ESTRUCTURA

1.- Fundamentos Teóricos de OOP.  
   - Definir que es el: Polimorfismo, herencia, encapsulación.  
   - Que es una clase abstracta interfaz y para q se usan.  
   - Que es una clase y u objeto.

- Clases Internas y Locales.  
  
2.- Preguntas Técnicas del Lenguaje en cuestión (C++, Java).  
   - como se declaran constantes.  
   - como se utilizan las hebras, excepciones, etc...  
   - como definir una clase abstracta, método abstracto, interfaz, método abstracto.  
   - q son las clases, atributos o métodos finales y  como se definen.

- Clases Internas.

- Tipos de Casting en C++.  
  
3.- Bases de Datos. (Allí usan MySQL con paquete XSQL).  
   - como harías un trigger o un procedimiento PL/SQL.

- Conceptos: Primary Key, Foreign Key, Índices, Problemas asociados al definir muchos índices, ….  
   - como detectas que una consulta va lenta y q hacer para optimizarla.

BLOQUE 1: POO

1. **Definición**:

La POO es una metodología de programación que consiste en la abstracción de un problema en base a un conjunto de objetos que interactúan entre sí. Los pilares básicos de la POO son: Abstracción, Herencia, Polimorfismo y la Encapsulación.

1. **Abstracción**:

Capacidad de definir algo del mundo real a través de objetos que mantienen lo esencial para el problema en cuestión y lo que los diferencian unos de otros.

1. **Herencia**:

Establece una jerarquía de clases: SuperClase – SubClase.

La herencia permite que se puedan definir nuevas clases basadas en clases existentes, lo cual facilita re-utilizar código previamente desarrollado. Si una clase deriva de otra hereda todas sus variables y métodos. La clase derivada puede ***añadir*** nuevas variables y métodos y/o ***redefinir*** las variables y métodos heredados.

1. **Polimorfismo**:

Los objetos de distintas clases pertenecientes a una misma jerarquía o que

implementan una misma interface pueden tratarse de una forma general e individualizada, al mismo tiempo.

1. **Encapsulación**:

La encapsulación es un mecanismo que consiste en organizar datos y métodos de un objeto, evitando el acceso a datos por cualquier otro medio distinto a los especificados. Por lo tanto, la encapsulación garantiza la integridad de los datos que contiene un objeto.

Esta ligada a la ocultación y por tanto a la restricción del acceso.

Las clases pueden ser declaradas como públicas (***public***) y como ***package***

(accesibles sólo para otras clases del ***package***). Las variables miembro y los métodos pueden ser ***public***, ***private***, ***protected*** y ***package***. De esta forma se puede controlar el acceso y evitar un uso inadecuado.

1. **Sobrecarga**:

Consiste en definir métodos con el mismo nombre pero con parámetros diferentes, ya sea en número o tipos. Es una alternativa al polimorfismo estático o por vinculación temprana.

1. **Objeto**:

Es una abstracción de algo que tiene propia identidad en el dominio de un problema a solucionar con POO y que se compone de:

Atributos: datos asociados, características o propiedades. Sus valores concretos determinan el estado. Pueden ser propios de cada objeto o comunes a todos. También se pueden llamar variables.

Métodos: comportamiento que tiene el objeto relacionado con la forma de tratar sus atributos, que a su vez define la interface a la hora de interactuar con el mismo. Estos comportamientos pueden ser propios de cada objeto o comunes al mismo “Tipo” de objetos.

Un objeto es una instancia de una clase.

1. **Clases**:

Las clases son la definición formal de los objetos , es decir, el patrón con el que se crean.

1. **Clase Abstracta**:

Es una clase que se utiliza únicamente para la definición de otras en la jerarquía según la herencia. No permiten que se instancien objetos.

1. **Interfaz**:

La interfaz de un objeto son el conjunto de métodos con los que se puede interactuar con dicho objeto.

Con respecto a los lenguajes de POO representa la definición de un conjunto de métodos que se deben implementar en la clase que lo implemente. En c++ y java se tratan de forma diferente, no existiendo propiamente el concepto en c++ debiéndose utilizar clases abstractas.

Son útiles en términos de polimorfismo y además en java son una alternativa a la herencia múltiple implementada de serie en c++.

BLOQUE 2: DETALLES TÉCNICOS DE JAVA Y C++

La principal diferencia entre ambos lenguajes es que en la práctica JAVA define cada clase en un fichero “.java” en el nombre de la clase pública debe coincidir con el del respectivo fichero. En C++ no existe esta restricción y permite la creación de las clases que se quieran en un mismo fichero, pero desdobla la definición e implementación en dos ficheros: “.hpp” o “.h” y “.cpp”.

1. Clases:

|  |  |
| --- | --- |
| C++ | JAVA |
| **.h:**  **Namespace NombreNameEspace**  **{**  **class** <identificador de clase> [<:lista de clases base>] {  <lista de miembros>  } [<lista de identificadores de objetos>];  }  Las clase por defecto son públicas. | package nombrePaq;  import java.lang.\*;  [public] **class** Classname extends SuperClase implements Interface  {  // definición de variables y métodos  ...  }  Nota: Las clases pueden ser públicas o de paquete (accesible por clases del mismo package, opción por defecto) |

1. Interfaces:

|  |  |
| --- | --- |
| C++ | JAVA |
| **.h:**  **class** <identificador de clase> [<:lista de clases base>] {  **//TODAS LOS MÉTODOS SON VIRTUALES PUROS =0.**  }; | import java.awt.Graphics;  public interface Dibujable  {  public void setPosicion(double x, double y);  public void dibujar(Graphics dw);  } |

1. Clases Abstractas:

|  |  |
| --- | --- |
| C++ | JAVA |
| **.h:**  **class** <identificador de clase> [<:lista de clases base>] {  **//COMO MÍNIMO UN MÉTODO VIRTUAL PURO.**  }; | public abstract class NombreClass  { ... }  NOTA:  - Puede tener unos o todos lo métodos abstractos y no se definen.  - Se definen con “abstract”.  - Métodos static no pueden ser abst. |

1. Clases/Atributos/Métodos Finales:

|  |  |
| --- | --- |
| C++ | JAVA |
| **Atributo Final o Constante:** Atributo que no puede cambiar. | |
| Palabra reservada “const”. | Palabra reservada “final”. Si no se inicializa a partir de la primera vez que se instancia adquiere un valor constante. |
| **Clase Final: Clase que no puede ser clase derivada o superclase de otra.** | |
| **NO EXISTE DIRECTAMENTE** | public final class NombreClass  { ... } |
| **Método Final: Método que no se puede redefinir.** | |
| **NO EXISTE** | public final void nombreMetodo()  { ... } |

1. Excepciones:

Una excepción es un error o malfuncionamiento en tiempo de ejecución. El tratamiento o manejo de excepciones consiste en controlar estas situaciones e intentar tomar una determinación al respecto.

|  |  |
| --- | --- |
| C++ | JAVA |
| **try, catch y throw.**  Tipos primitivos o clases. Normalmente se derivan de la clase exception.  En la definición del método se puede indicar las excepciones que se disparan.  class exception {  public:  exception() throw() { }  virtual ~exception() throw();  virtual const char\* what() const throw();  };  Excepciones predefinidas heredan de exception:  std::bad\_alloc // Al operador new std::bad\_cast // Al operador dynamic\_cast<> std::bad\_typeid // Al operador typeid std::bad\_exception // Cuando se viola una especificación  Captura genérica catch(…)  No existe jerarquía en las excepciones como en java, al lanzarse se busca de dentro a fuera el catch que captura el tipo en concreto.  Ejemplo:  try {  x = new int[y];  delete[] x;  }  catch(std::bad\_alloc&) {  *cout* << "Memoria insuficiente" << *endl*;  } | ***try***,  ***catch***, ***throw***, ***throws*** y ***finally***  En JAVA las excepciones se establecen con clases que derivan de Exception que a su vez deriva de Throwable dentro del paquete java.lang.  Tipos de excepciones:   * **Implícitas**: a las que no se obligan a capturar y que se gestionan con RuntimeException. Pe: java.lang.NullpointerException. * **Explícitas**: el resto, que si obliga a gestionar. |

1. Hebras:

|  |  |
| --- | --- |
| C++ | JAVA |
| POSIX: pthread  WINDOWS: ws2\_32.dll |  |

1. Sockets:

C++. SERVER.

/\* Estos son los ficheros de cabecera usuales \*/

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#define PORT 3550 /\* El puerto que será abierto \*/

#define BACKLOG 2 /\* El número de conexiones permitidas \*/

main()

{

int fd, fd2; /\* los ficheros descriptores \*/

struct sockaddr\_in server;

/\* para la información de la dirección del servidor \*/

struct sockaddr\_in client;

/\* para la información de la dirección del cliente \*/

int sin\_size;

/\* A continuación la llamada a socket() \*/

if ((fd=socket(AF\_INET, SOCK\_STREAM, 0)) == -1 ) {

printf("error en socket()\n");

exit(-1);

}

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

/\* ¿Recuerdas a htons() de la sección "Conversiones"? =) \*/

server.sin\_addr.s\_addr = INADDR\_ANY;

/\* INADDR\_ANY coloca nuestra dirección IP automáticamente \*/

bzero(&(server.sin\_zero),8);

/\* escribimos ceros en el reto de la estructura \*/

/\* A continuación la llamada a bind() \*/

if(bind(fd,(struct sockaddr\*)&server,

sizeof(struct sockaddr))==-1) {

printf("error en bind() \n");

exit(-1);

}

if(listen(fd,BACKLOG) == -1) { /\* llamada a listen() \*/

printf("error en listen()\n");

exit(-1);

}

while(1) {

sin\_size=sizeof(struct sockaddr\_in);

/\* A continuación la llamada a accept() \*/

if ((fd2 = accept(fd,(struct sockaddr \*)&client,

&sin\_size))==-1) {

printf("error en accept()\n");

exit(-1);

}

printf("Se obtuvo una conexión desde %s\n",

inet\_ntoa(client.sin\_addr) );

/\* que mostrará la IP del cliente \*/

send(fd2,"Bienvenido a mi servidor.\n",22,0);

/\* que enviará el mensaje de bienvenida al cliente \*/

close(fd2); /\* cierra fd2 \*/

}

}

BLOQUE 4: ORM SQL

Glosario

¿Qué es un ORM?

Object-Relational mapping, o lo que es lo mismo, **mapeo de objeto-relacional**, es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Desde hace muchos años el lenguaje más usado para acceder a las bases de datos relacionales ha sido el SQL. ¿Por qué entonces cambiar y pasarse a un ORM?

**¿Por qué usar un ORM?**

Aunque el lenguaje SQL se usa para acceder a muchas de las bases de datos existentes, existen múltiples varianzas en las funciones que los distintos SGBD han usado. Un ejemplo muy sencillo sería delimitar el número de registros de una consulta:

SELECT TOP 10 \* FROM usuarios //SqlServer

SELECT \* FROM usuarios LIMIT 10 //MySQL

SELECT \* FROM usuarios WHERE rownum<=20; //Oracle

Tres de las bases de datos más importantes, y como veis, para algo tan fácil vemos diferencias. Esto para el programador supone tener que conocer el lenguaje para cada Base de datos, y más importante aún, si en un futuro se desea migrar la aplicación, habría que reescribir gran número de las consultas.

Esto el ORM al tener un capa intermedia, **abstrae al programador de la base de datos y le centra en el desarrollo de la aplicación**.

Otro punto importante es la **facilidad de trabajo**, un ORM, nos facilita las labores básicas de cualquier acceso a datos , el CRUD (Create, Read, Update y Delete). Realizando todas estas labores a través de un lenguaje de alto nivel orientado a objetos. Ahora que ya sabemos **que es un ORM** y **porque usarlo**, vamos a ver sus ventajas y desventajas.

**Ventajas y desventajas de un ORM**

* Ventajas
  + Facilidad y velocidad de uso
  + Abstracción de la base de datos usada.
  + Seguridad de la capa de acceso a datos contra ataques.
* Desventajas
  + En entornos con gran carga poner una capa más en el proceso puede mermar el rendimiento.
  + Aprender el nuevo lenguaje del ORM.

**ORMs más utilizados**

Casi todos los lenguajes de alto nivel actualmente disponen de alguna solución de este tipo, una de las más conocidas es Hibernate para JAVA, pero existen muchas más:

* Java => Hibernate, iBatis, Ebean, etc..
* .NET=> Entity Framework, nHibernate, etc..
* PHP=> Doctrine, Propel, ROcks, Torpor, etc..

BLOQUE 4: C# .NET

### 2. What are strong references and weak references in GC?

The garbage collector cannot collect an object in use by an application while the application’s code can reach that object. The application is said to have a strong reference to the object.

A weak reference permits the garbage collector to collect the object while still allowing the application to access the object. A weak reference is valid only during the indeterminate amount of time until the object is collected when no strong references exist. When you use a weak reference, the application can still obtain a strong reference to the object, which prevents it from being collected. However, there is always the risk that the garbage collector will get to the object first before a strong reference is re-established.

Weak references are useful for objects that use a lot of memory, but can be recreated easily if they are reclaimed by garbage collection.

### 5. What is the difference between Var and Dynamic in C#?

|  |  |
| --- | --- |
| **var** | **dynamic** |
| Introduced in C# 3.0 | Introduced in C# 4.0 |
| Statically typed – This means the type of variable declared is decided by the compiler at compile time. | Dynamically typed – This means the type of variable declared is decided by the compiler at run time. |
| var type of variables are required to be initialized at the time of declaration or else they encounter the compile time error: Implicitly-typed local variables must be initialized. | No need to initialize at the time of declaration. |
| e.g., var str=”I am a string”; | e.g., dynamic str; |
| Looking at the value assigned to the variable str, the compiler will treat the variable str as string. | str=”I am a string”; //Works fine and compiles |
| Errors are caught at compile time. | Errors are caught at runtime |
| Since the compiler knows about the type and the methods and properties of the type at the compile time itself | Since the compiler comes to about the type and the methods and properties of the type at the run time. |
| Intellisense help is available for the var type of variables. This is because, its type is inferred by the compiler from the type of value it is assigned and as a result, the compiler has all the information related to the type | Intellisense help is not available for dynamic type of variables since their type is unknown until run time. So intellisense help is not available. Even if you are informed by the compiler as “This operation will be resolved at run-time”. |
| It will  throw a compilation error since the variable is not initialized. The compiler needs that this variable should be initialized so that it can infer a type from the value. | It Will compile successfully |

### 6. What is the difference between action and func in C#?

Action is a delegate (pointer) to a method, that takes zero, one or more input parameters, but does not return anything.  
   
Func is a delegate (pointer) to a method, that takes zero, one or more input parameters, and returns a value (or reference).

### Example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class Program  {      static void Main(string[] args)      {          Action<int> myAction = new Action<int>(DoSomething);          myAction(123);           // Prints out "123"                                   // can be also called as myAction.Invoke(123);            Func<int, double> myFunc = new Func<int, double>(CalculateSomething);          Console.WriteLine(myFunc(5));   // Prints out "2.5"      }        static void DoSomething(int i)      {          Console.WriteLine(i);      }        static double CalculateSomething(int i)      {          return (double)i/2;      }  } |

### 11. What is the difference between Boxing and Unboxing?

#### **Boxing:**

* The operation of Converting a Value Type to a Reference Type is called **Boxing**
* Boxing is used to store value types in the garbage-collected heap.
* Boxing is an implicit conversion of a value type to the type object or to any interface type implemented by this value type.
* Boxing a value type allocates an object instance on the heap and copies the value into the new object



|  |  |
| --- | --- |
| 1  2 | int Val = 10;  Object Obj = Val; //Boxing |

The first line we created a Value Type Val and assigned a value to Val.

The second line , we created an instance of Object Obj and assign the value of Val to Obj.

From the above operation (Object Obj = i ) we saw converting a value of a Value Type into a value of a corresponding Reference Type .

These types of operation is called Boxing.

#### **Unboxing:**

* Unboxing is simply the opposite of boxing.
* In it values are again shifted from the heap to the stack.
* In unboxing first of all it checks for the boxed object value and then it goes for replica creation
* It converts an object type back into the value type.
* It is an explicit operation using C-style casting.



|  |  |
| --- | --- |
| 1  2  3  4 | int Val = 1;  Object Obj = Val; //Boxing    int i = (int)Obj; //Unboxing |

The first two line shows how to Box a Value Type .

The next line (int i = (int) Obj) shows extracts the Value Type from the Object .

That is converting a value of a Reference Type into a value of a Value Type. This operation is called UnBoxing.

Boxing and UnBoxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed , also the cast required for UnBoxing is also expensive computationally.

Example:

Let’s look at the reference example.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | using System;  using System.Windows.Forms;    namespace WindowsApplication1  {   public partial class Form1 : Form      {          public Form1()          {              InitializeComponent();          }          private void button1\_Click(object sender, EventArgs e)          {              int Val = 10;              Object Obj = Val;       //Boxing              int i = (int)Obj;       //Unboxing              MessageBox.Show("The value is   : " + i);          }      }  } |

So boxing is the process of taking a value type, putting it inside a new object on the heap,and storing a reference to it on the stack. Unboxing is the exact opposite: It takes the item from the heap and returns a value type that contains the value from the heap.

If you execute an invalid unbox operation, the runtime will throw an InvalidCastException.

when boxing and unboxing happen, you need to explicitly cast your object from a reference to a value type.

There are some performance implications with each box and unbox operation.

The boxing and unboxing operations can hurt performance; however, now that you have generic support in the .NET Framework, this is less of an issue because you can store value types in a collection without boxing them.

### 12. What is the difference between Compile time Exception and Runtime Exception?

Compile time is where your compiler transforms your source code to a machine understandable language.

During the compile time, it processes through various stages:  
Creation of Symbol table, Syntax analysis, Semantic analysis, Code optimization, Code Generation & Error Handling.

Runtime is during the execution process(Eg: Page request is made. or looping through a variable instances, etc). Runtime errors are handles after the successful compilation.

Example:  
The static variables are allocated with memory in the compile time.  
The variables that are created at runtime(during the execution process), the memory is allocated for them at run time.

The compile time errors may occur with an error in syntax.

This run time errors may be based on the user input like divide by zero exception, stack over flow, pointer unavailability, wrong address reference, referring null string, etc.

### 13. What is the difference between Error and Exception in C#?

An exception is an Object of a type deriving from the System.Exception class. SystemException is thrown by the CLR (Common Language Runtime) when errors occur that are nonfatal and recoverable by user programs.

Exception syntax:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try    {    //write your code here    }    Catch (exception type)    {    //writ your code here    } |

Errors are unchecked exception and is difficult to handle in code.

### 14. Why to use Lock statement in C#?

Using Lock statement you can ensure only one thread can be executed at any point of time.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | using System;  using System.Threading;  using System.Threading.Tasks;  namespace RaceCondition  {      class Program      {          static object locker = new object();          private static int counter;          static void Main(string[] args)          {              new Thread(PrintStar).Start();              new Thread(PrintPlus).Start();          }            static void PrintStar()          {              lock (locker) // Thread safe code              {                  for (counter = 0; counter < 5; counter++)                  {                      Console.Write(" \* " + "\t");                  }              }          }            static void PrintPlus()          {              lock (locker) // Thread safe code              {                  for (counter = 0; counter < 5; counter++)                  {                      Console.Write(" + " + "\t");                  }              }          }      }  } |

### 15. What is the difference between Yield and Return in C#?

#### **return statement in C#:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | static int SimpleReturn()  {      return 1;      return 2;      return 3;  }  static void Main(string[] args)  {      Console.WriteLine(SimpleReturn());      Console.WriteLine(SimpleReturn());      Console.WriteLine(SimpleReturn());      Console.WriteLine(SimpleReturn());  } |

Output:  
1  
1  
1  
1  
In above example int SimpleReturn function three returns are mentioned but no matter how many times this function is called from anywhere, here from main the function will always return first value i.e 1.

#### **Yield statement in C#:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | static IEnumerable<int> YieldReturn()  {      yield return 1;      yield return 2;      yield return 3;  }  static void Main(string[] args)  {      foreach (int i in YieldReturn())      {          Console.WriteLine(i);      }  } |

Output:  
1  
2  
3  
The only difference between yield and return is whenever yield statement is encountered in a function, the execution of function is suspended and a value is send back to the caller but because of yield whenever the function is called again, the execution of function begin where it left off previously. When resumed, the function continues execution immediately after the last yield run. Thus yield allows a function to produce a series of values over time. The only requirement for yield return statement is that the function containing yield should return an IEnumerable and no matter from where that function is called it should be called from an iteration block i.e foreach statement.

### ¿Que es inyección de dependencias?

Basicamente es quitarle la responsabilidad a nuestra clase principal de crear instancias a sus dependencias. En pocas palabras, en vez de que nuestra clase haga una instancia internamente a otra clase que depende, mejor le pasamos la clase que va a implementar como un parámetro adicional, de esta manera evitamos la dependencia directa. ¿Se entendió?, bueno ... espero que con el ejemplo se entienda mejor.

### Nuestro ejemplo

En nuestro ejemplo vamos a suministrar a un soldado que arma debería usar en el combate. Para ello, hemos creado las siguientes clases:

#### Las armas convertidas a Clase

public class Revolver

{

public string Disparar()

{

return "Pum Pum ..";

}

}

public class Rifle

{

public string Disparar()

{

return "Pum pum pum pum pum ..";

}

}

public class Escopeta

{

public string Disparar()

{

return "pum PUMMM !! ..";

}

}

Como se darán cuenta, las armas implementan el método**Disparar()**, la cual el disparo varía dependiendo del tipo de arma que se elija.

#### Nuestra clase Soldado

Esta clase implementa los métodos necesarios para realizar los disparos para cada arma.

public class Soldado

{

public string DispararRevolver()

{

return new Revolver().Disparar();

}

public string DispararRifle()

{

return new Rifle().Disparar();

}

public string DispararEscopeta()

{

return new Escopeta().Disparar();

}

}

* **¿Cual es el problema?,**que la dependencia la tiene que implementar el mismo Soldado.
* **¿Y eso en que nos afecta?**
  + Vamos a tener que modificar nuestra clase Soldado en un futuro si queremos agregar más armas, ahora esta simple porque solo hay 3 armas y cada una tiene un método. ¿Pero si fuerán 200 armas y cada una tiene 100 métodos?.
  + Realizar una prueba unitaria sería bastante tedioso, hay que probar método por método.
  + La clase a futuro va a ser un dolor de cabeza, teniendo un código que no permita su mantenimiento/mejora a futuro.
  + Al momento de instanciar la clase, vamos a tener que **llamar a c/u de los métodos**para manipular el arma seleccionada. Eso nos daría un serio problema, porque tendríamos que **modificar el código constantemente** si es que quisieramos cambiar de arma o hacer un uso **excesivo y vulgar de los IF o SWITCH para saber con que arma se debe trabajar**.

### ¿Cual es la solución?

Implementar una **interface** yque el **constructor de la clase Soldado la reciba** como parámetro.

#### Paso #1

Crear la interface y hacer que las armas implementen dicha interface.

public interface IArma {

string Disparar();

}

public class Revolver : IArma

{

public string Disparar()

{

return "Pum Pum ..";

}

}

public class Rifle : IArma

{

public string Disparar()

{

return "Pum pum pum pum pum ..";

}

}

public class Escopeta : IArma

{

public string Disparar()

{

return "pum PUMMM !! ..";

}

}

#### Paso #2

Modificar nuestra clase Soldado para que reciba dicha interface.

public class Soldado

{

protected IArma arma;

public Soldado(IArma \_arma)

{

this.arma = \_arma;

}

public string Disparar()

{

return this.arma.Disparar();

}

}

Si se dan cuenta ahora, solo tenemos un **método "Disparar"**. Nuestra clase Soldado sabe que debe disparar un Arma, es nuestro constructor el encargado de setear que arma debe usar.

Veamos como se instancia nuestra clase Soldado usando como arma principal el Revolver:

var soldado = new Soldado(new Revolver());

soldado.Disparar();

Si queremos usar la escopeta:

var soldado = new Soldado(new Escopeta());

soldado.Disparar();

¿Y el rifle?

var soldado = new Soldado(new Rifle());

soldado.Disparar();

¿Se dieron cuenta lo lindo que ha quedado nuestro código?. Al final lo que hemos hecho es eliminar la dependenia de una clase dentro de otra clase.

**NOTA**: Me imagino que si has comprendido el ejemplo, tu mente ahora procesa varios casos para la cual usaría la **inyección de dependencia.**Sino fuera así, deja un comentario y te puedo orientar.

**Dependency property**: A property that is backed by a [**DependencyProperty**](http://msdn.microsoft.com/en-us/library/system.windows.dependencyproperty.aspx).

The purpose of dependency properties is to provide a way to compute the value of a property **based** on the **value of other inputs**.

These other inputs might include system properties such as themes and user preference, just-in-time property determination mechanisms such as data binding and animations/storyboards, multiple-use templates such as resources and styles, or values known through parent-child relationships with other elements in the element tree.

KeyWords

* C#
* C++
* Java
* SQL

Interviews Questions:

1. Experience and Importance of the support, testing and develop procedures.
2. How to solve the problems of replication between two Databases.
   * Pooling or Observer
3. What is a BTree.

. Arbol balanceado:  a **B-tree** is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The **B-tree** is a generalization of a binary search tree in that a node can have more than two children.

Uses: Index in data bases.

1. Stateless and statefull protocol, what type is HTTP?
   1. Stateless: Sin estado. Asincrono. HTTP.
   2. Statefull: Con estado. Syncrono. FTP.
2. What is AJAX and uses?
   1. **JavaScript Asíncrono y XML (AJAX)** no es una tecnología por sí misma, es un término que describe un nuevo modo de utilizar conjuntamente varias tecnologías existentes. Esto incluye: [HTML](https://developer.mozilla.org/es/docs/Web/Guide/es/HTML) o [XHTML](https://developer.mozilla.org/es/docs/Web/Guide/es/XHTML), [CSS](https://developer.mozilla.org/es/docs/Web/Guide/es/CSS), [JavaScript](https://developer.mozilla.org/es/docs/Web/Guide/es/JavaScript), [DOM](https://developer.mozilla.org/es/docs/Web/Guide/es/DOM), [XML](https://developer.mozilla.org/es/docs/Web/Guide/es/XML), [XSLT](https://developer.mozilla.org/es/docs/Web/Guide/es/XSLT), y el objeto [XMLHttpRequest](https://developer.mozilla.org/es/docs/Web/Guide/es/XMLHttpRequest). Cuando estas tecnologías se combinan en un modelo AJAX, es posible lograr aplicaciones web capaces de actualizarse continuamente sin tener que volver a cargar la página completa. Esto crea aplicaciones más rápidas y con mejor respuesta a las acciones del usuario.
3. Desgin Pattern: Singleton, Observe.
   1. Singleton:

A class of which only a single instance can exist and it’s globaly accessible. Methods: static Singleton getInstance(), the constructor is private.

* 1. Observe:

A way of notifying change to a number of classes. Communicate the changes in one class to the reset connected. 2 Interfaces: ISubject and IObserver. Methods: ISubject.AttachObserber(), ISubject.DettachObserber() and ISubject.SendNotification().

* 1. Factory- Abstract Factory:

The abstract class or interface constructor must be protected. Used in the List of the User Interface to group the same family. Abstract-Factory = Factory of factories (grouping different families). getType().

1. Create a small numerical function.

### [Senior Software Engineer at Amazon was asked...](https://www.glassdoor.ca/Interview/Amazon-Senior-Software-Engineer-Interview-Questions-EI_IE6036.0,6_KO7,31.htm)

18 Oct, 2011

|  |
| --- |
| Rand(7) from Rand(5)  [7 Answers](https://www.glassdoor.ca/Interview/Rand-7-from-Rand-5-QTN_200619.htm)  The simple solution is to implement RandBin() using Rand5(). int RandBin() { }  Sorry this UI posts without warning int RandBin() { int rand5Res = Rand5(); return rand5Res &lt; 2 ? 0 : rand5Res &lt; 4 ? 1 : RandBin(); } And then use RandBin() to implement Rand7; int Rand7() { int rand = RandBin() &lt;&lt; 2 | RandBin() &lt;&lt; 1 | RandBin(); return rand &lt; 7 ? rand : Rand7(); }  int rand7() { while(1) { int n = ((rand5()%2)\*4 + (rand5()%2)\*2 + (rand5()%2)\*1); if(n == 0) continue; return n; } } The rand5()%2 will generate 0 and 1 with equal probability and we need 3 bits since we are going from 000 upto 111. So we call this function thrice for each bit position. |

C++

int rand7(){int x=8;while(x>7)x=rand5()+5\*rand5()-5;return x;}

Desgin Pattern:

# Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

### Uses of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

### [Creational design patterns](https://sourcemaking.com/design_patterns/creational_patterns)

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

[](https://sourcemaking.com/design_patterns/abstract_factory)

* [**Abstract Factory**](https://sourcemaking.com/design_patterns/abstract_factory)  
  Creates an instance of several families of classes
* [**Builder**](https://sourcemaking.com/design_patterns/builder)  
  Separates object construction from its representation
* [**Factory Method**](https://sourcemaking.com/design_patterns/factory_method)  
  Creates an instance of several derived classes
* [**Object Pool**](https://sourcemaking.com/design_patterns/object_pool)  
  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
* [**Prototype**](https://sourcemaking.com/design_patterns/prototype)  
  A fully initialized instance to be copied or cloned
* [**Singleton**](https://sourcemaking.com/design_patterns/singleton)  
  A class of which only a single instance can exist

### [Structural design patterns](https://sourcemaking.com/design_patterns/structural_patterns)

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

[](https://sourcemaking.com/design_patterns/decorator)

* [**Adapter**](https://sourcemaking.com/design_patterns/adapter)  
  Match interfaces of different classes
* [**Bridge**](https://sourcemaking.com/design_patterns/bridge)  
  Separates an object’s interface from its implementation
* [**Composite**](https://sourcemaking.com/design_patterns/composite)  
  A tree structure of simple and composite objects
* [**Decorator**](https://sourcemaking.com/design_patterns/decorator)  
  Add responsibilities to objects dynamically
* [**Facade**](https://sourcemaking.com/design_patterns/facade)  
  A single class that represents an entire subsystem
* [**Flyweight**](https://sourcemaking.com/design_patterns/flyweight)  
  A fine-grained instance used for efficient sharing
* **[](https://sourcemaking.com/design_patterns/proxy)**

[**Private Class Data**](https://sourcemaking.com/design_patterns/private_class_data)  
Restricts accessor/mutator access

* [**Proxy**](https://sourcemaking.com/design_patterns/proxy)  
  An object representing another object

### [Behavioral design patterns](https://sourcemaking.com/design_patterns/behavioral_patterns)

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

[](https://sourcemaking.com/design_patterns/interpreter)

* [**Chain of responsibility**](https://sourcemaking.com/design_patterns/chain_of_responsibility)  
  A way of passing a request between a chain of objects
* [**Command**](https://sourcemaking.com/design_patterns/command)  
  Encapsulate a command request as an object
* [**Interpreter**](https://sourcemaking.com/design_patterns/interpreter)  
  A way to include language elements in a program
* [**Iterator**](https://sourcemaking.com/design_patterns/iterator)  
  Sequentially access the elements of a collection
* [**Mediator**](https://sourcemaking.com/design_patterns/mediator)  
  Defines simplified communication between classes
* [**Memento**](https://sourcemaking.com/design_patterns/memento)  
  Capture and restore an object's internal state
* [**Null Object**](https://sourcemaking.com/design_patterns/null_object)  
  Designed to act as a default value of an object
* [**Observer**](https://sourcemaking.com/design_patterns/observer)  
  A way of notifying change to a number of classes
* **[](https://sourcemaking.com/design_patterns/state)**

[**State**](https://sourcemaking.com/design_patterns/state)  
Alter an object's behavior when its state changes

* [**Strategy**](https://sourcemaking.com/design_patterns/strategy)  
  Encapsulates an algorithm inside a class
* [**Template method**](https://sourcemaking.com/design_patterns/template_method)  
  Defer the exact steps of an algorithm to a subclass
* [**Visitor**](https://sourcemaking.com/design_patterns/visitor)  
  Defines a new operation to a class without change

### Criticism

The concept of design patterns has been criticized by some in the field of computer science.

#### Targets the wrong problem

The need for patterns results from using computer languages or techniques with insufficient abstraction ability. Under ideal factoring, a concept should not be copied, but merely referenced. But if something is referenced instead of copied, then there is no "pattern" to label and catalog. Paul Graham writes in the essay [**Revenge of the Nerds**](http://www.paulgraham.com/icad.html).

Peter Norvig provides a similar argument. He demonstrates that 16 out of the 23 patterns in the Design Patterns book (which is primarily focused on C++) are simplified or eliminated (via direct language support) in Lisp or Dylan.

#### Lacks formal foundations

The study of design patterns has been excessively ad hoc, and some have argued that the concept sorely needs to be put on a more formal footing. AtOOPSLA 1999, the Gang of Four were (with their full cooperation) subjected to a show trial, in which they were "charged" with numerous crimes against computer science. They were "convicted" by ⅔ of the "jurors" who attended the trial.

#### Leads to inefficient solutions

The idea of a design pattern is an attempt to standardize what are already accepted best practices. In principle this might appear to be beneficial, but in practice it often results in the unnecessary duplication of code. It is almost always a more efficient solution to use a well-factored implementation rather than a "just barely good enough" design pattern.

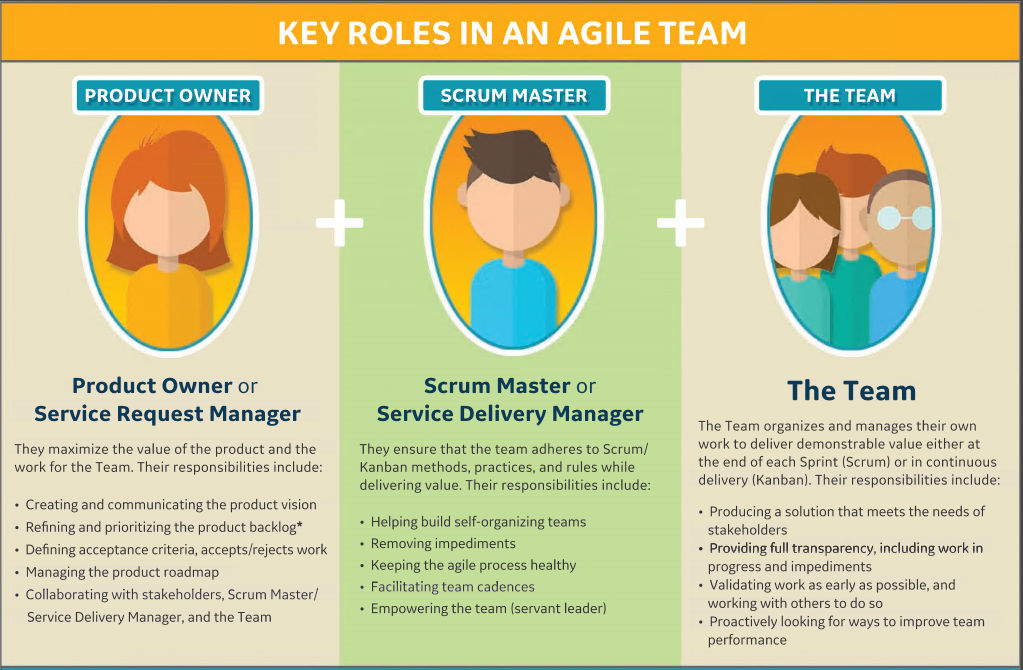
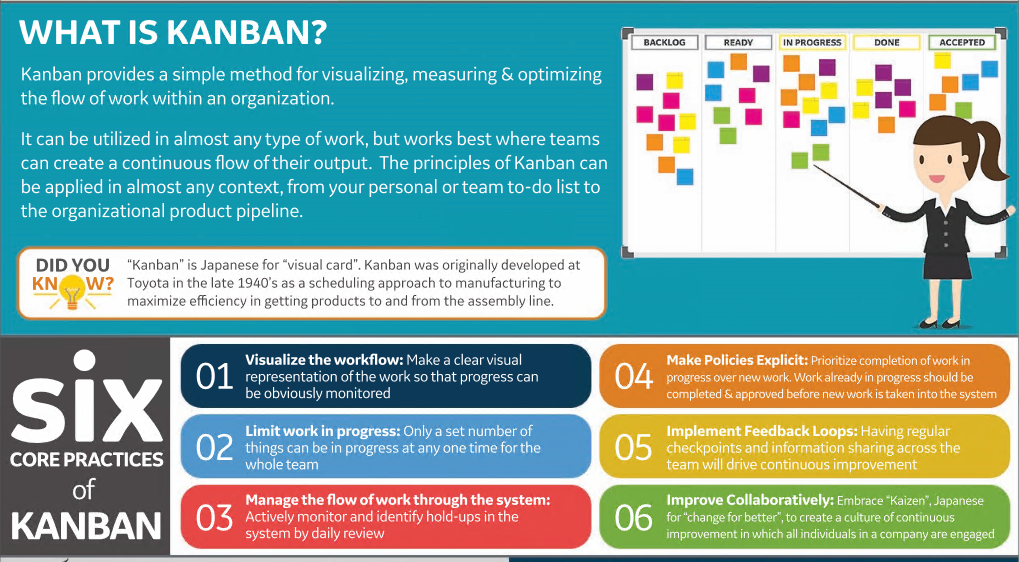
#### Does not differ significantly from other abstractions

Some authors allege that design patterns don't differ significantly from other forms of abstraction, and that the use of new terminology (borrowed from the architecture community) to describe existing phenomena in the field of programming is unnecessary. The Model-View-Controller paradigm is touted as an example of a "pattern" which predates the concept of "design patterns" by several years. It is further argued by some that the primary contribution of the Design Patterns community (and the Gang of Four book) was the use of Alexander's pattern language as a form of documentation; a practice which is often ignored in the literature.

<https://sourcemaking.com/design_patterns>

# Agile

# 

# SOLID

<https://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp>

### Modelo vista controlador (MVC)

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

* El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
* La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
* El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

#### **El modelo es el responsable de:**

* Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
* Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
* Lleva un registro de las vistas y controladores del sistema.
* Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes  que actualiza los datos, un temporizador que desencadena una inserción, etc.).

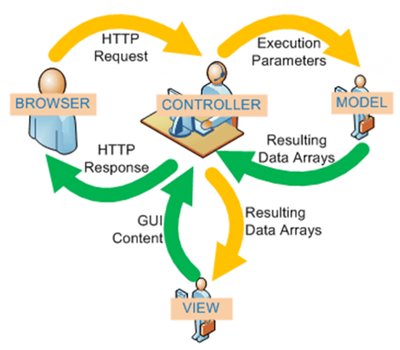
#### **El controlador es responsable de:**

* Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
* Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener\_tiempo\_de\_entrega ( nueva\_orden\_de\_venta )".

#### **Las vistas son responsables de:**

* Recibir datos del modelo y los muestra al usuario.
* Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
* Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

El flujo que sigue el control generalmente es el siguiente:



1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Most Popular Database Interview Questions And Answers

Given below is a list of most popular Database interview questions and answers for your reference.

**Q #1) What do you understand by ‘Database’?**

**Ans:** Database is an organized collection of related data where the data is stored and organized to serve some specific purpose.

For ***Example***, A librarian maintains a database of all the information related to the books that are available in the library.

**Q #2) Define DBMS.**

**Ans:**DBMS stands for Database Management system. It is a collection of application programs which allow the user to organize, restore and retrieve information about data efficiently and as effectively as possible.

Some of the popular DBMS's are MySql, Oracle, Sybase, etc.

**Q #3) Define RDBMS.**

**Ans:**Relational Database Management System(RDBMS) is based on a relational model of data that is stored in databases in separate tables and they are related to the use of a common column. Data can be accessed easily from the relational database using Structured Query Language (SQL).

**Q #4) Enlist the advantages of DBMS.**

**Ans: The Advantages of DBMS includes:**

* Data is stored in a structured way and hence redundancy is controlled.
* Validates the data entered and provide restrictions on unauthorized access to the database.
* Provides backup and recovery of the data when required.
* Provides multiple user interfaces.

**Q #5) What do you understand by Data Redundancy?**

**Ans:**Duplication of data in the database is known as Data redundancy. As a result of Data Redundancy, duplicated data is present at various locations, hence it leads to wastage of the storage space and the integrity of the database is destroyed.

**Q #6) What are the various types of relationships in Database? Define them.**

**Ans: There are 3 types of relationships in Database:**

* **One-to-one:** One table has the relationship with another table having the similar kind of column. Each primary key relates to only one or no record in the related table.
* **One-to-many:** One table has a relationship with another table that has primary and foreign key relation. The primary key table contains only one record that relates to none, one or many records in the related table.
* **Many-to-many:** Each record in both the tables can relate to many numbers of record in another table.

**Q #7) Explain Normalization and De-Normalization.**

**Ans*:*Normalization** is the process of removing the redundant data from the database by splitting the table in a well-defined manner in order to maintain data integrity. This process saves much of the storage space.

**De-normalization** is the process of adding up redundant data on the table in order to speed up the complex queries and thus achieve better performance.

**Q #8) What are the different types of Normalization?**

**Ans: Different Types of Normalization are:**

* **First Normal Form (1NF):** A relation is said to be in 1NF only when all the entities of the table contain unique or atomic values.
* **Second Normal Form (2NF):**A relation is said to be in 2NF only if it is in 1NF and all the non-key attribute of the table is fully dependent on the primary key.
* ***Third Normal Form (3NF):***A relation is said to be in 3NF only if it is in 2NF and every non-key attribute of the table is not transitively dependent on the primary key.

**Q #9) What is BCNF?**

**Ans:**BCNF is Boyce Code Normal form. It is the higher version of 3Nf which does not have any multiple overlapping candidate keys.

**Q #10) What is SQL?**

**Ans:**Structured Query language, SQL is an ANSI(American National Standard Institute) standard programming language which is designed specifically for storing and managing the data in the relational database management system (RDBMS) using all kinds of data operations.

**Q #11) How many SQL statements are used? Define them.**

**Ans:**SQL statements are basically divided into three categories, DDL, DML, and DCL.

**They can be defined as:**

**Data Definition Language (DDL)**commands are used to define the structure that holds the data. These commands are auto-committed i.e. changes done by the DDL commands on the database are saved permanently.

**Data Manipulation Language (DML)** commands are used to manipulate the data of the database. These commands are not auto-committed and can be rolled back.

**Data Control Language (DCL)**commands are used to control the visibility of the data in the database like revoke access permission for using data in the database.

**Q #12) Enlist some commands of DDL, DML, and DCL.**

**Ans: Data Definition Language (DDL) commands:**

* CREATE to create a new table or database.
* ALTER for alteration.
* Truncate to delete data from the table.
* DROP to drop a table.
* RENAME to rename a table.

**Data Manipulation Language (DML) commands:**

* INSERT to insert a new row.
* UPDATE to update an existing row.
* DELETE to delete a row.
* MERGE for merging two rows or two tables.

**Data Control Language (DCL) commands:**

* COMMIT to permanently save.
* ROLLBACK to undo the change.
* SAVEPOINT to save temporarily.

**Q #13) Define DML Compiler.**

**Ans:**DML compiler translates DML statements in a query language into a low-level instruction and the generated instruction can be understood by Query Evaluation Engine.

**Q #14) What is DDL interpreter?**

**Ans:**DDL Interpreter interprets the DDL statements and records the generated statements in the table containing metadata.

**Q #15) Enlist the advantages of SQL.**

**Ans: Advantages of SQL are:**

* Simple SQL queries can be used to retrieve a large amount of data from the database very quickly and efficiently.
* SQL is easy to learn and almost every DBMS supports SQL.
* It is easier to manage the database using SQL as no large amount of coding is required.

**Q #16) Explain the terms ‘Record’, ‘Field’ and ‘Table’ in terms of database.**

**Ans: Record:** Record is a collection of values or fields of a specific entity. Eg. An employee, Salary account, etc.

**Field:** A field refers to an area within a record which is reserved for a specific piece of data. Eg. Employee ID.

**Table:** Table is the collection of records of specific types. E.g. Employee table is a collection of record related to all the employees.

**Q #17) What do you understand by Data Independence? What are its two types?**

**Ans:**Data Independence refers to the ability to modify the schema definition in one level in such a way that it does not affect the schema definition in the next higher level.

**The 2 types of Data Independence are:**

* **Physical Data Independence**: It modifies the schema at the physical level without affecting the schema at the conceptual level.
* **Logical Data Independence:**It modifies the schema at the conceptual level without affecting or causing changes in the schema at the view level.

**Q #18) Define the relationship between ‘View’ and ‘Data Independence’.**

**Ans:**View is a virtual table which does not have its data on its own rather the data is defined from one or more underlying base tables.

Views account for logical data independence as the growth and restructuring of base tables is not reflected in views.

**Q #19) What are the advantages and disadvantages of views in the database?**

**Ans: Advantages of Views:**

* As there is no physical location where the data in views is stored, it generates output without wasting resources.
* Data access is restricted as it does not allow commands like insertion, updation, and deletion.

**Disadvantages of Views:**

* View becomes irrelevant if we drop a table related to that view.
* More memory is occupied when the view is created for large tables.

**Q #20) What do you understand by Functional dependency?**

**Ans:**A relation is said to be in Functional dependency when one attribute uniquely defines another attribute.

**For Example,** R is a Relation, X and Y are two attributes. T1 and T2 are two tuples. Then,

T1[X]=T2[X] and T1[Y]=T2[Y] means the value of component X uniquely define the value of component Y.

Also, X->Y means Y is functionally dependent on X.

**Q #21) When is functional dependency said to be the fully functional dependency?**

**Ans:**To fulfill the criteria of fully functional dependency, the relation must meet the requirement of functional dependency.

A functional dependency ‘A’ and ‘B’ is said to be fully functional dependent when removal of any attribute say ‘X’ from ‘A’ means the dependency does not hold anymore.

**Q #22) What do you understand by E-R model?**

**Ans:**E-R model is an Entity-Relationship model which defines the conceptual view of the database.

E-R model basically shows the real world entities and their association/relations. Entities here represent the set of attributes in the database.

**Q #23) Define Entity, Entity type, and Entity set.**

**Ans: Entity** can be anything, be it a place, class or object which has an independent existence in the real world.

**Entity type** represents a set of entities which have similar attributes.

**Entity set** in the database represents a collection of entities having a particular entity type.

**Q #24) Define Weak Entity set.**

**Ans:**Weak entity set is the one whose primary key comprises of its partial key as well as the primary key of its parent entity.

This is the case because the entity set may not have sufficient attributes to form a primary key.

**Q #25) Explain the terms ‘Attribute’ and ‘Relations’**

**Ans: Attribute** describes the properties or characteristics of an entity. For ***Example***, Employee ID, Employee Name, Age, etc., can be attributes of the entity Employee.

**Relation** is a two-dimensional table containing a number of rows and columns where every row represents a record of the relation. Here, rows are also known as ‘Tuples’ and columns are known as ‘Attributes’.

**Q #26) What are VDL and SDL?**

**Ans: VDL**is View Definition language which represents user views and their mapping to the conceptual schema.

**SDL**is Storage Definition Language which specifies the mapping between two schemas.

**Q #27) Define Cursor and its types.**

**Ans:**Cursor is a temporary work area which stores the data as well as the result set occurred after manipulation of data retrieved. A cursor can hold only one row at a time.

**The 2 types of Cursor are:**

**Implicit cursors** are declared automatically when DML statements like INSERT, UPDATE, DELETE is executed.

**Explicit cursors** have to be declared when SELECT statements which are returning more than one row are executed.

**Q #28) What is Database transaction?**

**Ans:**Sequence of operation performed which changes the consistent state of the database to another is known as the database transaction. After the completion of the transaction, either the successful completion is reflected in the system or the transaction fails and no change is reflected.

**Q #29) Define Database Lock and its types.**

**Ans:**Database lock basically signifies the transaction about the current status of the data item i.e. whether that data is being used by other transactions or not at the present point of time.

There are two types of Database lock which are **Shared Lock and Exclusive Lock.**

**Q #30) What is Data Warehousing?**

**Ans:**The storage as well as access to data, that is being derived from the transactions and other sources, from a central location in order to perform the analysis is called Data Warehousing.

**Q #31) What do you understand by Join?**

**Ans:**Join is the process of explaining the relationship between different tables by combining columns from one or more table having common values in each. When a table joins with itself, it is known as Self Join.

**Q #32) What do you understand by Index hunting?**

**Ans:**Index hunting is the process of boosting the collection of indexes which help in improving the query performance as well as the speed of the database.

**Q #33) How to improve query performance using Index hunting?**

**Ans: Index hunting help in improving query performance by:**

* Using query optimizer to coordinate queries with the workload.
* Observing the performance and effect of index and query distribution.

**Q #34) Differentiate between ‘Cluster’ and ‘Non-cluster’ index.**

**Ans:**Clustered Index alters the table and reorders the way in which the records are stored in the table. Data retrieval is made faster by using the clustered index.

A Non-clustered index does alter the records that are stored in the table but creates a completely different object within the table.

**Q #35) What are the disadvantages of a Query?**

**Ans: Disadvantages of a Query are:**

* Indexes are not present.
* Stored procedures are excessively compiled.
* Difficulty in interfacing.

**Q #36) What do you understand by Fragmentation?**

**Ans:**Fragmentation is a feature which controls the logical data units, also known as fragments that are stored at different sites of a distributed database system.

**Q #37) Define Join types.**

**Ans:**Given below are the types of Join, which are explained with respect to the tables as an ***Example***:

**employee table:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee-table.jpg)

**employee\_info table:**

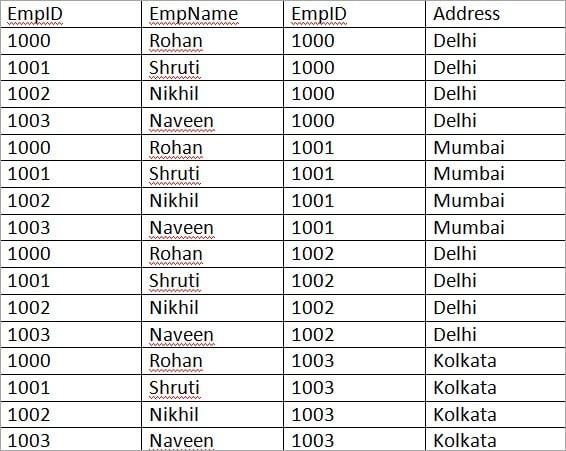
[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee_info-table.jpg)

**1) Inner JOIN:** Inner JOIN is also known as a simple JOIN. This SQL query returns result from both the tables having a common value in rows.

**SQL Query:**

SELECT \* from employee, employee\_info WHERE employee.EmpID = employee\_info.EmpID ;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/Inner-Join-Example.jpg)

**2) Natural JOIN:** This is a type of Inner JOIN which returns results from both the tables having same data values in the columns of both the tables to be joined.

**SQL Query:**

SELECT \* from employee NATURAL JOIN employee\_info;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Natural-JOIN.jpg)

**3) Cross JOIN:** Cross JOIN return results as all the records where each row from the first table is combined with each row of the second table.

**SQL Query:**

SELECT \* from employee CROSS JOIN employee\_info;

**Result:**

Let us do some modification in the above tables to understand Right JOIN, Left JOIN, and Full JOIN.

**employee table:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee-table-new.jpg)

**employee\_info table:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee_info-table-new.jpg)

**1) Right JOIN:** Right JOIN is also known as Right Outer JOIN. This returns all the rows as a result from the right table even if the JOIN condition does not match any records in the left table.

**SQL Query:**

SELECT \* from employee RIGHT OUTER JOIN employee\_info on (employee.EmpID = employee\_info.EmpID);

**Result:**

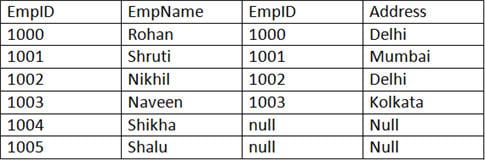
[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/Right-Join-Example.jpg)

**2) Left JOIN:** Left JOIN is also known as Left Outer JOIN. This returns all the rows as a result of the left table even if JOIN condition does not match any records in the right table. This is exactly the opposite of Right JOIN.

**SQL Query:**

SELECT \* from employee LEFT OUTER JOIN employee\_info on (employee.EmpID = employee\_info.EmpID);

**Result:**

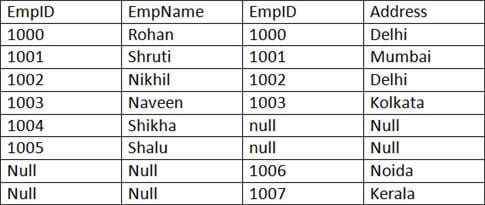
[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Left-JOIN.jpg)

**3) Outer/Full JOIN:** Full JOIN return results in combining the result of both the Left JOIN and Right JOIN.

**SQL Query:**

SELECT \* from employee FULL OUTER JOIN employee\_info on (employee.EmpID = employee\_info.EmpID);

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Outer-Full-JOIN.jpg)

**Q #38) What do you understand by ‘Atomicity’ and ‘Aggregation’?**

**Ans: Atomicity** is the condition where either all the actions of the transaction are performed or none. This means, when there is an incomplete transaction, database management system itself will undo the effects done by the incomplete transaction.

**Aggregation** is the concept of expressing the relationship with the collection of entities and their relationships.

**Q #39) Define Phantom deadlock.**

**Ans:**Phantom deadlock detection is the condition where the deadlock does not actually exist but due to a delay in propagating local information, deadlock detection algorithms identify the deadlocks.

**Q #40) Define checkpoint.**

**Ans:**Checkpoint declares a point before which all the logs are stored permanently in the storage disk and is the inconsistent state. In the case of crashes, the amount of work and time is saved as the system can restart from the checkpoint.

**Q #41) What is Database partitioning?**

**Ans:**Database partitioning is the process of partitioning tables, indexes into smaller pieces in order to manage and access the data at a finer level.

This process of partitioning reduces the cost of storing a large amount of data as well as enhances the performance and manageability.

**Q #42) Explain the importance of Database partitioning.**

**Ans: The importance of Database partitioning are:**

* Improves query performance and manageability.
* Simplifies common administration tasks.
* Acts as a key tool for building systems with extremely high availability requirements.
* Allows accessing a large part of a single partition.

**Q #43) Explain Data Dictionary.**

**Ans:**Data dictionary is a set of information describing the content and structure of the tables and database objects. The job of the information stored in the data dictionary is to control, manipulate and access the relationship between database elements.

**Q #44) Explain Primary Key and Composite Key.**

**Ans: Primary key** is that column of the table whose every row data is uniquely identified. Every row in the table must have a primary key and no two rows can have the same primary key. Primary key value can never be null nor can be modified or updated.

**Composite Key**is a form of the candidate key where a set of columns will uniquely identify every row in the table.

**Q #45) What do you understand by Unique key?**

**Ans:**A Unique key is same as the primary key whose every row data is uniquely identified with a difference of null value i.e. Unique key allows one value as NULL value.

**Q #46) What do you understand by Database Triggers?**

**Ans:**A set of commands that automatically get executed when an event like Before Insert, After Insert, On Update, On Delete of row occurs in a table is called as Database trigger.

**Q #47) Define Stored procedures.**

**Ans:**A Stored procedure is a collection of pre-compiled SQL Queries, which when executed denotes a program taking input, process and gives the output.

**Q #48) What do you understand by B-Trees?**

**Ans:**B-Tree represents the data structure in the form of a tree for external memory that reads and writes large blocks of data. It is commonly used in databases and file systems where all the insertions, deletions, sorting, etc., are done in logarithmic time.

**Q #49) Name the different data models that are available for database systems.**

**Ans: Different data models are:**

* Relational model
* Network model
* Hierarchical model

**Q #50) Differentiate between ‘DELETE’, ‘TRUNCATE’ and ‘DROP’ commands.**

**Ans:**After the execution of **‘DELETE’** operation, COMMIT and ROLLBACK statements can be performed to retrieve the lost data.

After the execution of**‘TRUNCATE’** operation, COMMIT, and ROLLBACK statements cannot be performed to retrieve the lost data.

**‘DROP’** command is used to drop the table or key like the primary key/foreign key.

**Q #51) Based on the given table, solve the following queries.**

**Employee table**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Employee-table-1.jpg)

**1)** Write the SELECT command to display the details of the employee with empid as 1004.

**Ans:**

SELECT empId, empName, Age, Address from Employee WHERE empId = 1004;

**Result:**

[SELECT command](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/SELECT-command.jpg)

**2)** Write the SELECT command to display all the records of table Employee.

**Ans:**

SELECT \* from Employee;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/display-all-records.jpg)

**3)** Write the SELECT command to display all the records of the employee whose name starts with the character ‘R’.

**Ans:**

SELECT \* from Employee WHERE empName LIKE ‘R%’;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/name-starts-with-character-R.jpg)

**4)** Write a SELECT command to display id, age and name of the employees with their age in both ascending and descending order.

**Ans:**

SELECT empId, empName, Age from Employee  ORDER BY Age;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employees-with-their-age-in-ascending.jpg)

SELECT empId, empName, Age from Employee  ORDER BY Age Desc;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employees-with-their-age-in-descending.jpg)

**5)** Write the SELECT command to calculate the total amount of salary on each employee from the below Emp table.

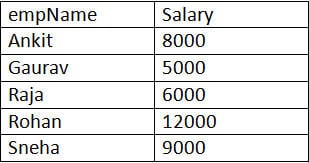
**Emp table:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Emp-table-1.jpg)

**Ans:**

SELECT empName, SUM(Salary) from Emp GROUP BY empName;

**Result:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Result.jpg)

Conclusion

These are the set of Database interview questions and answers which are mostly asked in the interview.

Mostly the basics of every subject are questioned in the interviews. It is a well-known fact to everyone that, if your basics are clear, you can reach top heights.

***However, there may be some more tricky questions. Just be confident and face each question with clarity in your subject knowledge.***

# CURRICULUM VITAE

**MANUEL ROJAS SUAREZ**

* ADDRESS: 1410 Waterloo – Allée de l’Aqueduc 5, Belgium
* NATIONALITY: Spanish (EU)
* DATE OF BIRTH: 28 – 06 – 1978
* TELEPHONE: +32 487 32 74 01
* EMAIL: manuel.rojas.suarez@gmail.com

WORK EXPERIENCE

**01/12/2017 – Ongoing: *GE Additive.*** Avenue George Lemaître 54, 6041 Gosselies (Belgium).

**Senior Software Engineer:**

* Lead developer in Virfac iAM®. Simulation software for additive manufacturing process. C#, .NET 4.6, WPF, MVVM, Visual Studio 2015/2017, Git, Windows OS, Agile Scrum, project management.

**13/01/2014 – 30/11/2017: *GeonX.*** Avenue George Lemaître 54, 6041 Gosselies (Belgium).

**Software Engineer:**

* Developer in Virfac®. Simulation software for Industrial process such as: Welding, heat treatment or additive manufacturing. C#, .NET 4.6, WPF, MVVM, Visual Studio 2012/2015, Git, Windows OS, Agile methodologies.

**10/10/2011 – 31/12/2013: DT *(ACCENTURE Spanish delivery center).*** Juan López Peñalver, 28 PTA - 29590 Málaga (Spain)

**Software Engineer:**

* Developer in Adobe Illustrator plugin to INDITEX Company. C++, Web services (gSoap).
* Developer of management software to FREMAP healthcare insurance company. Visual Studio 6.0, C, Informix (4GL), UNIX (Solaris).

**01/04/2011 – 09/10/2011: *INCIDE.*** Carrión, 8. 29013 Málaga (Spain).

**Software Developer:**

* Development a Human Resources Manage System. C#, SQL Server.

**01/06/2006 – 31/03/2011: *Fundación por la Solidaridad Tecnológica (SOLITEC-Solydi).*** Marea Baja, 19. 29006 Málaga (Spain).

**Software Developer:**

* Development a LED outdoor lamps System to control every outdoor lamps in a town, C#, .NET 3.0, UML, TCP, SQL, MySQL.
* Development and maintaining of a distributed multithreaded software to management status monitoring of LED light displays. C++, TCP, SQL, MySQL.
* Maintaining of the information system for business management. Java, JSP, JavaScript, HTML, Apache TomCat, SQL, MySQL, Batch.

**Programmer / Internship:**

* Development of an information system for business management. Java, JSP, JavaScript, HTML, Apache TomCat, SQL, MySQL, Batch.

EDUCATION

Computer Sciences (2007, *Ingeniero Técnico*, equivalence to Bachelor). *Superior Technical College of Computing Engineering. University of Málaga.*

PROFESSIONAL SKILLS

* More than 5 years:
  + C#, C++, SQL, MySQL.
* More than 2 years:
  + C, Java, JSP, JavaScript, HTML, XML, UML, 4GL, Informix.

**LANGUAGE SKILLS**

* **Spanish:** Native, C2.
* **English:** High, C1.
* **French:** Basic, A2.

**ADDITIONAL SKILLS**

* *European Driving license B.*

**HOBBIES**

* *Guitar player.*

Attn: Human Resources Section

Ref:

For the last five years I have been working in the GeonX – GE Additive as Software Engineer. In this time I have discovered the interaction and personal fulfilment in a multi-cultural and multi-lingual environment. This has been one of the most enriching and positive experiences of my career. Also I have had the opportunity of work in the Defense and Aerospace industries. So I believe that this position in would be an exciting challenge.

As you can see from my resume, I have been working in recent years as a Software Engineer. I have developed in a large number of programming languages. So I am capable to adapt quickly to new environments.

I am very proactive, but my major strength is my problem-solving ability. For this reason I believe this makes me a good candidate for the job.

I would very much appreciate the opportunity of an interview. Please feel free to contact me via telephone or email.

Thank you for your consideration. I am looking forward to hearing from you.

Yours sincerely,

Manuel Rojas Suarez