Database vs DataWarehouse

what is a collection of interrelated data organized in specific forms.
• a. DataMart
• b. DataItem
• c. DataWarehouse
• d. Database

The correct answer is: **d. Database**

Explanation:

A **Database** is:

- A **collection of interrelated data** that is organized in a **structured format** (tables, rows, columns, etc.).
- Designed to efficiently **store, manage, retrieve, and update** data.
- Examples: MySQL, PostgreSQL, Oracle, MongoDB.

Why not the other options?

- **a. DataMart**:
  A smaller version of a data warehouse, focused on a **specific business area** (like sales or finance). It's not a general-purpose structured collection like a database.
- **b. DataItem**:
  This usually refers to a **single unit or field** of data, not a collection.
- **c. DataWarehouse**:
  A large-scale system used for **reporting and data analysis**, and it often integrates data from multiple databases—but it is **broader and more analytical** than a database.

So, a **Database** is the most direct and accurate answer. ✅

give shorter explanations fot the next following ones:Copy constructor takes
arguments).
• a. two
• b. arbitrary number of
• c. one
O d. no

Correct answer: c. one

Short Explanation:

A **copy constructor** takes **one argument** – an object of the same class to create a copy of it.
Example in C++:

```
cpp

ClassName(const ClassName &obj);
```

Data structure representation in memory is termed as:
• a. Abstract data type
• b. Recursive data type
• c. Storage structure
• d.
File structure

Correct answer: c. Storage structure

Short Explanation:

Storage structure refers to how a data structure is **represented in computer memory** (RAM).
It's the actual implementation of the data (like arrays, linked lists, etc.).

dentify the addressing mode of the given instruction: R3 ← R3 + M[R1 + R2]
Question 4Answer

a.
Register

b.
Indexed

c.
Direct

d.
Immediate

Correct answer: b. Indexed

Short Explanation:

The instruction `R3 ← R3 + M[R1 + R2]` uses the **sum of two registers (R1 + R2)** to access memory.
This is a classic example of **indexed addressing mode**, where the effective address is calculated using a **base register + index register**.

> also explain other addressing modes in short

Sure! Here are short explanations of common **addressing modes**:

✅ 1. Register Addressing

- **Instruction uses only registers.**
- **Example:** `R1 ← R2 + R3`
- **Explanation:** Operands are in **CPU registers**, fast to access.

✅ 2. Immediate Addressing

- **Instruction uses a constant value.**
- **Example:** `R1 ← R2 + 5`
- **Explanation:** Operand is **hardcoded** in the instruction.

✅ 3. Direct Addressing

- **Uses a memory address directly.**
- **Example:** `R1 ← M[1000]`
- **Explanation:** Operand is at a **specific memory location**.

✅ 4. Indirect Addressing

- **Address is stored in a register or memory.**
- **Example:** `R1 ← M[R2]`
- **Explanation:** Get the actual operand **from the address stored** in R2.

✅ 5. Indexed Addressing

- **Uses base + offset (index) to find address.**
- **Example:** `R3 ← M[R1 + R2]`
- **Explanation:** Useful for **arrays or loops**.

✅ 6. Base Register Addressing

- **Similar to indexed, but usually one register is fixed as base.**
- **Example:** `R1 ← M[Base + Offset]`
- **Explanation:** Often used in **dynamic memory access**.

✅ 7. Relative Addressing

- **Used with program counter (PC).**
- **Example:** `Jump 20` means **PC + 20**
- **Explanation:** Mostly used in **branching or jumping**.

Let me know if you want a diagram or quick quiz on this!

↳
✅ **4\. Indirect Addressing** * **Address is stored in a register or memory.** * **Example:** `R1 ← M[R2]` * **Explanation:** Get the actual operand **from the address stored** in R2. * * * ### ✅ **5\. Indexed Addressing** * **Uses base + offset (index) to find address.** * **Example:** `R3 ← M[R1 + R2]` * **Explanation:** Useful for **arrays or loops**. * * * ### ✅ **6\. Base Register Addressing**
what is the clear difference between these three

Great question! Here's a **clear comparison** of the differences between **Indirect**, **Indexed**, and **Base Register Addressing**:

✅ 1. Indirect Addressing

- **What it does:**
  The register contains the **memory address** where the data is stored.
- **Example:** `R1 ← M[R2]`
  → Fetch data from the **address stored in R2**.
- **Use case:**
  Pointer dereferencing (e.g., accessing dynamic memory or linked lists).

✅ 2. Indexed Addressing

- **What it does:**
  Uses the **sum of a base address and an index** (both typically in registers) to calculate the **effective memory address**.
- **Example:** `R3 ← M[R1 + R2]`
  → R1 is base, R2 is index.
- **Use case:**
  Array traversal, where R2 is the element offset.

✅ 3. Base Register Addressing

- **What it does:**
  Similar to indexed, but typically:
  - One register is **dedicated as the base** (e.g., frame pointer)
  - The offset is a **constant** or displacement.
- **Example:** `R1 ← M[R2 + 4]`
  → R2 is base, 4 is constant offset.
- **Use case:**
  Accessing **structure fields, stack frames**, or **local variables**.

🧠 Quick Summary Table:

Mode	Address Calculation	Use Case
Indirect	`M[R]`	Pointers, linked lists
Indexed	`M[R1 + R2]`	Arrays, loops
Base Register	`M[Base + Constant Offset]`	Stack, structs

Let me know if you want a visual example!

give shorter explanations fot the next following ones:In the C++ code provided, what is the error (if any)?

```
class sample {
   mutable int a;       // line 1
public:
   sample() { a = 10; }
   void func() const {   // line 2
      a++;
      cout << a;       // line 3
   }
};
int main() {
   sample s1;
   s1.func();
   return 0;
}
```

Question 5Answer

a.
Error at line 3 – mutable int a cannot be overwritten

b.
There is no error in the code. The output is 11

c.
Error at line 2 – function func cannot be declared as const

d.
Error at line 1 – int a cannot be declared as mutable

Correct answer: b. There is no error in the code. The output is 11

Short Explanation:

- `mutable` allows **modifying a variable even inside a `const` function**.
- So `a++` inside `func()` is **valid**, and the code works.
- Output: **11** (since `a` starts at 10 and is incremented). ✅

SNMP is an application program that allows:
1. Retrieve value of object in agent
2. Store value in agent
3. Agent sends alarm to manager.
Choose correct:
Question 6Answer

a.
All 1, 2 and 3

b.
Only 1 and 2

c.
Only 1

d.
Only 2 and 3

Correct answer: a. All 1, 2 and 3

Short Explanation:

SNMP (Simple Network Management Protocol) allows:

1. ✅ **Retrieving** data from network devices (Get request)
2. ✅ **Storing** or updating data (Set request)
3. ✅ Devices (agents) can **send alerts (traps/alarms)** to managers

So **all three statements are correct**.

What is the maximum degree of any vertex in a simple graph with n vertices?
Question 7Answer

a.
n

b.
n–1

c.
n+1

d.
2n–1

Correct answer: b. n – 1

Short Explanation:

In a **simple graph** (no self-loops or multiple edges),
a vertex can be connected to **at most every other vertex**, which is **(n – 1)** other vertices.
So, **maximum degree = n – 1**. ✅

what is degree of a vertex

✅ Degree of a vertex (in graph theory):

The **degree** of a vertex is the **number of edges connected to it**.

🔹 In Undirected Graphs:

- Degree = **total number of edges** attached to the vertex.
- Example: If vertex A is connected to 3 other vertices, `deg(A) = 3`.

🔸 In Directed Graphs:

There are **two types** of degree:

- **In-degree** = number of **incoming edges** to a vertex.
- **Out-degree** = number of **outgoing edges** from a vertex.

💡 Example:

```
text

A —— B
|
C
```

- **`deg(A) = 2`** (connected to B and C)
- **`deg(B) = 1`**
- **`deg(C) = 1`**

Let me know if you want a visual example!

What is the only function all C++ programs must contain?
Question 8Answer

a.
system()

b.
program()

c.
start()

d.
main()

Correct answer: d. main()

Short Explanation:

Every C++ program **must have a `main()` function**, as it's the **starting point** of execution.
Without **`main()`**, the program **won't compile or run**. ✅

What is the order of Quick Sort in the worst case?
Question 9Answer

a.
O(n log n)

b.
O(log n)

c.
O (n)

d.
$O(n^2)$

Correct answer: d. $O(n^2)$

Short Explanation:

In the **worst case**, Quick Sort picks the **worst pivot** every time (e.g., smallest or largest element), causing **unbalanced partitions**.

This leads to **recursive calls on almost all elements**, resulting in **$O(n^2)$** time.

> ⚠️ Worst case often happens when the array is already sorted (or reverse sorted) without a good pivot strategy.

What will be the output of the following code?
#include <stdio.h>
#define scanf "SCAR"
int main(void) {
printf(scanf, scanf);
return 0;
}
Question 10Answer

a.
Compilation Error

b.
SCARSCAR

c.
AR

d.
SCAR

Correct answer: a. Compilation Error

Short Explanation:

In the code:

```
c

#define scanf "SCAR"
```

This replaces **every occurrence of `scanf`** with the string **`"SCAR"`**.

So this line:

```c
printf(scanf, scanf);
```

Becomes:

```c
printf("SCAR", "SCAR");
```

Now **`"SCAR"`** is treated as the **format string** in **`printf`**, but it **has no format specifiers** (like **`%s`**, **`%d`**, etc.).

However, **`"SCAR"`** is also being passed as a second argument, which **is unused** in the format string.

👉 Most compilers will **allow this with a warning**, but some **strict compilers treat it as an error** due to mismatched format and arguments.

So **technically**, the answer could be:

- **d. SCAR** on lenient compilers (since **`"SCAR"`** is printed)
- But for MCQ exams or standards-based answers, it's usually marked as:

  ✅ **a. Compilation Error**

Let me know if you want to test it online!

What would be the maximum number of hosts per subnet in a class B network with subnet mask 255.255.248.0?
Question 11Answer

a.
2046

b.
1022

c.
2047

d.
1023

Correct answer: a. 2046

🔍 Short Explanation:

- **Class B default subnet mask:** **`255.255.0.0`** → 16 host bits

- **Given subnet mask:** `255.255.248.0`

🧮 Step-by-step:

- `248` in binary = `11111000` → 5 bits used for subnetting
- So, **remaining host bits** = 8 (last octet) – 5 = **3 bits** + full 8 bits of last octet = **11 host bits**
- **Maximum hosts per subnet** = `2^11 – 2` = **2046**

  Subtracting 2 for **network address** and **broadcast address**.

✅ Answer: a. 2046

Which is the most appropriate way of defining primary key?
Question 12Answer

a.
CREATE TABLE table_name PRIMARY KEY (Column_name Constraints, ...);

b.
CREATE TABLE table_name (column_name PRIMARY KEY data_type NOT NULL, ...);

c.
None of the mentioned options

d.
CREATE TABLE table_name(column_name data_type constraint, PRIMARY KEY (column_name), ...);

Correct answer: d. CREATE TABLE table_name(column_name data_type constraint, PRIMARY KEY (column_name), ...);

✅ Short Explanation:

This is the **correct SQL syntax** to define a **primary key** separately after defining the columns:

```sql
CREATE TABLE table_name (
    column_name data_type [constraint],
    ...,
    PRIMARY KEY (column_name)
);
```

- It **clearly separates** column definitions and constraints.
- Allows **composite keys** (e.g., `PRIMARY KEY (col1, col2)`).

🛑 The other options either have syntax errors or incorrect order.

Which of the following is correct as per the classification under Class B Address of IP address?
Question 13Answer

a.
Network bit – 18, Host bit – 16

b.
Network bit – 14, Host bit – 16

c.
Network bit – 12, Host bit – 14

d.
Network bit – 16, Host bit – 14

Correct answer: d. Network bit – 16, Host bit – 14

✅ Short Explanation:

- **Class B IP Address Range:** `128.0.0.0` to `191.255.255.255`
- **Default subnet mask:** `255.255.0.0`
  - This means **16 bits** for **network**
  - And **16 bits** for **host**

But in **default Class B**, the first **2 bits (10)** are fixed → leaves **14 bits** for the **network ID**.

So:

- **Network bits:** 16 (including the fixed 2 bits for Class B)
- **Host bits:** 14 (usable for addressing within each network)

✅ Hence, correct answer: **d. Network bit – 16, Host bit – 14**

Which of the following layers functions as a liaison between user support layers and network support layers?
Question 14Answer

a.
Physical layer

b.
Transport layer

c.
Session layer

d.
Network layer
Question 15

Which of the following OSI model layers are host-to-host layers?
Question 15Answer

a.
Datalink, Network, Transport, Session

b.
Physical, Datalink, Network, Transport

c.
Network, Transport, Session, Presentation

d.
Transport, Session, Presentation, Application

✅ Question 14 Answer: b. Transport layer

Short Explanation:
The **Transport layer** acts as a **liaison (bridge)** between:

- **User support layers** (Session, Presentation, Application)
- And **network support layers** (Network, Data Link, Physical)

It ensures **end-to-end communication**, **reliability**, and **flow control** between systems.

✅ Question 15 Answer: d. Transport, Session, Presentation, Application

Short Explanation:
The **top four layers** of the OSI model are considered **host-to-host layers**, meaning they operate between **end devices**:

1. **Transport** – Reliable delivery (TCP/UDP)
2. **Session** – Manages sessions/connections
3. **Presentation** – Data translation, encryption
4. **Application** – Interface for user apps (HTTP, FTP, etc.)

✅ Correct option: **d**

Which of the following statements correctly defines the operation performed by the short-term scheduler?
Question 16Answer

a.
It selects which process has to be executed next and allocates CPU

b.
It selects which process has to be removed from memory by swapping

c.
It selects which process has to be brought into the ready queue

d.
None of the mentioned options

Correct answer: a. It selects which process has to be executed next and allocates CPU

✅ Short Explanation:

The **short-term scheduler**, also known as the **CPU scheduler**, is responsible for:

- **Selecting a process** from the **ready queue**
- **Allocating the CPU** to that process for execution

  It runs **frequently** (every few milliseconds) to decide which process gets the CPU next.

🔁 Comparison:

- **Long-term scheduler** → brings processes into memory (ready queue)
- **Medium-term scheduler** → swaps processes in/out of memory (for suspension)

✅ So, **short-term scheduler = CPU allocator**.