INTEGRANTES:

Gabriel Restrepo Giraldo

Juan Camilo Gonzalez Torres

Método de la ingeniería:

Contexto del problema:

Se plantea el desarrollo de un software que permita gestionar eficientemente las

operaciones de crear, leer, actualizar y eliminar los registros de personas en una base de datos que refleja las condiciones reales de todo el continente de América, esto, teniendo en cuenta que el continente tiene poco más de mil millones de personas hasta el 2020. En dicho registro cada persona debe contar con un código, nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y una fotografía. El sistema debe permitir generar de forma automática dicha magnitud de datos. Además estos datos deben ser persistentes en el tiempo. El usuario final podrá interactuar con la base de datos generada por el software mediante una interfaz desarrollada para cumplir con la inserción, edición, eliminación y consulta de cada registro.

Definición del problema

Se debe lograr gestionar eficientemente las operaciones CRUD (Create, Read, Update y Delete) sobre una base de datos representativa de todo el continente de América.

Etapa 1: Requerimientos funcionales y no funcionales

Funcionales

- **R1:** El programa debe generar una base de datos de aproximadamente 1000 millones de personas. Cada persona tiene un código autogenerado, nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía. Para generar la base de datos, el programa

emplea un “data-set” de nombres y otro de apellidos, cuya combinación será la cantidad total de registros. El sexo será distribuido de forma que haya la misma cantidad de hombres y mujeres. La fecha de nacimiento y la estatura son generadas aleatoriamente, de forma que correspondan con la distribución de edad de Estados Unidos y con un intervalo que tenga sentido, respectivamente. La nacionalidad se asignará de forma que se conserve la proporción de personas de cada país de América. La fotografía de la persona se generará aleatoriamente por medio de un asistente de IA.

- **R2:** El programa debe tener una opción para empezar a generar los datos siguiendo las especificaciones anteriores. Debe tener una barra de progreso si el proceso tarda más de 1 segundo en terminar, y debe indicar cuánto tiempo se demoró la operación. La opción de generar debe tener un campo de texto en el cual se pueda digitar cuántos registros se desea generar. Por defecto, debe estar en el campo el máximo valor posible.
- **R3:** El programa debe permitir al usuario agregar una nueva persona con todos sus datos.
- **R4:** El programa debe permitir al usuario eliminar a una persona del registro.
- **R5:** El programa debe permitir al usuario actualizar los datos de una persona en el registro.
- **R6:** El programa debe permitir al usuario realizar una búsqueda por criterio de Nombre, Apellido, Nombre Completo o código.

No funcionales

- **R7:** Una vez los datos se generan, debe haber una opción para guardarlos en la base de datos del programa, y así poderlos consultar posteriormente.
- **R8:** Todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).
- **R9:** La interfaz con el usuario deberá contar con un formulario para agregar a una nueva persona, otro para buscar a una persona por los criterios mencionados más adelante y uno más para actualizar los campos de una persona existente. Este último formulario debe también tener una opción para eliminar a una persona.

- **R10:** El formulario para agregar debe tener todos los campos requeridos (menos el código, que es autogenerado) para la información de una persona y la opción de Guardar.
- **R11:** El formulario para actualizar una persona debe tener todos los campos editables (menos el código, que no se puede actualizar) de información de una persona, cargados con su información actual, la opción de Actualizar (para guardar los cambios, si hubo) y la opción Eliminar (si se desea eliminar a esta persona).
- **R12:** El programa debe permitir que, en la medida en que se digite los caracteres de búsqueda en el campo (para los criterios búsqueda por Nombre, Apellido o Nombre Completo), vayan apareciendo en una lista emergente debajo del campo, máximo 100 nombres (parametrizable) de la base de datos, que empiecen con los caracteres digitados hasta el momento.
- **R13:** Mientras se hace la búsqueda, debe mostrarse al lado del campo donde se digita la cadena, un número que indica la cantidad total de elementos que hasta el momento coinciden con el prefijo digitado.
- **R14:** En el momento en que haya 20 (parametrizable) o menos elementos que coinciden con la búsqueda, debe desplegarse un listado con los registros que coinciden y un botón al lado de cada registro con la opción de Editar, que nos llevará al formulario con la posibilidad de modificar o eliminar ese registro.
- **R15:** La interfaz del programa debe tener un componente menú en la parte superior de la ventana que permite cambiar todos los elementos de la ventana cada vez que se esté trabajando en opciones diferentes.

Etapa 2: Investigación de estructuras modelar el problema.

• **Marco teórico**

- **Estructura de datos:** Son una forma de almacenar y ordenar información estructuradamente con el fin de realizar sobre estas distintas operaciones de manera eficiente. Existen distintos tipos de estructuras de datos que son útiles de acuerdo con el contexto del problema. Por ejemplo, un

- **AVL Tree:** Un árbol AVL es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-Velskii y Landis. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó.
- **Binary Search Tree (BST):** Un árbol binario de búsqueda (BTS) es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x ,y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x.
- **Trie:** es una estructura de datos que se utiliza en strings. Se puede utilizar para realizar búsquedas de manera óptima, con una complejidad casi lineal (dependiendo de la implementación y las restricciones del problema). La idea del Trie es armar un árbol, donde los nodos son las letras de las palabras. Las letras que están en rojo indican los nodos donde termina una palabra. Es decir, si recorremos el árbol desde la raíz hasta ese nodo, formaremos una de las palabras iniciales. Si lo recorremos en Preorden, podemos ordenar alfabéticamente a las palabras que forman el Trie. Los números escritos expresan el orden alfabético de las palabras.
- **Árbol rojo-negro:** Un árbol rojo-negro es un tipo abstracto de datos. Concretamente, es un árbol binario de búsqueda equilibrado, una Estructura de datos utilizada en informática y ciencias de la computación. La estructura original fue creada por Rudolf Bayer en 1972, que le dio el nombre de “árboles-B binarios simétricos”, pero tomó su nombre moderno en un trabajo de Leo J. Guibas y Robert Sedgewick realizado en 1978. Es complejo, pero tiene un buen peor caso de tiempo de ejecución para sus operaciones y es eficiente en la práctica. Puede buscar, insertar y borrar en un tiempo O(log n), donde n es el número de elementos del árbol.

  Un árbol rojo-negro es un árbol binario de búsqueda en el que cada nodo tiene un atributo de color cuyo valor es rojo o negro. En adelante, se dice

que un nodo es rojo o negro haciendo referencia a dicho atributo.

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

1. Todo nodo es o bien rojo o bien negro.

2. La raíz es negra.

3. Todas las hojas (NULL) son negras.

4. Todo nodo rojo debe tener dos nodos hijos negros.

Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros. Estas reglas producen una regla crucial para los árboles rojo-negro: el camino más largo desde la raíz hasta una hoja no es más largo que dos veces el camino más corto desde la raíz a una hoja. El resultado es que dicho árbol está aproximadamente equilibrado.

Fuentes:

- OpenWebinars.net. 2020. ¿Qué Son Las Estructuras De Datos Y Por Qué Son Útiles?. [online] Available at: https://openwebinars.net/blog/queson-las-estructuras-de-datos-y-por-que-son-tan-utiles/
- Sites.google.com. 2020. Arboles AVL - Estructuras De Datos En Java. [online] Available at: https://sites.google.com/a/espe.edu.ec/programacionii/home/arboles/arboles-avl
- Decsai.ugr.es. 2020. ÁRBOLES BINARIOS DE BÚSQUEDA. [online] Available at: https://ccia.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/abb.htm
- TIC Portal. 2020. Base De Datos : ¿Qué Tipos Hay Y Cómo Funciona Conectada A Un Software?. [online] Available at: https://www.ticportal.es/glosario-tic/base-datos-database
- Es.wikipedia.org. 2020. CRUD. [online] Available at:
- https://es.wikipedia.org/wiki/CRUD

- Árbol rojo-negro. Es.wikipedia.org. (2020). Retrieved 7 November 2020, [online] Available at: https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro.

Etapa 3: Alternativas de solución

1. Idea 1: Recibir los datos a partir de los cuales se crean 1000 millones de combinaciones mediante un ArrayList que almacene ambos conjuntos (nombres y apellidos). Fusionarlos en un arraylist y manejar las operaciones CRUD.

2. Idea 2: Diseñar e implementar un software que utilice una LinkedList y sea capaz de generar registros en base a lo requerido y tenga persistencia que aplique las operaciones CRUD.

3. Idea 3: Diseñar e implementar un software que utilice como estructura de datos un Array de tamaño definido que almacene los usuarios generados aleatoriamente y permita realizar las operaciones CRUD sobre la base de datos, para después realizar serialización mediante archivos de texto.

4. Idea 4: Diseñar e implementar un software que por cada usuario generado lo haga persistente en un archivo csv de tal forma que cuando se cree un usuario inmediatamente se escriba en el archivo y cuando se vayan a realizar las operaciones CRUD ingrese al archivo busque el usuario y modifique su campo.

5. Idea 5: Diseñar e implementar un software que utilice como estructuras de

datos árboles balanceados y poder realizar las operaciones CRUD en la base de datos. al mismo tiempo generar búsquedas de usuarios y cuando los resultados sean menores o iguales a 20 desplegar un menú de sugerencia de posibles usuarios buscados.

6. Idea 6: Diseñar e implementar un software que utilice como estructura de datos un Trie que almacene los usuarios generados aleatoriamente, sea persistente mediante archivos de texto y aplique las operaciones CRUD sobre la base de datos.

Etapa 4: Ideas descartadas.

Ideas descartadas: Se descarta el uso de estructuras como ArrayList, Arrays y LinkedList, dado que estas estructuras saturan en sobremanera la capacidad de memoria ram dedicada del jdk para procesos gestionados por la java collector. (ideas 1, 2 y 3)

Ideas posibles:

- **Idea 2: Archivos CSV**

Se propone esta alternativa con fines de trabajar directamente sobre el archivo final en formato csv. Esto supone el manejo de la escritura sobre el mismo todo el tiempo. El tiempo de búsqueda sobre este formato sería algo lento, pero consideramos que puede ser manejable.

• Idea 5: Árboles

A esta alternativa la podríamos enfocar en árboles AVL o rojinegros, que son estructuras de datos eficientes y facilitan la realización de las diferentes operaciones en un menor tiempo.

• Idea 6: Trie

Usando la estructura de datos Trie. Se puede utilizar para realizar búsquedas de manera óptima, con una complejidad casi lineal, es una muy bueno opción para realizar operaciones CRUD pero no tenemos mucho conocimiento acerca de la implementación.

Etapa 5: Evaluación y selección de soluciones.

Al final nos hemos decantado por usar el AVL tree, pues para la búsqueda entre tal magnitud de datos requiere una estructura cuya complejidad temporal de búsqueda sea $O(log\ n)$. Quedaría de esta manera descartada la LinkedList, pues su complejidad temporal de búsqueda es de $O(n)$. Por otro lado un CRUD nos fué difícil de imaginar una implementación.