[07 de junio de 2019]

Oscar Lenin Espinoza Alvarez

PROYECTO DE TITULACIÓN

DE LA CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

USO DE HERRAMIENTAS OPEN SOURCE PARA LA GESTIÓN DE RECURSOS INFORMÁTICOS

Instituto Nacional de Investigaciones Forestales, Agrícolas y Pecuarias: Laboratorio Nacional de Modelaje y Sensores Remotos.

Dr. Víctor Manuel Rodríguez Moreno

M.A.T.I. Jorge Norberto Mondragón Reyes

07 de Junio de 2019

Capítulo 1

2. Agradecimientos.

Quiero agradecer a cada una de las personas: amigos, familia y compañeros presentes en mi vida durante esta etapa tan importante para mí, por cada consejo, cada aliento, y por cada día que me hicieron sentir que no estaba solo.

Especialmente a mis padres, por haberme inculcado los valores por los cuales me rijo, por cada día o noche de trabajo para que yo pudiera estudiar y no me faltara lo necesario, este logro es también suyo, siéntanse orgullosos de quien soy, porque es gracias a ustedes.

Al amor de mi vida por acompañarme siempre en este viaje, por ser el motor de mi vida, y la luz para mí en todo momento.

A mí bisabuela por siempre haber creído en mis capacidades, siempre me motivaste a seguir, estés donde estés, tenías razón; todo se puede.

También quiero dar infinitas gracias al personal del LNMySR, el Doctor Víctor Manuel Rodríguez Moreno, los ingenieros Efrén Emmanuel Prado López, Edwin Celestino García Alcocer, y Jorge Ernesto Mauricio Ruvalcaba por el excelente trato que me brindaron, por el compañerismo y el apoyo hacia mí y mi proyecto. Gracias por esta oportunidad y por todo lo que me enseñaron.

3. Resumen.

Este reporte está conformado inicialmente por la investigación necesaria sobre cursos Open-Source enfocados a base de datos y al Big data, esta investigación sirvió para elegir un nuevo gestor de base de datos que solucionara todos los problemas relacionados con la gestión de la información en el servidor de base de datos del LNMySR el cual contiene la base de datos “estaciones” la cual almacena todos los datos de las estaciones pertenecientes a la Red Nacional de Estaciones Agrometeorológicas Automatizadas (RNEAA) de INIFAP, uno de los principales problemas son que el gestor actualmente utilizado es obsoleto y no se cuentan con los recursos necesarios para su actualización, se realizaron pruebas de comportamiento para justificar si la opción elegida es óptima, a la par de la investigación se desarrollaron programas para migrar y consultar los datos de ambos servidores para la comparativa de respuesta, eficiencia y eficacia.

4. Índice.

Contenido

Lista de Tablas

Lista de Figuras

Capítulo 2: Generalidades del proyecto

5.- Introducción

En la actualidad existe un sinfín de nuevas tecnologías para la optimización de recursos informáticos, tomando como recurso informático cualquier elemento tangible o intangible relacionado con la informática que sea utilizado para un fin, ya sea con o sin fines de lucro.

Las nuevas tecnologías emergentes crecen de una manera rápida, ya que se diseñan para cubrir las necesidades de la actualidad por lo tanto, se vuelven tendencia al innovar de manera que quien las use encuentra maneras más fáciles y/o optimas de utilizar sus recursos.

Este proyecto buscó ayudar al LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos) a encontrar en nuevas tecnologías que cubran sus necesidades a bajo costo y que permitan migrar la base de datos de sus estaciones climáticas y así sentar las bases para cumplir sus objetivos a largo plazo de incursionar en el mundo del Big data.

Como propuesta para los objetivos planteados, surgió MongoDB un motor de base de datos orientados a documentos, parte de los motores emergentes de lenguaje NoSQL los cuales han ganado terreno por su eficacia ante el alta demanda.

Este proyecto conllevo varias etapas las cuales fueron:

- Investigación de tecnologías Open Source con la capacidad de un Big data.
- Selección y evaluación de la mejor opción para migrar la Base de Datos.
- Propuesta de Migración en MongoDB.
- Pruebas de migración a MongoDB usando programación en java.
- Documentación técnica que permita la migración de la Base de datos completa.

6. Descripción de la empresa u organización y del puesto o área del trabajo del estudiante.

Ilustración 1 Logotipo Oficial INIFAP

NOMBRE O RAZÓN SOCIAL: INIFAP (Instituto Nacional de Investigación Forestales, Agrícolas y Pecuarias (CIR NORTE CENTRO)
RAMO: Investigación Científica y Tecnológica
DIRECCIÓN: Campo Experimental Pabellón. Km 32.5 Carretera Aguascalientes – Zacatecas, Pabellón de Arteaga, Ags.
TELÉFONO: 01 800 088 22 22 Extensión 82536
MISIÓN: Generar conocimientos científicos y tecnologías que contribuyan al desarrollo sustentable de los subsectores forestal, agrícola y pecuario del país.

VISIÓN: Institución líder en ciencia y tecnología, con capacidad de respuesta en la atención a las demandas y necesidades de los subsectores forestal, agrícola y pecuario, que privilegia el trabajo en equipo, la superación de su personal y la satisfacción de sus usuarios.

ÁREA DE TRABAJO: Laboratorio Nacional de Modelaje y Sensores Remotos (LNMySR).

Ilustración 2 Organigrama Centro de Investigaciones Norte Centro

7. Problemas a resolver.

El principal problema que se buscó mitigar con la migración de la base de datos, es el hecho de que las tecnologías implementadas actualmente son obsoletas, el motor de base de datos por el cual se pagó licencia de uso de software ha dejado de tener la eficiencia que tenía para soportar los proyectos generados en el LNMySR por lo que se buscaron opciones para migrar la base de datos.

Además de lo antes mencionado en el laboratorio existe una visión a futuro para utilización de las tecnologías Big Data, y el software implementado hasta el momento como gestor de base de datos no es lo suficientemente robusto para este objetivo.

Por ultimo como problema se debe mencionar el hecho de que el motor de base de datos usado desde hace algunos años ha sido el mismo ya que no se ha destinado recurso para la actualización del software por ello fue que se comenzó a pensar en la posibilidad de usar software OpenSource.

8. Objetivos (General y Específicos)

Objetivo general: Realizar la documentación necesaria para la migración de una base de datos de software de licencia a un software Open Source.

Objetivos Específicos:

- La documentación deberá ser clara y precisa, deberá permitir que el personal capacitado pueda migrar los datos, sin pérdida de información y conservando la integridad de los mismos.
- Elaboración de un programa en Java el cual permita obtener, analizar y migrar los datos de una base de datos a otra.
- Realizar pruebas de migración de los datos para el análisis de los mismos, conservando su integridad y certeza, además de ayudar al análisis de la veracidad de los datos históricos almacenados.
- Realizar pruebas de tiempos de consulta, disponibilidad de los datos y tiempo de migración para justificar que la elección de software fue la correcta.

9. Justificación

Para resolver todos los problemas antes mencionados se realizó una investigación sobre software Open Source que permitirán tener la eficiencia que se requiere, buscando el cubrir las necesidades del Laboratorio en cuanto a gestión y uso de base de datos se refiere. Al migrar la base de datos a un sistema de licencia libre, se ahorran los costos que produce el pago de una licencia para otro motor de base de datos y soporte técnico, además de esto existen software libres que tienen el enfoque a Big data por lo que tener una investigación previa en este ambiente permite sentar las bases de un proyecto más grande y ambicioso.

Además el uso de este software permitirá el crecimiento exponencial de sus recursos sin afectar a los recursos ya existentes, y así con el paso del tiempo agregar más recursos e información a la base de datos sin afectar la disponibilidad de los que ya existentes.

CAPÍTULO 3: MARCO TEÓRICO

10. Marco Teórico (fundamentos teóricos).

Base de datos

Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro. (Valdés, 2007)

Ilustración 3 Ilustración de base de datos

Base de datos NoSQL

Aquella que no requiere de estructuras de datos fijas como tablas; no garantizan completamente las características ACID y escalan muy bien horizontalmente. Se utilizan en entornos distribuidos que han de estar siempre disponibles y operativos y que gestionan un importante volumen de datos.

Agregar el año en que los autores de los fundamentos teóricos hicieron la aportación o publicación.

Podríamos resumir las principales características de una base de datos NoSQL como sigue:

- El lenguaje estándar no tiene porqué ser SQL.
- El esquema de datos es flexible o no tiene un esquema predefinido, lo que permite el tratamiento de datos heterogéneos
- Las propiedades ACID no siempre están garantizadas
- Mayor coherencia entre los datos de programas y bases de datos
- Diseñadas para ser escalables generalmente de forma horizontal
- Suelen ser distribuidas
- Frecuentemente son de código abierto con grandes comunidades de desarrollo detrás (Rochina, 2016)

Propiedades ACID

ACID, conformado por las siglas provenientes de Atomicity, Consistency, Isolation y Durability. En español, Atomicidad, Consistencia, Aislamiento y Durabilidad, son un conjunto de propiedades necesarias para que un conjunto de instrucciones, sean consideradas como una transacción en un sistema de gestión de bases de datos. (Brito, 2012)

Transacción

Es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, es decir, en forma indivisible o atómica. Un ejemplo de una transacción compleja es la transferencia de fondos de una cuenta a otra, la cual implica múltiples operaciones individuales.

Si un sistema supera la prueba ACID, significa que es fiable. (Brito, 2012)

- **Atomicidad:** Significa que el sistema permite operaciones atómicas. Una operación atómica es aquella que si está formada por operaciones más pequeñas, se consideran como un paquete indivisible. Deben ejecutarse todas correctamente, o en el caso de que alguna de ellas no pueda hacerlo, el efecto de las que ya se han ejecutado no debe hacerse notar, debe deshacerse, como si el conjunto de las operaciones no se hubieran realizado. La atomicidad está íntimamente ligada al concepto de transacción de los sistemas gestores de bases de datos. En un SGBD, cuando se indica que un conjunto de operaciones forman una transacción, o se ejecutan todas correctamente, o el SGBD deshará los cambios, como si la transacción nunca se hubiera iniciado. No obstante, atomicidad y transacción no son sinónimos. Mientras atomicidad es una propiedad, la transacción es el mecanismo que utilizan los SGBD para lograr la atomicidad.

- **Consistencia:** Integridad. Esta propiedad asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. Sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.

- **Aislamiento:** Propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.

- **Durabilidad:** Propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema. (Brito, 2012)

Mongo DB

Es el sistema de base de datos desarrollada en 10gen por Geir Magnusson y Dwight Merriman. Es una base de datos orientada a documentos JSON, salvo que está diseñada para ser una verdadera base de datos de objetos, más que para un almacenamiento de clave/valor puro. Es una base de datos no relacional, es decir, no utiliza SQL. El nombre viene del término inglés “humongous” (colosal) y puede ser definida como una base de datos documental sin esquema, escalable y de alto rendimiento.

Forma en que opera: Para almacenar los documentos, utiliza una serialización binaria de JSON, llamada BSON, que es una lista ordenada de elementos simples. El núcleo de la base de datos es capaz de interpretar su contenido, de modo que lo que a simple vista parece un contenido binario, realmente es un documento que contiene varios elementos. Estos datos están limitados a un tamaño máximo de 4 MB; para tamaños superiores se requiere del uso de GridFS. (EcuRed, 2011)

MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Es algo bastante lógico, dado que los propios documentos se almacenan en BSON.

MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, podemos utilizar muchas de las funciones propias de JavaSciprt. En la consola también podemos definir variables, funciones o utilizar bucles. (Genbeta, 2014)

Ilustración 4 Logotipo MongoDB

Consultas (aggregate) Mongo DB

Tabla 1 Comando utilizados en consultas "aggregate" de MongoDB

MongoDB	Descripción	Ejemplo Consulta
Pipeline	Es el nombre utilizado para toda la instrucción incluida en las consultas tipo "aggregate"	
$project	Esta función proyecta los campos existentes del documento Mongo o los campos evaluados en el "pipeline" en que se usa. En esto, los desarrolladores usan "1" o "tru" si quieren incluir el Campo y "0" o "false" si quieren excluir un campo.	Db.estado1.aggregate([{"$project": { "_id" : 0, "Nombre_estado" : 1, "Nombre_municipio" : 1, "Nombre_estacion : 1, "_ID_estacion" : 1 }}])
$match	Esta función filtra los documentos de una colección que coinciden con los criterios especificados y pasan solo los documentos coincidentes a la siguiente instrucción del "Pipeline".	db.estado1.aggregate([{"$match": { "Nombre_municipio": "Aguascalientes" }}])
$limit	Delimita la consulta a n número de documentos posibles a proyectar en el "pipeline"	db.estado1.aggregate([{"$match": { ID_estacion":20671}}, {"$limit": 1}])
$group	Esta función agrupa los documentos de una colección de Mongo por criterios específicos y los pasa a la siguiente opción del "pipeline".	db.estado1.aggregate([{"$group": {"_id": {"Nombre de la estacion" : "$Nombre_estacion"}, "Velv": {"$sum": 1}}}])
$sort	Esta función reordena los documentos de una colección o consulta de Mongo en orden ascendente o descendente.	db.estado1.aggregate([{"$match": { "Nombre_municipio": 1 }}, {"$sort" : {"Nombre_municipio": 1}}])
$lookup	Esta función realiza una unión externa izquierda con otra colección en la misma base de datos de Mongo.	db.estado1.aggregate([{$lookup: {from: "estado2", localField: "Nombre_estado", foreignField: "_id", as: "ID_estado" }}])
$out	Esta función escribe el resultado calculado en una colección específica en Mongo. Este operador debe ser la última opción en el "pipeline".	db.estado1.aggregate([{"$match": { "Nombre_municipio": "Aguascalientes" }}, {"$sort" : {"Nombre_estacion": 1}}, {"$out" : "Estaciones_ags"}])
$unwind	Esta función deconstruye un campo de matriz del documento ingresado para generar un documento para cada elemento.	db.estado1.aggregate([{"$match": { "nombre_estado": "Aguascalientes" }}, {"$unwind": "$Nombre_municipio"}])

(UCM, 2017)

Studio 3T

Studio 3T es una GUI e IDE para desarrolladores e ingenieros de datos que trabajan con MongoDB. Las funciones de administración de datos, como la edición in situ y las conexiones de base de datos sencillas, se combinan con la generación de código de consulta políglota, shell avanzado con autocompletado, importación / exportación de SQL y autenticación a nivel empresarial con LDAP y Kerberos.

Studio 3T está diseñado para equipos profesionales en crecimiento. Permite el desarrollo rápido del equipo al ofrecer vistas de árbol de la tabla y JSON de sus datos, y una variedad de formas de interrogarlo, incluyendo agregaciones sofisticadas, extensiones nativas de Mongo JSON, consultas SQL tradicionales y un generador de consultas de arrastrar y soltar. (Mongodb.com, s.f.)

Ilustración 5 Logotipo Studio 3T

JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo. Una lista ordenada de valores.
- En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias. (Json.org, 2016)

```
[
      {
        "description": "quarter",
        "mode": "REQUIRED",
        "name": "qtr",
        "type": "STRING"
      },
      {
        "description": "sales representative",
        "mode": "NULLABLE",
        "name": "rep",
        "type": "STRING"
      },
      {
        "description": "total sales",
        "mode": "NULLABLE",
        "name": "sales",
        "type": "INTEGER"
      }
    ]
```

Ilustración 6 Ejemplo de documento Json

Java

Sun Microsystems desarrolló, en 1991, el lenguaje de programación orientado a objetos que se conoce como Java. El objetivo era utilizarlo en un set-top box, un tipo de dispositivo que se encarga de la recepción y la descodificación de la señal televisiva. El primer nombre del lenguaje fue Oak, luego se conoció como Green y finamente adoptó la denominación de Java. (Gardey, 2013)

La aplicación de Java es muy amplia. El lenguaje se utiliza en una gran variedad de dispositivos móviles, como teléfonos y pequeños electrodomésticos. Dentro del ámbito de Internet, Java permite desarrollar pequeñas aplicaciones (conocidas con el nombre de applets) que se incrustan en el código HTML de una página, para su directa ejecución desde un navegador; cabe mencionar que es necesario contar con el plug-in adecuado para su funcionamiento, pero la instalación es liviana y sencilla. (Gardey, 2013)

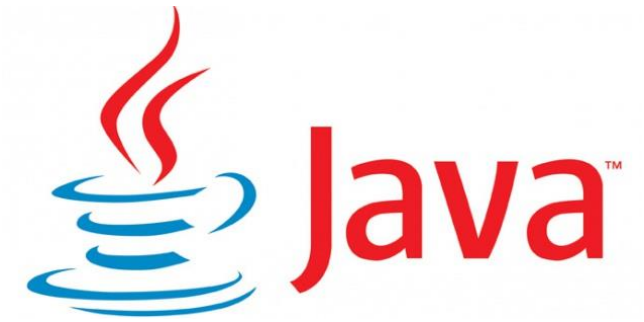

Ilustración 7 Logotipo de java

NetBeans

NetBeans IDE es un entorno de desarrollo integrado (IDE), modular, de base estandar (normalizado), escrito en el lenguaje de programación Java. El proyecto NetBeans consiste en un IDE de código abierto y una plataforma de aplicación, las cuales pueden ser usadas como una estructura de soporte general (framework) para compilar cualquier tipo de aplicación. (Oracle_Corporation, s.f.)

Ilustración 8 Logotipo NetBeans

Base de datos "Estaciones".

La base de datos "red_clima" contiene la información generada por las estaciones meteorológicas pertenecientes a la red de estaciones de INIFAP a nivel nacional, consta de 99 tablas, usadas para la ubicación de las estaciones, el registro de las misma y para el tratamiento de datos para usarse en diferentes aplicaciones y/o programas, existen 67 que podrían considerarse como las principales, existen dos por cada estado, en una se registran los valores de cada estación cada 15 minutos y en la segunda se registran los acumulados de cada estación por cada día, además de las tablas de estaciones, estados y municipios, las cuales guardan información más específica de cada estación, estado y municipio esto por los reglas de normalización. Para las 64 tablas de registros de los estados existe aproximadamente 315,766,198 de registros, en total la base de datos

pesa aproximadamente 30.01 GB almacenando datos históricos de hasta 10 años de las estaciones climatológicas. (INIFAP, 2018)

Las estructuras presentadas para cada tabla marcada como importante son las siguientes:

Registro cada 15 min.

Nombre tabla: Estado + numeró de estado en la tabla estados.

Ejemplo: “Estado1”

Tabla 2 Estructura de las tablas registros cada 15 min.

Campos	Tipo de dato	Definición del contenido del campo
numero	Int	Clave de la estación en la tabla estaciones
fecha	smalldatetime	Fecha y hora de en qué se registró la información
prec	Decimal(8,2)	Precipitación pluvial
temt	Decimal(8,2)	Temperatura ambiental
dirv	Decimal(8,2)	Dirección del viento
velv	Decimal(8,2)	Velocidad del viento
radg	Decimal(8,2)	Radiación solar
humr	Decimal(8,2)	Humedad relativa
humh	Decimal(8,2)	Humedad horaria
eto	Decimal(8,2)	Evapotranspiración

(INIFAP, 2018)

Registros diarios

Nombre tabla: “Estado” + numeró de estado en la tabla estados +”diarios”

Ejemplo: “Estado1diarios”

Tabla 3 Estructura de las tablas registros diarios.

Campos	Tipo de dato	Definición del contenido del campo.
numero(PK)	Int	Clave de la estación en la tabla estaciones.
fecha(PK)	smalldatetime	Fecha y hora de en qué se registró la información.
prec	Decimal(8,2)	Sumatoria de precipitación diaria.
tmin	Decimal(8,2)	Temperatura mínima en el día.
tmax	Decimal(8,2)	Temperatura máxima en el día.
tmed	Decimal(8,2)	Temperatura media del día.
velvmax	Decimal(8,2)	Velocidad máxima del día.
velv	Decimal(8,2)	Promedio de la velocidad del viento en el día.
dirvmax	Decimal(8,2)	Dirección del viento máxima del día.
dirv	Decimal(8,2)	Promedio de la dirección del viento del día
radg	Decimal(8,2)	Promedio de radiación solar del día.
humr	Decimal(8,2)	Promedio de la humedad relativa del día.
eto	Decimal(8,2)	Promedio de

		evapotranspiración del día.

(INIFAP, 2018)

Estaciones

Tabla 4 Estructura de la tabla estaciones.

Campos	Tipo de dato	Definición del contenido del campo.
numero(PK)	Int	Clave numérica única de identificación de la estación
idred	Int	Desconocido, no es utilizado.
nombre	Varchar(255)	Nombre de la estación
estadoid	Int	Clave del estado donde se encuentra la estación.
municipioid	Int	Clave del municipio donde se encuentra la estación
productor	Varchar(255)	Productor encargado de la estación
latitud	Float	Latitud de la ubicación exacta de la estación.
longitud	Float	Longitud de la ubicación exacta de la estación.
altitud	Float	Altitud de la ubicación exacta de la estación.
alturasensor	Smallmoney	Desconocido, no es utilizado.
inicio	smalldatetime	Fecha de inicio de funcionamiento de la

		estación.
activa	Bit	Estado de actividad de la estación.
mes1	Float	Desconocido, no es utilizado.
mes2	Float	Desconocido, no es utilizado.
mes3	Float	Desconocido, no es utilizado.
mes4	Float	Desconocido, no es utilizado.
mes5	Float	Desconocido, no es utilizado.
mes6	Float	Desconocido, no es utilizado.
mes7	Float	Desconocido, no es utilizado.
mes8	Float	Desconocido, no es utilizado.
mes9	Float	Desconocido, no es utilizado.
mes10	Float	Desconocido, no es utilizado.
mes11	Float	Desconocido, no es utilizado.
mes12	Float	Desconocido, no es utilizado.
baja	Bit	Estado de inactividad de la estación.
pertenencia	Varchar(50)	Persona u organización a quien pertenece la

		estación.
condición	Nvarchar(300)	Condición física en la que se encuentra la estación.
gratis	bit	Desconocido, no es utilizado.
xmlregen	Char(10)	Desconocido, no es utilizado.
ingenioId	Int	Desconocido, no es utilizado.
marquesina_foto	Varchar(100)	Desconocido, no es utilizado.
clasificaciónXRegion	int	Clave del identificador de la región donde se encuentra la estación
nodoid	int	Clave de identificación de la estación en su base de datos nativa.

(INIFAP, 2018)

Estados

Tabla 5 Estructura de la tabla estados.

Campos	**Tipo de dato**	**Definición del contenido del campo.**
indice(PK)	Int	Clave única de identificación de estados.
nombre	Varchar(30)	Nombre del estado
activo	Bit	Estado de actividad del estado
es_Canhero	Bit	Desconocido, no es

		utilizado.
inicio	Int	Año de inicio
funpro	Varchar(35)	Desconocido, no es utilizado.
marquesina	Bit	Desconocido, no es utilizado.
permisohist	Bit	Desconocido, no es utilizado.
prec2	Float	Desconocido, no es utilizado.
velv2	Float	Desconocido, no es utilizado.
temp1	Float	Desconocido, no es utilizado.
temp2	Float	Desconocido, no es utilizado.
tempvc1	Float	Desconocido, no es utilizado.
tempvc2	Float	Desconocido, no es utilizado.
superficieKm2	Decimal(18,2)	Registra el tamaño de la superficie territorial abarcada por el estado en km^2

(INIFAP, 2018)

Municipio

Tabla 6 Estructura de las tabla de municipios.

Campos	Tipo de dato	Definición del contenido del campo.
indice(PK)	Int	Clave única de identificación del municipio
idedo	Int	Clave única de identificación de estados.
nombre	Varchar(50)	Nombre del municipios
superficieKm2	Decimal(18,2)	Registra el tamaño de la superficie territorial abarcada por el estado en km^2

(INIFAP, 2018)

CAPÍTULO 4: DESARROLLO

Cronograma de actividades

Tabla 7 Cronograma de actividades

Actividades por Quincena	Ago -1a	Ago- 2a	Sept – 1a	Sept – 2a	Oct – 1a	Oct- 2a	Nov – 1a	Nov . – 2a	Dic- 1a
Investigar y analizar las tecnologías aplicadas actualmente y las posibilidades a aplicar									
Creación de copias de seguridad de algunas tablas de la base de datos original, para realizar pruebas de migración									
Propuesta de estructura para migración de la base de datos y desarrollo de programa en Java para migrar de SQL Server a MongoDB									
Pruebas de tiempos de migración de cada tabla y de toda la base de datos, tiempos de respuesta en consultas para comparación de ambas tecnologías.									
Realizar pruebas de consultas periódicas necesarias para el funcionamiento de los recursos del Laboratorio									

Redacción de informe final del proyecto, resultados y documentación necesaria para la migración.									

11. Procedimiento y descripción de las actividades realizadas.

Para lograr los objetivos principales del proyecto se comenzó por investigar las ventajas y desventajas de migrar la base de datos a una tecnología No SQL, de la cual surgió la idea MongoDB, tecnología que cumplió con los requerimientos: recursos Open Source, alta disponibilidad, enfoque big data y reducción de costos, planteados en el laboratorio para la elección de la nueva tecnología a ser empleada.

Después de decidir que sería MongoDB la tecnología a emplear, se comenzó la investigación de la instalación, uso de este manejador de base de datos, la forma de realizar consultas, la estructura de los datos, la sintaxis usada, los recursos de software y hardware necesarios para el correcto funcionamiento de MongoDB.

Al tener el conocimiento del funcionamiento del nuevo manejador, se realizaron pruebas que permitirán poner en práctica lo investigado y así obtener conocimiento propio del uso de esta tecnología, lo que después serviría para dar las bases necesarias para el programa de migración. Se realizaron pruebas de las consultas utilizadas en MongoDB así como la inserción, eliminación y actualización de los datos.

Teniendo el conocimiento propio del funcionamiento de MongoDB, se comenzó a desarrollar el código en java que permitiera la migración, para ello se estudió la estructura de la Base de Datos en SQL Server y se propuso una nueva estructura desnormalizada de la base de datos, la cual es la siguiente presentada en un documento tipo Json.

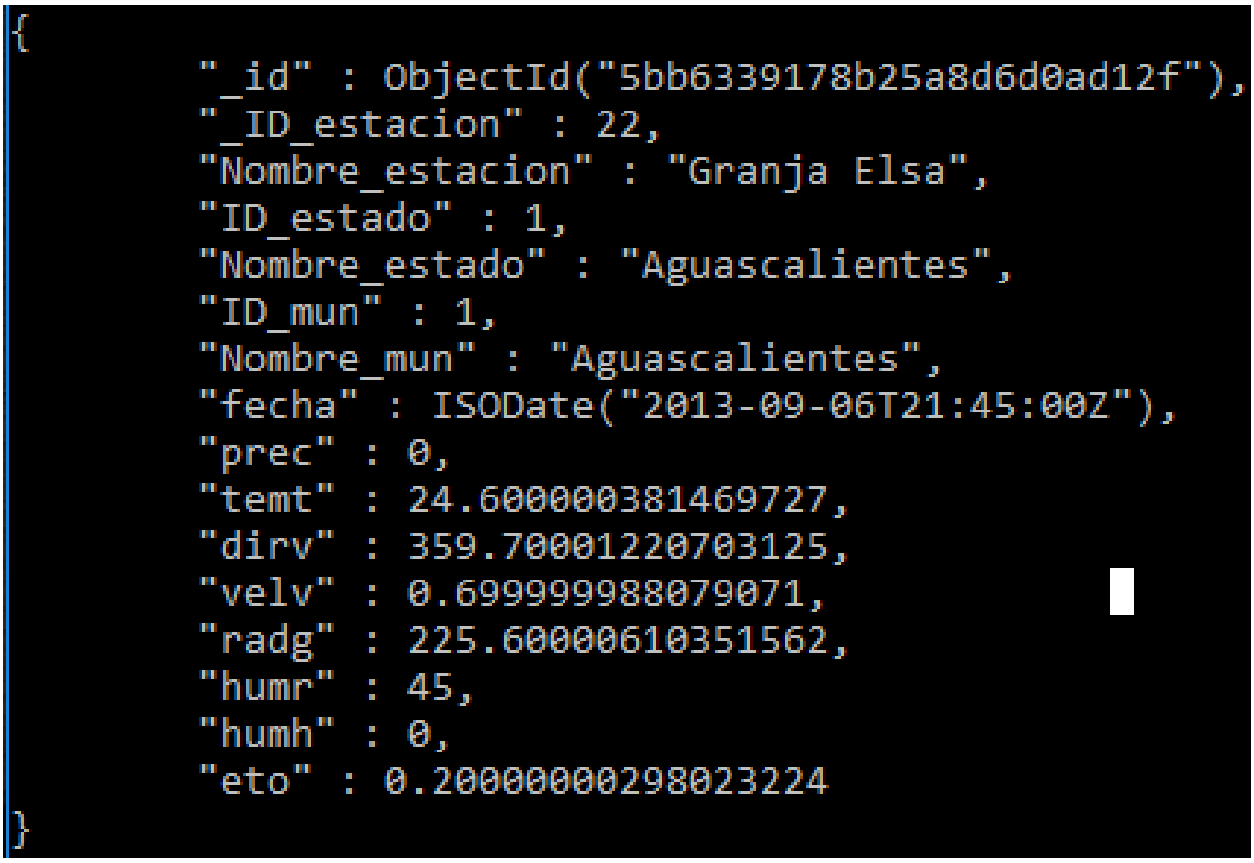

Ilustración 9 Propuesta de migración presentada en Mongo Shell

Para lograr esa estructura de los datos en SQL Server para después ser migrados a MongoDB, se creó una vista con las tablas de cada estado con sus registros de cada 15, las tablas de municipios, estados y estaciones, cada estado tiene su propia vista, todas con el mismo nombre, cambiando solo el número de estado, ejemplo:

Tabla 8 Ejemplo nombres de las vistas

Estado a migrar	Nombre de la vista
estado1	Vista_RC15E1
estado2	Vista_RC15E2
estado3	Vista_RC15E3
estado4	Vista_RC15E4
estado5	Vista_RC15E5
estado6	Vista_RC15E6

La estructura de las vistas es igual para cada estado y es la siguiente:

```
create view vista_RC15 as select Est.numero as ID_estacion,
Est.nombre as Nombre_estacion,
Std.indice as ID_estado,
Std.nombre as Nombre_estado,
Mun.indice as ID_mun,
Mun.nombre as Nombre_mun,
R_15.fecha,
R_15.prec,
R_15.temt,
R_15.dirv,
R_15.velv,
R_15.radg,
R_15.humr,
R_15.humh,
R_15.eto
from estaciones as Est, estados as Std, estado1 as R_15 ,municipios as Mun

where Est.municipioid=Mun.indice and
Est.estadoid=Std.indice and
Est.numero=R_15.numero
```

Ilustración 10 Vista de SQL

Donde cada tabla recibe un alias de la siguiente manera:

Tabla 9 Alias de las tablas en las vistas

Nombre de la tabla	Alias
Estaciones	Est
Estados	Std
Estado1	R_15
Municipios	Mun

Después de haber generado la vista con la estructura deseada en MongoDB, se generó un código en java utilizando el IDE Netbeans el cual funciona de la siguiente manera: Recorre una consulta, registro por registro, por lo que con un ciclo, cada iteración en los registros, realiza las mismas funciones, obtiene la información de ese registro, es parseada a los tipos de datos que se utilizaran en MongoDB, y después son insertados uno a uno en MongoDB, esto permite agregar cualquier tipo de validación de los datos antes de ser migrados, para eliminar o actualizar datos errores.

El diagrama de flujo que sigue el programa es el siguiente:

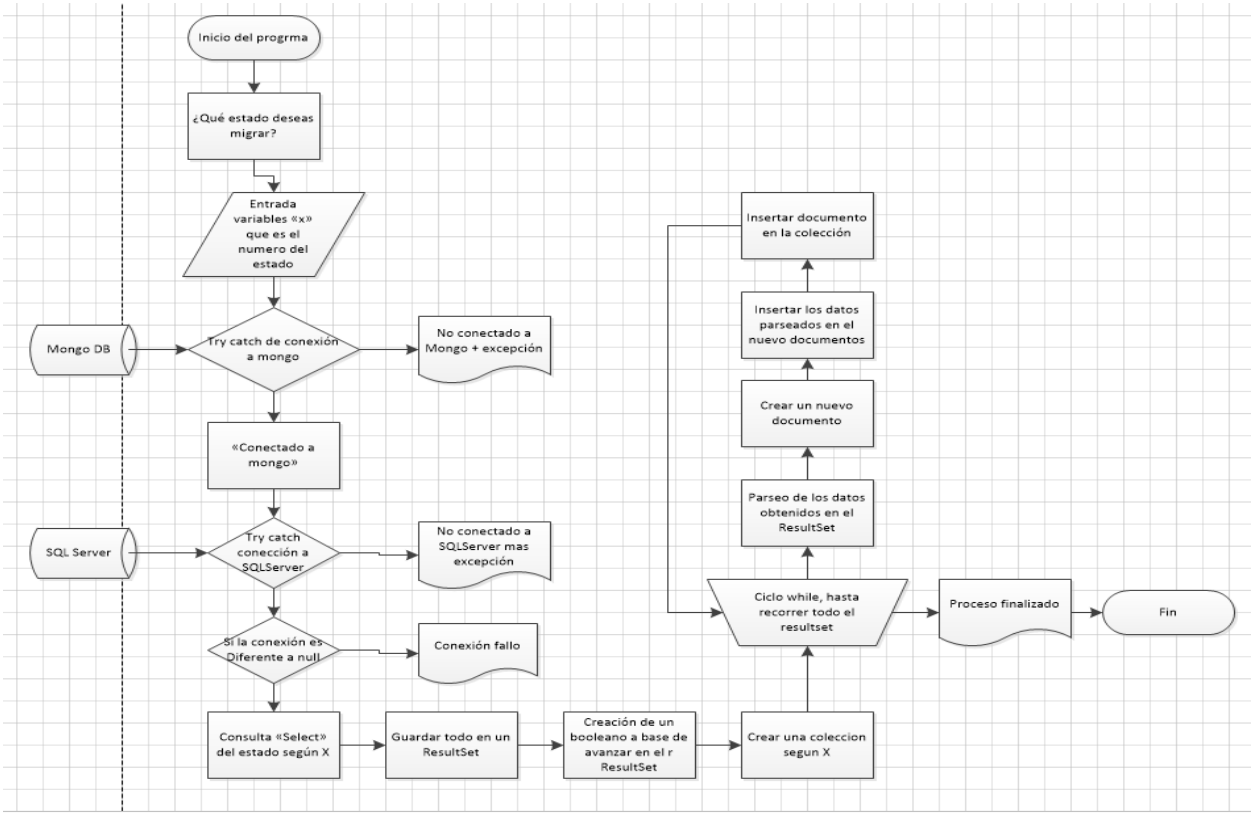

Ilustración 11 Diagrama de flujo programa migración

El programa utiliza dos clases para funcionar, el usuario ingresa el número de estado que desea migrar a MongoDB en la clase principal, con base al número de estado que se desea migrar dado por el usuario la clase publica “tablas” usa dos métodos el primero genera un String con la consulta a realizarse en SQL Server de la vista generada, como cada estado tiene su vista, este método devuelve el query de consulta a la vista basado solo en el número de estado a migrar, el segundo método genera el String del nombre de la colección en donde se insertara en Mongo DB.

Para que el programa funcione se necesitan dos drivers de conexión, el de SQL Server desde donde se obtendrán los datos y el de MongoDB a donde se va migrar.

El código de la clase “tablas” es el siguiente

```
public class Tablas {
    public static String tabla(int x){
        String numero=Integer.toString(x);
        String tablaSql="select * from vista_RC15E"+numero;
        return tablaSql;
    }
    public static String coleccion(int x){
        String numero=Integer.toString(x);
        String coleccion="estado"+numero;
        return coleccion;
    }
}
```

Ilustración 12 Código de la clase tablas

Donde el método “tabla” obtiene un número entero, este es parseado a String, para ser concatenado junto con el query de selección para obtener los datos de la vista, y regresa ese String
El método colección sigue un procedimiento similar, donde se obtiene un número entero que es parseado y concatenado en un String para nombrar la colección de MongoDB donde serán insertados los datos.

Código de la clase principal “RC15”

Las siguientes librerías son las utilizadas en el programa, para el control de excepciones, conexiones a las bases de datos y tratamiento de los datos.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import com.mongodb.DB;
import com.mongodb.BasicDBObject;
import com.mongodb.DBCollection;
import com.mongodb.Mongo;
import java.net.UnknownHostException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;/*importacion de librerias
```

Ilustración 13 Librerías utilizadas

El programa inicia con la impresión en consola de la lista de los estados con su número en la base de datos SQL Server, y pide ingresar al usuario el número de estado que desea migrar, este número es obtenido por un escáner y guardado en un variable entera llamada “x”

```
public class RC15 {
    public static void main(String[] args) throws ClassNotFoundException, SQLException, UnknownHostException,
            ParseException, InterruptedException {
        Scanner sc = new Scanner(System.in);
     int x;
      System.out.println("Elige un estado\n"
                + "1.-Aguascalientes\n"
                + "2.-Baja california\n"
                + "3.-Baja california sur\n"
                + "4.-Campeche\n"
                + "5.-Coahuila\n"
                + "6.-Colima\n"
                + "7.-Chiapas\n"
                + "8.-Chihuaha\n"
                + "9.-Durango\n"
                + "10.-Estado de México\n"
                + "11.-Guanajuato\n"
                + "12.-Guerrero\n"
                + "13.-Hidalgo\n"
                + "14.-jalsco\n"
                + "15.-Michoacan\n"
                + "16.-Morelos\n"
                + "17.-Nayarit\n"
                + "18.-Nuevo Leon\n"
                + "19.-Oaxaca\n"
                + "20.-Puebla\n"
                + "21.-Queretaro\n"
                + "22.-Quintana Roo\n"
                + "23.-San Luis Potosi\n"
                + "24.-Sinaloa\n"
                + "25.-Sonora\n"
                + "26.-Tabasco\n"
                + "27.-Tamaulipas\n"
                + "28.-Tlaxcala\n"
                + "29.-veracruz\n"
                + "30.-Yucatan\n"
                + "31.-Zacatecas\n"
                + "32.-CDMX"
      );

      x = sc.nextInt();/*seleccion del estado a migrar*/
```

Ilustración 14 Lista de estados a migrar

Al tener el número de estado a migrar se comienza por probar las conexiones a base de datos, inicia con MongoDB, inicializa una variable de tipo Mongo (variables de conexión) de nombre “mongo” donde se introduce la dirección IP del servidor mongo y el puerto por el cual desea entrar y salir la conexión, por medio de un Try-Catch se define si la conexión se ha realizado o no, si la conexión se ha completado con éxito, se imprime un mensaje en consola indicándolo, de lo contrario se imprime un mensaje de no conexión y el error por el cual no se ha realizado dicha conexión.

```
        x = sc.nextInt();/*seleccion del estado a migrar*/

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver"); /*declaracion del driver de conexion a sq
try{ /* cachamos errores en la conexion de mongo */
Mongo mongo = new Mongo("10.20.55.227",83);
System.out.println("Conectado a mongo");/*conexion a mongo*/
mongo.close();/*cerramos la comexion*/
}
catch (UnknownHostException ex){
        System.out.println("No conectado a mongo"+ex);
        }
```

Ilustración 15 Try-Catch conexión a MongoDB

Continúa y se realiza un proceso parecido para la conexión en SQL Server, donde en un Try-Catch contiene el proceso de conexión a la base de datos, por medio de un String de conexión donde se especifica la ruta del servidor así como las credenciales del usuario de SQL Server para conectarse al servidor, se declara una variable de tipo connection donde se especifica el String de conexión al driver de conexión, posteriormente con una condicional se dice que, si existe una conexión realice los siguientes procesos, declaración de la conexión, declaración del String que contendrá el query de selección de la vista. En este punto, se llama a la clase “Tablas” en su método tabla, donde el parámetro x, será igual la variable con el mismo nombre que fue obtenida de lo ingresado por el usuario, después con el uso de un ResulSet se ejecuta la consulta obtenida y se imprime un mensaje en consola informando que la consulta ha sido ejecutada. Continua por declarar una variable booleana de nombre “r” la cual será “true” mientras el ResulSet siga iterando resultados.

```
try{ /*cachamos el error en la conexion a sql server */
    String connectionUrl = ("jdbc:sqlserver://10.20.55.223\\INIFAP;databaseName=redclima_pruebas;User=lenin;Password=Mclenin14.");
     Connection con = DriverManager.getConnection(connectionUrl);
     Statement sqll=null;
     ResultSet rs=null;
     if (con != null){
     System.out.println("Conexion realizada con exito");
     sqll=con.createStatement();
     String tabla =Tablas.tabla(x);
     rs=sqll.executeQuery(tabla);
     System.out.println("CONSULTA EJECUTADA");
     boolean r=rs.next();
```

Ilustración 16 Try-Catch conexión a SQL Server

Posterior a esto se realiza una nueva conexión a MongoDB la cual servirá para insertar, además se declara una variable tipo DB donde se almacenara el nombre de la base de datos donde se insertara, cabe mencionar que si en el servidor MongoDB no existe alguna base de datos con el nombre especificado aquí, automáticamente es creada al ejecutar un comando de creación de alguna colección. Se llama el método “colección”

de la clase tablas para crear una nueva colección donde será insertada la información, a base del número ingresado por el usuario.

```
Mongo mongo = new Mongo("10.20.55.227",83);
DB db = mongo.getDB("red_clima_pruebas"); /*nueva conexion de a mongo directamente a la collecion*/
   DBCollection collection =null; /* declaracion de variable tipo collecion*/
   collection=db.getCollection(Tablas.coleccion(x));
```

Ilustración 17 Conexión a una base de datos en servidor MongoDB

Para insertar en la base de datos MongoDB se utiliza el resultset donde se guardaron los resultados se la consulta en SQL Server, de manera que en un ciclo while, que para al obtener un falso de la variable booleana “r” declarada antes, obtiene los datos de cada registro, columna por columna en un String y estos son parseados al tipo de dato en que serán guardados en mongo DB, por ejemplo, se guarda el resultado de la columna ”ID_estacion” en un String, que es parseado a una variable de tipo int, con el mismo nombre, la cual será el valor insertado en
MongoDB.

```
String numer1=rs.getString("ID_estacion");
int ID_estacion=Integer.parseInt(numer1); /*conversiona entero los ID's*/
String Nombre_estacion=rs.getString("Nombre_estacion");/*se guardan los registros consultados en una variable*/
String numer2=rs.getString("ID_estado");
int ID_estado=Integer.parseInt(numer2);
String Nombre_estado=rs.getString("Nombre_estado");
String numer3=rs.getString("ID_mun");
int ID_mun=Integer.parseInt(numer3);
String Nombre_mun=rs.getString("Nombre_mun");
String fecha=rs.getString("fecha");
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.S");
Date date= formatter.parse(fecha);
double prec=rs.getFloat("prec");
double temt=rs.getFloat("temt");
double dirv=rs.getFloat("dirv");
double velv=rs.getFloat("velv");
double radg=rs.getFloat("radg");
double humr=rs.getFloat("humr");
double humh=rs.getFloat("humh");
double eto=rs.getFloat("eto");
```

Ilustración 18 Captura y parseo de los datos

Después de esto se crea una variable de tipo BasicDBObject que será el documento que será insertado en MongoDB, continua por insertar las variables guardadas en este documento, ingresando primero el nombre de la columna y después la variable.

```
BasicDBObject documento = new BasicDBObject(); /*se crea el documento a insertar*/

documento.put("_ID_estacion",ID_estacion);/*insercion de los string en el documento*/
documento.put("Nombre_estacion",Nombre_estacion);
documento.put("ID_estado",ID_estado);
documento.put("Nombre_estado",Nombre_estado);
documento.put("ID_mun",ID_mun);
documento.put("Nombre_mun",Nombre_mun);
documento.put("fecha",date);
documento.put("prec",prec);
documento.put("temt",temt);
documento.put("dirv",dirv);
documento.put("velv",velv);
documento.put("radg",radg);
documento.put("humr",humr);
documento.put("humh",humh);
documento.put("eto",eto);
```

Ilustración 19 Documento a insertar en MongoDB

Para finalizar el programar se inserta en la colección el documento generado y se imprime, después movemos el cursor al siguiente registro SQL. Todo esto dentro del Try-Catch de la conexión a SQL Server, si existe un error, se imprime “Error al ejecutar la sentencia en SQL Server” más la excepción que arroje SQL Server.

```
            collection.insert(documento);
            System.out.println(documento);
            r=rs.next();
            }
      }
    } catch (SQLException e) {
    System.out.println("Error al ejecutar la sentencia en SQL Server"+e);
       }
      }
    }
```

Ilustración 20 Inserción del documento y fin del programa

5: RESULTADOS

12. Resultados

Como resultados a los objetivos propuestos se realizaron varias acciones, las cuales son descritas en la siguiente tabla:

Tabla 10 Tabla de objetivos-resultados

Objetivo Propuesto	Resultado
Elaboración de la documentación que deberá permitir que el personal capacitado pueda migrar los datos, sin pérdida de información y conservando la integridad de los mismos.	Reporte de trabajo bajo el esquema propuesto por INIFAP, para el LNMySR.
Elaboración de un programa en Java el cual permita obtener, analizar y migrar los datos de una base de datos a otra.	Programa que permita migrar los datos de SQL Server a MongoDB
Realizar pruebas de migración de los datos para el análisis de los mismos, como que conserven su integridad y certeza, además de ayudar al análisis de la veracidad de los datos históricos almacenados.	Realizar reportes con el análisis de los datos, así como tablas de los tiempos de migración y respuesta.
Realizar pruebas de tiempos de consulta, disponibilidad de los datos y tiempo de migración para justificar que la elección de software haya sido la más óptima.	Documentación que justifique la migración, tablas comparativas entre SQL Server y MongoDB.

Se realizaron pruebas de migración por cada estado en la cual se contabilizo el tiempo que tardo el programa en migrar dicha tabla, se elaboró una tabla comparativa, que permite analizar el tiempo estimado de migración y el tiempo real en que tardo la migración, estas pruebas fueron realizadas en un pequeño servidor, por lo que se espera que la respuesta en un servidor de alta gama los resultados sean más óptimos.

Tabla 11 Tabla tiempos de migración por estado

Estado	Tiempo estimado min.	Tiempo estimado seg.	Registros	Estatus	Tiempo real
1	124.0317575	7441.9	16,950,283	migrado	124 min.
2	4.43	265.8	605,463	migrado	4 min. 27 seg.
3	24.73	1484.0	3,380,041	migrado	39 min. 58 seg.
4	54.97	3298.5	7,512,877	migrado	84 min. 5 seg.
5	76.72	4603.4	10,484,977	migrado	70 min. 41 seg.
6	34.13	2048.0	4,664,715	migrado	35 min. 45 seg.
7	46.89	2813.1	6,407,371	migrado	42 min. 18 seg.
8	174.69	10481.2	23,872,848	migrado	168. min 17 seg.
9	69.27	4156.0	9,466,033	migrado	60 min. 24 seg.
10	29.11	1746.4	3,977,760	migrado	28 min. 29 seg.
11	122.53	7351.8	16,745,165	migrado	110 min. 14 seg.
12	24.85	1491.0	3,396,005	migrado	23 min. 34 seg.
13	107.07	6424.2	14,632,372	migrado	96 min. 54 seg.
14	81.41	4884.9	11,126,219	migrado	64 min. 21 seg.
15	44.13	2647.6	6,030,274	migrado	39 min. 51 seg.
16	74.86	4491.8	10,230,917	migrado	64 min. 54 seg.
17	70.38	4222.6	9,617,712	migrado	64 min. 49 seg.
18	81.86	4911.4	11,186,487	migrado	76 min. 30 seg.
19	74.77	4486.4	10,218,686	migrado	69 min. 46 seg.
20	70.52	4231.0	9,636,859	migrado	63 min. 38 seg.
21	3.91	234.5	534,099	migrado	7 min. 29 seg.
22	37.76	2265.6	5,160,419	migrado	34 min. 22 seg.
23	96.97	5818.5	13,252,656	migrado	102 min. 13 seg.
24	114.13	6847.5	15,596,469	migrado	96 min. 36 seg.
25	235.16	14109.6	32,137,101	migrado	200 min. 13 seg.
26	14.01	840.5	1,914,316	migrado	24 min. 29 seg.
27	64.43	3865.6	8,804,679	migrado	60 min. 13 seg.

28	24.35	1461.0	3,327,672	migrado	22 min. 54 seg.
29	158.53	9512.0	21,665,228	migrado	138 min. 6 seg.
30	42.75	2565.3	5,842,825	migrado	33 min. 10 seg.
31	124.71	7482.5	17,042,794	migrado	119 min. 52 seg.
32	2.52	151.4	344,876	migrado	2 min. 37 seg.

En total se migraron 315,766,198 registro de 32 tablas, el tiempo estimado y el tiempo real de migración, fue el siguiente:

Tabla 12 Tabla tiempos de migración completos

	Tiempo estimado	**Tiempo real**
Minutos	2310.58	2204.65
Horas	38.51	36.74
Días	1.6	1.5

Además de esto se realizaron consultas de prueba en ambos servidores para la comparación de la eficiencia y eficacia de la nueva tecnología implementada.
Las consultas utilizadas son las siguientes:

1. Acumulaciones diarias a base de los registros de cada 15 minutos.
2. Qué municipio presento la mayor precipitación histórica, y en que mes y año sucedió.
3. En qué municipio se presentó la temperatura más baja de todos los meses de diciembre y que fecha fue.
4. Número de registros del estado 5.
5. Número de estaciones del estado 25.
6. Acumulaciones por hora a base de los registros de cada 15 minutos.

7.

La consulta número 1 en ambos gestores fue la siguiente:

```
select std.nombre as Nombre_estado,
std.indice as ID_estado,
mun.indice as mun_ID,
mun.nombre as Nombre_mun,
est.nombre as nombre_estacion,
est.numero as ID_estacion,
datepart(year,e1.fecha) as anio,
datepart(month,e1.fecha) as mes,
datepart(day,e1.fecha) as dia,
SUM(e1.prec) as prec,
MAX(e1.temt) as Tmax,
MIN(e1.temt) as Tmin,
AVG(e1.temt) as Tmed,
MAX(e1.velv) as VelvMax,
AVG(e1.velv) as Velv,
MAX(e1.dirv) as Dirvmax,
AVG(e1.dirv) as Dirv,
AVG(e1.radg) as Radg,
AVG(e1.humr) as Humr,
AVG(e1.eto) as eto,
from estado1 as e1, estaciones as est, estados as std, municipios as mun
where e1.numero=est.numero and est.municipioid=mun.indice and mun.idedo=std.indice
group by std.nombre,std.indice,mun.indice,mun.nombre,est.numero,est.nombre,
DATEPART(year,fecha), DATEPART(month,fecha), DATEPART(day,fecha)
```

Ilustración 21 Consulta 1 en SQL Server

```
db.getCollection("estado1").aggregate(
    [
        {"$group" : {
                "_id" : {
                    "Nombre_estado" : "$Nombre_estado",
                    "Nombre_mun" : "$Nombre_mun",
                    "Nombre_estacion" : "$Nombre_estacion",
                    "Anio" : {"$year" : "$fecha"},
                    "Mes" : {"$month" : "$fecha"},
                    "Dia" : {"$dayOfMonth" : "$fecha"}
                },
                "Prec" : {"$sum" : "$prec"},
                "Tmax" : {"$max" : "$temt"},
                "Tmin" : {"$min" : "$temt"},
                "Velvmax" : {"$max" : "$velv"},
                "Velv" : {"$avg" : "$velv"},
                "Dirvmax" : {"$max" : "$dirv"},
                "Dirv" : {"$avg" : "$dirv"},
                "Radg" : {"$avg" : "$radg"},
                "Humr" : {"$avg" : "$humr"},
                "Eto" : {"$avg" : "$eto"}
            }},
        { "$project" : {
                "_id" : NumberInt(0),
                "Nombre_estado" : "$_id.Nombre_estado",
                "Nombre_mun" : "$_id.Nombre_mun",
                "Nombre_estacion" : "$_id.Nombre_estacion",
                "Anio" : "$_id.Anio",
                "Mes" : "$_id.Mes",
                "Dia" : "$_id.Dia",
                "Prec" : "$Prec",
                "Tmax" : "$Tmax",
                "Tmin" : "$tmin",
                "Velvmax" : "$Velvmax",
                "Velv" : "$Velv",
                "Dirvmax" : "$Dirvmax",
                "Dirv" : "$Dirv",
                "Radg" : "$Radg",
                "Humr" : "$Humr",
                "Eto" : "Eto"
            }
        }
    ],
    {
        "allowDiskUse" : true
    }
);
```

Ilustración 22 Consulta 1 en MongoDB

La consulta número 2 en ambos gestores fue la siguiente:

```
select datepart(year,e.fecha) as anio,DATEPART(month,e.fecha) as mes,
max(e.prec), mun.nombre
from estado1 as e, municipios as mun, estaciones as es
where e.numero=es.numero and es.municipioid=mun.indice
group by datepart(year,e.fecha),datepart(month,e.fecha),mun.nombre
order by max(prec) desc;
```

Ilustración 23 Consulta 2 en SQL Server

```
db.getCollection("estado1").aggregate(
    [
        {
            "$group" : {
                "_id" : {
                    "Anio" : {"$year":"$fecha"},
                    "Mes" : {"$month":"$fecha"} ,
                    "Nombre_mun" : "$Nombre_mun"
                },
                "MAX(prec)" : {"$max" : "$prec"}
            }
        },
        {
            "$project" : {
                "_id" : NumberInt(0),
                "Anio" : "$_id.Anio",
                "Mes" : "$_id.Mes" ,
                "Nombre_mun" : "$_id.Nombre_mun",
                "Precipitacion" : "$MAX(prec)"
            }
        },
        {
            "$sort" : {
                "Precipitacion" : NumberInt(-1)
            }
        }
    ],
    {
        "allowDiskUse" : true
    }
);
```

Ilustración 24 Consulta 2 en MongoDB

La consulta número 3 en ambos gestores fue la siguiente:

```
select MIN(temt) as temt_min, mun.nombre, DATEPART(year,fecha) as anio, DATEPART(MONTH,fecha)
as mes, DATEPART(day,fecha)
from estado1  as std, municipios as mun, estaciones as st
where DATEPART(month,fecha)=12 and std.numero=st.numero and st.municipioid=mun.indice
group by mun.nombre, DATEPART(year,fecha),DATEPART(MONTH,fecha),DATEPART(DAY,fecha)
order by MIN(temt) asc
```

Ilustración 25 Consulta 3 en SQL Server

```
db.getCollection("estado1").aggregate(
    [
        {
            "$project" : {
                "Anio" : {"$year":"$fecha"},
                "Mes" : {"$month":"$fecha"},
                "Dia" : {"$dayOfMonth":"$fecha"},
                "temt":"$temt",
                "Nombre_muni" : "$Nombre_mun"
            }
        },
        {
            "$match" : {
                "Mes" : NumberInt(12),

            }
        },
        {
            "$group" : {
                "_id" : {
                    "Anio" : "$Anio",
                    "Mes" : "$Mes" ,
                    "Dia" : "$Dia",
                    "Nombre_municipio" : "$Nombre_muni"
                },
                "Tmin" : {"$min":"$temt"}
            }
        },
        {
            "$project" : {
                "_id" : NumberInt(0),
                "Anio" : "$_id.Anio",
                "Mes" : "$_id.Mes" ,
                "Dia" :"$_id.Dia",
                "Nombre_mun" : "$_id.Nombre_municipio",
                "Tmin" : "$Tmin"
            }
        },
        {
            "$sort" : { "Tmin" : NumberInt(1)}
        }
    ],
    {
        "allowDiskUse" : true
    }
);
```

Ilustración 26 Consulta 3 en MongoDB

La consulta número 4 en ambos gestores fue la siguiente:

```
select avg(temt) as temt_med, st.nombre, DATEPART(MONTH,fecha)
 as mes, DATEPART(day,fecha) as dia , datepart(hour,fecha) as hora
 from estado1  as std, estaciones as st
 where DATEPART(YEAR,fecha)=2017 and std.numero=st.numero
 group by st.nombre,DATEPART(MONTH,fecha),DATEPART(DAY,fecha),DATEPART(hour,fecha)
order by st.nombre,DATEPART(month,fecha),DATEPART(DAY,fecha),DATEPART(HOUR,fecha)
```

Ilustración 27 Consulta 4 en SQL Server

```
db.getCollection("estado1").aggregate(
    [
        {
            "$project" : {
                "Anio" : {"$year":"$fecha"},
                "Mes" : {"$month":"$fecha"},
                "Dia" : {"$dayOfMonth":"$fecha"},
                "Hora" : {"$hour":"$fecha"},
                "temt":"$temt",
                "Nombre_estacion" : "$Nombre_estacion"
            }
        },
        {
            "$match" : {
                "Anio" : NumberInt(2017),
                "Nombre_estacion":"CEPAB"

            }
        },
        {
            "$group" : {
                "_id" : {
                    "Mes" : "$Mes" ,
                    "Dia" : "$Dia",
                    "Hora" : "$Hora",
                    "Nombre_estacion" : "$Nombre_estacion"
                },
                "Tmed" : {"$avg":"$temt"}
        }},
        {
            "$project" : {
                "_id" : NumberInt(0),
                "Anio" : "$_id.Anio",
                "Mes" : "$_id.Mes" ,
                "Dia" :"$_id.Dia",
                "Hora":"$_id.Hora",
                "Nombre_mun" : "$_id.Nombre_estacion",
                "Tmed" : "$Tmed"
        }},
        {
            "$sort" : {"Nombre_estacion":NumberInt(1),
                       "Mes":NumberInt(1),
                       "Dia":NumberInt(1),
                       "Hora":NumberInt(1)
            }}
    ],
    {
        "allowDiskUse" : true
    }
);
```

Ilustración 28 Consulta 4 en MongoDB

La consulta número 5 en ambos gestores fue la siguiente:

```
select st.nombre, MAX(fecha)
from estado1  as std, estaciones as st
where  std.numero=st.numero
group by st.nombre
order by st.nombre
```

Ilustración 29 Consulta 5 en SQL Server

```
db.getCollection("estado1").aggregate(
    [

        {
            "$group" : {
                "_id" : {
                    "Nombre_estacion" : "$Nombre_estacion"
                },
                "fecha_max" : {"$max":"$fecha"}
        }},
        {
            "$project" : {
                "_id" : NumberInt(0),
                "Nombre_mun" : "$_id.Nombre_estacion",
                "ultima":"$fecha_max"
        }},
    ],
```

Ilustración 30 Consulta 5 en MongoDB

La consulta número 6 en ambos gestores fue la siguiente:

```
select std.nombre as Nombre_estado,
std.indice as ID_estado,
mun.indice as mun_ID,
mun.nombre as Nombre_mun,
est.nombre as nombre_estacion,
est.numero as ID_estacion,
datepart(year,e1.fecha) as anio,
datepart(month,e1.fecha) as mes,
datepart(day,e1.fecha) as dia,
DATEPART(HOUR,e1.fecha) as hora,
SUM(e1.prec) as prec,
MAX(e1.temt) as Tmax,
MIN(e1.temt) as Tmin,
AVG(e1.temt) as Tmed,
MAX(e1.velv) as VelvMax,
AVG(e1.velv) as Velv,
MAX(e1.dirv) as Dirvmax,
AVG(e1.dirv) as Dirv,
AVG(e1.radg) as Radg,
AVG(e1.humr) as Humr,
AVG(e1.eto) as eto,
from estado1 as e1, estaciones as est, estados as std, municipios as mun
where e1.numero=est.numero and est.municipioid=mun.indice and mun.idedo=std.indice
```

Ilustración 31 Consulta 6 en SQL Server

```
use red_clima;
db.getCollection("estado1").aggregate(
    [
        {
            "$group" : {
                "_id" : {
                    "Nombre_estado" : "$Nombre_estado",
                    "Nombre_mun" : "$Nombre_mun",
                    "Nombre_estacion" : "$Nombre_estacion",
                    "Anio" : {"$year" : "$fecha"},
                    "Mes" : {"$month" : "$fecha"},
                    "Dia" : {"$dayOfMonth" : "$fecha"},
                    "Hora" : {"$hour" : "$fecha"}},
                "Prec" : {"$sum" : "$prec"},
                "Tmax" : {"$max" : "$temt"},
                "Tmin" : {"$min" : "$temt"},
                "Velvmax" : {"$max" : "$velv"},
                "Velv" : {"$avg" : "$velv"},
                "Dirvmax" : {"$max" : "$dirv"},
                "Dirv" : {"$avg" : "$dirv"},
                "Radg" : {"$avg" : "$radg"},
                "Humr" : {"$avg" : "$humr"},
                "Eto" : {"$avg" : "$eto"}}},
        { "$project" : {
                "_id" : NumberInt(0),
                "Nombre_estado" : "$_id.Nombre_estado",
                "Nombre_mun" : "$_id.Nombre_mun",
                "Nombre_estacion" : "$_id.Nombre_estacion",
                "Anio" : "$_id.Anio",
                "Mes" : "$_id.Mes",
                "Dia" : "$_id.Dia",
                "Hora" : "$_id.Hora",
                "Prec" : "$Prec",
                "Tmax" : "$Tmax",
                "Tmin" : "$tmin",
                "Velvmax" : "$Velvmax",
                "Velv" : "$Velv",
                "Dirvmax" : "$Dirvmax",
                "Dirv" : "$Dirv",
                "Radg" : "$Radg",
                "Humr" : "$Humr",
                "Eto" : "$Eto"
            }}
    ],
    { "allowDiskUse" : true}
);
```

Ilustración 32 Consulta 6 en MongoDB

Tabla 13 Tiempos de consultas

Consulta No.	**SQL Server**	**Mongo DB**
1	01:16	01:20.3
2	00:04	00:38.3
3	00:05	01:08.9
4	00:05	00:10.5
5	00:05	00:16.1
6	02:43	02:07.6

Al notar la variación de ambos gestores, se realizó una nueva comparación, con la consulta número 6 que es la más extensa, por cada uno de los estados, y se contabilizaron los resultados:

Tabla 14 Comparación en acumuladores por hora

Estado	**Tiempo SQL Server**	**Conteo SQL Server**	**Tiempo MongoDB**	**Conteo MongoDB**
1	02:43.0	4,282,231	02:07.6	4,238,280
2	00:04.0	114,176	00:04.4	109,377
3	00:22.0	824.289	00:16.1	819,888
4	01:19.0	1,860,087	00:55.4	1,856,042
5	01:38.0	2,655,186	01:21.0	2,628,698
6	00:30.0	1,178,029	00:34.7	1,166,183
7	01:11.0	1,576,261	00:47.4	1,566,267
8	04:10.0	6,080,775	02:58.4	5,968,195
9	01:48.0	2,397,063	01:12.5	2,364,087
10	00:27.0	1,016,055	00:32.2	994,454
11	03:31.0	4,195,195	02:02.2	4,186,085

12	00:19.0	814,617	00:25.8	808,710
13	02:26.0	3,684,148	01:50.1	3,658,408
14	01:52.0	2,742,029	01:22.4	2,728,666
15	01:00.0	1,463,669	00:43.6	1,462,126
16	01:31.0	2,557,797	01:15.9	2,557,636
17	01:39.0	2,406,058	01:11.3	2,404,917
18	02:00.0	2,829,473	01:26.2	2,808,999
19	01:52.0	2,550,891	01:19.9	2,523,886
20	01:34.0	2,419,543	01:14.5	2,409,257
21	00:03.0	125,721	00:04.1	119,486
22	00:55.0	1,263,574	00:40.6	1,260,610
23	02:44.0	3,349,672	01:44.2	3,313,179
24	03:36.0	3,918,713	01:55.8	3,899,102
25	05:41.0	8,158,240	03:58.1	8,033,459
26	00:09.0	446,671	00:14.1	444,036
27	01:24.0	2,180,037	00:06.3	2,174,169
28	00:21.0	827,892	00:25.4	813,448
29	03:31.0	5,419,315	02:42.4	5,416,323
30	00:44.0	1,470,693	00:43.3	1,460,787
31	02:47.0	4,317,361	02:08.7	4,261,614
32	00:03.0	76,161	00:02.0	76,161

Con base a los datos recabados de esta comparación, se realizó un cálculo de tiempo en que tarda cada manejador en obtener un registro en segundos

Tabla 15 Comparación búsqueda por segundo

Estado	**Registros por segundo SQL Server**	**Registros por segundo MongoDB**
1	26271	33215
2	28544	24858
3	37	50925
4	23545	33503
5	27094	32453
6	39268	33608
7	22201	33044
8	24323	33454
9	22195	32608
10	37632	30884
11	19882	34256
12	42875	31345
13	25234	33228
14	24482	93794
15	24394	33535
16	28108	33697
17	24304	33730
18	23579	32587
19	22776	31588
20	25740	32339
21	41907	29143
22	22974	31050
23	20425	31796
24	18142	33671
25	23924	33740
26	49630	31492
27	25953	345106
28	39423	32026
29	25684	33352
30	33425	33736
31	25852	33113
32	25387	38081
Promedio	27038	44842

13. Actividades Sociales realizadas en la empresa u organización.

1. Apoyo técnico en el desarrollo de talleres y cursos impartidos por el personal del LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos).
2. Apoyo técnico a los asistentes de los cursos de “Estadística con Python” y “Domótica con Arduino” con búsqueda y resolución de problemas en la programación desarrollada durante el curso.
3. Apoyo técnico a los asistentes de los cursos de “Domótica con Arduino” con busque y resolución de problemas en el diseño de circuitos desarrollada durante el curso.
4. Complementación de presentación del LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos) para alumnos de la UTA y Grupo Modelo
5. Presentación ante investigadores INIFAP del proyecto realizado.
6. Apoyo técnico a los asistentes de los cursos de “Programación en Android” y “Pre procesamiento de datos con SQL Server” con búsqueda y resolución de problemas en la programación desarrollada durante el curso.
7. Apoyo en limpieza del área de trabajo.

CAPÍTULO 6: CONCLUSIONES

14. Conclusiones del Proyecto

Con este proyecto podemos concluir que MongoDB es la opción buscada por el LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos) ya que con las pruebas se comprobó que los tiempos de consulta son más cortos al incrementar la complejidad de una consulta y el tamaño de datos buscados, esto quiere decir que cubrirá la necesidad en cuanto a eficacia, el conocimiento adquirido para la compresión y uso de MongoDB se obtuvo de sus manuales oficiales y de la comunidad de desarrolladores que lo utilizan por lo que una capacitación con costos no fue necesaria.

Además de esto es una herramienta de fácil instalación y configuración por lo que no será difícil montarlo en un servidor de altos recursos, además de esto, el manejador que se utilizó, no necesito un pago de licencia por lo que el uso de esta tecnología reduce los gastos de software a poco o nulos, y así reducir el costo en general para mantener la base de datos ya que con esto solo se necesita invertir en el software necesario para su uso.

El LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos) es una dependencia federal para el desarrollo de investigaciones que ayuden a los sectores pecuarios, forestales y agropecuarios por lo cual, el tener recursos informáticos menos costos ayuda a la disponibilidad de la información para el sector en que se enfoca.

Además de las estaciones climatológicas el LNMySR (Laboratorio Nacional de Modelaje y Sensores Remotos) cuenta con tecnologías WRF que permiten generar pronósticos horarios climatológicos, MongoDB permitirá guardar esa información y así aplicar el Big data de varias bases de datos.

CAPÍTULO 7: COMPETENCIAS DESARROLLADAS

15. Competencias desarrolladas y/o aplicadas.

1. Se aplicó habilidades analíticas para el estudio de los datos que se trataron en este proyecto.
2. Se emprendió la búsqueda de recursos que cumplieran los requerimientos.
3. Se diseñó un algoritmo de migración que permita migrar en varios lenguajes de programación.
4. Se diseñaron estructuras de datos innovadores que permita una búsqueda y control más óptimo de los datos.
5. Se aplicaron habilidades de lógica en estructura de datos para el desarrollo de consultas más optimizadas.
6. Se aplicaron habilidades matemáticas para el análisis de resultados de tiempos de migración
7. Se gestionaron de manera correcta los recursos para el proyecto con el fin de mejorar el rendimiento.
8. Se aplicaron métodos cuantitativos y cualitativos en el análisis e interpretación de datos para la optimización de los recursos informáticos.
9. Se aplicaron habilidades descriptivas para el sustento de la viabilidad del proyecto.
10. Se gestionaron de manera constitucional recursos licencias gratuitas para el uso de software necesario para el uso de las nuevas tecnologías.
11. Se aplicaron conocimientos sobre base de datos para el previo análisis de los recursos a cambiar.
12. Se aplicaron habilidades de trabajo en equipo para el correcto uso de los recursos y la optimización de actividades a desarrollar.

CAPÍTULO 8: FUENTES DE INFORMACIÓN

16. Fuentes de información

Referencias

Brito, F. (19 de 09 de 2012). *Blog Spot*. Obtenido de http://basededatosfrancisbrito.blogspot.com/2012/09/propiedades-acid.html

EcuRed. (18 de 11 de 2011). Obtenido de https://www.ecured.cu/MongoDB

Gardey, J. P. (2013). *definicion.de* . Obtenido de https://definicion.de/java/

Genbeta. (03 de 02 de 2014). Obtenido de https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

INIFAP. (09 de 09 de 2018). Como funciona la base de datos red clima. (O. L. Alvarez, Entrevistador)

Json.org. (2016). *Json.org*. Obtenido de https://www.json.org/json-es.html

Mongodb.com. (s.f.). Obtenido de https://www.mongodb.com/partners/studio-3t

Oracle_Corporation. (s.f.). *Netbeans.org*. Obtenido de https://netbeans.org/community/releases/61/index_es.html

Rochina, P. (14 de 12 de 2016). *Revista digital INESEM*. Obtenido de https://revistadigital.inesem.es/informatica-y-tics/bases-datos-nosql-mongodb/

UCM. (2017). Obtenido de http://gpd.sip.ucm.es/rafa/docencia/nosql/Agregando.html

Valdés, D. P. (26 de 10 de 2007). *Maestros de la web*. Obtenido de http://www.maestrosdelweb.com/que-son-las-bases-de-datos/

CAPÍTULO 9: ANEXOS

17. Anexos

SAGARPA
SECRETARÍA DE AGRICULTURA, GANADERÍA, DESARROLLO RURAL, PESCA Y ALIMENTACIÓN

inifap
Instituto Nacional de Investigaciones Forestales, Agrícolas y Pecuarias

CENTRO DE INVESTIGACIÓN REGIONAL NORTE CENTRO
CAMPO EXPERIMENTAL PABELLÓN

ASUNTO: RESIDENCIA PROFESIONAL.

Pabellón de Arteaga, Ags., 20 de agosto de 2018.

MLI. JULISSA ELAYNE COSME CASTORENA.
JEFA DE DEPARTAMENTO GESTIÓN DE TECNOLÓGICA Y VINCULACIÓN
INSTITUTO TECNOLÓGICO DE PABELLÓN
P R E S E N T E

Por medio del presente, me permito informarle que aceptamos la propuesta para que el alumno C. OSCAR LENIN ESPINOZA ÁLVAREZ, de la carrera de Ingeniería en Tecnologías de la Información y Comunicaciones con número de control 131050154, realice su proyecto de Residencias Profesionales en esta unidad administrativa cubriendo un total de 500 hrs., en el periodo del 20 de agosto a 30 de noviembre del presente, bajo la supervisión del Dr. Víctor Manuel Rodríguez Moreno.

Sin otro particular por el momento, aprovecho la ocasión para enviarle un cordial saludo.

ATENTAMENTE
DIRECTOR DE COORDINACIÓN Y VINCULACIÓN EN AGUASCALIENTES

DR. ALFONSO PEÑA RAMOS

INSTITUTO NACIONAL DE INVESTIGACIONES FORESTALES, AGRICOLAS Y PEC CAMPO EXPERIMENTAL PABELLON, AGS.

Campo Experimental Pabellón
Km 32.5 Carretera Aguascalientes-Zacatecas
Pabellón de Arteaga, Aguascalientes C.P. 20671
Tel.: (55) 3871 8700 ext. 82534 y 82531
www.inifap.gob.mx