**Data Structures and Algorithms**

A data structure is a way of organizing and storing data in a computer so that it can be accessed and manipulated easily and efficiently.

Q). Explain the process behind storing a variable in memory.

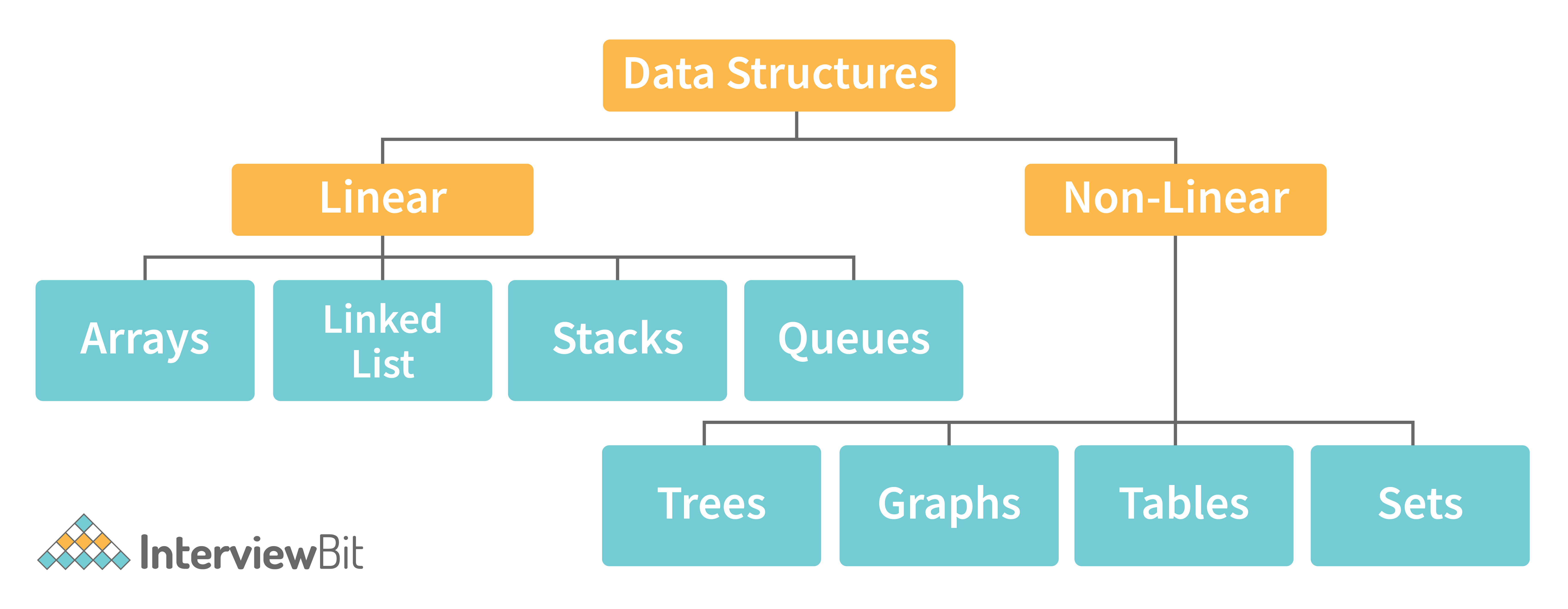
* A variable is stored in memory based on the amount of memory that is needed.
  + The required amount of memory is assigned first.
  + Then, it is stored based on the data structure being used.

Q). Explain the difference between file structure and storage structure?

* File Structure: Representation of data into secondary or auxiliary memory say any device such as a hard disk or pen drive that stores data which remains intact until manually deleted is known as a file structure representation.
* Storage Structure: In this type, data is stored in the main memory i.e RAM, and is deleted once the function that uses this data gets completely executed.

The difference is that the storage structure has data stored in the memory of the computer system, whereas the file structure has the data stored in the auxiliary memory.

Q). What are the types of data structures ?



* Linear Data Structure: A data structure that includes data elements arranged sequentially or linearly, where each element is connected to its previous and next nearest elements, is referred to as a linear data structure. Arrays and linked lists are two examples of linear data structures.
* Non-Linear Data Structure: Non-linear data structures are data structures in which data elements are not arranged linearly or sequentially. We cannot walk through all elements in one pass in a non-linear data structure, as in a linear data structure. Trees and graphs are two examples of non-linear data structures.

Q).What is an array ?

An array data structure is a collection of elements of the same type that are stored in contiguous memory locations. Each element in an array is identified by an index, which is a number that indicates the element's position in the array.

A diagram of a rectangular object with text

Description automatically generated

Q). Classification of Types of Arrays

A diagram of different types of array

Description automatically generated

Fixed Sized Arrays:

We cannot alter or update the size of this array. Here only a fixed size of memory will be allocated for storage. (i,e. the size that is mentioned in square brackets []).

int[] arr1 = new int [5];

int[] arr2 = {1, 2, 3, 4, 5};

Dynamic Sized Arrays:

The size of the array changes as per user requirements during execution of code. The memory is mostly dynamically allocated and de-allocated in these arrays.

ArrayList<Integer> arr = new ArrayList<>();

[One-dimensional array (1-D arrays)](https://www.geeksforgeeks.org/difference-between-one-dimensional-and-two-dimensional-array/):

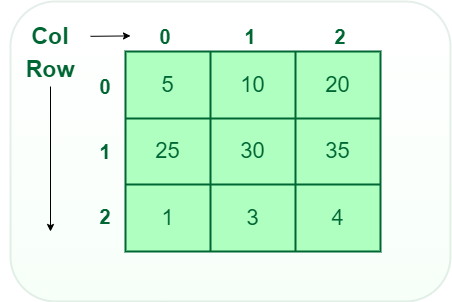
You can imagine a 1d array as a row, where elements are stored one after another.

data\_type array\_name[array\_size];

int nums[5];

Two-dimensional (2D) array:

2D arrays can be considered as an array of arrays or as a matrix consisting of rows and columns.

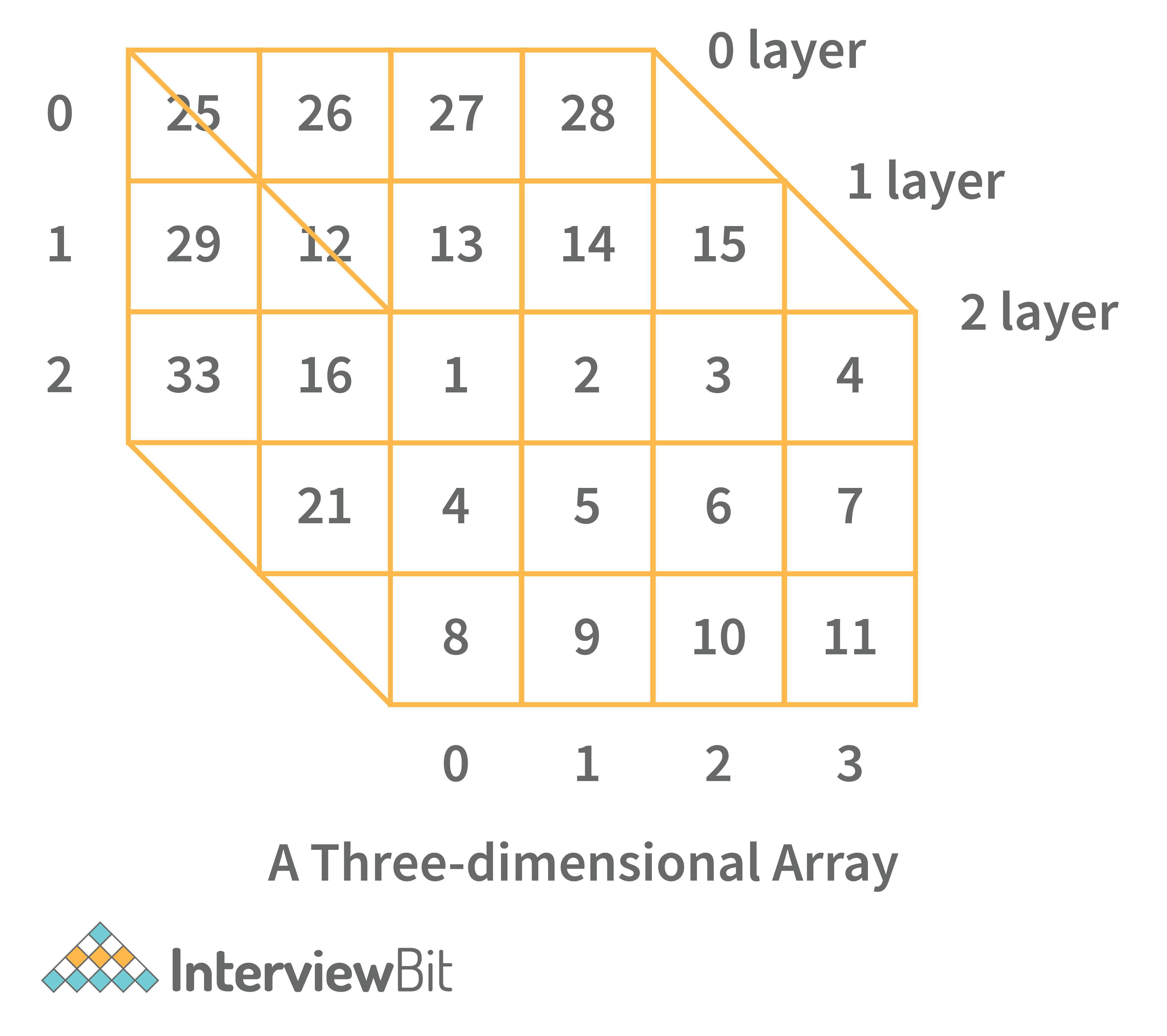


data\_type array\_name[sizeof\_1st\_dimension][sizeof\_2nd\_dimension];

int nums[5][10];

Three-dimensional array:

A 3-D array contains three dimensions, so it can be considered as an array of two-dimensional arrays.



data\_type array\_name[sizeof\_1st\_dimension][sizeof\_2nd\_dimension][sizeof\_3rd\_dimension];

int nums[5][10][2];

Q).Explain the difference between an array and an ArrayList in Java.

An array in Java has a fixed size, while an ArrayList can dynamically resize itself. Arrays can store primitives and objects, whereas ArrayLists can only store objects.

Q). What is an array index out of bounds exception?

This error occurs when an attempt is made to access an element at an index that is outside the bounds of the array (e.g., negative index or greater than the array size).

Q). Can an array be resized at runtime?

In some programming languages, arrays can be resized dynamically, while in others, such as[C,](https://www.geeksforgeeks.org/c-programming-language) the size is fixed.

Q). Explain the concept of a jagged array.

A [jagged array](https://www.geeksforgeeks.org/jagged-array-in-java) is an array of arrays, where each sub-array could be of a different length.

Q). Discuss the advantages and disadvantages of using arrays.

* Advantages: Constant time access, simple implementation, and efficient storage for contiguous data.
* Disadvantages: Fixed size, no support for dynamic growth, inefficient for insertions and deletions.

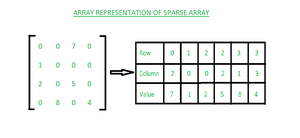
Time complexity : Read – O(1)

Insert – O(n)

Delete – O(n)

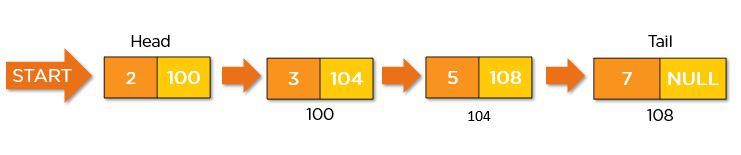
Q). What is meant by Sparse Array?

* A sparse array or sparse matrix is an array in which most of the elements have the same value (the default value is zero or null).
* Elements do not have contiguous indexes starting at zero.
* Sparse arrays are used over arrays when there are lesser non-zero elements. Sparse arrays require lesser memory to store the elements and the computation time can be saved.



Q).What is a linked list ?

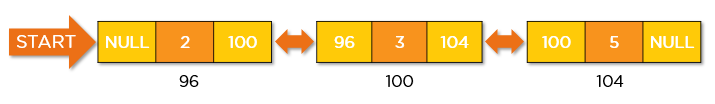
A linked list is a [data structure](https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-data-structure) that stores a sequence of elements. Each element in the list is called a node, and each node has a reference to the next node in the list. The first node in the list is called the head, and the last node in the list is called the tail.



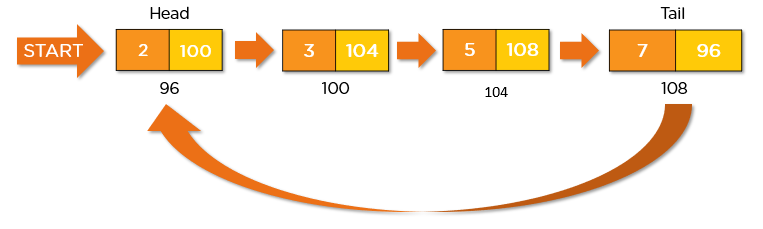
Types of Linked Lists :

A [singly linked list](https://www.simplilearn.com/tutorials/data-structure-tutorial/singly-linked-list) is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node.

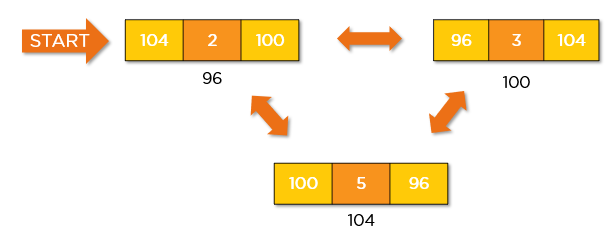
A [doubly linked list](https://www.simplilearn.com/tutorials/data-structure-tutorial/doubly-linked-list) is a bi-directional linked list. So, you can traverse it in both directions. Unlike singly linked lists, its nodes contain one extra pointer called the previous[pointer.](https://www.simplilearn.com/tutorials/c-tutorial/pointers-in-c) This pointer points to the previous node.



A [circular Linked list](https://www.simplilearn.com/tutorials/data-structure-tutorial/circular-linked-list) is a unidirectional linked list. So, you can traverse it in only one direction. But this type of linked list has its last node pointing to the head node. So while traversing, you need to be careful and stop traversing when you revisit the head node.



A circular doubly linked list is a mixture of a doubly linked list and a circular linked list. Like the doubly linked list, it has an extra pointer called the previous pointer, and similar to the circular linked list, its last node points at the head node. This type of linked list is the bi-directional list. So, you can traverse it in both directions.



Time complexity : Read – O(n)

Insert – O(1)

Delete – O(1)

Q). What is stack data structure?

A[stack](https://www.geeksforgeeks.org/stack-data-structure) is a linear data structure that consists of a series of items that are added to the top of the stack and then removed from the top.

LIFO (Last In First Out) or FILO (First In Last Out) are two possible orders.

The common operations on a stack are push (insert an element), pop (remove the top element), and peek (view the top element).

Time Complexity – Push , pop, peek – O(1)

A stack of clothes with text

Description automatically generated

Different operations available in stack:

* push: This adds an item to the top of the stack. The overflow condition occurs if the stack is full.
* pop: This removes the top item of the stack. Underflow condition occurs if the stack is empty.
* top: This returns the top item from the stack.
* isEmpty: This returns true if the stack is empty else false.
* size:  This returns the size of the stack.

### Q).What is a Queue?

A[queue](https://www.geeksforgeeks.org/queue-data-structure) is a linear data structure that follows the **First-In-First-Out (FIFO)** principle, where elements are added at the rear (enqueue) end and removed from the front end (dequeue).

A screenshot of a video game

Description automatically generated

 different operations available in queue :

* enqueue: This adds an element to the rear end of the queue.  Overflow conditions occur if the queue is full.
* dequeue: This removes an element from the front end of the queue. Underflow conditions occur if the queue is empty.
* isEmpty: This returns true if the queue is empty or else false.
* rear: This returns the rear end element without removing it.
* front: This returns the front-end element without removing it.
* size: This returns the size of the queue.

Types of Queues :

* [Simple Queue](https://www.geeksforgeeks.org/introduction-to-queue-data-structure-and-algorithm-tutorials)
* [Circular Queue](https://www.geeksforgeeks.org/introduction-to-circular-queue)
* [Priority Queue](https://www.geeksforgeeks.org/priority-queue-set-1-introduction)
* [Double-Ended Queue (Deque)](https://www.geeksforgeeks.org/deque-set-1-introduction-applications)

time complexity :

Enqueue: O(1)

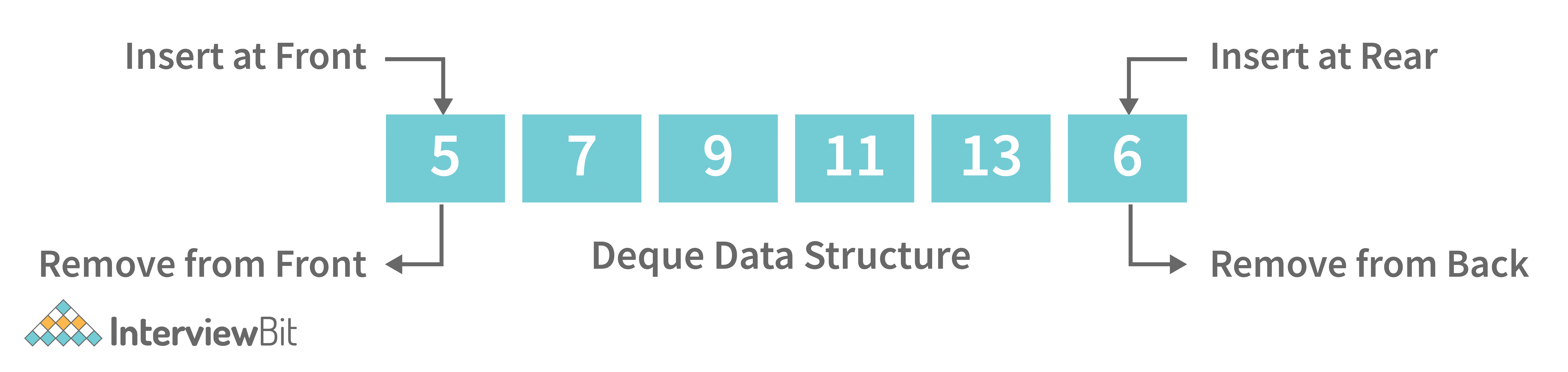
Dequeue: O(1) for simple and circular queues, O(n) for priority queues

Q). What is a priority queue?

A priority queue is a queue where elements are assigned priorities and are dequeued based on their priorities.  If the elements have the same priority, they are served in the order they appear in the queue.

Q). What is a double-ended queue (Deque)?

A[deque](https://www.geeksforgeeks.org/deque-set-1-introduction-applications) is a queue that allows insertions and deletions from both ends.



There are two types of deque:

* Input Restricted Deque: Insertion operations are performed at only one end while deletion is performed at both ends in the input restricted queue.
* Output Restricted Deque: Deletion operations are performed at only one end while insertion is performed at both ends in the output restricted queue.

Q). What is a Tree?

A[tree](https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials) is a non-linear data structure consisting of nodes connected by edges. Each node contains data and[references](https://www.geeksforgeeks.org/references-in-c) to its child nodes. It has one special node called the root, with no parent, and leaf nodes with no children.

Different types of trees :

* [Binary Tree:](https://www.geeksforgeeks.org/binary-tree-data-structure) Each node has at most two children (left and right).
* [Full Binary Tree:](https://www.geeksforgeeks.org/full-binary-tree)Every node except leaves has two children.
* [Complete Binary Tree:](https://www.geeksforgeeks.org/complete-binary-tree)All levels are filled except possibly the last, and nodes are filled left to right.
* [Perfect Binary Tree:](https://www.geeksforgeeks.org/perfect-binary-tree)Every node has two children, and all leaves are at the same level.
* [AVL Tree:](https://www.geeksforgeeks.org/introduction-to-avl-tree)Self-balancing binary search tree with a height difference of at most 1 between subtrees.
* [Red-Black Tree:](https://www.geeksforgeeks.org/introduction-to-red-black-tree)Self-balancing binary search tree with specific coloring rules to maintain balance.
* [B-Tree:](https://www.geeksforgeeks.org/introduction-of-b-tree-2)Generalization of a binary search tree with more than two children per node.

Basic operations performed on a tree:

* Insertion: Add a new node to the tree while maintaining its properties.
* Deletion: Remove a node from the tree while preserving its structure.
* Traversal: Visit each node in the tree exactly once in a specific order (preorder, inorder, postorder).
* Searching: Find a specific node with a given value based on search criteria.

Q). Explain the concept of a binary search tree.

A [binary search tree](https://www.geeksforgeeks.org/binary-search-tree-data-structure)has a specific ordering property: the data in the left subtree is less than the root, and the data in the right subtree is greater than the root. This allows for efficient searching by comparing values with the root and navigating left or right accordingly.

Q). What are tree traversals?

Tree traversal is the process of visiting all the nodes of a tree. Since the root (head) is the first node and all nodes are connected via edges (or links) we always start with that node. There are three ways to traverse a tree :

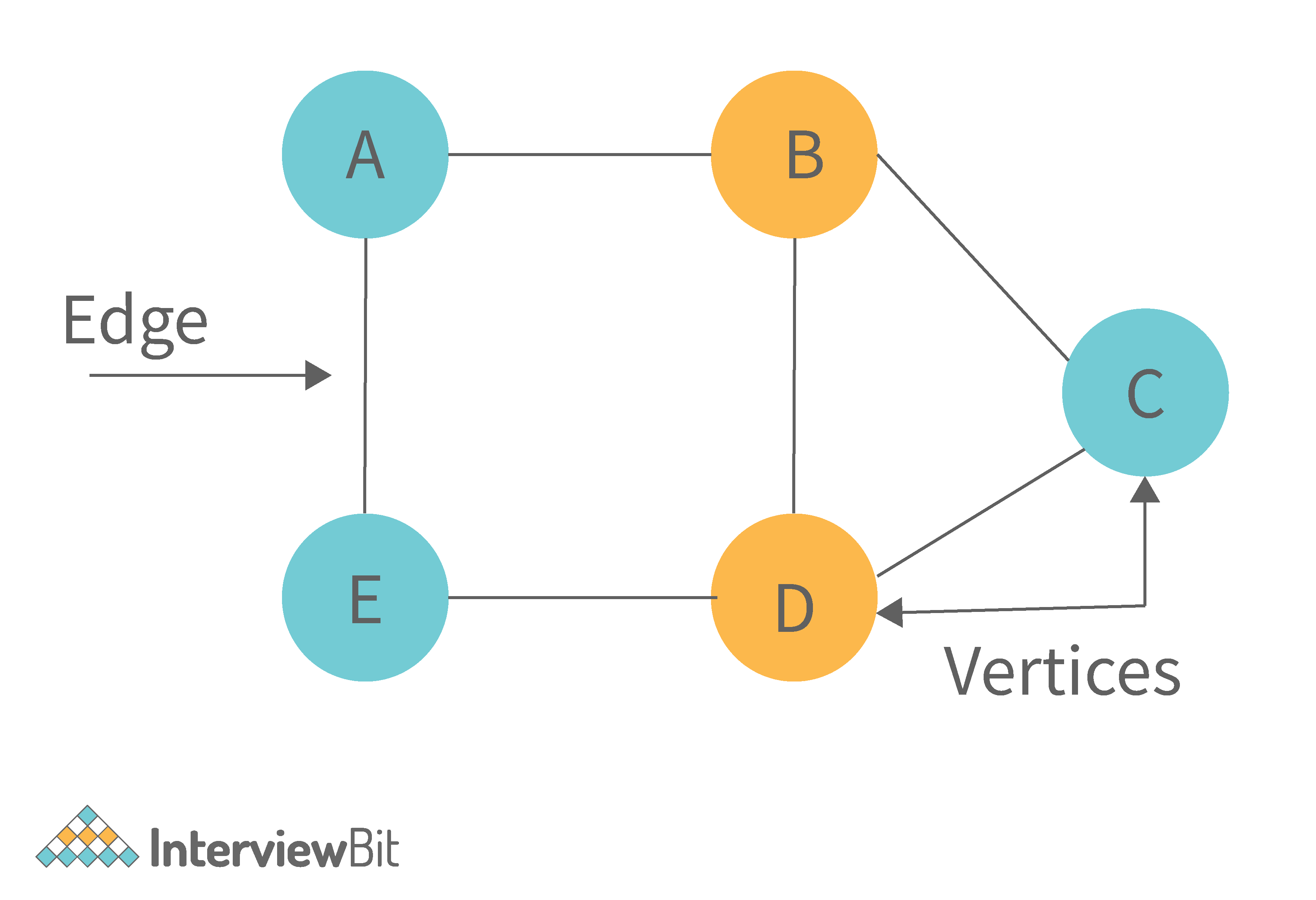
[Preorder:](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder) Visit root, then left subtree, then right subtree. ( Root L R )

[Inorder:](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder) Visit left subtree, then root, then right subtree. ( L Root R )

[Postorder:](https://www.geeksforgeeks.org/postorder-traversal-of-binary-tree)Visit left subtree, then right subtree, then root. ( L R Root )

Q). What is graph data structure and its representations?

A graph is a type of non-linear data structure made up of nodes and edges. The nodes are also known as vertices, and edges are lines or arcs that connect any two nodes in the graph.



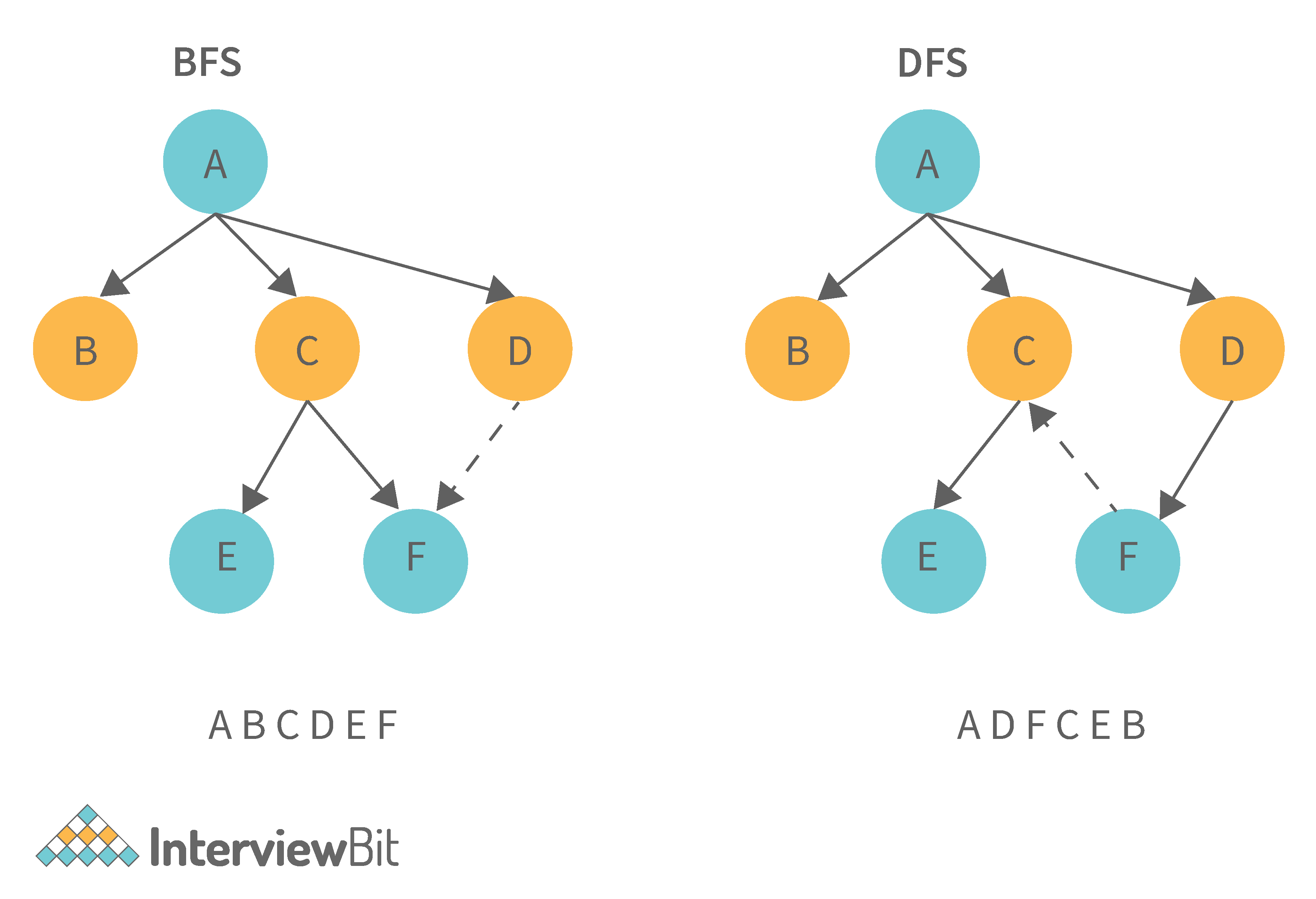
common graph representations.

* [Adjacency matrix:](https://www.geeksforgeeks.org/adjacency-matrix-meaning-and-definition-in-dsa) A 2D array where rows and columns represent nodes, and values indicate the existence of an edge between them. Efficient for space usage, but can be slow for sparse graphs.
* [Adjacency list:](https://www.geeksforgeeks.org/adjacency-list-meaning-definition-in-dsa) An array of linked lists or other data structures, where each list stores nodes connected to a specific node. Efficient for sparse graphs and adjacency queries, but may require more space.

[**Directed graphs:**](https://www.geeksforgeeks.org/what-is-directed-graph-directed-graph-meaning) Edges have a direction, signifying one-way relationships.

[**Undirected graphs:**](https://www.geeksforgeeks.org/what-is-unidrected-graph-undirected-graph-meaning)Edges have no direction, representing bidirectional relationships.

Q). What is the difference between the Breadth First Search (BFS) and Depth First Search (DFS)?



| Breadth First Search (BFS) | Depth First Search (DFS) |
| --- | --- |
| BFS (Breadth First Search) finds the shortest path using the Queue data structure. | DFS (Depth First Search) finds the shortest path using the Stack data structure. |
| We walk through all nodes on the same level before passing to the next level in BFS. | DFS begins at the root node and proceeds as far as possible through the nodes until we reach the node with no unvisited nearby nodes. |
| When compared to DFS, BFS is slower. | When compared to BFS, DFS is faster. |
| BFS performs better when the target is close to the source. | DFS performs better when the target is far from the source. |
| BFS necessitates more memory. | DFS necessitates less memory. |
| Nodes that have been traversed multiple times are removed from the queue. | When there are no more nodes to visit, the visited nodes are added to the stack and then removed. |
| Backtracking is not an option in BFS. | The DFS algorithm is a recursive algorithm that employs the concept of backtracking. |
| It is based on the FIFO principle (First In First Out). | It is based on the LIFO principle (Last In First Out). |

Q). What is AVL tree data structure?

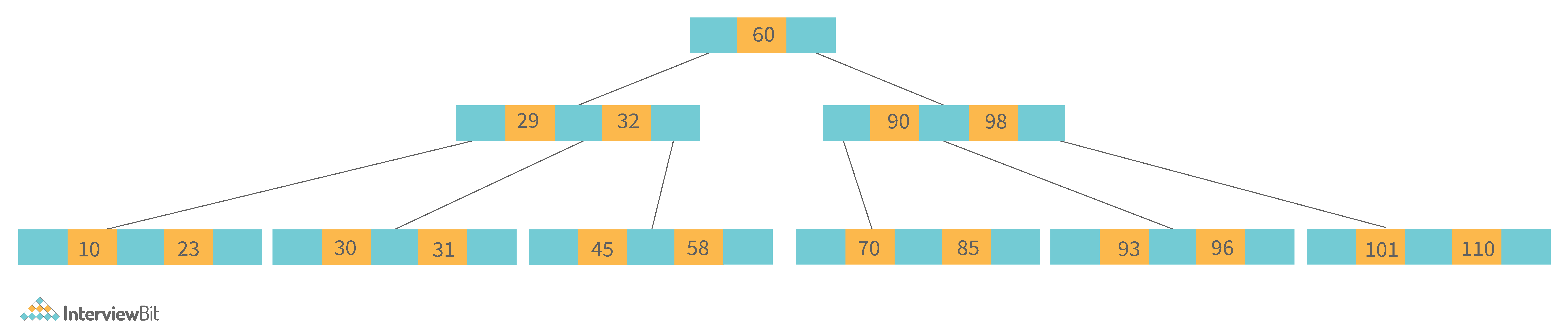
AVL trees are height balancing binary search trees. The AVL tree compares the heights of the left and right subtrees and ensures that the difference is less than one. This distinction is known as the Balance Factor.

BalanceFactor = height(left-subtree) − height(right-subtree)

A diagram of numbers and circles

Description automatically generated

Q). What is a B-tree data structure?

The B Tree is a type of m-way tree that is commonly used for disc access. A B-Tree with order m can only have m-1 keys and m children. One of the primary reasons for using a B tree is its ability to store a large number of keys in a single node as well as large key values while keeping the tree's height relatively small.

Q).What is B+ tree ?

The [B+ tree](https://www.simplilearn.com/tutorials/data-structure-tutorial/b-plus-tree-in-data-structure) is an advanced self-balanced tree since every path from the tree's root to its leaf is the same length. The fact that all leaf nodes are the same length indicates that they all occur at the same level.

Q).  Define Red-Black Tree and its applications

Red Black Trees are a type of self-balancing binary search tree.A red-black tree is a Binary tree in which each node has a colour attribute, either red or black. By comparing the node colours on any simple path from the root to a leaf, red-black trees ensure that no path is more than twice as long as any other, ensuring that the tree is generally balanced

A diagram of a tree

Description automatically generated

Every Red-Black Tree Obeys the Following Rules:

* Every node is either red or black.
* The tree's root is always black.
* There are no two red nodes that are adjacent.
* There is the same number of black nodes on every path from a node to any of its descendant's NULL nodes.
* All of the leaf nodes are black.

Q). Explain minimum spanning trees

It finds a subset of edges that connects all nodes with minimum total weight while avoiding cycles.

Q). What is a heap data structure?

A[heap](https://www.geeksforgeeks.org/heap-data-structure) is a complete[binary tree](https://www.geeksforgeeks.org/binary-tree-data-structure) that satisfies the heap property: each node’s value is greater than or equal to its children’s values.

[Max-heap](https://www.geeksforgeeks.org/introduction-to-max-heap-data-structure) and[min-heap](https://www.geeksforgeeks.org/introduction-to-min-heap-data-structure). In a max-heap, the root node has the maximum value, while in a min-heap, the root node has the minimum value.

Time complexity : Insertion and deletion : O(log n)

Q). What is a hash data structure?

A hash data structure is a data structure that stores key-value[pairs,](https://www.geeksforgeeks.org/pair-in-cpp-stl) where the keys are hashed to determine the location of the value in the data structure.

A HashMap can store at most one null key and multiple null values

A collision occurs when two different keys hash to the same value.

Q). **How does a HashMap handle collisions in Java?**

[HashMap in Java](https://www.hirist.tech/blog/top-20-hashmap-interview-questions-with-answers/) handles collisions by using a technique called chaining. In chaining, elements that hash to the same index are stored in a linked list. If multiple elements hash to the same index, they are added to the same linked list.

Q). What is LinkedHashMap?

A Linked HashMap is a Map implementation in Java that combines the characteristics of a HashMap and a Linked List. In Java, a Linked HashMap is a special type of HashMap that maintains the order in which key-value pairs were inserted.

* Order Preservation: Unlike regular Hash Maps, which don't guarantee any specific order, LinkedHashMaps maintain the order of insertion. When you iterate over a LinkedHashMap, you'll get the elements in the same order as they were added.
* Hashing and Linked List: LinkedHashMaps internally use a combination of hashing and a doubly linked list. The hashing mechanism is used for efficient lookups, while the doubly linked list maintains the insertion order.
* Slower than HashMap: Due to the overhead of maintaining the linked list, LinkedHashMaps can be slightly slower than HashMaps for some operations, especially for frequent insertions and deletions.

Comparison Table: When to Use HashMap vs LinkedHashMap

| Criteria | HashMap | LinkedHashMap |
| --- | --- | --- |
| Order of Elements | Does not maintain any order. | Maintains insertion order or access order (if configured). |
| Performance | Faster for basic operations (put(), get(), etc.). | Slightly slower due to additional linked list for ordering. |
| Memory Usage | More memory efficient (no linked list). | Uses more memory to maintain the linked list structure. |
| When to Use | When order doesn't matter. | When order of insertion or access matters. |
| Use Case Examples | Fast lookups, dictionaries, or hash tables. | Caching systems, ordered collections, or maintaining access history. |

Q). Which data structures are used for implementing LRU cache?

LRU cache or Least Recently Used cache allows quick identification of an element that hasn’t been put to use for the longest time by organizing items in order of use. In order to achieve this, two data structures are used:

* Queue – This is implemented using a doubly-linked list. The maximum size of the queue is determined by the cache size, i.e by the total number of available frames. The least recently used pages will be near the front end of the queue whereas the most recently used pages will be towards the rear end of the queue.
* Hashmap – Hashmap stores the page number as the key along with the address of the corresponding queue node as the value.

Q). What is difference between treeMap and hashMap?

* In Java, both TreeMap and HashMap are part of the Map interface, but they have different internal structures and behaviors. Below are the key differences between TreeMap and HashMap:

| Property | HashMap | TreeMap |
| --- | --- | --- |
| Implementation | Implements HashTable using a hash table. | Implements NavigableMap and is based on a Red-Black tree. |
| Order of Elements | Does not guarantee any specific order of elements. | Maintains elements in a sorted order, according to the natural ordering of the keys or by a comparator provided at the time of creation. |
| Sorting | Does not sort the keys. | Keys are always sorted (ascending order by default or based on a custom comparator). |
| Performance (Time Complexity) | O(1) for basic operations like get(), put(), and remove() (average case). | O(log n) for get(), put(), and remove() due to the underlying Red-Black tree structure. |
| Null Keys and Values | Allows one null key and multiple null values. | Does not allow null keys but allows null values. |
| Thread Safety | Not thread-safe by default. You need to use Collections.synchronizedMap() or other synchronization mechanisms for thread safety. | Not thread-safe by default. Thread safety requires external synchronization, just like HashMap. |
| Use Case | Suitable when you don't care about the order of elements. Best for quick lookups. | Suitable when you need the elements to be ordered (either naturally or via a custom comparator). |
| Key Comparisons | Uses the hash code of the keys for comparisons. | Uses natural ordering or a custom Comparator for key comparisons. |
| Memory Consumption | Generally uses less memory compared to TreeMap. | Uses more memory due to the need to maintain tree structure for ordering. |

Summary:

* HashMap is generally faster and more memory-efficient for basic operations, but it does not maintain any order of elements.
* TreeMap is slower but guarantees sorted keys and is useful when you need elements to be in a specific order or need to perform range operations.

**DBMS**

Q).What is DBMS and what is its utility?

DBMS stands for [Database Management System](https://www.interviewbit.com/blog/components-of-dbms/). It is a set of applications or programs that enable users to create and maintain a database. It is software that enables the storage of data more compactly and securely as compared to a file-based system. A DBMS system helps a user to overcome problems like data inconsistency, data redundancy, etc. in a database.

Q). Explain RDBMS with examples.

RDBMS stands for **Relational Database Management System**. It is used to access and store data more efficiently than DBMS. RDBMS stores data in the form of tables as compared to DBMS which stores data as files. Storing data as rows and columns makes it easier to locate specific values in the database and makes it more efficient as compared to DBMS.

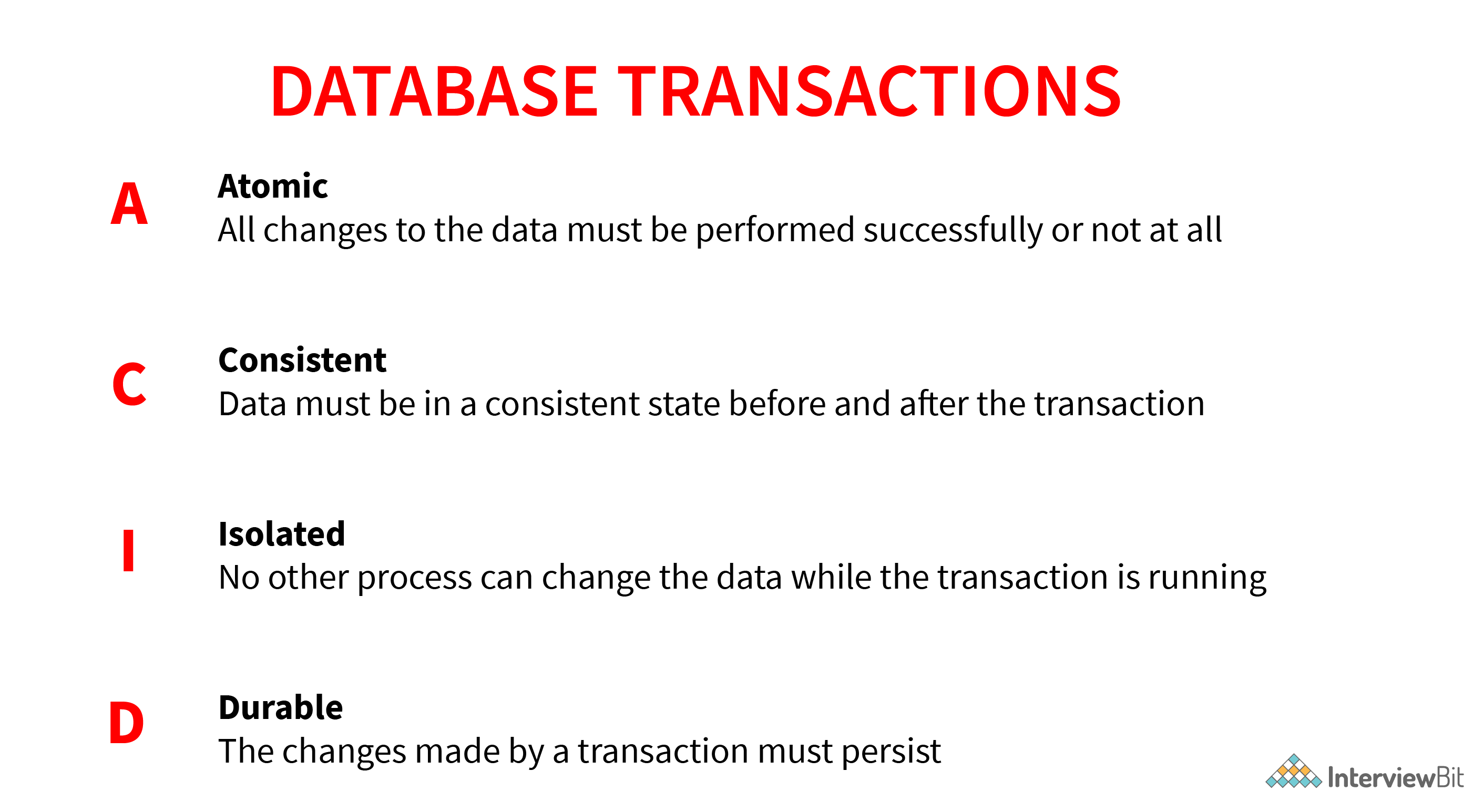
**Examples** of popular RDBMS systems are MySQL, Oracle DB, etc.

**Different languages present in DBMS**

* **DDL(Data Definition Language):**  It contains commands which are required to define the database.  
  E.g., CREATE, ALTER, DROP, TRUNCATE, RENAME, etc.
* **DML(Data Manipulation Language):** It contains commands which are required to manipulate the data present in the database.  
  E.g., SELECT, UPDATE, INSERT, DELETE, etc.
* **DCL(Data Control Language):** It contains commands which are required to deal with the user permissions and controls of the database system.  
  E.g., GRANT and REVOKE.
* **TCL(Transaction Control Language):**  It contains commands which are required to deal with the transaction of the database.  
  E.g., COMMIT, ROLLBACK, and SAVEPOINT

**Q). What is meant by ACID properties in DBMS?**

ACID stands for Atomicity, Consistency, Isolation, and Durability in a DBMS these are those properties that ensure a safe and secure way of sharing data among multiple users.



**Durability:** This property ensures that the data is not lost in cases of a system failure or restart and is present in the same state as it was before the system failure or restart.

Q). **Are NULL values in a database the same as that of blank space or zero?**

No, a NULL value is very different from that of zero and blank space as it represents a value that is unknown, unavailable, or not applicable as compared to blank space which represents a character and zero represents a number.

Q). **What is meant by an entity-relationship (E-R) model? Explain the terms Entity, Entity Type, and Entity Set in DBMS.**

An [**entity-relationship model**](https://www.interviewbit.com/blog/er-model-in-dbms/) is a diagrammatic approach to a database design where real-world objects are represented as entities and relationships between them are mentioned.

* **Entity:** An entity is defined as a real-world object having attributes that represent characteristics of that particular object. For example, a student, an employee, or a teacher represents an entity.
* **Entity Type:** An entity type is defined as a collection of entities that have the same attributes. One or more related tables in a database represent an entity type. For example, a student represents an entity that has attributes such as student\_id, student\_name, etc.
* **Entity Set:** An entity set can be defined as a set of all the entities present in a specific entity type in a database. For example, a set of all the students, employees, teachers, etc. represent an entity set

**Q).** **What is meant by normalization and denormalization?**

[**Normalization**](https://www.scaler.com/topics/normalization-in-dbms/) is a process of reducing redundancy by organizing the data into multiple tables. Normalization leads to better usage of disk spaces and makes it easier to maintain the database.

**Denormalization** is the reverse process of normalization as it combines the tables which have been normalized into a single table so that data retrieval becomes faster. JOIN operation allows us to create a denormalized form of the data by reversing the normalization.

Q). **What is a lock. Explain the major difference between a shared lock and an exclusive lock during a transaction in a database.**

A database lock is a mechanism to protect a shared piece of data from getting updated by two or more database users at the same time. When a single database user or session has acquired a lock then no other database user or session can modify that data until the lock is released.

* **Shared Lock:**A shared lock is required for reading a data item and many transactions may hold a lock on the same data item in a shared lock. Multiple transactions are allowed to read the data items in a shared lock.
* **Exclusive lock:** An exclusive lock is a lock on any transaction that is about to perform a write operation. This type of lock doesn’t allow more than one transaction and hence prevents any inconsistency in the database.

Q). Difference between Drop, Delete and Truncate.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **DROP** | **DELETE** | **TRUNCATE** |
| **Definition** | It completely removes the table from the database. | It removes one or more records from the table. | It removes all the rows from the existing table |
| **Type of Command** | It is a DDL command | It is a DML command | It is a DDL command |
| **Syntax** | DROP TABLE table\_name; | DELETE FROM tble\_nameWHERE conditions; | TRUNCATE TABLE table\_name; |
| **Memory Management** | It completely removes the allocated space for the table from memory. | It doesn’t free the allocated space of the table. | It doesn’t free the allocated space of the table. |
| **Effect on Table** | Removes the entire table structure. | Doesn’t affect the table structure | Doesn’t affect the table structure |
| **Speed and Performance** | It is faster than DELETE but slower than TRUNCATE as it firstly deletes the rows and then the table from the database. | It is slower than the DROP and TRUNCATE commands as it deletes one row at a time based on the specified conditions. | It is faster than both the DELETE and DROP commands as it deletes all the records at a time without any condition. |
| **Use with WHERE clause** | Not applicable as it operates on the entire table | Can be used | It can’t be used as it is applicable to the entire table |

Q). **Explain different types of relationships amongst tables in a DBMS.**

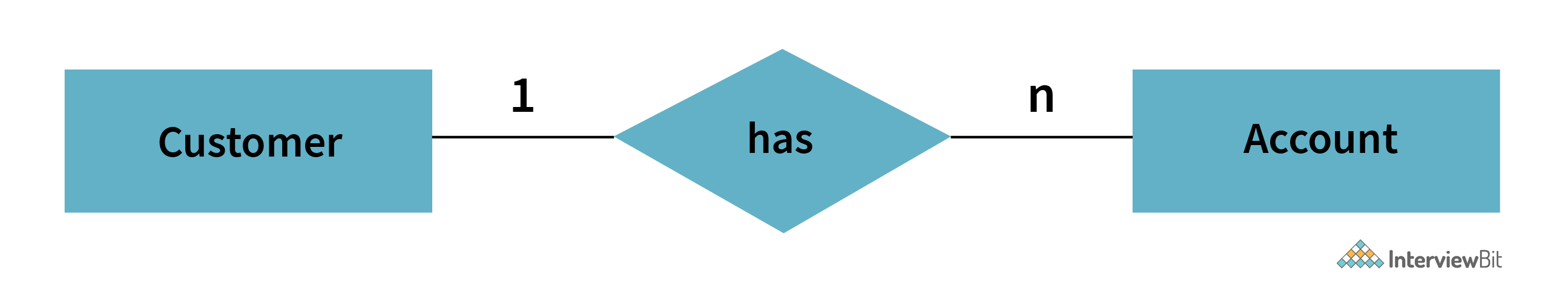
Following are different types of relationship amongst tables in a DBMS system:

* **One to One Relationship:** This type of relationship is applied when a particular row in table X is linked to a singular row in table Y.

A blue diamond with black text

Description automatically generated

* **One to Many Relationship:** This type of relationship is applied when a single row in table X is related to many rows in table Y.

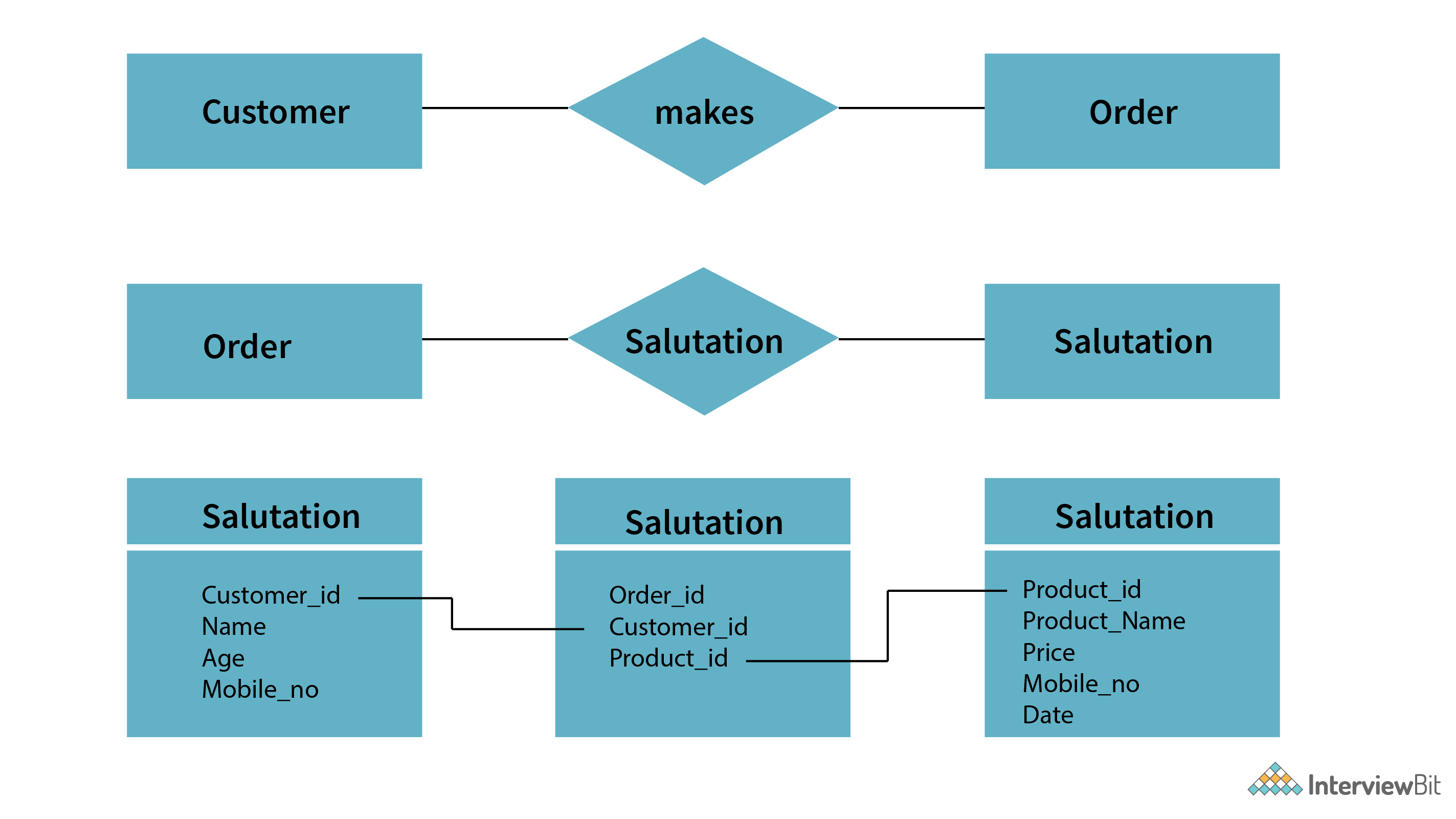


* **Many to Many Relationship:**This type of relationship is applied when multiple rows in table X can be linked to multiple rows in table Y.

A blue diamond with black text

Description automatically generated

* **Self Referencing Relationship:**This type of relationship is applied when a particular row in table X is associated with the same table.



Q). **Explain different levels of data abstraction in a DBMS.**

The process of hiding irrelevant details from users is known as data abstraction. Data abstraction can be divided into 3 levels:

A screenshot of a diagram

Description automatically generated

* **Physical Level:**  it is the lowest level and is managed by DBMS. This level consists of data storage descriptions and the details of this level are typically hidden from system admins, developers, and users.
* **Conceptual or Logical level:**  it is the level on which developers and system admins work and it determines what data is stored in the database and what is the relationship between the data points.
* **External or View level:** it is the level that describes only part of the database and hides the details of the table schema and its physical storage from the users. The result of a query is an example of View level data abstraction.  A view is a virtual table created by selecting fields from one or more tables present in the database

Q). **What is Data Warehousing?**

The process of collecting, extracting, transforming, and loading data from multiple sources and storing them in one database is known as [**data warehousing**](https://www.interviewbit.com/data-warehouse-interview-questions/).

Q). **Explain different types of keys in a database.**

There are mainly 7 types of keys in a database:

* **Candidate Key:** The candidate key represents a set of properties that can uniquely identify a table. Each table may have multiple candidate keys. One key amongst all candidate keys can be chosen as a primary key. In the below example since studentId and firstName can be considered as a Candidate Key since they can uniquely identify every tuple.
* **Super Key:** The super key defines a set of attributes that can uniquely identify a tuple. Candidate key and primary key are subsets of the super key, in other words, the super key is their superset.

A diagram of a key process

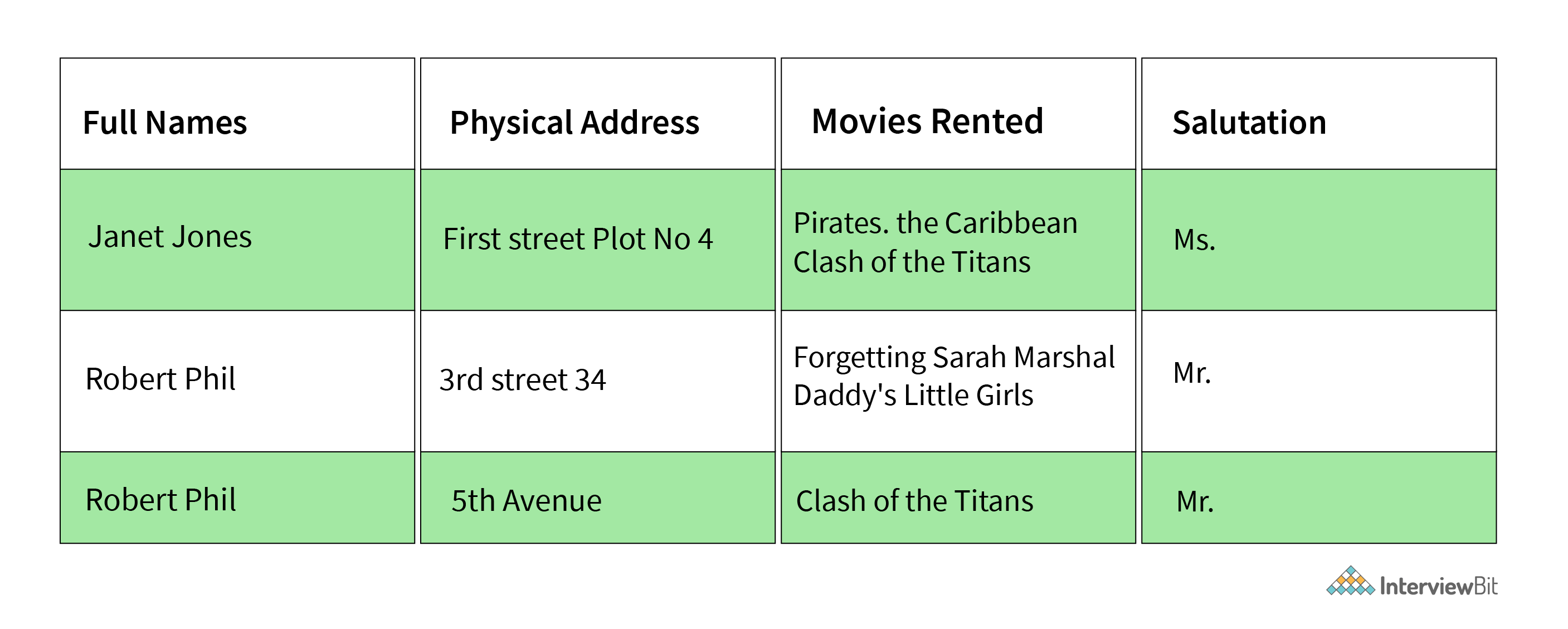
Description automatically generated

* **Primary Key:** The primary key defines a set of attributes that are used to uniquely identify every tuple. In the below example studentId and firstName are candidate keys and any one of them can be chosen as a Primary Key. In the given example studentId is chosen as the primary key for the student table.
* **Unique Key:** The unique key is very similar to the primary key except that primary keys don’t allow NULL values in the column but unique keys allow them. So essentially unique keys are primary keys with NULL values.
* **Alternate Key:** All the candidate keys which are not chosen as primary keys are considered as alternate Keys. In the below example, firstname and lastname are alternate keys in the database.
* **Foreign Key:**  The foreign key defines an attribute that can only take the values present in one table common to the attribute present in another table. In the below example courseId from the Student table is a foreign key to the Course table, as both, the tables contain courseId as one of their attributes.
* **Composite Key:**  A composite key refers to a combination of two or more columns that can uniquely identify each tuple in a table. In the below example the studentId and firstname can be grouped to uniquely identify every tuple in the table.

Q). **Explain different types of Normalization forms in a DBMS.**

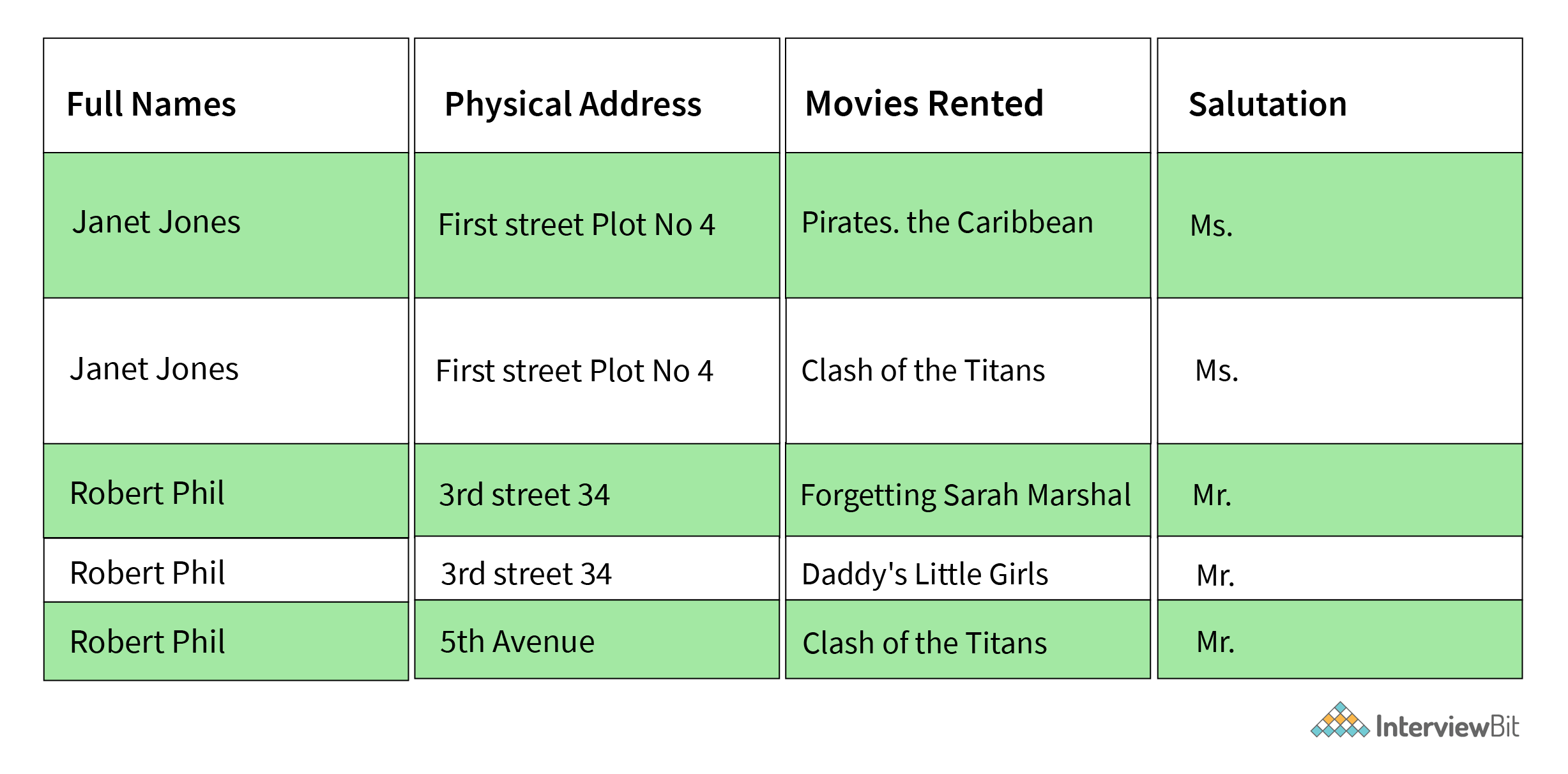
Following are the major normalization forms in a DBMS:

A diagram of a diagram

Description automatically generated with medium confidence

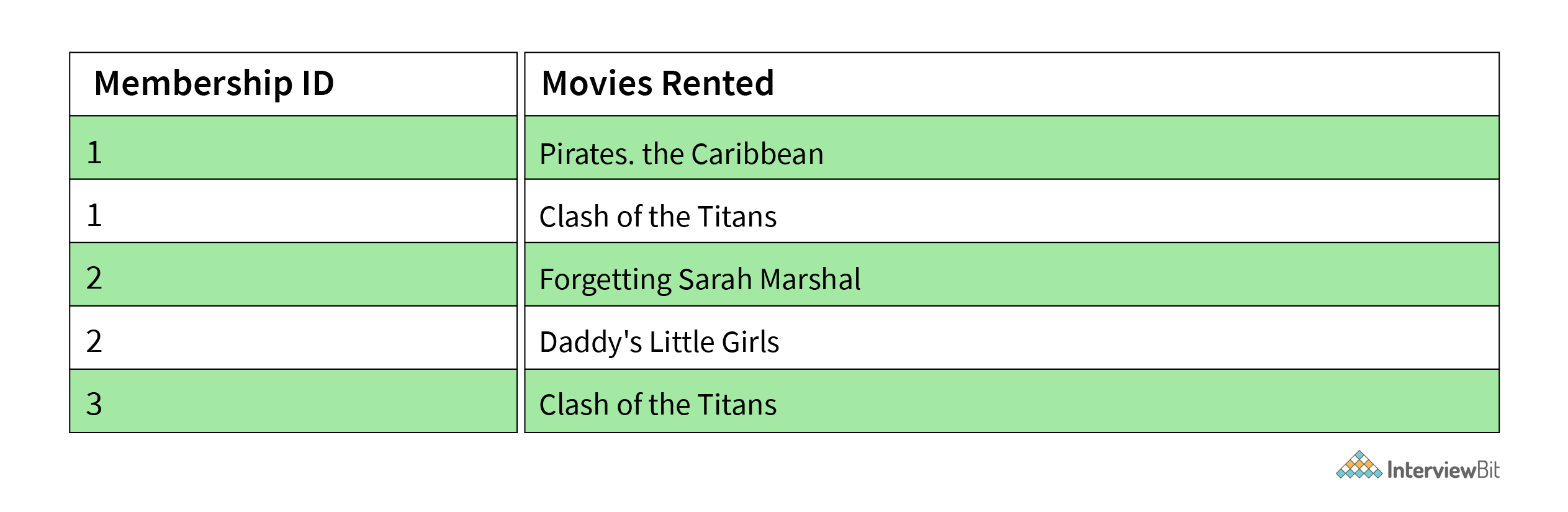
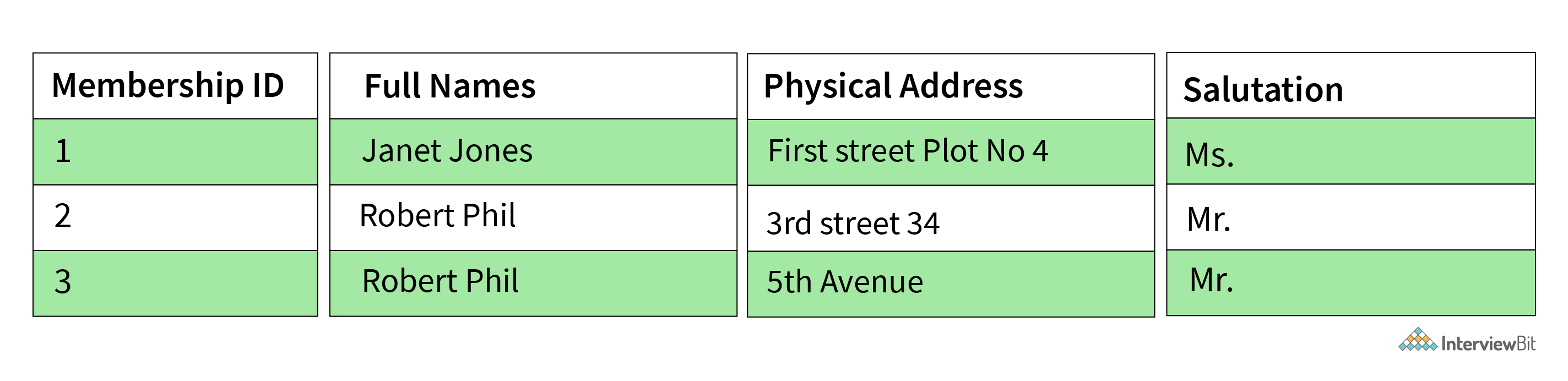
**Considering the above Table-1 as the reference example for understanding different normalization forms.**

* 1NF: It is known as the first normal form and is the simplest type of normalization that you can implement in a database. A table to be in its first normal form should satisfy the following conditions:
  + Every column must have a single value and should be atomic.
  + Duplicate columns from the same table should be removed.
  + Separate tables should be created for each group of related data and each row should be identified with a unique column.

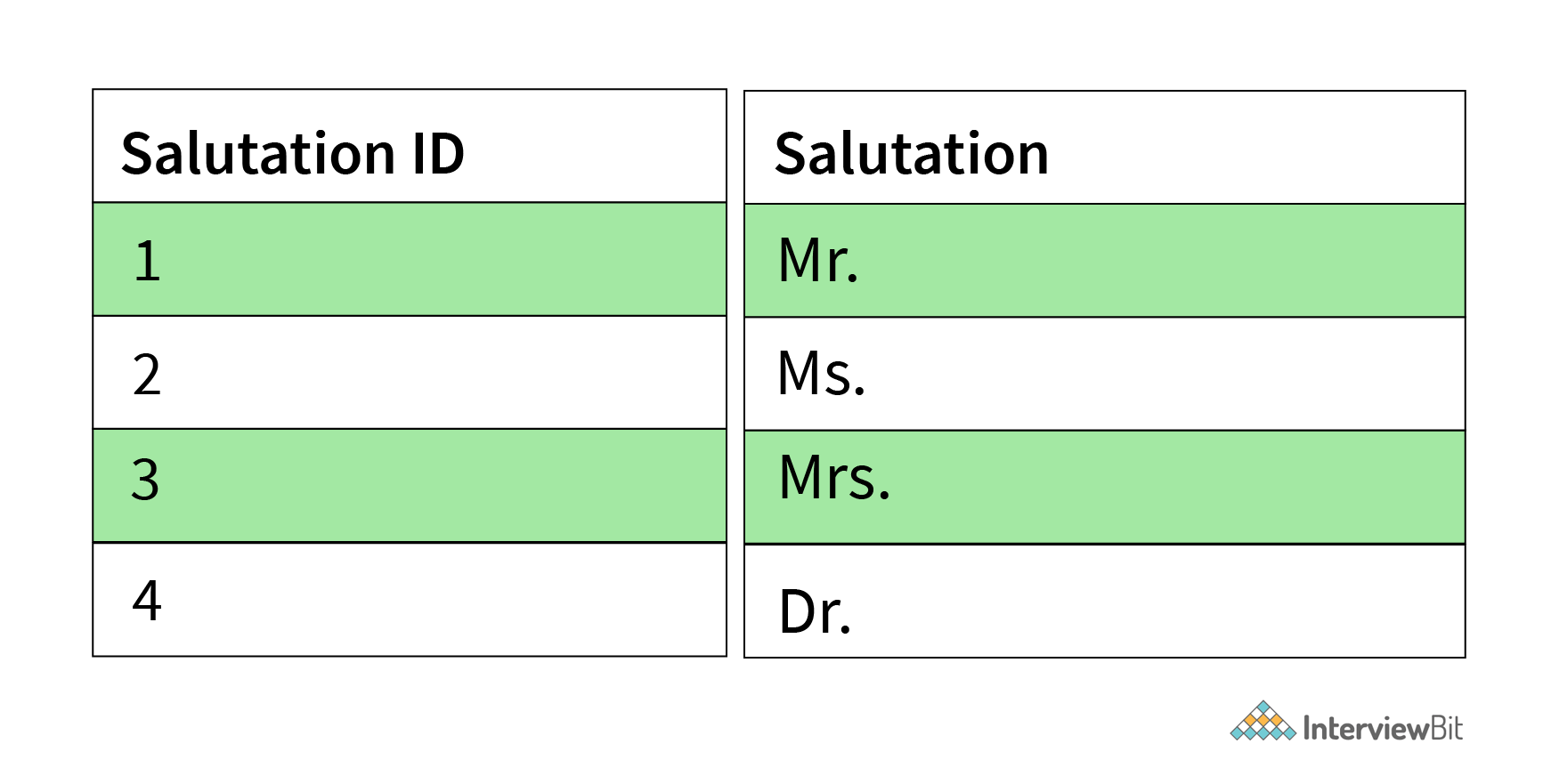
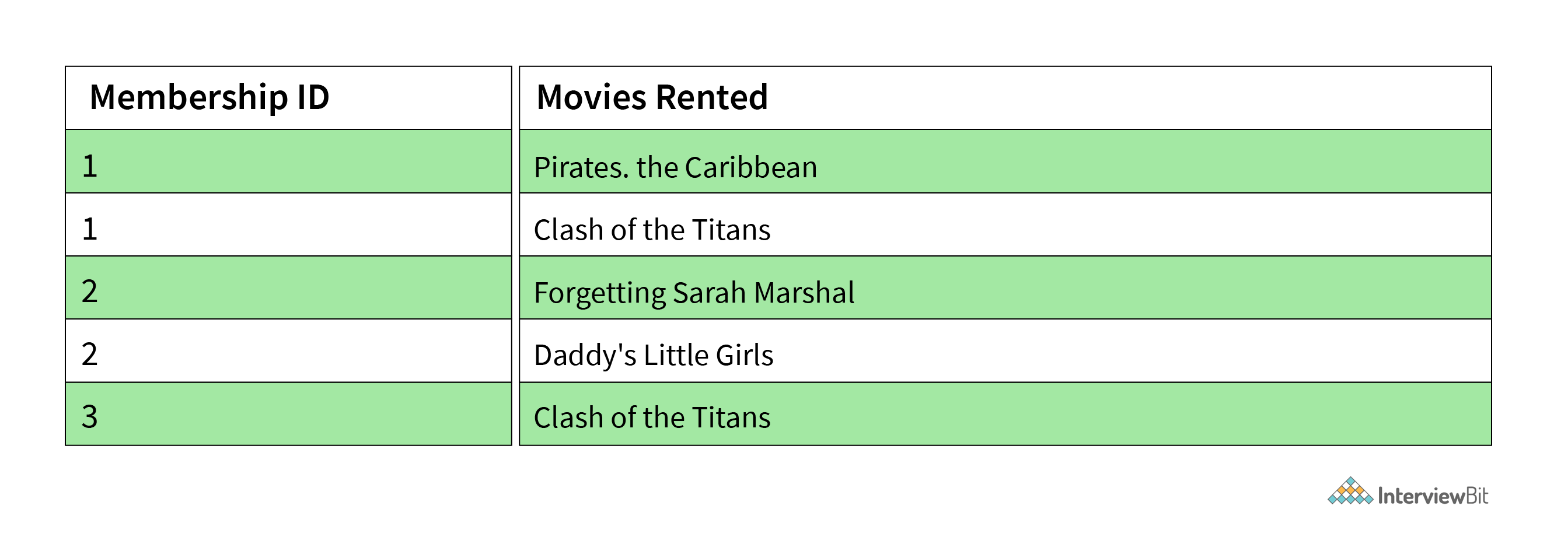
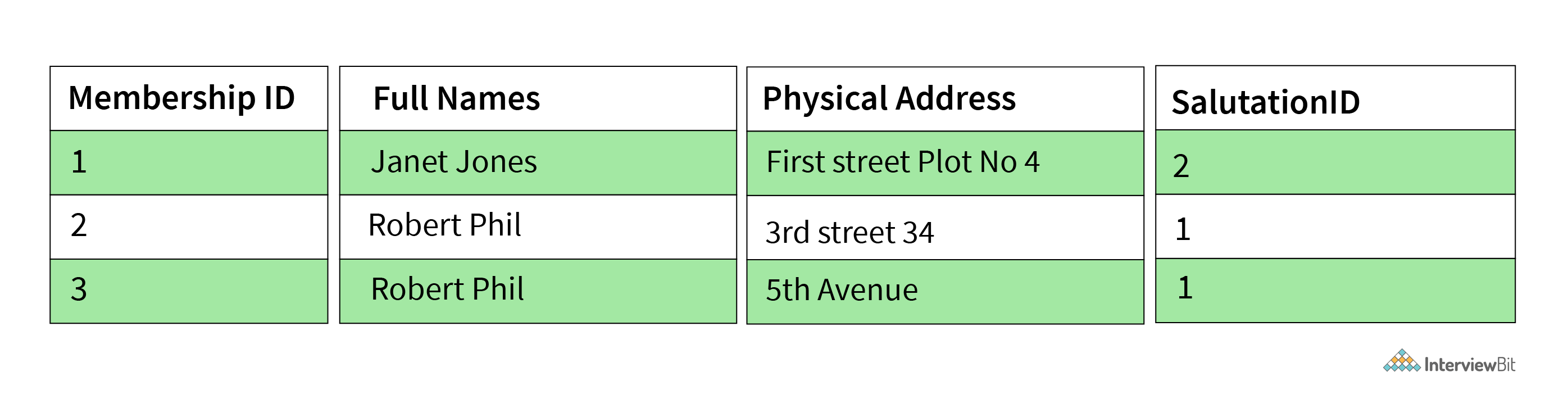


***Table-1 converted to 1NF form***

* **2NF:**It is known as the second normal form. A table to be in its second normal form should satisfy the following conditions:
  + The table should be in its 1NF i.e. satisfy all the conditions of 1NF.
  + Every non-prime attribute of the table should be fully functionally dependent on the primary key i.e. every non-key attribute should be dependent on the primary key in such a way that if any key element is deleted then even the non\_key element will be saved in the database.



***Breaking Table-1 into 2 different tables to move it to 2NF.***

* 3NF: It is known as the third normal form. A table to be in its third normal form should satisfy the following conditions:
  + The table should be in its 2NF i.e. satisfy all the conditions of 2NF.
  + There is no transitive functional dependency of one attribute on any attribute in the same table.

***Breaking Table-1 into 3 different tables to move it to 3NF.***

* **BCNF:** BCNF stands for [Boyce-Codd Normal Form](https://www.scaler.com/topics/bcnf-in-dbms/) and is an advanced form of 3NF. It is also referred to as 3.5NF for the same reason. A table to be in its BCNF normal form should satisfy the following conditions:
  + The table should be in its 3NF i.e. satisfy all the conditions of 3NF.
  + For every functional dependency of any attribute A on B  
    (A->B), A should be the super key of the table. It simply implies that A can’t be a non-prime attribute if B is a prime attribute.

Q). What is a checkpoint in DBMS?

The Checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.

Q). What do you mean by transparent DBMS?

The transparent DBMS is a type of DBMS which keeps its physical structure hidden from users. Physical structure or physical storage structure implies to the memory manager of the DBMS, and it describes how the data stored on disk.

Q). What are the integrity rules in DBMS?

Data integrity is one significant aspect while maintaining the database. So, data integrity is enforced in the database system by imposing a series of rules. Those set of integrity is known as the integrity rules.

**There are two integrity rules in DBMS:**

**Entity Integrity** : It specifies that "Primary key cannot have a NULL value."

**Referential Integrity**: It specifies that "Foreign Key can be either a NULL value or should be the Primary Key value of other relation

Q). What is a stored procedure, and when should it be used?

This is one of the common interview questions on DBMS. A stored procedure is a collection of precompiled [SQL](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql) statements kept in the database. It can be reused and carried out as needed. Stored procedures increase performance by decreasing data transfer between applications and the database, as the logic is executed on the database server.

They are important for automating repetitive activities, increasing security (since users can only run procedures without direct access to tables), and maintaining consistency in complex operations.

Q). What is a deadlock in DBMS, and how can it be avoided?

A deadlock occurs when two or more transactions are waiting on each other to release locks, creating a cycle of dependency that halts progress. Deadlocks can be avoided using several strategies:

* Lock timeout: Automatically rolling back one transaction after a certain period.
* Deadlock prevention: Enforcing a strict order in acquiring locks to prevent cyclic dependencies.
* Deadlock detection and resolution: Identifying deadlocks and resolving them by rolling back one of the involved transactions.

Q). What is a join in SQL, and what are the different types of joins?

A JOIN in SQL is used to combine rows from two or more tables based on a related column between them. The JOIN operation allows us to retrieve data from multiple tables and combine them into a single result set based on a logical relationship between those tables.

INNER JOIN: The INNER JOIN returns only the rows that have matching values in both tables. If there is no match, the row is not included in the result.

* LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and the matched rows from the right table. If there’s no match, NULL values are returned for columns of the right table.
* RIGHT JOIN (or RIGHT OUTER JOIN): This function returns all rows from the right table and the matched rows from the left table. If there’s no match, NULL values are returned for the columns of the left table.
* FULL JOIN (or FULL OUTER JOIN): Returns rows when there is a match in one of the tables. It effectively combines the results of both LEFT JOIN and [RIGHT JOIN](https://www.simplilearn.com/right-outer-join-in-sql-article).
* CROSS JOIN : returns the Cartesian product of two tables, meaning it will combine every row from the first table with every row from the second table. This can result in a very large result set, especially if both tables contain many rows.
* SELF JOIN is a join where a table is joined with itself. This is useful when a table contains hierarchical data or when you need to compare rows within the same table.

Q). Describe the difference between the HAVING and WHERE clause.

* WHERE Clause: This clause filters rows before groupings are made. It applies conditions to individual records in the table(s) involved in the SQL statement. The WHERE clause cannot be used with aggregate functions.
* HAVING Clause: This clause filters groups after applying the GROUP BY clause. It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to filter the results of a GROUP BY operation.

Q). What are aggregate functions in SQL, and can you give some examples?

Aggregate functions perform calculations on a set of values and return a single value. They are commonly used with the GROUP BY clause to summarize data. Some common aggregate functions include:

* COUNT: Counts the number of rows in a result set.
* SUM: Calculates the total of a numeric column.
* AVG: Computes the average value of a numeric column.
* MAX: Finds the maximum value in a column.
* MIN: Retrieves the minimum value in a column.

Q). GROUP BY: Groups rows based on specified columns and allows the use of aggregate functions (e.g., SUM(), AVG(), COUNT()).

ORDER BY: Sorts the results in ascending or descending order based on specified columns.

SUM(): An aggregate function that calculates the total sum of a numeric column or expression.

Q). **What is the main difference between UNION and UNION ALL?**

UNION and UNION ALL are used to join the data from 2 or more tables but UNION removes duplicate rows and picks the rows which are distinct after combining the data from the tables whereas UNION ALL does not remove the duplicate rows, it just picks all the data from the tables.

Q). **What are temporary tables? When are they useful?**  
 Temporary tables exist solely for a particular session, or whose data persists for the duration of the transaction. The temporary tables are generally used to support specialized rollups or specific application processing requirements. Unlike a permanent table, space is not allocated to a temporary table when it is created. Space will be dynamically allocated for the table as rows are inserted. The CREATE GLOBAL TEMPORARY TABLE command is used to create a temporary table in Oracle.

Q).

| Feature | MySQL | NoSQL |
| --- | --- | --- |
| Type | Relational Database (RDBMS) | Non-Relational Database |
| Schema | Fixed, structured schema (SQL) | Flexible schema, often schema-less |
| Query Language | SQL | Varies (MongoDB Query Language, etc.) |
| Data Integrity | ACID-compliant (strong consistency) | BASE model (eventual consistency) |
| Use Cases | Transactional apps, complex queries | Big data, real-time, unstructured data |
| Performance | Slower for large data sets | High throughput, low latency |
| Examples | MySQL, MariaDB, Percona | MongoDB, Cassandra, Redis, Neo4j |
| Transaction Support | Full ACID transactions | Limited, eventual consistency |
| Data Types | Structured (tables, columns) | Unstructured or semi-structured |

**OOPS**

Q). **What is meant by the term OOPs?**

OOPs refers to Object-Oriented Programming. It is the programming paradigm that is defined using objects. An object is a collection of data and the methods which operate on that data. The most widely used OOPS Languages are: Python, Java, C++, C#.

Q).**Explain the 4 pillars of OOP.**

The four pillars of Object-Oriented Programming (OOP) are Abstraction, Encapsulation, Inheritance, and Polymorphism.

Abstraction: Abstraction is the process of hiding complex implementation details and showing only the necessary information to the user. It allows the user to focus on what the object does instead of how it does it. Abstraction is achieved through abstract classes and interfaces.

Encapsulation: Encapsulation is the process of wrapping data and methods into a single unit, called a class. It provides data security by preventing unauthorized access to the data. Encapsulation also helps in code maintenance and reusability.

Inheritance: Inheritance is the process of creating a new class from an existing class. The new class inherits the properties and methods of the existing class. Inheritance allows the creation of a hierarchy of classes, where the child classes inherit the properties and methods of the parent class.

Polymorphism: Polymorphism is the ability of an object to take on many forms. It allows objects of different classes to be treated as if they were objects of the same class. Polymorphism is achieved through method overloading and method overriding.

Q). What are the differences between object-oriented programming and structural programming?

|  |  |
| --- | --- |
| **Object-oriented Programming** | **Structural Programming** |
| It follows a bottom-up approach. | It follows a top-down approach. |
| It provides data hiding. | Data hiding is not allowed. |
| It is used to solve complex problems. | It is used to solve moderate problems. |
| It allows reusability of code that reduces redundancy of code. | Reusability of code is not allowed. |
| It is based on objects rather than functions and procedures. | It provides a logical structure to a program in which the program is divided into functions. |
| It provides more security as it has a data hiding feature. | It provides less security as it does not support the data hiding feature. |
| More abstraction more flexibility. | Less abstraction less flexibility. |
| It focuses on data. | It focuses on the process or logical structure. |

Q). **What is the difference between a class and an object?**

In object-oriented programming, a class is a blueprint or template for creating objects, while an object is an instance of a class. A class defines a set of attributes and methods that are common to all objects of that class. Class is a logical entity that describes the properties and behavior of a group of objects. On the other hand, an object is a physical entity that represents a specific instance of a class. It has its own set of values for the attributes defined in the class and can perform the methods defined in the class. In simpler terms, a class is like a blueprint, while an object is like a building created from that blueprint.

Q). **What are access specifiers and what is their significance?**

Access specifiers are special types of keywords that are used to specify or control the accessibility of entities like classes, methods, and so on. **Private**, **Public**, and **Protected** are examples of access specifiers or access modifiers.  
The key components of OOPs, encapsulation and data hiding, are largely achieved because of these access specifiers.

Q). What are the differences between the constructor and the method in Java?

|  |  |
| --- | --- |
| **Constructor** | **Method** |
| Constructor has the same name as the class name. | The method name and class name are not the same. |
| It is a special type of method that is used to initialize an object of its class. | It is a set of instructions that can be invoked at any point in a program. |
| It creates an instance of a class. | It is used to execute Java code. |
| It is invoked implicitly when we create an object of the class. | It gets executed when we explicitly called it. |
| It cannot be inherited by the subclass. | It can be inherited by the subclass. |
| It does not have any return type. | It must have a return type. |
| It cannot be overridden in Java. | It can be overridden in Java. |
| It cannot be declared as static. | It can be declared as static. |
| Java compiler automatically provides a default constructor. | Java compiler does not provide any method by default. |

Q). What is Coupling in OOP and why it is helpful?

It defines how closely two objects are connected together. There are two types of coupling, **loose** coupling, and **tight** coupling.

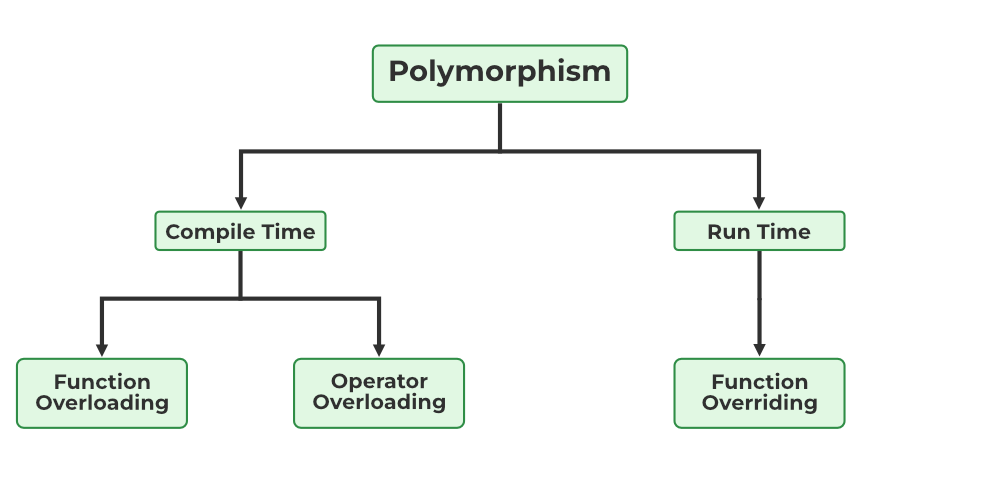
Objects that are independent of one another and do not directly modify the state of other objects is called loosely coupled. Loose coupling makes the code more flexible, changeable, and easier to work with.

Objects that depend on other objects and can modify the states of other objects are called tightly coupled. It creates conditions where modifying the code of one object also requires changing the code of other objects. The reuse of code is difficult in tight coupling because we cannot separate the code.

Q). **What are the different types of Polymorphism?**

Polymorphism can be classified into two types based on the time when the call to the object or function is resolved. They are as follows:

1. Compile Time Polymorphism
2. Runtime Polymorphism



*Types of Polymorphism*

**A) Compile-Time Polymorphism**

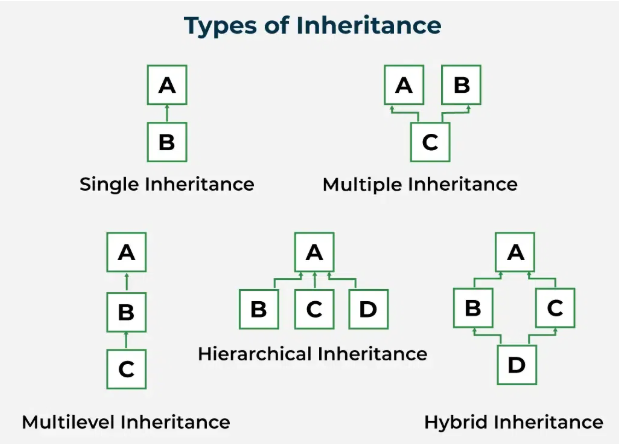
Compile time polymorphism, also known as static polymorphism or early binding is the type of polymorphism where the binding of the call to its code is done at the compile time. **Method overloading** or **operator overloading** are examples of compile-time polymorphism.

**B) Runtime Polymorphism**

Also known asdynamic polymorphismor late binding, runtime polymorphism is the type of polymorphism where the actual implementation of the function is determined during the runtime or execution. **Method overriding** is an example of this method.

Q). **What different types of Inheritance are there?**

Inheritance can be classified into 5 types which are as follows:



1. **Single Inheritance:** Child class derived directly from the base class
2. **Multiple Inheritance:** Child class derived from multiple base classes.
3. **Multilevel Inheritance:**Child class derived from the class which is also derived from another base class.
4. **Hierarchical Inheritance:** Multiple child classes derived from a single base class.
5. **Hybrid Inheritance:** Inheritance consisting of multiple inheritance types of the above specified.

***Note:*** *Type of inheritance supported is dependent on the language. For example, Java does not support multiple inheritance.*

Q). **What is an interface?**

A unique class type known as an interface contains methods but not their definitions. Inside an interface, only method declaration is permitted. You cannot make objects using an interface. Instead, you must put that interface into use and specify the procedures for doing so.

**Q). How is an abstract class different from an interface?**

Both abstract classes and interfaces are special types of classes that just include the declaration of the methods, not their implementation. An abstract class is completely distinct from an interface, though. Following are some major differences between an abstract class and an interface.

| **Abstract Class** | **Interface** |
| --- | --- |
| **An abstract class is a class that cannot be instantiated and is typically used as a base class for other classes to inherit from.** | **An interface, on the other hand, is a collection of abstract methods that can be implemented by any class.** |
| Abstract classes can have constructors. | Interfaces cannot have constructors. |
| A class that is abstract can have both abstract and non-abstract methods. | An interface can only have abstract methods. |
| Fields in an abstract class can have any access modifiers (e.g., private, protected, public) and can be initialized or not. | The interface has only static and final variables. They must be initialized at the time of declaration |
| Abstract class doesn’t support multiple inheritance | An interface supports multiple inheritance. |

Q). **What is Constructor?**

A constructor is a block of code that initializes the newly created object. It has the same name as the class.

Q). **What are the various types of constructors in C++?**

1. **Default Constructor**
2. **Non-Parameterized Constructor**
3. **Parameterized Constructor**
4. **Copy Constructor**

**1. Default Constructor**

The default constructor is a constructor that doesn’t take any arguments. It is a non-parameterized constructor that is automatically defined by the compiler when no explicit constructor definition is provided.

It initializes the data members to their default values.

**2. Non-Parameterized Constructor**

It is a user-defined constructor having no arguments or parameters.

**3. Parameterized Constructor**

The constructors that take some arguments are known as parameterized constructors.

**4. Copy Constructor**

A copy constructor is a member function that initializes an object using another object of the same class.

In Python, we do not have built-in copy constructors like Java and C++ but we can make a workaround using different methods.

**Q).What is a destructor?**

A destructor is a method that is automatically called when the object is made out of scope or destroyed.  
In Java, the garbage collector automatically deletes the useless objects so there is no concept of destructor in Java.

Q). What is the final keyword?

* Final variable: Once initialized, its value cannot be changed.
* Final method: The method cannot be overridden by subclasses.
* Final class: The class cannot be subclassed.

Q). Explain the difference between throw and throws.

* throw: Used to explicitly throw an exception in the code.
* throws: Declares the exceptions that a method might throw. It is used in the method signature.