**Technical QuestionsAndAnswers**

**Return type of executeQuery() & executeUpdate():-**

execute( ) and executeUpdate( ) methods returns **boolean** and **int.**

executeQuery( ), returns a **ResultSet** object.

**Statement vs Prepared Statement:-**

Statement interface cannot accept parameters and useful when you are using static SQL statements at runtime. If you want to run SQL query only once then this interface is preferred over PreparedStatement.  
You can not pass parameters at runtime.  
Performance is very low.  
It is base interface.   
Used to execute normal SQL queries.

It is used when you want to use SQL statements many times.  
The PreparedStatement interface accepts input parameters at runtime.  
You can pass parameters at runtime.  
Performance is better than Statement because it is a precompiled statement i.e. the query is compiled and stored in the database, using place holders (?) instead of values and values to these place holders are supplied later. Thus, avoiding unnecessary compilation and execution of the statement again and again.  
It extends statement interface.  
Used to execute dynamic SQL queries.

**ArrayLis Vs LinkedLIst:-**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

**How to Convert collection to array**

Method of java utility class can be used to change a collection to an array in java. Following methods can be used to convert Collection to arrays:

1. Using list.add() method  
   Syntax: public void add(int index, E element);
2. Using list.toArray() method  
   Syntax: public Object[] toArray() ;

Ex:  
**import** java.util.Arrays;

**import** java.util.ArrayList;

**import** java.util.Arrays;

// Or simply add all generic lava libraries

**import** java.util.\*;

**public** **class** GFG {

    // Main driver method

**public** **static** **void** main(String[] args)

    {

        // Creating arrayList list dynamically

        List<String> list = **new** ArrayList<String>();

        // List is created

        // Adding elements to the list

        list.add("Geeks ");

        list.add("for ");

        list.add("Geeks ");

        list.add("is ");

        list.add("the ");

        list.add("Best.");

        // Converting list to an array

        String[] str = list.toArray(**new** String[0]);

        // Iterating over elements of array

**for** (**int** i = 0; i < str.length; i++) {

            String data = str[i];

            // Printing elements of an array

            System.out.print(data);

        }

    }

}

**Convert Array to SET:-**

**class** GFG {

    // Function to convert array to set

**static** Set<Integer> convert(**int**[] array)

    {

        // Hash Set Initialisation

        Set<Integer> Set = **new** HashSet<>();

        // Iteration using enhanced for loop

**for** (**int** element : array) {

            Set.add(element);

        }

        // returning the set

**return** Set;

    }

    // Function to print the set

**static** **void** print(Set<Integer> Set)

    {

        // Implement to iterator the Set

        Iterator<Integer> \_iterator = Set.iterator();

        // Iterate the elements of Set

**while** (\_iterator.hasNext()) {

            // print the element of the Set

            System.out.print(\_iterator.next() + " ");

        }

    }

**public** **static** **void** main(String[] args)

    {

        // Array taken for consideration

**int** array[] = { 1, 2, 3, 4, 5, 6 };

        // Calling function to convert the array

        Set<Integer> Set = convert(array);

        // print the set

        print(Set);

    }

}

**Types of JDBC Driver:-**

1. Type 1 - JDBC-ODBC bridge driver
2. Type 2 - Native-API driver (partially java driver)
3. Type 3 - Network Protocol driver
4. Type 4 – Database Protocol Driver / Thin driver (fully java driver)

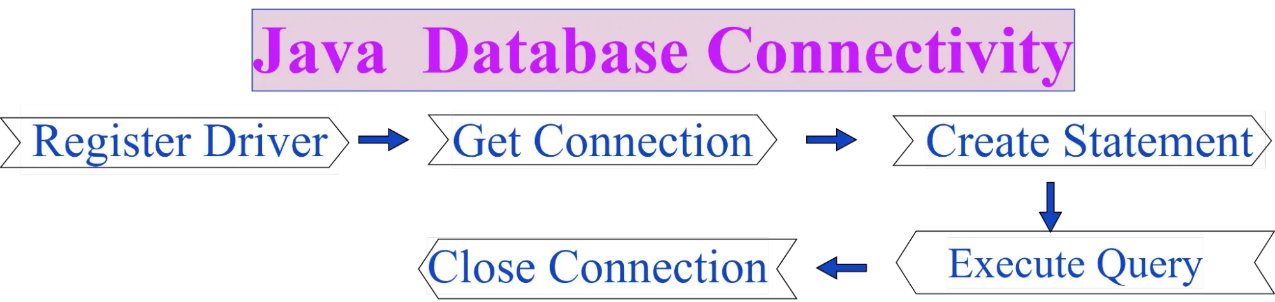
The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Mockito: Why not use junit: Junit is non-persistent whereas Mockito is persistent. So, Data should be presistent means Actual database should not change while testing.

What is the default type of ResultSet in JDBC?  
 The default ResultSet type is TYPE\_FORWARD\_ONLY.

JDBC components

The JDBC core comes with the following interfaces and classes:

* *Driver:* This is the interface that controls communication with the database server. It also withdraws information associated with driver objects.
* *Driver Manager:* It manages any required set of JDBC drivers
* *Connection:* This is an interface or session that houses all the methods to connect to any database.
* *Statements:* This is used to carry out a static SQL statement
* *ResultSet:* This is used to access the result row-by-row  
  

**HashMap Vs HashSet**

| **HashMap** | **HashSet** |
| --- | --- |
| Hashmap is the implementation of Map interface. | Hashset on other hand is the implementation of set interface. |
| Hashmap internally do not implements hashset or any set for its implementation. | Hashset internally uses Hashmap for its implementation. |
| HashMap Stores elements in form of key-value pair i.e each element has its corresponding key which is required for its retrieval during iteration. Syntax:  HashMap<Integer, String> hm = new HashMap<Integer, String>(); | HashSet stores only objects no such key value pairs maintained.   Syntax:  HashSet<String> hs = new HashSet<String>(); |
| Put method of hash map is used to add element in hashmap. | On other hand add method of hashset is used to add element in hashset. |
| Single null key and any number of null value can be inserted in hashmap without any restriction. | On other hand Hashset allows only one null value in its collection,after which no null value is allowed to be added. |

**Primary Key Vs Foreign Key:-**

Primary Key: A primary key is used to ensure that data in the specific column is unique. A column cannot have NULL values. It is either an existing table column or a column that is specifically generated by the database according to a defined sequence.

Foreign Key: A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It is a column (or columns) that references a column (most often the primary key) of another table.   
Example: Refer the figure – STUD\_NO in STUDENT\_COURSE is a foreign key to STUD\_NO in STUDENT relation.

**Primary Key Vs Unique Key:**

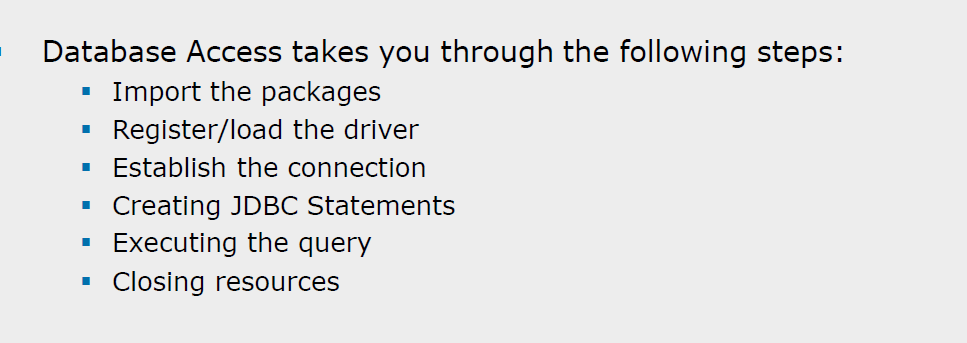
A **primary key** is a column of table which uniquely identifies each tuple (row) in that table. Primary key enforces integrity constraints to the table. Only one primary key is allowed to use in a table. The primary key does not accept the any duplicate and NULL values. The primary key value in a table changes very rarely so it is chosen with care where the changes can occur in a seldom manner. A primary key of one table can be referenced by foreign key of another table.   
Cannot accept NULL values.  
Only one primary key  
A Primary key supports auto increment value.  
We cannot change or delete values stored in primary keys.

**Unique key** constraints also identifies an individual tuple uniquely in a relation or table. A table can have more than one unique key unlike primary key. Unique key constraints can accept only one NULL value for column. Unique constraints are also referenced by the foreign key of another table. It can be used when someone wants to enforce unique constraints on a column and a group of columns which is not a primary key.Can accept NULL values.  
More than one primary key.  
A unique key does not supports auto increment value.  
We can change unique key values.

**Types of SQL Command:-**

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.  
Data Definition Language  
Data Manipulation Language  
Data Control Language  
Transaction Control Language  
Data Query Language

 **Rollback**: Rollback command is used to undo transactions that have not already been saved to the database.



**Statement Vs Prepared Statement Vs Callable Statement**

|  |  |
| --- | --- |
| Statement | Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters. |
| PreparedStatement | Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime. |
| CallableStatement | Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters. |