 Database Concepts | Korth
 Database Concepts | Sudarshan
 Learning MongoDB | Chodorow
(4 rows)
adm_mp=# SELECT title FROM books WHERE 'NoSQL' = ANY (keyword_set);
      title
-----
 Learning MongoDB
(1 row)
adm_mp=# SELECT title, ARRAY(SELECT DISTINCT k FROM unnest(keyword_set) AS k
ORDER BY k) AS keyword_set_as_set FROM books WHERE title LIKE 'Data%';
      title      | keyword_set_as_set
-----+-----
 Database Concepts | {SQL,database}
(1 row)

```

Operations: array[i] — element access; cardinality() — number of elements; UNNEST() — converts a collection into rows (unnesting); for multisets SQL also gives MEMBER OF, SET() (removes duplicates), FUSION, COLLECT, INTERSECTION. Note that the keyword ‘database’ appears twice — a multiset allows duplicates, and SET()/DISTINCT removes them.

(Array 2 M, multiset 2 M, example 2 M)

b) Advantages and disadvantages of distributed database.

6 M

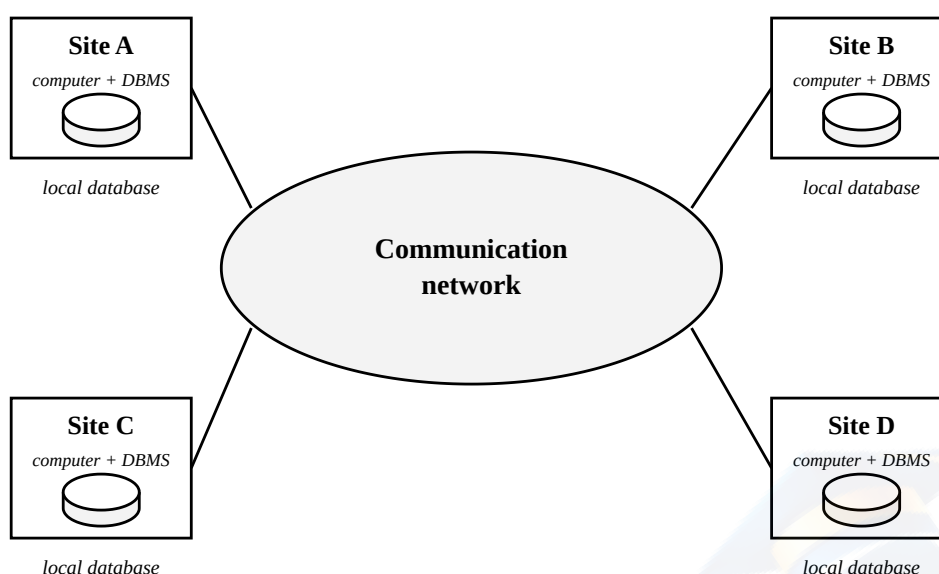


Fig.: Distributed database system architecture

Advantages:

1. **Sharing of data:** a user at one site can access data at other sites.
2. **Reliability and availability:** failure of one site does not stop the system; replicated data remains available.
3. **Faster local processing and local autonomy:** data is stored near users; each site controls its own data.

Disadvantages:

1. **Complexity and cost:** software development, design (fragmentation/allocation) and administration are costly and complex.
2. **Processing overhead:** message exchange and coordination (e.g. two-phase commit) add overhead; bugs are more likely.
3. **Security and integrity:** data spread across sites and networks is harder to secure and to keep consistent.

(Three advantages 3 M, three disadvantages 3 M)

c) XQuery — FLWOR with sorting, join, nested query.

6 M

Assumed documents:

```

<books>
  <book><isbn>B1</isbn><title>Advance DBMS</title><pub_id>P1</pub_id>
<price>450</price></book>
  <book><isbn>B2</isbn><title>Learning XML</title><pub_id>P2</pub_id>
<price>620</price></book>
  <book><isbn>B3</isbn><title>MongoDB Basics</title><pub_id>P1</pub_id>
<price>540</price></book>
  <book><isbn>B4</isbn><title>Data Mining</title><pub_id>P2</pub_id>
<price>780</price></book>
</books>
  
```

```
<publishers>
  <publisher><pub_id>P1</pub_id><name>Tech Press</name><city>Pune</city>
</publisher>
  <publisher><pub_id>P2</pub_id><name>Vision Books</name><city>Mumbai</city>
</publisher>
</publishers>
```

i) Titles of books with price > 500, sorted on price:

XQuery:

```
for $b in doc("books.xml")/books/book
where $b/price > 500
order by xs:decimal($b/price)
return <title price="{ $b/price}">{ $b/title/text() }</title>
```

Output:

```
<title price="540">MongoDB Basics</title>
<title price="620">Learning XML</title>
<title price="780">Data Mining</title>
```

ii) Join — book title with city of its publisher:

XQuery:

```
for $b in doc("books.xml")/books/book,
   $p in doc("publishers.xml")/publishers/publisher
where $b/pub_id = $p/pub_id
return <book_city>{ $b/title }{ $p/city }</book_city>
```

Output:

```
<book_city>
  <title>Advance DBMS</title>
  <city>Pune</city>
</book_city>
<book_city>
  <title>Learning XML</title>
  <city>Mumbai</city>
</book_city>
<book_city>
  <title>MongoDB Basics</title>
  <city>Pune</city>
</book_city>
<book_city>
  <title>Data Mining</title>
  <city>Mumbai</city>
</book_city>
```

iii) Nested query — every publisher with titles of its books:

XQuery:

```
<result>{
  for $p in doc("publishers.xml")/publishers/publisher
  return <publisher name="{ $p/name}">{
    for $b in doc("books.xml")/books/book[pub_id = $p/pub_id]
    return $b/title
  }
}
```

```
}</publisher>
}</result>
```

Output:

```
<result>
  <publisher name="Tech Press">
    <title>Advance DBMS</title>
    <title>MongoDB Basics</title>
  </publisher>
  <publisher name="Vision Books">
    <title>Learning XML</title>
    <title>Data Mining</title>
  </publisher>
</result>
```

The join is written by two for variables and a where condition on the common element pub_id; the nested query has a FLWOR expression inside the return clause of another FLWOR expression.

(Each query 2 M)

Q.6

a) Transaction-server process structure.

6 M

In a **transaction-server** (query-server) system, clients send requests (SQL queries or remote procedure calls) to the server, the server executes the transactions and sends results back. A typical transaction server consists of several processes accessing data in **shared memory**:

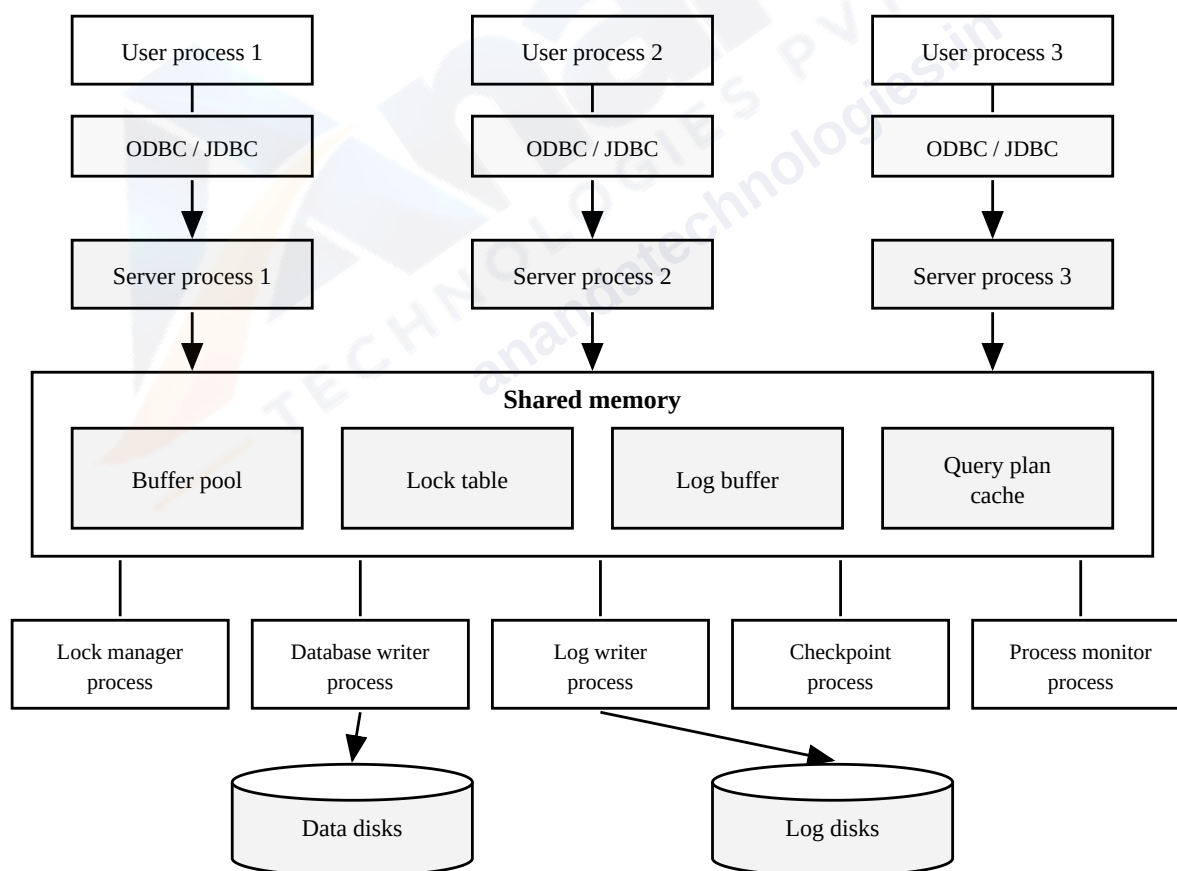


Fig.: Transaction-server (query-server) process structure

1. **Server processes:** receive user queries (transactions), execute them and send results. They may be multi-threaded so that one process serves many users.
2. **Lock manager process:** grants and releases locks and detects deadlocks (lock table is in shared memory).
3. **Database writer process:** continuously writes modified buffer blocks back to disk.
4. **Log writer process:** writes log records from the log buffer to stable storage; server processes only add records to the log buffer.
5. **Checkpoint process:** performs periodic checkpoints.
6. **Process monitor process:** monitors other processes; if one fails, it performs recovery actions (e.g. aborts its transaction) and restarts it.

Shared memory contains all shared data: the **buffer pool**, **lock table**, **log buffer** (log records waiting to be output) and **cached query plans**. Access to shared memory is controlled by mutual exclusion (semaphores / atomic instructions) so that two processes do not update the same structure at the same time.

(Diagram 3 M, explanation of processes and shared memory 3 M)

b) MongoDB commands on products collection.

6 M

```
test> use shop
switched to db shop
shop> db.products.insertMany([
  { _id: 1, name: "Keyboard", price: 700, qty: 12 },
  { _id: 2, name: "Mouse", price: 450, qty: 30 },
  { _id: 3, name: "Monitor", price: 7500, qty: 0 },
  { _id: 4, name: "Pen Drive", price: 399, qty: 25 },
  { _id: 5, name: "Webcam", price: 1800, qty: 0 }
])
{
  acknowledged: true,
  insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }
}
shop> db.products.find()
[
  { _id: 1, name: 'Keyboard', price: 700, qty: 12 },
  { _id: 2, name: 'Mouse', price: 450, qty: 30 },
  { _id: 3, name: 'Monitor', price: 7500, qty: 0 },
  { _id: 4, name: 'Pen Drive', price: 399, qty: 25 },
  { _id: 5, name: 'Webcam', price: 1800, qty: 0 }
]
shop> db.products.updateOne({ name: "Mouse" }, { $set: { price: 550 } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
shop> db.products.deleteMany({ qty: 0 })
{ acknowledged: true, deletedCount: 2 }
shop> db.products.find()
[
  { _id: 1, name: 'Keyboard', price: 700, qty: 12 },
  { _id: 2, name: 'Mouse', price: 550, qty: 30 },
```

```
{ _id: 4, name: 'Pen Drive', price: 399, qty: 25 }
]
```

use `shop` switches to (and creates on first insert) the database; `insertMany()` creates collection `products` automatically; `$set` changes only the given field; `deleteMany({qty: 0})` removed 2 documents (Monitor, Webcam).

(Each command 1 M)

c) Classification and clustering.

6 M

1) Classification is a *supervised* data-mining technique: from training data whose class labels are known, a model (classifier) is built which then predicts the class of new items.

- Steps: (i) collect training data with known classes, (ii) build the classifier (decision tree, Bayesian, neural network, SVM), (iii) validate it on test data (accuracy, precision, recall), (iv) classify new data.
- **Example:** a bank classifies loan applicants as *good risk* or *bad risk* using income, age, credit history. A decision tree may say: if income > 50,000 and no previous default → good risk.

2) Clustering is an *unsupervised* technique: items are grouped into clusters so that items in the same cluster are similar (close) and items in different clusters are dissimilar. Class labels are not given in advance.

- Methods: partitioning (k-means), hierarchical (agglomerative / divisive), density-based.
- **Example — k-means (k = 2):** eight students (S1–S8) with (hours of study per day, assignments submitted). Initial centroids C1 = S1 (2, 3), C2 = S5 (9, 8). Each student joins the nearest centroid (Euclidean distance):

Student	Point	Distance to C1	Distance to C2	Cluster
S1	(2, 3)	0.00	8.60	C1
S2	(3, 4)	1.41	7.21	C1
S3	(4, 3)	2.00	7.07	C1
S4	(3, 2)	1.41	8.49	C1
S5	(9, 8)	8.60	0.00	C2
S6	(10, 9)	10.00	1.41	C2
S7	(8, 9)	8.49	1.41	C2
S8	(9, 10)	9.90	2.00	C2

New centroids: C1 = (3.00, 3.00), C2 = (9.00, 9.00); re-assignment gives no change, so the algorithm stops after 2 iterations. Cluster 1 = S1–S4 (low effort group), cluster 2 = S5–S8 (high effort group). (Values computed by program.)

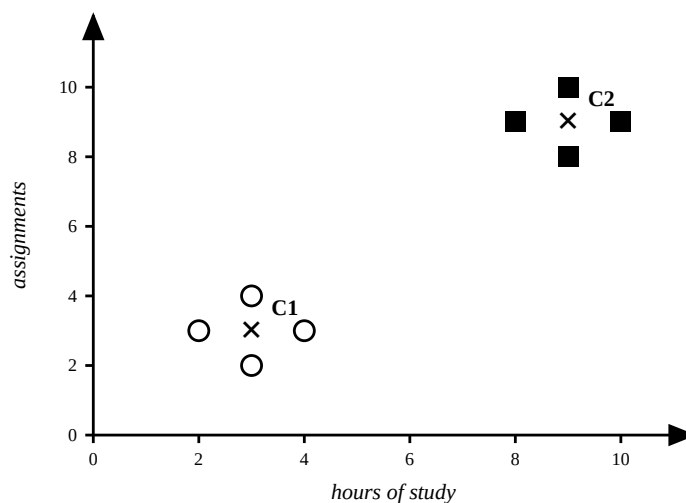


Fig.: Two clusters found by k-means ($\times$ = centroid)

Classification	Clustering
Supervised — classes known in advance	Unsupervised — groups discovered from data
Needs training data	No training data
e.g. spam / not spam e-mail	e.g. customer segmentation

(Classification 3 M, clustering 3 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