**Método de la Ingeniería**

**Gestión Eficiente de Base de Datos de Personas**

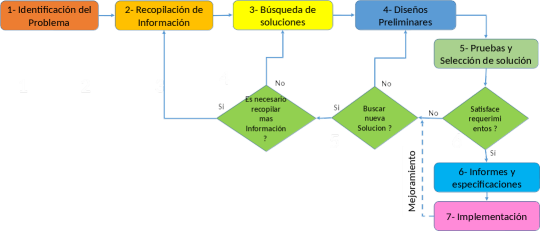
**Contexto Problemático**

Una monitoria en un proyecto de investigación interna de la Universidad Icesi, como parte del Equipo VIP de Simulación. Es necesario el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD sobre una base de datos de personas de nuestro continente, y para esto es necesario desarrollar un programa que lleve a cabo la generación de todos estos registros de personas de acuerdo con condiciones especiales.

**Desarrollo de la Solución**

A lo largo del presente documento se evidenciará el uso del método de ingeniería para resolver de la mejor forma el problema que se busca solucionar.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, el método se llevara a cabo de la siguiente forma dividido en las 7 fases.



**Paso 1. Identificación del Problema**

Se reconocen de manera concreta las necesidades propias de la situación problemática, así como sus síntomas y condiciones bajo las cuales debe ser resuelta.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre o identificador | R1-Autgenerar todos los datos de las personas | | |
| Resumen | El sistema debe permitir tener una opción para que se autogenere toda la información posible para simular que la base de datos este llena, debe autogenerar toda la información por cada persona en la base de datos(El usuario puede seleccionar cuantas personas deseara autogenerar). | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
| NumPersonas | int |  |
| Actividades generales necesarias para obtener los resultados | Ingresar al apartado de autogeneración de datos. | | |
| Resultado o postcondición | El sistema muestra un mensaje cuando se han generado satisfactoriamente todos los datos. | | |
| Salidas | Nombre salida | Tipo de dato | Condición de selección o repetición |
| Mensaje sobre autogeneración | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre o identificador | R2-Agregar una nueva persona | | |
| Resumen | El sistema debe permitir al usuario el registro de una persona a la base de datos, cada persona cuenta con código, pero este siempre es autogenerado, nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía. | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
| Nombre | String |  |
| Apellido | String |  |
| Sexo | String |  |
| Fecha de nacimiento | Date |  |
| Estatura | String | No puede ser negativo. |
| Nacionalidad | String |  |
| Actividades generales necesarias para obtener los resultados | Ingresar al apartado de registrar una persona en la base de datos | | |
| Resultado o postcondición | La persona será almacenada en la base de datos satisfactoriamente. | | |
| Salidas | Nombre salida | Tipo de dato | Condición de selección o repetición |
| MensajeAgregado | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre o identificador | R3-Buscar en la información almacenada | | |
| Resumen | El sistema debe permitir al usuario buscar por cualquiera de los parámetros existentes a una o un grupo de personas que el usuario requiera. | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
| tipoBusqueda | String |  |
| busqueda | String |  |
| Actividades generales necesarias para obtener los resultados | Ingresar al apartado de búsqueda en la base de datos del programa. | | |
| Resultado o postcondición | Salen los primeros 20 resultados de búsqueda. | | |
| Salidas | Nombre salida | Tipo de dato | Condición de selección o repetición |
| ListaPersonas | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre o identificador | R4-Modificar la información de una persona especifica | | |
| Resumen | El sistema debe permitir al usuario modificar la información que quiera de una persona previamente seleccionada por el mismo usuario. | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
| Nombre | String |  |
| Apellido | String |  |
| Sexo | String |  |
| Fecha de nacimiento | Date |  |
| Estatura | String | No puede ser negativo. |
| Nacionalidad | String |  |
| Actividades generales necesarias para obtener los resultados | Haber realizado la búsqueda de alguien, e ingresar a la parte de modificar información de la persona. | | |
| Resultado o postcondición | Datos modificados satisfactoriamente. | | |
| Salidas | Nombre salida | Tipo de dato | Condición de selección o repetición |
| MensajeModificacion | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre o identificador | R4-Eliminar la información de una persona especifica | | |
| Resumen | El sistema debe permitir al usuario eliminar a una persona de la base de datos previamente seleccionada por el mismo usuario. | | |
| Entradas | Nombre entrada | Tipo de dato | Condición de selección o repetición |
|  |  |  |
| Actividades generales necesarias para obtener los resultados | Haber realizado la búsqueda de alguien, e ingresar a la parte de eliminar información de la persona. | | |
| Resultado o postcondición | Datos Eliminados satisfactoriamente. | | |
| Salidas | Nombre salida | Tipo de dato | Condición de selección o repetición |
| MensajeEliminación | String |  |

*Definición del Problema*

Se requiere un programa que lleve a cabo auto generación de todos estos registros de personas, así como también permita buscar, editar, borrar y agregar registros a la base de datos.

**Paso 2. Recopilación de Información**

Con el objetivo de tener total claridad en los conceptos involucrados se hace una recopilación de toda la información que tenemos y si es necesario adicionar mas para resolver de manera efectiva el problema.

Principalmente tenemos la siguiente información del problema:

-El numero de habitantes de el continente americano es poco mas de mil millones de personas.

-Se tienen 6782 nombres entre los cuales 3437 son de hombre, y los 3435 nombres restantes perteneces a nombres de mujer.

-Se tienen 162255 apellidos.

-La distribución de las edades tiene que seguir las siguientes condiciones:

0-14 años: 18,62% (hombres 31.329.121 /mujeres 29.984.705)

15-24 años: 13,12% (hombres 22.119.340 /mujeres 21.082.599)

25-54 años: 39,29% (hombres 64.858.646 /mujeres 64.496.889)

55-64 años: 12,94% (hombres 20.578.432 /mujeres 22.040.267)

65 años y más: 16,03% (hombres 23.489.515 /mujeres 29.276.951).

**Operaciones CRUD**

En programación, se conoce como CRUD (Create, Read, Update y Delete) a las cuatro funciones básicas del almacenamiento persistente, funciones que es necesario que tenga el programa que se desarrollara.

Para desarrollar el programa se debe tener conocimiento de árboles binarios, AVL y rojo y negro.

-Un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho.

-Un árbol AVL es un tipo especial de árbol binario, ya que este se balancea solo.

-Un Árbol rojo-negro, un árbol de búsqueda binario, pero agregar un bit de almacenamiento a cada nodo indica el color del nodo, que puede ser Rojo o Negro.

Al restringir la coloración de cada nodo en cualquier camino desde la raíz hasta las hojas, el árbol rojo-negro asegura que ningún camino crecerá dos veces más que los otros caminos, por lo que está cerca del equilibrio.

**Paso 3. Búsqueda de Soluciones Creativas**

Para encontrar una solución al problema planteado, se tienen una serie de posibles soluciones para el programa, donde se centrarán las soluciones en la búsqueda, almacenamiento y generación de datos automáticamente:

* **Soluciones generación de datos:**

**Alternativa 1:**

Utilizar el archivo de nombres y el archivo de apellidos, y generar la máxima cantidad de combinaciones nombre-apellido posibles, de forma que quede un nombre con todos los apellidos posibles, y así con todos los nombres, nos daría que la máxima cantidad de nombres y apellidos generados serian, (6782\*162255) que nos daría 1100413410 nombres y apellidos generados lo cual es un poco más de mil millones.

**Alternativa 2:**

Utilizar Random para que los nombres generados sean más variados y aleatorios. Consiste en generar un numero aleatorio entre 1 y 6782 (cantidad de nombres) y otro numero aleatorio entre 1 y 162255 (cantidad de apellidos), y realizar este proceso mil millones de veces para tener una base de datos variada y aleatoria.

* **Soluciones búsqueda y almacenamiento:**

**Alternativa 1:**

Hacer uso de un árbol binario AVL para el almacenamiento de las personas por cada criterio de búsqueda, y que cada nodo contenga un objeto persona, y se organice con base en el criterio, de esta manera se podrá acceder al árbol del criterio requerido y realizar la búsqueda.

**Alternativa 2:**

Hacer uso de un árbol binario AVL que contenga más arboles AVL en sus nodos en el almacenamiento de la información de las personas por nombre y apellido, funcionaria haciendo un árbol principal que contenga nodos con todos los nombres sin repetir, y que cada nodo que a su vez es un árbol distinto contenga todos los apellidos que tienen ese nombre, y de esta forma tener un árbol para búsquedas por nombre, de la misma forma con los apellidos en un árbol distinto.

**Alternativa 3:**

Hacer uso de arboles binario AVL que contenga mas arboles AVL, funcionaria de forma que el árbol principal es un árbol de todo el abecedario, dentro de cada nodo con la letra del abecedario habrá otro árbol que posea todos los nombres que empiezan con esa letra, y cada nodo nombre tendrá otro árbol con nodos de apellido.

**Idea para mejorar 1:**

Realizar arboles binarios Rojo y negro para optimizar los algoritmos y la eficacia del programa.

**Paso 4.  Transición de las Ideas a los Diseños Preliminares**

**Generación de datos:**

Después de tener todas las ideas de diseño, se descartan las ideas menos factibles, como en la generación de datos, descartamos **la Alternativa 2** debido a que al estar realizando pruebas con la generación de datos se noto que al usar Random no se tenia un buen control de lo aleatorio del proceso, y se repetía muchas veces algunos números, esto se podría arreglar con condiciones y guardando los números aleatorios para evitar que se repitan muchas veces y de alguna forma tener un mayor control de estos, pero como esta alternativa necesitara más almacenamiento y al tener que repetir esto tantas veces podría ser un problema, por eso se usara la **alternativa 1-generacion de datos**.

* **Alternativa 1-Generacion de datos:**

Esta alternativa nos permite generar 1100413410 nombres por lo que la alternativa ya cumple con generar mil millones de personas, y como solo seria necesario 2 contadores y ciclos, por ahora representa la alternativa más efectiva.

**Búsqueda y almacenamiento:**

Se descarta **la alternativa 1**, es menos efectiva en eficiencia, porque este árbol solo tendría los mil millones de nombres, y forzaría al código a buscar desde la raíz cada nombre.

* **Alternativa 2-Busqueda y almacenamiento:**

-Permite acceder a todas las personas que comparten el criterio de búsqueda, pero este posee el problema de que se accede al árbol de las personas que comparten el nombre, y esto podría traer problemas al querer mostrar personas de mismas iniciales, pero diferentes nombres, o lo mismo con cualquier criterio de búsqueda.

* **Alternativa 3-Busqueda y almacenamiento:**

-Permite acceder a todas las personas que comparten la misma letra inicial, de esta forma los datos ya estarían más organizados, y se podría acceder fácilmente a cada persona.

**Paso 5. Evaluación y Selección de la Mejor Solución**

En el caso **de generación de datos** solo nos quedó la **alternativa 1**, por lo que esta es la mejor opción, y en el caso de **búsqueda y almacenamiento** tenemos l**a alternativa 2 y 3**, para los cuales se observa que las 2 alternativas poseen una cierta similitud, pero **la alternativa 3** finalmente es la que se puede ver como una mejora de la otra alternativa, lo que hace que esta sea mas precisa y logre agrupar mejor los datos de las personas, quedando **la alternativa 3** como la mejor solución.

Después de seleccionar la mejor solución, sabemos que esta es la mejor solución con árboles binarios AVL, pero también al indagar en los arboles binarios rojo y negro, podríamos realizar la idea de **la alternativa 3,** combinada con el hecho de usar arboles binarios Rojos y Negros.

**Solución Final:**

Almacenar la información de la forma que se dio en la alternativa 3, pero reemplazar los árboles AVL por arboles de búsqueda rojos y negros.

**Paso 6. Preparación de Informes y Especificaciones**

La solución permitirá suplir los requerimientos de la mejor forma encontrada.

* **Observaciones:**

-El número máximo de personas que plantea el problema, se puede lograr pero realizar esta solución trae consigo problemas en la memoria y tiempo de ejecución del código, ya que realizando pruebas, al serializar las mil millones de personas el archivo quedaría mas pesado que 50 gb de memoria, demorando este demasiado tiempo, se estima mas de 30 min en serializar y autogenerar.

Finalmente teniendo la mejor solución se realiza un diagrama de flujo que intenta mostrar el flujo del programa y sobre todo como interactúa con la base de datos.

Diagrama

Descripción generada automáticamente

**Paso 7. Implementación del Diseño**

**Ventanas:**

**Diagrama

Descripción generada automáticamente**

**Estructuras:**

**Diagrama, Esquemático

Descripción generada automáticamente**

**Codigo de un nodo de árbol rojo y negro:**

**public** **class** BRNode<K **extends** Comparable<K>,V> **extends** AVLNode<K,V>{

**private** Color color;

**private** BRNode<K, V> dad;

**private** BRNode<K, V> uncle;

**private** BRNode<K, V> left;

**private** BRNode<K, V> right;

**public** BRNode(K key, V value) {

**super**(key, value);

color = Color.***RED***;

}

**public** BRNode() {

**super**(**null**, **null**);

}

@Override

**public** BRNode<K, V> getDad() {

**return** dad;

}

**public** **void** setDad(BRNode<K, V> dad) {

**this**.dad = dad;

**if**(dad!=**null**) {

**if**(dad.getDad()!=**null**) {

**if**(dad.getDad().getRight()==dad)

**this**.uncle = dad.getDad().getLeft();

**else**

**this**.uncle= dad.getDad().getRight();

}

}

}

**public** BRNode<K, V> getUncle() {

**return** uncle;

}

**public** **void** setUncle(BRNode<K, V> uncle) {

**this**.uncle = uncle;

}

**public** BRNode<K, V> getLeft() {

**return** left;

}

**public** **void** setLeft(BRNode<K, V> left) {

**this**.left = left;

}

**public** BRNode<K, V> getRight() {

**return** right;

}

**public** **void** setRight(BRNode<K, V> right) {

**this**.right = right;

}

**public** Color getColor() {

**return** color;

}

**public** **void** setColor(Color color) {

**this**.color = color;

}

}

**Codigo de autogeneración de datos:**

**public** **void** createCombinations() **throws** IOException {

String[] names = *importNames*();

String[] lastNames = *importLastNames*();

**int** numI=1000;

**int** numJ=1000;

**int** total=num;

String[ ] paises = **new** String[] {"Canada","Estados Unidos","Mexico","Belice","Costa rica","El Salvador",

"Guatemala","Honduras","Nicaragua","Panamá","Argentina","Bolivia","Brasil","Chile","Colombia","Ecuador",

"Paraguay","Peru","Surinam","Trinidad y Tobago","Uruguay","Venezuela","Antigua y Barbuda","Bahamas","Barbados",

"Cuba","Dominica","Granada","Guyana","Haiti","Jamaica","República Dominicana","San Cristóbal y Nieves","San Vicente y las Granadinas","Santa Lucia"};

**double**[] poblacionesDouble=**new** **double**[] {0.03\*total,0.25\*total,0.10\*total,0.01\*total,0.01\*total,0.01\*total,0.02\*total,0.01\*total,0.01\*total,0.01\*total,0.04\*total,0.01\*total,0.18\*total,0.02\*total,0.05\*total,0.02\*total,0.01\*total,0.03\*total,0.01\*total,0.01\*total,0.01\*total,0.02\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total,0.01\*total};

**int** [] poblaciones=**new** **int**[poblacionesDouble.length];

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

poblaciones[i]=(**int**) poblacionesDouble[i];

}

**double** contador=1;

**double** c=0;

**int** k = 0;

**int** contadorNames=0;

**boolean** out2=**false**;

**for**(**int** i = 0; i<numI && !out2; i++) {

**for**(**int** j = 0; j<numJ && !out2; j++) {

String name = names[i].split(",")[0];

String sexString = names[i].split(",")[1];

String lastName = lastNames[j];

String code = k+"";

//System.out.println(name);

**if**(name!=**null** && sexString!=**null** && lastName!=**null** && code!=**null**) {

Sex sex = **null**;

**if**(sexString.equalsIgnoreCase("BOY")) {

sex = Sex.***MALE***;

} **else** **if**(sexString.equalsIgnoreCase("GIRL")) {

sex = Sex.***FEMALE***;

}

LocalDate startDate = LocalDate.*of*(1990, 1, 1);

**long** start = startDate.toEpochDay();

LocalDate endDate = LocalDate.*now*();

**long** end = endDate.toEpochDay();

ZoneId defaultZoneId = ZoneId.*systemDefault*();

**long** randomEpochDay = ThreadLocalRandom.*current*().longs(start, end).findAny().getAsLong();

Date date = Date.*from*(LocalDate.*ofEpochDay*(randomEpochDay).atStartOfDay(defaultZoneId).toInstant());

**double** randomHeight=*getRandom*(100,210);

randomHeight=randomHeight/100;

String nationality="";

**if**(total>100) {

**boolean** out=**false**;

**while**(!out) {

**int** randomNationality=*getRandom*(1, 35);

**if**(poblaciones[randomNationality]>0) {

nationality=paises[randomNationality];

out=**true**;

}

}

}**else** {

**int** randomNationality=*getRandom*(1, 35);

nationality=paises[randomNationality];

}

Person person = **new** Person(k+"", name, lastName, sex, date, randomHeight, nationality);

Main.*addPerson*(k,person);

c=contador/100;

**double** t=total\*c;

**if**(k/t==1) {

autoGeneDataPB.setProgress(contador/100);

contador=contador+1;

}

k++;

Main.*setCodeAux*(k);

**if**(k==num) {

out2=**true**;

}

}

}

}

autoGeneDataPB.setProgress(1);

closeBTN.setDisable(**false**);

}