GOBIERNO DE MÉXICO | CONAHCYT CONSEJO NACIONAL DE HUMANIDADES CIENCIAS Y TECNOLOGÍAS | INFOTEC

BIBLIOTECA INFOTEC
VISTO BUENO DE TRABAJO TERMINAL

Maestría en Gestión de Innovación de las Tecnologías de Información y Comunicación (MGITIC)

Ciudad de México, a 5 de enero de 2024

UNIDAD DE POSGRADOS
PRESENTE

Por medio de la presente se hace constar que el trabajo de titulación:

"Manual de estrategias de búsqueda en expedientes electrónicos"

Desarrollado por el alumno: **Genaro Antonio Arcos Orozco**, bajo la modalidad del **Diplomado en Derecho, TIC e Innovación del INFOTEC** cumple con el formato de Biblioteca, así mismo, se ha verificado la correcta citación para la prevención del plagio; por lo cual, se expide la presente autorización para entrega en digital del proyecto terminal al que se ha hecho mención. Se hace constar que el alumno no adeuda materiales de la biblioteca de INFOTEC.

No omito mencionar, que se deberá anexar la presente autorización al inicio de la versión digital del trabajo referido, con el fin de amparar la misma.

Sin más por el momento, aprovecho la ocasión para enviar un cordial saludo.

Mtro. Carlos Josué Lavandeira Portillo
Director Adjunto de Innovación y Conocimiento

CJLP/jah

C.c.p. Felipe Alfonso Delgado Castillo.- Gerente de Capital Humano.- Para su conocimiento
Genaro Antonio Arcos Orozco.- Alumno de la Maestría en Gestión de Innovación de las Tecnologías de Información y Comunicación (MGITIC).- Para su conocimiento.

MGITIC

MAESTRÍA EN GESTÍON DE INNOVACIÓN DE LAS TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN

INFOTEC CENTRO DE INVESTIGACIÓN E INNOVACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

DIRECCIÓN ADJUNTA DE INNOVACIÓN Y CONOCIMIENTO
GERENCIA DE CAPITAL HUMANO
POSGRADOS

“MANUAL DE ESTRATEGIAS DE BÚSQUEDA EN EXPEDIENTES ELECTRÓNICOS”

Bajo la modalidad de Diplomado
Que para obtener el grado de MAESTRO EN GESTIÓN DE INNOVACIÓN DE LAS TECNOLOGÍAS DE INFORMACIÓN Y COMUNICACIÓN

Presenta:

Genaro Antonio Arcos Orozco

Seattle, Washington, Estados Unidos de América, Noviembre 2023

CONACYT

INFOTEC POSGRADOS

MANUAL DE ESTRATEGIAS DE BÚSQUEDA EN EXPEDIENTES ELECTRÓNICOS

Strategies Guide for Searching in Electronic Files

Genaro Antonio Arcos Orozco[1*]

RESUMEN

Es común que en los sistemas informáticos se implementen expedientes electrónicos para dar seguimiento a procesos de negocio. Lo que no es común es implementar funcionalidad de búsqueda que sea similar a los portales de Google Search o Microsoft Bing. Esta manual presenta, explica, analiza, compara y propone algunos algoritmos de distancia de edición que son base para implementar funcionalidad similar a dichos portales.

PALABRAS CLAVE: motor de búsqueda, algoritmos de búsqueda, expedientes electrónicos, búsqueda aproximada de texto.

ABSTRACT

It is usual that on information systems, to implement electronic files for keeping track of business processes. What does not always happen is to implement a search functionality over those files in a similar way to the public web sites Google Search or Microsoft Bing. This guide presents, explains, analyze, compare, and proposes some edit distance algorithms that are the base for implementing a similar functionality as the ones on said web sites.

KEYWORDS: search engine, search algorithms, electronic files, approximate string matching.

[1*] Ingeniero en Tecnologías Computacionales, genaro.arcos@gmail.com;

1. Introducción

1.1. Descripción del proyecto

Este proyecto busca generar una manual para facilitar la implementación búsquedas en sistemas de expedientes electrónicos. De tal manera que estas sean similares en comportamiento y resultados a los motores de búsqueda en Internet como son Google Search y Microsoft Bing. Se enumeran algunos algoritmos de búsqueda, se explica su funcionamiento y la lógica detrás del mismo. Se realizará una comparativa de los diversos algoritmos de búsqueda aproximada de texto con criterios enfocados a la su implementación en motores de bases de datos que implementen el lenguaje SQL, como pueden ser Microsoft SQL Server, MySQL, PostgreSQL, entre otros. Se elije SQL al ser este un estándar industrial muy popular para la implementación de sistemas de negocios. La tabla de la comparativa por tipo de dato es la base para generar una discusión sobre el uso del algoritmo. Esta discusión utiliza mi experiencia profesional como desarrollador de software para emitir una recomendación final, y en algunos casos comentarios sobre las mejores prácticas.

La audiencia de este manual es toda aquella persona que tenga en su responsabilidad: planeación, diseño y/o implementación de sistemas de software de negocio personalizados que incluyan expedientes electrónicos como unidad de información de trabajo. Tales profesiones incluyen: profesionales en desarrollo de software, profesionales en sistemas de información, profesionales en sistemas administrativos y similares.

1.2. Problemática

Típicamente las búsquedas en expedientes electrónicos se realizan comparando el parámetro de búsqueda contra todos los campos de datos del expediente. Estas búsquedas regresan resultados donde el parámetro es una coincidencia exacta relativo al parámetro. Dicho tipo de búsqueda simple en SQL puede proveer buenos resultados al usuario solamente cuando este ya sabe qué está buscando. En caso contrario esta búsqueda es inflexible y no necesariamente va a dar valor al usuario del sistema.

En la captura inferior se muestra la pantalla de búsqueda de cédulas profesionales de la Secretaría de Educación Pública de México.

Link:
https://www.cedulaprofesional.sep.gob.mx/cedula/presidencia/indexAvanzada.action

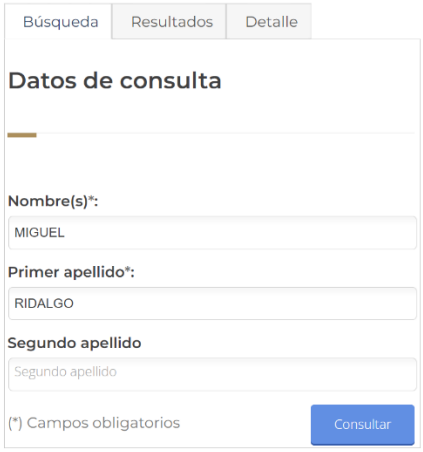


Ilustración 1 Panel de Búsqueda de cédulas profesionales. Se aprecian 3 campos de texto para ejecutar una búsqueda. Fuente: (Secretaria de Educacion Publica, Gobierno de Mexico, 2023)

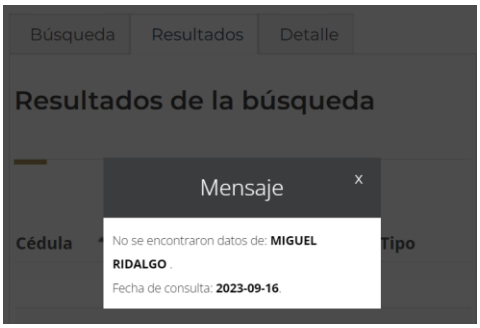


Ilustración 2 Búsqueda de cédulas profesionales, sin resultados para "Miguel Ridalgo". Fuente: (Secretaria de Educacion Publica, Gobierno de Mexico, 2023)

Esta búsqueda utiliza una sentencia WHERE nombre = "miguel" AND primerapellido = "ridalgo". Haciendo la corrección de "ridalgo" a "hidalgo" nos muestra estos resultados:

Ilustración 3 Resultados de buscar Nombre="Miguel" y PrimerApellido="Hidalgo", al menos dos coincidencias que difieren solamente en SegundoApellido. Fuente: (Secretaria de Educacion Publica, Gobierno de Mexico, 2023)

Los Motores de Búsqueda en internet como son Google Search y Microsoft Bing son el nuevo estado del arte en cuanto a las búsquedas de información con sistemas informáticos. Las búsquedas que implementan dichos sistemas son más simples al mostrar un solo campo de texto para la búsqueda. Por ejemplo, en la captura de pantalla inferior, buscamos "miguel ridalgo", y el motor de Bing provee resultados preliminares para "miguel hidalgo".

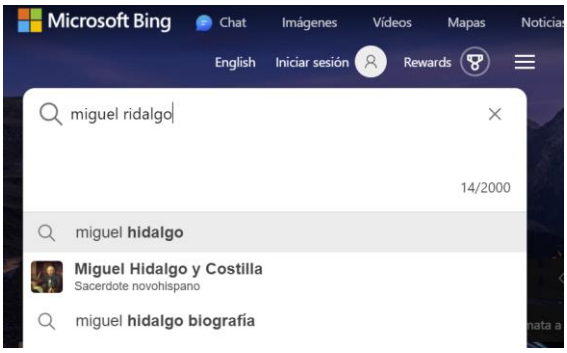


Ilustración 4 Panel de Bing para "miguel ridalgo", muestra resultados preliminares y extendidos del texto introducido. Fuente: (Microsoft Corporation, 2023)

Ilustración 5 Bing, página de resultados de buscar "miguel ridalgo", muestra “Miguel Hidalgo y Costilla”, además de preguntar si realmente está buscando “miguel ridalgo”. Fuente: (Microsoft Corporation, 2023)

Presionando la tecla Enter, Bing muestra resultados para “miguel hidalgo”, aclarando que está mostrando resultados sobre un parámetro propuesto similar al proporcionado, y adicionalmente pregunta al usuario si realmente está buscando el parámetro original “miguel ridalgo”.

Esta implementación de búsqueda es flexible y provee un valor mayor al usuario, especialmente cuando este no sabe exactamente lo que está buscando. En contraste, una búsqueda utilizando SQL con una sentencia WHERE nombre = “miguel ridalgo” va a regresar cero resultados en el caso de que Ridalgo no exista en la base de datos, tal como se mostró en el primer ejemplo.

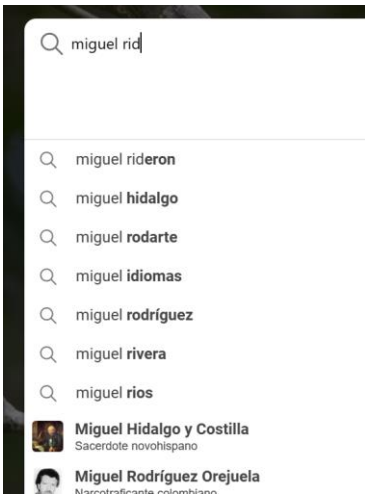


Ilustración 6 la función Autocompletar de Bing mostrando resultados que autocompletan lo escrito por el usuario, "miguel rid" como "miguel rideron", y otros resultados. Fuente: (Microsoft Corporation, 2023)

Dichos motores de búsqueda implementan 2 principales características

1. Búsqueda mientras escribes (search as you type), también llamado autocompletar. Esta capacidad permite ir mostrando 2 tipos de resultados: auto completar la frase que se está escribiendo y resultados preliminares de la consulta principal sugerida. En la ilustración 6 se muestra que escribiendo "miguel rid" el autocompletar muestra "miguel rideron", "miguel hidalgo", "miguel rodarte", etc.
2. Resultados por búsqueda tipo coincidencia aproximada de cadenas. En la ilustración 5 se muestra que buscando "miguel ridalgo" el motor muestra resultados para "Miguel Hidalgo", especificando que los resultados mostrados son para esta sugerencia, y adicionalmente la opción de buscar exclusivamente "Miguel Hidalgo".

La búsqueda mientras escribes y autocompletar, en mi experiencia, es una característica relativamente fácil de implementar. Existen frameworks de desarrollo web que lo incluyen en sus controles de textbox, de tal forma que el desarrollador tiene que unir un sistema backend con la característica de autocomplete en la interfaz gráfica. Al existir muchas implementaciones y recursos al respecto no amerita escribir un artículo más al respecto.

En contraste con el autocompletar, para la búsqueda por coincidencia aproximada de cadenas, Gonzalo Navarro (Navarro, 2001) se enfoca en analizar la teoría de cómo funcionan algunos algoritmos de distancias entre cadenas de texto. La mayoría de los recursos que encontré se enfocan en implementar búsquedas en contenido de texto para sistemas de Content Management, donde el contenido de búsqueda son amplios campos de texto del contenido que administran, como son: periódicos, foros, etc. Esto no aplica directamente para expedientes electrónicos.

1.3. Objetivo General

Crear un manual para facilitar la implementación de búsquedas en sistemas de expedientes electrónicos que tengan un comportamiento y resultados similares a los de los motores de búsqueda en Internet como son Google Search o Microsoft Bing, de tal forma que los resultados de la misma búsqueda sean más relevantes para el usuario.

1.4. Objetivos Específicos

Durante la creación del manual:

- Enumerar los diferentes tipos de registros de datos de los expedientes electrónicos a los cuales aplicar los algoritmos.
- Explicar brevemente el funcionamiento de los algoritmos de búsqueda en base a los registros de los expedientes.
- Explicar el funcionamiento de cada estrategia de búsqueda.

2. Marco Teórico

2.1. Antecedentes

La Real Academia Española (RAE), en su Diccionario de la lengua española, define “expediente” como “*conjunto de todos los papeles correspondientes a un*

asunto o negocio” (Real Academia Española, 2022). De manera similar, en el Diccionario panhispánico del español jurídico define “expediente electrónico” como “*conjunto de documentos electrónicos correspondientes a un procedimiento administrativo, cualquiera que sea el tipo de información que contengan*” (Real Academia Española, 2023). Siendo la principal diferencia lo referente al almacenaje electrónico de los documentos.

La utilización de los expedientes electrónicos se ha incrementado considerablemente debido a la facilidad de almacenar y procesar la información gracias a los mismos sistemas computacionales, producto de las Tecnologías de Información y Comunicaciones (TICs). Tanto la iniciativa privada como entidades de gobierno han adoptado la utilización de expedientes electrónicos como una forma de optimizar los procesos y servicios que ofrecen. De acuerdo con Sanchez Torres, C.A., *el concepto de “gobierno electrónico” incluye todas aquellas actividades basadas en las modernas tecnologías informáticas, en particular Internet, que el Estado desarrolla para aumentar la eficiencia de la gestión pública, mejorar los servicios ofrecidos a los ciudadanos y proveer a las acciones del gobierno de un marco mucho más transparente*, (Sánchez Torres & Rincón Cárdenas, 2004).

En general, para la implementación de expedientes electrónicos, el almacenamiento de la información de este se logra utilizando sistemas de bases de datos relacionales. El término “base de datos relacional” fue definido por E. F. Codd en 1970 en su artículo “Un Modelo Relacional de Datos para Grandes Bancos de Datos Compartidos”, (Codd, 1970). La mayoría de las implementaciones de bases de datos relacionales comerciales utilizan el lenguaje SQL (Structured Query Language). Este lenguaje fue adoptado como estándar internacional por la International Organization for Standardization (ISO) en 1987 (International Organization for Standardization, 1987). El lenguaje SQL permite realizar consultas donde el valor a buscar puede un valor exacto (columna = valor), no ser un valor

exacto (columna <> valor), un rango de valores (columna between valor1 and valor2), valores menos o mayores a uno especifico (columna > valor; columna < valor), o contenido dentro de un set de valores (columna IN (<set de valores>)).

La implementación particular de expedientes electrónicos depende completamente de la estructura del mismo expediente. En la definición de expediente como un conjunto de documentos, esto implica gran diversidad del contenido. El estándar ISO SQL define a nivel general 4 tipos de datos predefinidos: numéricos, caracteres, datetime e intervalo (International Organization for Standardization, 1987), y a su vez de cada categoría emanan diferentes tipos derivados: números enteros (exactos), números decimales, texto, fecha, fecha y hora, marca de tiempo (timestamp), por mencionar algunos, (Melton, 1996). Al ser un estándar todos los motores de bases de datos los implementan.

Todos estos tipos de datos se utilizan para definir los elementos del expediente electrónicos. Diseñando un ejemplo rápido, un expediente clínico electrónico puede tener:

- Datos del paciente:
    - Nombre (texto), Apellido(s) (texto), Fecha de nacimiento (fecha), tipo de sangre (catalogo), sexo (catalogo),
- Listado de Domicilios:
    - Calle (texto), número (texto), ciudad (texto o catalogo), estado, país (texto o catalogo), código postal (número o texto).
- Listado de Diagnósticos.
- Listado de Recetas de medicamentos.
- Listado de Citas
- Listado de pruebas de laboratorio.

No hay límite en la complejidad de la estructura de los diversos documentos y las relaciones entre los mismos. Además de las optimizaciones del esquema que pudieran definirse durante el desarrollo de la base de datos. Todo lo anterior impacta en cómo se va a implementar una búsqueda de los expedientes.

La búsqueda de expedientes electrónicos dependerá de:

- En qué datos se va a buscar, ejemplo, Nombre, Ciudad, Fecha de nacimiento, código postal, comentarios, etc.
- La naturaleza del dato a buscar, texto, fecha, numero, código postal, teléfono.

Existen diferentes algoritmos de distancia de edición entre cadenas de texto, los cuales son la base para la búsqueda aproximada. Este manual explorará los algoritmos de distancia de edición más comunes.

Ejemplo, si el usuario teclea la búsqueda “Robert”, se pueden utilizar diferentes algoritmos de distancia de edición entre cadenas de texto, como la distancia de Levenshtein o de Hamming que puede regresar “Roberto”, los cuales son similares en texto. Todos estos algoritmos se agrupan como Búsqueda Aproximada de Texto (Approximate String Matching)

2.2. Conceptos Clave

Búsqueda Aproximada de Texto, Approximate String Matching, Algoritmo Soundex, Distancia de Edición, Distancia de Levenshtein, Distancia de Jaro-Winkler, Distancia de Hamming, búsqueda FullText en Microsoft SQL Server.

2.3. Teoría

Distancia de Levenshtein, distancia de edición que permite inserciones, borrados y sustituciones de caracteres en una palabra. Es “*el mínimo número de inserciones,*

borrados y substituciones para volver iguales 2 cadenas de texto" (Navarro, 2001). El resultado de la función es un numero entero que representa la distancia entre 2 cadenas. Ejemplo

Cadena 1: casa Cadena 2: calle Resultado: 3	1. casa -> cala (sustitución S -> L) 2. cala -> calla (inserción L) 3. calla -> calle (sustitución A -> E)

Distancia de Hamming, distancia de edición que "*solo permite sustituciones* (de caracteres) *con costo 1 en su definición simplificada. […] es llamada 'coincidencias en cadenas de texto con k errores'*" (Navarro, 2001). Esta limitante lo hace menos flexible que Levenshtein.

Ejemplo con Hamming:

Cadena 1: arroz Cadena 2: zorra Resultado: 4	Posición: 1 A -> Z 2 R -> O 4 O -> R 5 Z -> A

Hamming encuentra 4 diferencias entre "arroz" y "zorra".

Ejemplo con Levenshtein:

Cadena 1: arroz Cadena 2: zorra Resultado: 2	1. arroz -> zrroa (sustitución A -> Z) 2. zrroa -> zorra (sustitución R -> O)

Levenshtein encuentra 2 diferencias entre "arroz" y "zorra"".

Microsoft Full-Text Search, es un conjunto de tecnologías contenidas en el motor de base de datos Microsoft SQL Server. Este permite configurar columnas de texto para realizar diferentes tipos búsquedas sobre las mismas, como son: palabras específicas, por prefijos, por sinónimos, etc. "*Las consultas Full-Text realizan las búsquedas lingüísticas en los datos de texto de los índices de Full-Text sobre las palabras y frases basándose en las reglas de un idioma determinado, […] Una consulta de Full-Text devuelve todos los documentos que contienen por lo menos*

una coincidencia (también se conoce como acierto).” (Microsoft Corporation, 2023). A deferencia de los algoritmos anteriores, y al ser un producto comercial completo, este no depende de explícitamente programar funciones adicionales en el motor de base de datos. Su implementación es labor de configurar los índices adecuados y seguir las instrucciones al respecto en la documentación provista por Microsoft. También, por ser un producto de caja, su costo es el de la licencia. En los casos donde el sistema de expedientes no está usando MS SQL Server no puede ser utilizado. En caso afirmativo es más rápido configurar esta característica siguiendo la documentación del producto.

3. Metodología

Al consistir el proyecto en un manual, este incluye los diferentes tipos de datos de los que típicamente consiste un expediente electrónico, así como las estrategias para cada uno de estos tipos de datos. Este hecho de analizar y comparar diferentes estrategias hace viable utilizar una Metodología de Trabajo Comparado. Resultado de la comparativa se establecerán las mejores prácticas para cada tipo de dato del expediente.

Para este proyecto seguiremos parcialmente las fases utilizadas por (Acevedo Villaroel & Rioseco Reinoso, 2011) las cuales para este documento son: definición de criterios, selección de los algoritmos de búsqueda, comparación, resultados y discusión, y conclusiones.

Primeramente, implementaremos los algoritmos para realizar búsquedas aproximadas de texto. El código fuente será tomado de fuentes públicas, con mínimas modificaciones para poder ser ejecutadas en una base de datos MS SQL Server.

Posteriormente utilizaremos una base de datos de pruebas. Microsoft distribuye gratuitamente una llamada Contoso, (Microsoft Corporation, 2010). Disponible en esta liga:

https://www.microsoft.com/en-us/download/details.aspx?id=18279

En esta base de datos seleccionaremos una tabla para la cual realizar las pruebas. La tabla de Empleados es suficiente (DimEmployee).

Crearemos un Stored Procedure para realizar búsquedas aproximadas de texto, al cual podamos cambiar rápidamente el algoritmo de búsqueda simplemente comentando/descomentando la llamada al stored procedure del mismo. Tentativamente lo llamaremos BusquedaEmpleado.

Para la comparativa utilizaremos los siguientes criterios:

- Tiempo de ejecución.
- Complejidad de implementación.

Tiempo de ejecución: En un motor de base de datos moderno toda búsqueda tiene un costo computacional en las siguientes métricas: tiempo de CPU, uso de memoria, uso de capacidad de I/O. Típicamente en un sistema computacional las búsquedas están optimizadas de diferentes formas, esto no necesariamente sucede con los algoritmos de búsqueda que propone este documento. Si bien para un administrador de base de datos es importante conocer el detalle del costo de ejecución de estos algoritmos en todas sus características, esto es demasiado complejo para el alcance limitado de este documento. Para simplificar este proceso nos limitaremos al tiempo aproximando de ejecución de cada algoritmo estableciendo 3 niveles de calificación: **alto, medio, bajo**.

Complejidad de implementación: Intrínsecamente, todo algoritmo consiste en código computacional. En algunos casos, el motor de la base de datos ya implementa nativamente algunos algoritmos, por lo que es relativamente fácil consumirlos en el proceso de búsqueda de expedientes. En caso contrario de no estar implementado, el desarrollador del sistema debe realizar la implementación completa a partir de código fuente público. Y extraordinariamente algunos

algoritmos necesitan el soporte de sistemas externos al motor de base de datos, lo que aumenta la complejidad de su implementación. Para este documento definiremos la complejidad de implementación en: **alta, media y baja**.

Tomando algunos ejemplos de (Navarro, 2001), para los tipos de datos de texto seleccionaremos los siguientes algoritmos:

- Distancia de Levenshtein
- Distancia de Hamming

La comparación del tiempo de ejecución se realiza principalmente en la complejidad del algoritmo, la famosa T(n)=O(n). Se puede obtener habilitado las métricas de tiempo del motor de base de datos y tomando el tiempo de ejecución.

La evaluación de la complejidad de implementación es completamente subjetiva. En este caso tomaré la complejidad del código del algoritmo para emitir un juicio.

El resultado de la comparativa será una tabla con los resultados de cada algoritmo y su evaluación de ambos criterios. Se explica el motivo de otorgar cada calificación en base a la experiencia profesional. De existir parámetros, consideraciones o comentarios adicionales se precisarán en esta sección.

Para emitir las conclusiones utilizaremos la tabla de resultados, ya que este es el artefacto principal para recomendar algoritmos por cada tipo de dato. De existir variaciones o consideraciones adicionales se describirán en esta sección.

Al final, y en base a mi experiencia personal, se va a describir las circunstancias específicas de un expediente electrónico donde el algoritmo mejor calificado podría no ser más apropiado.

4. Propuesta

4.1. Registros de datos en Expedientes.

Las bases de datos relaciones siguen una jerarquía de datos. Shauna Roch et al (Roch, Fowler, Smith, & Bourgeois, 2014) la plantean de esta manera:

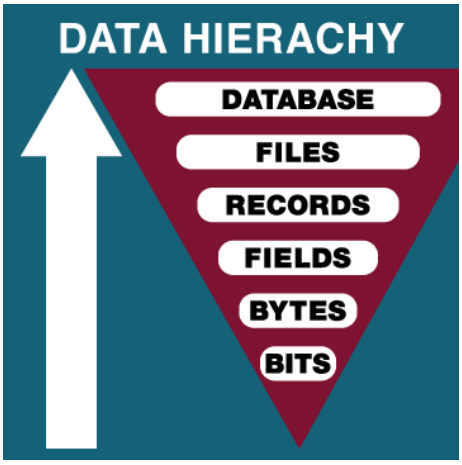


Ilustración 7, De abajo hacia arriba, de menor a mayor complejidad: bits, bytes, campos, registros, archivos/expedientes, base de datos. Fuente: (Roch, Fowler, Smith, & Bourgeois, 2014)

El concepto que Roch llama archivos (files) como "*una colección de registros relacionados, por ejemplo, toda la información de un estudiante (ID, nombre, especialización, correo electrónico)*" (Roch, Fowler, Smith, & Bourgeois, 2014), es lo que llamamos expedientes electrónicos.

En base a mi experiencia puedo decir que los diferentes tipos de registros que puede tener un expediente son:

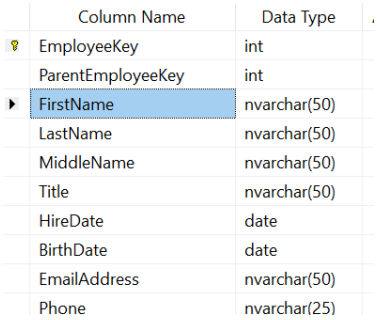
- Personas.
- Lugar, por ejemplo, domicilio.
- Texto libre.
- Fecha

Siguiendo el ejemplo del expediente clínico y subiendo el nivel de abstracción de campos a registros tenemos:

- Registro del paciente -> nombre, fecha de nacimiento.
- Listado de Domicilios -> listado de lugares.
- Listado de Diagnósticos -> listado de textos libres.

4.2. Búsqueda exacta con Operador LIKE

Para cada expediente es importante identificar cuales son los campos de importancia para las búsquedas, esto debido a que los algoritmos de búsqueda operan sobre campos de datos, no sobre los registros completos. Al ejecutarlos algoritmos debemos indicar columna por columna e ir guardando los que tengan una similitud adecuada. Adicional al uso de los algoritmos de búsqueda aproximada, es que se puede utilizar el operador LIKE de la sentencia WHERE. Para poder aprovecharla usaremos un truco que consiste en crear una columna de dato computado y persistido (*persisted computed column* en inglés) que concatene todos los campos de datos de interés de un registro. Usare de ejemplo la modificación de una tabla de Empleados.



Column Name	Data Type
EmployeeKey	int
ParentEmployeeKey	int
FirstName	nvarchar(50)
LastName	nvarchar(50)
MiddleName	nvarchar(50)
Title	nvarchar(50)
HireDate	date
BirthDate	date
EmailAddress	nvarchar(50)
Phone	nvarchar(25)

Ilustración 8 Parte del esquema de la tabla DimEmployee con las columnas que utilizaremos en los ejemplos. Fuente: Elaboración propia

Para buscar empleados nos interesan los siguientes campos: FirstName, LastName, MiddleName, Title, EmailAddress. Entonces agregamos una columna computada llamada “SearchEmployee” con la siguiente sentencia de SQL:

```
ALTER TABLE dbo.DimEmployee
       ADD SerchEmployee AS (CONCAT(
       FirstName
       , ' ',
       MiddleName, ' ',
       LastName, ' ',
       EmailAddress, ' ',
       Title)) PERSISTED
```

El tiempo de ejecución dependerá del tamaño de la tabla. Es importante concatenar un espacio en blanco entre cada campo. Posteriormente verificamos que la columna computada funcione adecuadamente con un query de prueba:

```
SELECT TOP 10
       e.FirstName, e.MiddleName, e.LastName
       , e.EmailAddress, e.Title, e.[SerchEmployee]
FROM DimEmployee e
```

Results | Messages

	FirstName	MiddleName	LastName	EmailAddress	Title	SerchEmployee
1	Kim	NULL	Abercrombie	guy1@contoso.com	Sales Region Manager	Kim Abercrombie guy1@contoso.com Sales Region Manager
2	Sagiv	NULL	Hadaya	kevin0@contoso.com	Sales Region Manager	Sagiv Hadaya kevin0@contoso.com Sales Region Manager
3	Luka	NULL	Abrus	roberto0@contoso.com	Sales Region Manager	Luka Abrus roberto0@contoso.com Sales Region Manager
4	Kirk	J.	Nason	KirkNason@contoso.com	Sales Region Manager	Kirk J. Nason KirkNason@contoso.com Sales Region Manager
5	Humberto	NULL	Acevedo	Humberto@contoso.com	Sales Region Manager	Humberto Acevedo Humberto@contoso.com Sales Region Manager
6	Yoichiro	NULL	Okada	thierry0@contoso.com	Sales State Manager	Yoichiro Okada thierry0@contoso.com Sales State Manager
7	Pilar	NULL	Ackerman	PilarAckerman@contoso.com	Sales State Manager	Pilar Ackerman PilarAckerman@contoso.com Sales State Manager
8	Aaron	M.	Painter	AaronPainter@contoso.com	Sales State Manager	Aaron M. Painter AaronPainter@contoso.com Sales State Manager
9	Terry	NULL	Adams	jolynn0@contoso.com	Sales State Manager	Terry Adams jolynn0@contoso.com Sales State Manager
10	David	NULL	Probst	ruth0@contoso.com	Sales State Manager	David Probst ruth0@contoso.com Sales State Manager

Ilustración 9 Resultado del query mostrando la nueva columna computada SearchEmployee. Los valores de las columnas de interés aparecen concatenados y separados por un espacio en blanco. Fuente: Elaboración propia

Como se observa, la nueva columna computada muestra la información del empleado que es relevante para su búsqueda en un solo campo de dato. Esto es un truco sencillo que permite buscar si una cadena está contenida en un registro. En este caso todos los campos son de texto y no se requiere código adicional. En caso contrario se debe hacer un CAST() de la columna que no lo sea para convertirla a texto. Es algo simple:

```
CAST(columnanotexto AS varchar(7))
```

Solamente hay que tener cuidado de especificar una longitud de varchar adecuada, en este caso se usa el 7.

Una posible desventaja es que, al ser una columna Persistida, se requiere espacio de almacenamiento adicional. De lo contrario cada vez que se realice una

búsqueda el motor de base de datos tendría que concatenar todos los campos. El almacenamiento es barato comparado con el uso intensivo del CPU, es un pequeño precio que pagar para permitir las búsquedas LIKE.

Ejemplo de búsqueda tipo LIKE en la columna computada, buscaremos “mik”

```
SELECT
       e.FirstName, e.MiddleName, e.LastName
       , e.EmailAddress, e.Title, e.[SerchEmployee]
FROM DimEmployee e
where e.[SerchEmployee] like '%mik%'
```

FirstName	MiddleName	LastName	EmailAddress	Title
Mike	NULL	Nash	ashvini0@contoso.com	Sales
Mike	NULL	Ray	britta0@contoso.com	Sales
Erik	NULL	Rucker	mikael0@contoso.com	Sales
Mike	NULL	Danseglio	belinda0@contoso.com	Sales
Jeff	NULL	Stammler	mike0@contoso.com	Sales
Mike	NULL	Gahrns	michael8@contoso.com	Sales

Ilustración 10 resultado de búsqueda LIKE del texto “mik”, se aprecian resultados “Mike” y “Mikael”. Fuente: Elaboración propia

Como se observa, al buscar “mik” sobre la columna computada nos da resultados rápidos y exactos. De todas formas, para implementar los algoritmos de búsqueda aproximada es necesario identificar los campos de interés y con esto ya estamos adelantando trabajo.

4.3. Implementando la Búsqueda Aproximada

Una vez teniendo identificados los campos de datos de interés para las búsquedas. En este ejemplo usaremos FirstName, LastName, Title y EmailAddress. Además, se necesita definir el porcentaje mínimo de similitud para considerar si una palabra es lo suficientemente parecida. Usaremos 60%, escrito como 0.6. Con esto procedemos a programar el stored procedure:

```sql
-- Autor: Genaro Arcos, 2023
-- Stored Procedure de Ejemplo para Búsqueda por Empleado
CREATE OR ALTER PROCEDURE BusquedaEmpleado
	-- Add the parameters for the stored procedure here
	@keyword nvarchar(MAX)
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

	DECLARE @keywordLike nvarchar(max)
	SET @keywordLike = CONCAT('%', @keyword, '%')
	PRINT @keywordLike

    -- Insert statements for procedure here
	-- Almacena los resultados de la busqueda a mostrar
	DECLARE @result TABLE
	(
		EmployeeKey		INT NOT NULL,
		FirstName		NVARCHAR(50) NOT NULL,
		LastName		NVARCHAR(50) NOT NULL,
		MiddleName		NVARCHAR(50) NULL,
		Title			NVARCHAR(50) NOT NULL,
		EmailAddress	NVARCHAR(50) NOT NULL,
		Similitud		NVARCHAR(50) NULL
	);

	-- busqueda LIKE
	INSERT INTO @result (EmployeeKey, FirstName, LastName, MiddleName, Title,
EmailAddress, Similitud)
		SELECT  e.EmployeeKey, e.FirstName, e.LastName, e.MiddleName,
e.EmailAddress, e.Title, 'Operador LIKE'
		FROM DimEmployee e
		WHERE e.SerchEmployee like @keywordLike;

	-- definimos porcentaje de similitud minimo para regresar resultado
	-- usaremos un flexible 60%
	DECLARE @limiteSimilitud FLOAT
	SET @limiteSimilitud = 0.6

	-- por cada campo de interes ejecutamos el algoritmo e insertamos los
resultados en la variable @results

	-- FirstName
	INSERT INTO @result (EmployeeKey, FirstName, LastName, MiddleName, Title,
EmailAddress, Similitud)
		SELECT  e.EmployeeKey, e.FirstName, e.LastName, e.MiddleName,
e.Title, e.EmailAddress, CONCAT('FirstName: ', dbo.PercentualTextDiff(e.FirstName,
@keyword))
		FROM DimEmployee e
		WHERE dbo.PercentualTextDiff(e.FirstName, @keyword) >=
@limiteSimilitud

	-- LastName
```

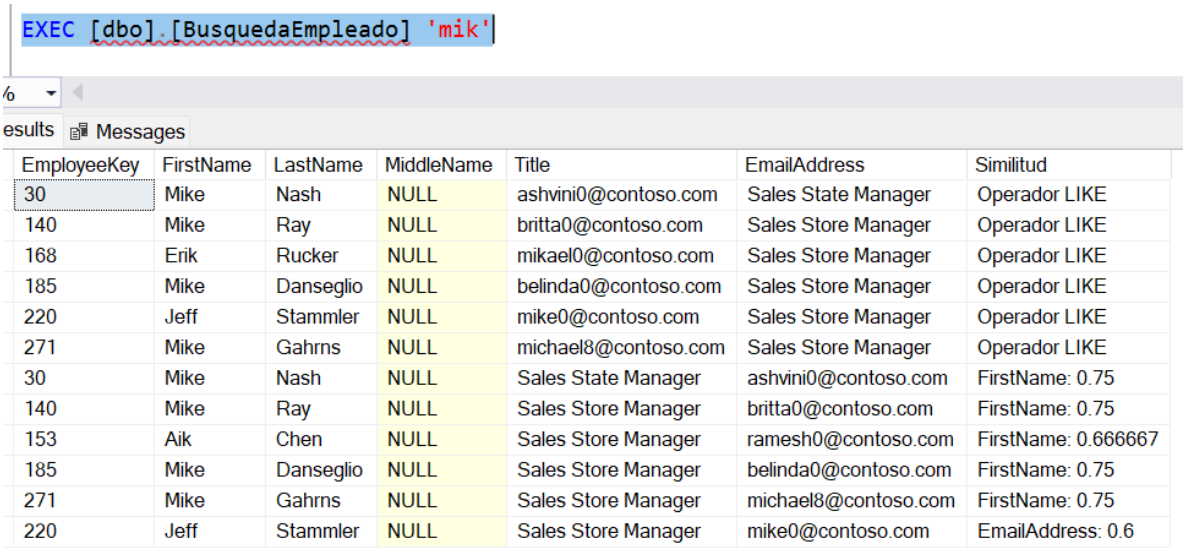
```
	INSERT INTO @result (EmployeeKey, FirstName, LastName, MiddleName, Title,
EmailAddress, Similitud)
		SELECT  e.EmployeeKey, e.FirstName, e.LastName, e.MiddleName,
e.Title, e.EmailAddress, CONCAT('LastName: ', dbo.PercentualTextDiff(e.LastName,
@keyword))
		FROM DimEmployee e
		WHERE dbo.PercentualTextDiff(e.LastName, @keyword) >=
@limiteSimilitud

	-- Title
	INSERT INTO @result (EmployeeKey, FirstName, LastName, MiddleName, Title,
EmailAddress, Similitud)
		SELECT  e.EmployeeKey, e.FirstName, e.LastName, e.MiddleName,
e.Title, e.EmailAddress, CONCAT('Title: ', dbo.PercentualTextDiff(e.Title,
@keyword))
		FROM DimEmployee e
		WHERE dbo.PercentualTextDiff(e.Title, @keyword) >= @limiteSimilitud

	-- EmailAddress
	INSERT INTO @result (EmployeeKey, FirstName, LastName, MiddleName, Title,
EmailAddress, Similitud)
		SELECT  e.EmployeeKey, e.FirstName, e.LastName, e.MiddleName,
e.Title, e.EmailAddress, CONCAT('EmailAddress: ',
dbo.PercentualTextDiff(LEFT(e.EmailAddress, CHARINDEX('@', e.EmailAddress)-1),
@keyword))
		FROM DimEmployee e
		WHERE dbo.PercentualTextDiff(LEFT(e.EmailAddress, CHARINDEX('@',
e.EmailAddress)-1), @keyword) >= @limiteSimilitud

	-- regresamos todos los resultados
	SELECT * FROM @result;
END
GO
```

Se define una variable tipo table donde se almacenarán temporalmente los resultados. La primera búsqueda es con el operador LIKE. Posteriormente y por cada campo de interés se realiza una búsqueda utilizando la función auxiliar PercentualTextDiff, que es la que realiza la búsqueda aproximada, abstrayendo el algoritmo especifico.

En este caso el campo EmailAddress necesita que se le remueva el @domain.com del mismo, para eso se utiliza la función LEFT. Es probable que otros campos necesiten un procesamiento especial durante la programación de la búsqueda.

4.4. Probando la Búsqueda Aproximada

```
EXEC [dbo].[BusquedaEmpleado] 'mik'
```



EXEC [dbo].[BusquedaEmpleado] 'mik'

esults Messages

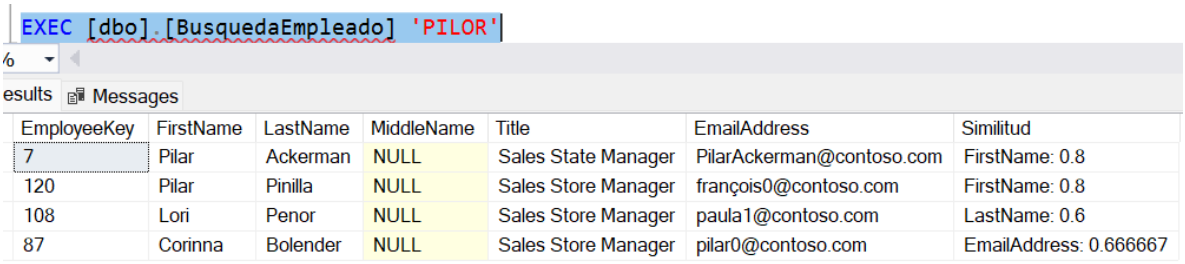
EmployeeKey	FirstName	LastName	MiddleName	Title	EmailAddress	Similitud
30	Mike	Nash	NULL	ashvini0@contoso.com	Sales State Manager	Operador LIKE
140	Mike	Ray	NULL	britta0@contoso.com	Sales Store Manager	Operador LIKE
168	Erik	Rucker	NULL	mikael0@contoso.com	Sales Store Manager	Operador LIKE
185	Mike	Danseglio	NULL	belinda0@contoso.com	Sales Store Manager	Operador LIKE
220	Jeff	Stammler	NULL	mike0@contoso.com	Sales Store Manager	Operador LIKE
271	Mike	Gahrns	NULL	michael8@contoso.com	Sales Store Manager	Operador LIKE
30	Mike	Nash	NULL	Sales State Manager	ashvini0@contoso.com	FirstName: 0.75
140	Mike	Ray	NULL	Sales Store Manager	britta0@contoso.com	FirstName: 0.75
153	Aik	Chen	NULL	Sales Store Manager	ramesh0@contoso.com	FirstName: 0.666667
185	Mike	Danseglio	NULL	Sales Store Manager	belinda0@contoso.com	FirstName: 0.75
271	Mike	Gahrns	NULL	Sales Store Manager	michael8@contoso.com	FirstName: 0.75
220	Jeff	Stammler	NULL	Sales Store Manager	mike0@contoso.com	EmailAddress: 0.6

Ilustración 11 resultados de la búsqueda utilizando el parámetro 'mik'. Primero aparecen los resultados de la búsqueda por el operador LIKE y posteriormente los resultados de la búsqueda aproximada. Fuente: Elaboración propia

Los resultados del operador LIKE se muestran primero, ya que si existe la cadena “mik” en los campos de interés. Posterior vienen los resultados de búsqueda aproximada, existen coincidencias del 66% y 75% del FirstName y del 60% en EmailAddress.

Hagamos otro ejemplo, quería buscar PILAR y por error escribí PILOR:

```
EXEC [dbo].[BusquedaEmpleado] 'PILOR'
```



EXEC [dbo].[BusquedaEmpleado] 'PILOR'

esults Messages

EmployeeKey	FirstName	LastName	MiddleName	Title	EmailAddress	Similitud
7	Pilar	Ackerman	NULL	Sales State Manager	PilarAckerman@contoso.com	FirstName: 0.8
120	Pilar	Pinilla	NULL	Sales Store Manager	françois0@contoso.com	FirstName: 0.8
108	Lori	Penor	NULL	Sales Store Manager	paula1@contoso.com	LastName: 0.6
87	Corinna	Bolender	NULL	Sales Store Manager	pilar0@contoso.com	EmailAddress: 0.666667

Ilustración 12 resultados de la búsqueda del parámetro ‘PILOR’. Todos son resultados de búsqueda aproximada. Fuente: Elaboración propia

Al no existir PILOR, el operador LIKE no regresa resultados. Y es aquí donde se muestra la utilidad de usar los algoritmos de búsqueda aproximada.

Este último ejemplo demuestra que el Stored Procedure de BusquedaEmpleado funciona correctamente. Hemos implementado una búsqueda aproximada que utiliza 1 solo parámetro.

5. Comparativa

Para ejecutar las pruebas se generó un stored procedure que ejecuta efectivamente 20 veces el stored procedure de BusquedaEmpleado. Antes de ejecutar el SP de prueba habilitamos las estadísticas de tiempo en SQL Server con el siguiente comando:

```
SET STATISTICS TIME ON
EXEC [dbo].SearchBenchmark 'PILOR'
```

```
-- Procedure para prueba benchmar de la busqueda
CREATE OR ALTER PROCEDURE SearchBenchmark
	@keyword nvarchar(MAX)
AS
BEGIN
	SET NOCOUNT ON;
	exec [dbo].[BusquedaEmpleado] @keyword; -- se repite 20 veces
		…
	exec [dbo].[BusquedaEmpleado] @keyword;
END
GO
```

Tiempo de ejecución para Distancia de Levenshtein: Consistentemente Levenshtein tarda 14.9 segundos

```
SQL Server Execution Times:
  CPU time = 11750 ms,  elapsed time = 14939 ms.
```

Ilustración 13 Tiempo de ejecución para distancia de Levenshtein, resultados obtenidos de Microsoft SQL Server Management Studio. Fuente: Elaboración propia

Tiempo de ejecución para Distancia de Hamming: En contraste, Hamming es veloz con 1.25 segundos.

```
SQL Server Execution Times:
  CPU time = 1047 ms,  elapsed time = 1257 ms.
```

Ilustración 14 Tiempo de ejecución para la distancia de Hamming. Fuente: Elaboración propia

Complejidad de implementación de la distancia de Levenshtein: El código implementado por (Gama & Oppolzer, 2010) cuenta con aproximadamente 80 líneas, contando su función auxiliar MIN3, lo que lo hace enorme comparado con Hamming. Contiene 2 ciclos while anidados, lo que hace que tenga un orden O(n2). Requiere de funciones auxiliares. Definitivamente su complejidad es ALTA.

Complejidad de implementación de la distancia de Hamming: La implementación de (Balazs, 2012) consiste solamente en 24 lídeas de código. Esto lo hace pequeño y simple. Utiliza solamente 1 ciclo while. Su complejidad es BAJA.

En esta comparativa no existió el caso donde la complejidad fuese MEDIA.

Tabla de resultados:

Algoritmo	Ejecución	Complejidad
Levenshtein	ALTA	ALTA
Hamming	BAJA	BAJA

Tabla 1 Fuente: Elaboración propia

Consideraciones adicionales

Los siguientes puntos sirven para ampliar las capacidades o dar mayor valor a la funcionalidad presentada en el ejemplo. Su implementación dependerá de las circunstancias particulares de quien utilice este manual. Mayormente provienen de

mi experiencia personal tanto desarrollando sistemas de negocio con este tipo de búsqueda y sistemas de software en general.

Búsqueda de múltiples palabras: En el ejemplo de la búsqueda de empleados se utilizó un solo parámetro. Los usuarios van a buscar múltiples palabras y es necesario procesarlo adecuadamente. Se recomienda el siguiente proceso:

1. Dividir el texto por espacios en blanco.
2. Por cada palabra:
    a. Ejecutar una búsqueda de la palabra en alguna función auxiliar que recorra todos los campos de interés.
    b. El resultado de cada palabra almacenarlo en una variable global de resultados.
3. Mostrar el resultset global con los resultados de todas las palabras.

Ejemplo “juan perez” -> buscar “juan” y guardar resultados en variable global, buscar “perez” y guardar resultados en variable global, mostrar contenido de variable global.

Columnas con tipos de datos diferentes a texto: Asegurarse de convertir a texto cualquier campo a buscar. Los algoritmos solo funcionan sobre cadenas de texto. También considerar la longitud del texto apropiada que puede representar el dato sin ser truncada.

Ordenar los resultados: En el ejemplo utilizamos una variable tipo tabla para almacenar los resultados. Se puede almacenar los resultados de múltiples búsquedas en otra variable y ordenar los resultados en base al porcentaje de similitud y en orden descendente. Para los resultados del operador LIKE se asigna el porcentaje de similitud al 100%.

Búsqueda en columnas de Email: Como se vio en el ejemplo. Es necesario remover del email la parte que contiene “*@dominio.com*” del texto. Una optimización

podría ser crear una columna computada y persistida que realice la extracción. Ejemplo: “miusuario@dominio.com” se convierte a “miusuario”.

Búsqueda en columnas de Teléfonos: Es necesario exponer al algoritmo de búsqueda los números telefónicos como texto. Una mejor práctica de la industria es almacenarlos de dicha forma. Esto por los formatos locales para expresar números telefónicos, por ejemplo: +52-555-123-4567, (555) 123-4567, 55-51-23-45-67, 55-51-23-45-67 extensión 123, etc. Para la búsqueda es mejor procesar dicho dato extrayendo solamente los dígitos y almacenarlos, al igual que el campo de email, en una columna computada y persistida. Ejemplo: “525551234567”. Se realizar la búsqueda sobre dicha columna computada.

Búsqueda en campos de URLs: Similar al email, se debe extraer el dominio y las palabras posteriores al mismo, por ejemplo “www.google.com/maps” -> “google” + “maps”, y crear una columna computada persistida que los almacene.

6. Conclusiones

En la sección de Propuesta implementamos como ejemplo una búsqueda aproximada en una sola tabla de la base de datos de ejemplo Contoso (Microsoft Corporation, 2010). Este ejemplo trata de explicar lo mínimo indispensable para implementar la búsqueda aproximada y sus algoritmos de distancia de edición. Con dicha base se procedió a realizar ejercicios para la comparativa de los algoritmos de distancia de edición. Se espera que esta base sea suficiente para que el lector interesado pueda implementar una búsqueda aproximada de mayor alcance en sus sistemas informáticos de negocio. Las consideraciones adicionales sirven como guía para expandir la base de acuerdo con las necesidades del lector interesado.

La comparativa sirve para determinar cuál algoritmo de distancia de edición es la mejor opción para integrarlo como parte del algoritmo de búsqueda aproximada. Al principio personalmente creí que el algoritmo de Levenshtein era la

mejor opción debido a la forma de encontrar sustituciones de caracteres en la misma palabra que el algoritmo de Hamming no hace. Pero por su alto tiempo de ejecución, mayor a 10 veces, es difícil justificarlo. El tiempo de ejecución aumentará conforme el desarrollador del sistema involucre más tablas y columnas en su búsqueda. En este ejemplo solamente se utilizó una tabla. Un sistema de expedientes electrónicos típicamente llevaría decenas de tablas y cientos de campos. Debido esto **el mejor algoritmo de distancia de edición es la Distancia de Hamming**.

7. Bibliografía

Acevedo Villaroel, R., & Rioseco Reinoso, C. (2011). Una comparación de metodologías para el modelado de aplicaciones web. *Revista Cubana de Ciencias Informáticas, 5*(2), págs. 1-9.

Balazs, F. (2012, Diciembre 9). *Hamming distance function in T-SQL*. Retrieved from Bits of BI: http://bitsofbi.blogspot.com/2012/12/hamming-distance-function-in-t-sql.html

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 377. Retrieved Julio 02, 2023, from https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Gama, J., & Oppolzer, H. (7 de Septiembre de 2010). *Calculating Levenshtein Distance in TSQL*. Obtenido de Devioblog: https://devio.wordpress.com/2010/09/07/calculating-levenshtein-distance-in-tsql/

International Organization for Standardization. (1987). 9075:1987 Information processing systems — Database language — SQL. Recuperado el 02 de Julio de 2023, de https://www.iso.org/standard/16661.html

Melton, J. (Marzo de 1996). SQL Language Summary. *ACM Computing Surveys, 28*(1), pág. 141. Recuperado el 03 de Julio de 2023, de https://dl.acm.org/doi/pdf/10.1145/234313.234374

Microsoft Corporation. (29 de Enero de 2010). *Contoso BI Demo Dataset for Retail Industry*. Obtenido de https://www.microsoft.com/en-us/download/details.aspx?id=18279

Microsoft Corporation. (Octubre de 2023). *Bing*. Obtenido de https://www.bing.com

Microsoft Corporation. (2023, 02 28). *Full-Text Search*. Retrieved from Microsoft Learning: https://learn.microsoft.com/en-us/sql/relational-databases/search/full-text-search?view=sql-server-ver16

Navarro, G. (2001, Marzo). A Guided Tour to Approximate String Matching. *ACM Computing Surveys, 33*(1), 37-38. Retrieved Julio 3, 2023, from https://dl.acm.org/doi/10.1145/375360.375365

Real Academia Española. (2022). Diccionario de la lengua española, 23a ed. *Versión 23.6 en linea*, pág. parrafo 3. Recuperado el 02 de Julio de 2023, de https://dle.rae.es/expediente

Real Academia Española. (2023). Diccionario panhispánico del español jurídico (DPEJ) [en línea]. Recuperado el 02 de Julio de 2023, de https://dpej.rae.es/lema/expediente-electronico

Roch, S., Fowler, J., Smith, B., & Bourgeois, D. (2014). *Information Systems for Business and Beyond, first canadian edition.* Ontario, Canada: eCampus Ontario. Recuperado el Octubre de 2023, de https://ecampusontario.pressbooks.pub/informationsystemscdn/

Sánchez Torres, C., & Rincón Cárdenas, E. (2004). Municipio digital y gobierno electrónico. *Vniversitas*, 7. Recuperado el 02 de Julio de 2023, de https://www.redalyc.org/articulo.oa?id=82510721

Secretaria de Educacion Publica, Gobierno de Mexico. (Octubre de 2023). *Registro Nacional de Profesionistas*. Obtenido de Consulta de cédulas profesionales: https://www.cedulaprofesional.sep.gob.mx/cedula/presidencia/indexAvanzada.action

8. Anexo: Código fuente público

8.1. Distancia de Hamming

Código tomado de (Balazs, 2012)

```
CREATE FUNCTION HammingDist
   (@value1 CHAR (8000), @value2 CHAR (8000))
   RETURNS INT
   AS
   BEGIN
         DECLARE @distance AS INT;
         DECLARE @i AS INT;
         DECLARE @len AS INT;
         SELECT @distance = 0,
               @i = 1,
               @len = CASE
    WHEN len(@value1) > len(@value2) THEN len(@value1) ELSE len(@value2)
    END;
        IF (@value1 IS NULL)
           OR (@value2 IS NULL)
            RETURN NULL;
        WHILE (@i <= @len)
            SELECT @distance = @distance + CASE
    WHEN substring(@value1, @i, 1) != substring(@value2, @i, 1) THEN 1 ELSE 0
    END,
                  @i = @i + 1;
        RETURN @distance;
    END
```

8.2. Distancia de Levenshtein

Código tomado de (Gama & Oppolzer, 2010)

```
--- Función auxiliar MIN3 para Levenshtein
CREATE FUNCTION [dbo].[MIN3](@a int, @b int, @c int)
returns int as
begin
  declare @min int
  set @min = @a
  if @b < @min set @min = @b
  if @c < @min set @min = @c
  return @min
end
```

```
--- Implementacion de Levenshtein
CREATE FUNCTION [dbo].[LEVENSHTEIN]( @s NVARCHAR(MAX), @t NVARCHAR(MAX) )
/*
Levenshtein Distance Algorithm: TSQL Implementation
by Joseph Gama
```

```
http://www.merriampark.com/ldtsql.htm

Returns the Levenshtein Distance between strings s1 and s2.
Original developer: Michael Gilleland http://www.merriampark.com/ld.htm
Translated to TSQL by Joseph Gama

Fixed by Herbert Oppolzer / devio
as described in http://devio.wordpress.com/2010/09/07/calculating-levenshtein-
distance-in-tsql
*/
RETURNS INT AS
BEGIN
  DECLARE @d NVARCHAR(MAX), @LD INT, @m INT, @n INT, @i INT, @j INT,
    @s_i NCHAR(1), @t_j NCHAR(1),@cost INT

  -- Conversion de parametros a MAYUSCULAS para ignorar el casing.
  SET @s = UPPER(@s)
  SET @t = UPPER(@t)

  --Step 1
  SET @n = LEN(@s)
  SET @m = LEN(@t)
  SET @d = REPLICATE(NCHAR(0),(@n+1)*(@m+1))
  IF @n = 0
  BEGIN
    SET @LD = @m
   GOTO done
  END
  IF @m = 0
  BEGIN
    SET @LD = @n
    GOTO done
  END

  --Step 2
  SET @i = 0
  WHILE @i <= @n BEGIN
    SET @d = STUFF(@d,@i+1,1,NCHAR(@i))        --d(i, 0) = i
    SET @i = @i+1
  END

  SET @i = 0
  WHILE @i <= @m BEGIN
    SET @d = STUFF(@d,@i*(@n+1)+1,1,NCHAR(@i))    --d(0, j) = j
    SET @i = @i+1
  END

  --Step 3
  SET @i = 1
  WHILE @i <= @n BEGIN
    SET @s_i = SUBSTRING(@s,@i,1)

    --Step 4
    SET @j = 1
```

```
    WHILE @j <= @m BEGIN
      SET @t_j = SUBSTRING(@t,@j,1)
      --Step 5
      IF @s_i = @t_j
        SET @cost = 0
      ELSE
        SET @cost = 1
      --Step 6
      SET @d = STUFF(@d,@j*(@n+1)+@i+1,1,
        NCHAR(dbo.MIN3(
          UNICODE(SUBSTRING(@d,@j*(@n+1)+@i-1+1,1))+1,
          UNICODE(SUBSTRING(@d,(@j-1)*(@n+1)+@i+1,1))+1,
          UNICODE(SUBSTRING(@d,(@j-1)*(@n+1)+@i-1+1,1))+@cost)
        ))
      SET @j = @j+1
    END
    SET @i = @i+1
  END

  --Step 7
  SET @LD = UNICODE(SUBSTRING(@d,@n*(@m+1)+@m+1,1))

done:
  RETURN @LD
END
```

8.3. Función propia PercentualTextDiff

```
-- Función auxiliar para PercentualTextDiff
-- Autor: Genaro Arcos, 2012
create function [dbo].[MAXVAL](@a int, @b int)
returns int as
begin
  declare @max int
  set @max = @a
  if @b > @max set @max = @b
  return @max
end
```

```
/*
Función que regresa el porcentaje de similitud
entre 2 cadenas de texto utilizando la funcion de
distancia de Levenshtein
Autor: Genaro Arcos, 2012
*/
CREATE OR ALTER FUNCTION [dbo].[PercentualTextDiff]( @source NVARCHAR(MAX),
@target NVARCHAR(MAX))
RETURNS FLOAT AS
BEGIN
```

```
IF @source = NULL AND @target = NULL
BEGIN
	RETURN 1.0;
END

IF @source = NULL OR @target = NULL
BEGIN
	RETURN 0.0;
END

DECLARE @sourcelength INT = LEN(@source)
DECLARE @targetlength INT = LEN(@target)

IF @sourcelength = 0 AND @targetlength = 0
BEGIN
	RETURN 1.0;
END

DECLARE @ldf FLOAT = CONVERT(FLOAT, dbo.[LEVENSHTEIN](@source, @target))
--DECLARE @ldf FLOAT = CONVERT(FLOAT, dbo.HammingDist(@source, @target))
DECLARE @maxlenf FLOAT = CONVERT(FLOAT, dbo.[MAXVAL](@sourcelength,
@targetlength))
DECLARE @percentage FLOAT

SET @percentage = 1.0 - (@ldf / @maxlenf)

RETURN @percentage

END
```