

Instituto Politécnico Nacional
Escuela Superior de Cómputo

Trabajo Terminal No. 2019-B052

Herramienta para transformar un modelo
entidad-relación a un modelo no relacional

Presentan:

Aparicio Quiroz Omar
Martínez Acosta Eduardo

Directoras:

M. en C. Ocotitla Rojas Nancy
D. en C. Chavarria Baez Lorena

Índice general

1. Introducción	8
1.1. Descripción del problema	9
1.2. Propuesta de solución	9
1.3. Justificación	10
1.4. Objetivo	10
1.5. Alcance y limitaciones	11
2. Estado del Arte	12
2.1. Kashliev Data Modeler	12
2.2. NoSQL Workbench for Amazon DynamoDB	13
2.3. HacKolade	13
2.4. Mortadelo	13
2.5. Schema Design for NoSQL Applications	14
2.6. Conclusiones	15
3. Marco Teórico	18
3.1. Modelos de datos para bases de datos	18
3.1.1. Modelo entidad-relación	18
3.1.1.1. Entidades	18
3.1.1.2. Atributos	19
3.1.1.3. Relaciones	21
3.1.1.4. Resumen de la notación para los diagramas ER	23
3.1.2. El modelo entidad-relación extendido	24
3.1.2.1. Especialización	24
3.1.2.2. Generalización	25
3.1.2.3. Restricciones	25
3.1.2.4. Jerarquías múltiples y herencia	25

3.1.2.5.	Unión	25
3.1.3.	Modelo relacional	26
3.1.3.1.	Relaciones	26
3.1.3.2.	Claves	28
3.1.3.3.	Restricciones de integridad	28
3.1.3.4.	Propiedades de las relaciones	28
3.1.3.5.	ACID	28
3.1.3.6.	Structured Query Language	29
3.1.4.	Modelos NoSQL	32
3.1.4.1.	Teorema CAP	32
3.1.4.2.	Clave-Valor	33
3.1.4.3.	Orientado a documentos	34
3.1.4.4.	Orientado a columnas	36
3.1.4.5.	Orientado a grafos	37
3.1.4.6.	Orientado a grafos	38
3.2.	Tecnologías a usar	39
3.2.1.	Hypertext Transfer Protocol	39
3.2.2.	HTML 5	40
3.2.3.	CSS vs SASS	41
3.2.4.	JavaScript vs TypeScript	43
3.2.5.	JavaScript Frameworks: Vue/Nuxt vs React vs Angular	45
3.2.6.	CSS Frameworks: Vuetify vs Bootstrap	47
3.2.7.	MongoDB vs Apache CouchDB	48
3.2.8.	GoJS vs mxGraph vs D3.js	49
3.2.9.	Python 3 vs Java	49
3.2.10.	Django vs Flask	51
4.	Análisis y Diseño del Sistema	53
4.1.	Metodología	53
4.1.1.	SCRUM	54
4.1.1.1.	Lista de Producto (<i>Product Backlog</i>)	55
4.1.1.2.	Historias de usuario	56
4.2.	Factibilidad técnica	59
4.3.	Factibilidad económica	60
4.3.1.	COCOMO II	60

4.3.1.1.	El modelo de composición de aplicación COCOMO	
	II	61
4.4.	Factibilidad operativa	63

Índice de figuras

3.1. Tabla en el modelo Relacional	27
3.2. <i>Bucket</i> en el almacenamiento clave-valor	34
3.3. Documento en el almacenamiento de documentos	35
3.4. Familia de columnas	36
3.5. modelo conceptual orientado a grafos	37
3.6. modelo conceptual orientado a grafos	38

Índice de tablas

3.1. Notación para los diagramas ER	23
3.2. Comandos de SQL.	30

Resumen

Capítulo 1

Introducción

En los últimos años, los sistemas *Not only SQL* o NoSQL han surgido como alternativa a los sistemas de bases de datos relacionales y se enfocan, principalmente, en buscar resolver problemáticas inherentes al usarlas en algunos escenarios.

Los sistemas NoSQL admiten diferentes modelos de datos, como los de clave-valor, documentos, columnas y grafos; con ello se enfocan en guardar datos de una manera apropiada de acuerdo a los requisitos actuales para la gestión de datos en la web o en la nube, enfatizando la escalabilidad, la tolerancia a fallos y la disponibilidad a costa de la consistencia.

De acuerdo a Google Trends, en 2009 aumentó el interés en los sistemas NoSQL y desde entonces, también empezaron a resaltar algunas problemáticas propias de estos sistemas, porque si bien resuelven algunos dilemas de las bases de datos relacionales, por enfocarse en la ya mencionada escalabilidad, la tolerancia a fallos o la flexibilidad para realizar cambios en el esquema de la base de datos, la heterogeneidad de los sistemas NoSQL ha llevado a una amplia diversificación de las interfaces de almacenamiento de datos, provocando la pérdida de un paradigma de modelado común o de un lenguaje de consultas estándar como SQL.

Respecto al modelado de datos, en el diseño de las bases de datos tradicionales, el modelo entidad-relación[1] es el modelo conceptual más usado y tiene procedimientos bien establecidos como resultado de décadas de investigación, sin embargo, para los sistemas NoSQL los enfoques tradicionales de diseño de bases de datos no proporcionan un soporte adecuado para satisfacer el modelado de sus diferentes modelos de datos y para abordar esta problemática se han creado varias metodologías de diseño para sistemas NoSQL en los últimos años, porque para modelar estos sistemas, los diseñadores de bases de datos deben tener en cuenta no solo qué datos se almacenarán en la base de datos, sino también cómo se accederán a ellos [2]-[4].

Para tener en cuenta cómo se accederán a los datos se debe conocer cómo se realizarán las consultas de los mismos y de acuerdo al trabajo de Mosquera[5], que es una investigación 1376 *papers* sobre el modelado de sistemas NoSQL, se muestra que de las metodologías propuestas la mayoría usa el lenguaje de modelado UML (*Unified Modeling Language*), mientras que algunos proponen su propio modelo conceptual; en resumen, en la literatura sobre el tema solo existen cinco herramientas propuestas para el modelado conceptual y, en general, no hay una tendencia en el modelado de datos NoSQL.

Lo que resta del capítulo está organizado de la siguiente manera: primero se muestra la problemática a resolver, después la propuesta de solución, la justificación, el objetivo del proyecto y por último se menciona el alcance con las limitaciones del mismo.

1.1. Descripción del problema

Como se ha visto en la introducción, son pocas las herramientas propuestas en la literatura para el modelado de sistemas NoSQL en sus tres niveles de abstracción (nivel conceptual, lógico y físico).

Solo en el modelado conceptual, el modelo entidad-relación puede considerarse como una tendencia, sin embargo, el modelo entidad-relación por sí mismo no es suficiente para representar cómo se consultarán los datos ni tampoco con qué frecuencia se accederán a ellos; por eso, para modelar sistemas NoSQL es necesario conocer enfoques de desarrollo como el *query-driven design*, el *domain-driven design*, el *data-driven design* o el *workload-driven design*.

Ahora bien, respecto a los modelos de datos NoSQL, no hay un modelo estándar ni tampoco existe un acuerdo sobre la mejor definición de reglas de transformación entre modelos en los tres niveles de abstracción. Además, en la literatura sobre el tema se pueden encontrar unos 36 estudios que proponen diferentes enfoques para las transformaciones.

Asimismo, por los pocos años que han pasado desde que el interés sobre el tema aumentó y que son usados estos sistemas de bases de datos, aún no se ven reflejados en los programas de estudio o solo se da una introducción del tema a los estudiantes de licenciatura en ingeniería en sistemas o carreras afines.

En consecuencia, para el estudiante es un problema tener que enfrentarse a diseñar un sistema NoSQL, porque no se le enseña ninguna metodología de desarrollo para sistemas NoSQL, tampoco conoce los distintos modelos de datos NoSQL que hay, mucho menos sus ventajas o desventajas para poder elegir el modelo más apropiado para su aplicación y, como resultado, no pueden visualizar el diseño de una base de datos no relacional a partir de los conocimientos adquiridos en la asignatura de Bases de Datos.

Además, la escasez de herramientas CASE para el diagramado de esquemas no relacionales o herramientas que apoyen la migración de un modelo entidad-relación o relacional a uno no relacional aumenta la complejidad para que el estudiante haga uso de una base de datos no relacional.

1.2. Propuesta de solución

Desarrollar una herramienta web que permita la edición de un diagrama de bases de datos bajo el modelo entidad-relación y realice la validación del mismo, posteriormente el diagrama podrá ser transformado al modelo relacional con la posibilidad de obtener el esquema de la base de datos en sentencias SQL, o bien,

obtendrá el esquema conceptual de base de datos en un modelo de datos NoSQL orientado a documentos y tendrá la posibilidad de implementar el modelo no relacional en MongoDB, que es un NoSQL RDBMS (*NoSQL Database Management System*).

1.3. Justificación

Las pocas herramientas para modelado conceptual NoSQL no ofrecen un modelado desde un modelo conceptual relacional, porque solo se enfocan en proponer modelos conceptuales para sistemas NoSQL, tampoco ofrecen validaciones para sus sistemas (porque son propuestas de modelos conceptuales) ni dan la posibilidad de obtener el esquema SQL, ya que no están enfocados en sistemas relacionales.

Asimismo, por los pocos años que han pasado desde que el interés y el uso de los sistemas NoSQL aumentó, estos sistemas todavía no están reflejados en los programas de estudio.

En consecuencia, para abordar esta problemática en la que el estudiante pueda diseñar un sistema NoSQL sin haber aprendido ninguna metodología de desarrollo para dichos sistemas y sin poder visualizar en primera instancia el diseño de una base de datos no relacional a partir de lo que ve en la asignatura de Bases de Datos, se propone una herramienta CASE que pueda apoyar al estudiante a diseñar una base de datos NoSQL a partir de los conocimientos adquiridos del estudiante.

Se usará como modelo conceptual el modelo entidad-relación por ser un modelo bien conocido para el estudiante de licenciatura en sistemas, está probado en el área de las bases de datos y es también el más usado en las investigaciones sobre el modelado conceptual de sistemas NoSQL.

Respecto al modelo físico que obtenga la herramienta a desarrollar, se probará en el esquema orientado a documentos en MongoDB, porque en ambos casos (el modelo orientado a documentos y MongoDB) son los más usados en la literatura sobre modelado NoSQL[5].

1.4. Objetivo

Desarrollar una herramienta web que permita la edición de un diagrama de bases de datos bajo el modelo entidad-relación y realice la validación del mismo.

El diagrama podrá ser transformado al modelo relacional con la posibilidad de obtener el esquema de la base de datos en sentencias SQL, o bien, obtendrá el esquema de base de datos en un modelo de datos no relacional y tendrá la posibilidad de implementar el modelo no relacional a un SGBDNR (Sistema Gestor de Base de Datos No Relacional).

1.5. Alcance y limitaciones

Se usará la notación oficial en los modelos de datos conceptuales entidad-relación y relacional, es decir, la notación de Chen y la notación de Codd, respectivamente.

Para la generación de scripts de bases de datos relacionales se usará el lenguaje de consultas SQL.

Asimismo, los scripts generados se probarán con MySQL, ya que es SGBD que se usa en la asignatura de Base de Datos y es con el que están más familiarizados los alumnos.

Para la transformación del modelo de datos conceptual del modelo entidad-relación al modelo relacional no se realizará ninguna normalización, porque se necesita un esquema no normalizado para la transformación al modelo conceptual de datos NoSQL.

Los scripts NoSQL generados por la aplicación solo serán probados y ejecutados en MongoDB, porque es un lenguaje de consultas propias de MongoDB.

El usuario podrá acceder con un correo o no válido, teniendo en cuenta de que si no crea su cuenta con un correo válido, no le llegará notificación de correo que le indica que se ha registrado exitosamente junto con su contraseña.

El diagramador entidad-relación solo podrá editar elementos del modelo entidad-relación, no se podrá diagramar elementos del modelo entidad-relación extendido, porque este último modelo no es indispensable para el objetivo del proyecto.

No se podrá editar visualmente el diagrama generado para el modelo relacional.

Tampoco se podrá editar visualmente el diagrama conceptual del modelo NoSQL orientado a documentos.

El modelo elegido para desarrollar la propuesta de modelo conceptual será independiente del modelo de datos NoSQL o enfocado en el modelo NoSQL orientado a documentos.

Asimismo, aunque la propuesta de modelo conceptual sea independiente del modelo de datos NoSQL, la generación de scripts será únicamente en el modelo de datos orientado a documentos.

El grado de partición en el modelo entidad-relación será máximo de dos entidades.

Capítulo 2

Estado del Arte

Para la elaboración de la presente propuesta de Trabajo Terminal, ha sido necesaria una investigación de herramientas similares en el ámbito académico y empresarial.

A continuación se presentan las ofertas similares y se finaliza con una tabla comparativa:

2.1. Kashliev Data Modeler

Es una herramienta de modelado desarrollada a partir del 2015 por Andrii Kashliev, profesor asistente del Departamento de informática de la Universidad del Oeste Michigan, que automatiza el diseño de esquemas para Apache Cassandra, una base de datos NoSQL orientada a columnas.

Por medio de una aplicación web, KDM ayuda al usuario en el modelado de datos, comenzando con un modelo conceptual de datos (de una notación familiar al modelo entidad-relación) y generando un modelo físico de datos o *script* CQL (Cassandra Query Language).

KDM automatiza:

1. El mapeo conceptual a lógico
2. El mapeo lógico a físico
3. La generación de *script* CQL

El modelo de datos usado en KDM es el propuesto por Chebotko[3] y por medio de un algoritmo, KDM automatiza el proceso de pasar del modelo de datos a un modelo de datos físicos.

Además, KDM cuenta con una versión de prueba de tiempo indefinido con características limitadas que permite la generación de un modelo lógico y guardar los proyectos de KDM.

2.2. NoSQL Workbench for Amazon DynamoDB

NoSQL Workbench for Amazon DynamoDB es una herramienta desarrollada por Oracle Corporation que proporciona funciones de modelado de datos, visualización de datos y desarrollo de consultas para ayudar a diseñar, crear, consultar y administrar bases de datos NoSQL de clave-valor y orientadas a documentos.

También se pueden crear nuevos modelos de datos o diseñar modelos basados en modelos de datos existentes que satisfagan los requerimientos de acceso a datos de su aplicación, así como importar y exportar el modelo de datos diseñado al final del proceso.

El visualizador de modelo de datos proporciona un lienzo donde puede asignar consultas y visualizar las facetas (parte de la base de datos) de la aplicación sin tener que escribir código.

Cada faceta corresponde a un patrón de acceso diferente en DynamoDB; puede agregar datos manualmente a su modelo de datos o importar datos desde MySQL.

Cuenta con un generador de operaciones para ver, explorar y consultar conjuntos de datos.

Por último, admite la proyección, la declaración de expresiones condicionales y permite generar código de muestra en varios idiomas.

2.3. HacKolade

Hackolade es un *software* con capacidad de representar objetos JSON anidados; la aplicación combina la representación gráfica de colecciones (término usado en las bases de datos orientados a documentos) y vistas en un diagrama entidad-relación.

La aplicación está basada en la desnormalización, el polimorfismo y matrices anidadas JSON; la representación gráfica de la definición del esquema JSON de cada colección está en una vista de árbol jerárquica.

Hackolade genera dinámicamente *scripts* MongoDB a medida que construye un modelo de datos, deriva modelos de datos por medio de ingeniería inversa de instancias MongoDB, dando facilidad para obtener descripciones, propiedades y restricciones.

Es una herramienta de modelado de datos para MongoDB, Neo4j, Cassandra, Couchbase, Cosmos DB, DynamoDB, Elasticsearch, HBase, Hive, Google BigQuery, Firebase/Firestore, MarkLogic, entre otros.

2.4. Mortadelo

Es un framework para el diseño automático de bases de datos NoSQL que ofrece un proceso de transformación basado en modelos con el objetivo de brindar un tratamiento homogéneo a los diferentes paradigmas sobre tecnologías bien conocidas.

Define un proceso de transformación el cual, a través de una serie de pasos transforma primero el modelo conceptual provisto denotado como GDM (generic data model) el cual representa una definición conceptual de la base de datos dada por el usuario en un modelo lógico dependiendo del paradigma NoSQL, para después generar scripts de implementación que instanciarían la tecnología NoSQL específica de ese paradigma

Parte de la validación del GDM compuesto por dos bloques que contienen la información sobre el dominio de las entidades y sus relaciones (Modelo de estructura) y la definición como los datos del modelo serán solicitados por el esquema (Consultas de acceso), posteriormente se traduce el GDM en las especificaciones lógicas NoSQL mediante la aplicación de un conjunto de reglas de transformación para finalmente transformar el modelo en texto proporcionando un diseño e implementación generados automáticamente para el almacén de datos NoSQL deseado.

Para la comprobación de su validez, se ha implementado una herramienta que admite la generación de almacenes de datos de columnas con transformaciones concretas para Cassandra, así mismo ofrece soporte preliminar de los basados en documentos.

2.5. Schema Design for NoSQL Applications

NoSE esta diseñado para ser utilizado en el desarrollo temprano de una aplicación, comienza con un modelo conceptual de los datos requeridos por una aplicación de destino, así como una descripción de cómo la aplicación usará los mismos. Luego recomienda un esquema de bases de datos extensible, es decir, un conjunto de definiciones de familia de columnas optimizadas para aplicación destino y pautas para desarrollar aplicaciones que usan este esquema.

Aunque el esquema de este tipo de bases de datos es flexible en el sentido de que la aplicación no necesita definir columnas específicas de antemano, aún es necesario decidir qué familias de columnas existirán en el almacén de datos y qué información codifica cada familia de columnas. Estas opciones son importantes porque el rendimiento de la aplicación depende en gran medida del esquema derivado.

Un esquema para una base de datos extensible consta de un conjunto de definiciones de familia de columnas. Cada familia de columnas tiene un nombre como identificador y su definición incluye los dominios de las llaves de partición, las llaves de agrupación y los valores de columna utilizados en esa familia de columnas. Dado un modelo conceptual (opcionalmente con estadísticas que describen la distribución de datos), una carga de trabajo de la aplicación (Workload) y una restricción de espacio opcional, el problema del diseño del esquema es recomendar un esquema tal que:

1. cada consulta en la carga de trabajo responda usando una o más solicitudes de obtención para agrupar familias en el esquema
2. se minimiza el costo total ponderado de responder las consultas, y opcionalmente

-
3. El tamaño agregado de las familias de columnas recomendadas está dentro de una restricción de espacio dada. Resolver este problema de optimización es el objetivo de nuestro asesor de esquemas.

Dando el modelo conceptual de una aplicación y un Workload, recomienda un plan específico para obtener una respuesta a la solicitud utilizando el siguiente esquema:

1. Enumeración de candidatos. Genera un conjunto de familias de columnas candidatas, en función de la carga de trabajo. Al inspeccionar la carga de trabajo, el asesor genera solo candidatos que pueden ser útiles para responder las consultas en la carga de trabajo.
2. Planificación de consultas. Genera un espacio de posibles planes de implementación para cada consulta. Estos planes hacen uso de las familias de columnas candidatas producidas en el primer paso.
3. Optimización del esquema. Genera un programa entero binario (BIP por sus siglas en ingles binary integer program) a partir de los candidatos y los espacios del plan. Luego, el BIP se entrega a un solucionador estándar que elige un conjunto de familias de columnas que minimiza el costo de responder las consultas.
4. Recomendación del plan. Elija un plan único del espacio del plan de cada consulta para ser el recomendado plan de implementación para esa consulta en función de las familias de columnas seleccionadas por el optimizador.

2.6. Conclusiones

Herramienta	Creador	Objetivo	Licencia	Sistema operativo	Publicación	Modelo			Modelado		
						Conceptual	Lógico	Físico	Transformación entre niveles de abstracción	Lenguaje	Metodología
KDM	Andrii Kashliev.	Comercial	Software privativo	Web based GUI	2018	E-R	Columnas	Cassandra DB	Reglas de mapeo. Flujo de trabajo de la aplicación. Patrones de mapeo	Diagramas a Chebotko a nivel logico	model-driven
Hackolade	IntegrIT SA / NV,	Comercial	Software privativo	Windows, Mac, Linux	2016	E-R	Multi-paradigma	Multi-plataforma	reglas de mapeo proceso de normalización y desnormalización	No definido	model-driven

Continuación de Tabla 3.2

Continuación de tabla		Oracle Corporation	En desarrollo	Software privativo	Windows, Mac	2019	E-R	llave-valor	Dynamo-DB	Reglas de mapeo	No definido	query-driven
NoSQL Workbench for Amazon DynamoDB		Oracle Corporation										
Mortadelo : A Model-Driven Framework for NoSQL Database Design		Alfonso de Vega, Diego García-Saiz, Carlos Blanco, Marta Zorrilla, and Pablo Sánchez	Investigación	implementación de software bajo licencia libre	-	2019	GDM	Columnas, documento	Cassandra, MongoDB	Reglas de mapeo	UML	model-driven
NoSE (Schema Design for NoSQL Applications)		Michael J. Mior, Kenneth Salem, Ashraf Aboulhagga and Rui Liu	Investigación	implementación de software bajo licencia libre	-	2017	E-R	Columnas	Cassandra	Carga de trabajo. Limitación de espacio	N/A	query-driven
Fin de Tabla												

Capítulo 3

Marco Teórico

3.1. Modelos de datos para bases de datos

De acuerdo a la bibliografía de Elmasri [6], un modelo de datos es una colección de conceptos que describen una estructura de una base de datos.

Los modelos de datos de alto nivel o conceptuales ofrecen conceptos visuales simples que representan un modelo de datos, mientras que los modelos de datos de bajo nivel o físicos ofrecen conceptos que describen los detalles de cómo se implementa el almacenamiento de los datos en el sistema de la base de datos.

Los modelos de datos conceptuales utilizan conceptos como entidades, atributos y relaciones o, en el caso de ser bases de datos no relacionales, no hay un estándar definido para modelar conceptualmente este tipo de bases de datos.

Una entidad representa un objeto o concepto del mundo real, un atributo representa alguna propiedad que describe a una entidad y una relación es una asociación entre entidades.

A continuación se presentan varios de los modelos de datos existentes.

3.1.1. Modelo entidad-relación

De acuerdo a Elmasri [6], el modelo entidad-relación, que fue creado y formalizado por Peter Chen en 1976[7], se utiliza con frecuencia para el diseño conceptual de las aplicaciones de base de datos.

En esta sección se describen los conceptos básicos y las restricciones del modelo ER.

3.1.1.1. Entidades

El objeto básico representado por el modelo ER es una entidad, que es una cosa del mundo real con una existencia independiente.

Una entidad puede ser un objeto con una existencia física (por ejemplo, una persona en particular, un coche, una casa o un empleado) o puede ser un objeto

con una existencia conceptual (por ejemplo, una empresa, un trabajo o un curso universitario).

Tipo de entidad

Un tipo de entidad define una colección (o conjunto) de entidades que tienen los mismos atributos.

La colección de todas las entidades de un tipo de entidad en particular de la base de datos se denomina conjunto de entidades; se usa el mismo nombre del tipo de entidad para hacer referencia al conjunto de entidades.

Tipos de entidades débiles

Los tipos de entidad que no tienen atributos clave propios se denominan tipos de entidad débiles. En contraposición, los tipos de entidad regulares que tienen un atributo clave se denominan tipos de entidad fuertes.

Las entidades que pertenecen a un tipo de entidad débil, se identifican como relacionadas con entidades específicas de otro tipo de entidad (llamada entidad identificado o propietario) en combinación con uno de sus valores de atributo.

Este tipo de relación que relaciona un tipo de entidad débil con su propietario lo podemos llamar relación identificativa del tipo de entidad débil.

Un tipo de entidad débil siempre tiene una restricción de participación total (dependencia de existencia) respecto a su relación identificativa, porque una entidad débil no puede identificarse sin una entidad propietaria. No obstante, no toda dependencia de existencia produce un tipo de entidad débil.

Un tipo de entidad débil normalmente tiene una clave parcial, que es el conjunto de atributos que pueden identificar sin lugar a dudas las entidades débiles que están relacionadas con la misma entidad propietaria.

En los diagramas ER, tanto el tipo de la entidad débil como la relación identificativa, se distinguen rodeando sus cuadros y rombos mediante unas líneas dobles.

El atributo de clave parcial aparece subrayado con una línea discontinua o punteada. Los tipos de entidades débiles se puede representar a veces como atributos complejos (compuestos, multivalor).

En general, se puede definir cualquier cantidad de niveles de tipos de entidad débil; un tipo de entidad propietaria puede ser ella misma un tipo de entidad débil.

Además, un tipo de entidad débil puede tener más de un tipo de entidad identificativa y un tipo de relación identificativa de grado superior a dos.

3.1.1.2. Atributos

Cada entidad tiene atributos, que son propiedades particulares que la describen y en el modelo ER hay varios tipos: simple frente a compuesto, monovalor frente a multivalor, almacenado frente a derivado y nulo.

Atributos compuestos frente a atributos simples Los atributos compuestos se pueden dividir en subpartes más pequeñas que representan atributos más básicos con significados independientes.

Los atributos que no son divisibles se denominan atributos simples o atómicos, mientras que los atributos compuestos pueden formar una jerarquía.

El valor de un atributo compuesto es la concatenación de los valores de sus atributos simples.

Atributos monovalor y multivalor La mayoría de los atributos tienen un solo valor para una entidad en particular; dichos atributos reciben el nombre de monovalor o de un solo valor.

En algunos casos, un atributo puede tener un conjunto de valores para la misma entidad y se denominan multivalor.

Un atributo multivalor puede tener límites superior e inferior para restringir el número de valores permitidos para cada entidad individual.

Atributos almacenados y derivados El atributo derivado puede calcularse u obtenerse a partir de otro atributo, que se denomina almacenado.

Atributos complejos Los atributos complejos son los atributos compuestos y multivalor que se anidan arbitrariamente.

Podemos representar el anidamiento arbitrario agrupando componentes de un atributo compuesto entre paréntesis () separando los componentes con comas, y mostrando los atributos multivalor entre llaves {}.

Por ejemplo, si una persona puede tener más de una residencia y cada residencia puede tener una sola dirección y varios teléfonos, el atributo TlfDir de una persona se puede especificar Como

```
{TlfDir({Tlf(CodÁrea,NumTlf)},
Dir(DirCalle(Número,Calle,NumApto),
Ciudad,Provincia,CP))}
```

Los atributos Tlf y Dir son compuestos.

Atributos clave de un tipo de entidad Una restricción importante de las entidades de un tipo de entidad es la clave o restricción de unicidad de los atributos.

Un tipo de entidad normalmente tiene un atributo cuyos valores son distintos para cada entidad del conjunto de entidades.

Los valores de un atributo en que se pueden utilizar para identificar cada entidad inequívocamente se denomina atributo clave.

En la notación diagramática ER cada atributo clave tiene su nombre subrayado dentro del óvalo y algunos tipos de entidad tienen más de un atributo clave.

Un tipo de entidad que carece de clave se le denomina tipo de entidad débil (que se explicará más adelante).

Conjuntos de valores (dominios) de atributos Cada atributo simple de un tipo de entidad está asociado con un conjunto de valor (o dominio de valores) que especifica el conjunto de los valores que se pueden asignar a ese atributo por cada entidad individual.

Los conjuntos de valores no se muestran en los diagramas ER; normalmente se especifican mediante los tipos de datos básicos disponibles en la mayoría de los lenguajes de programación, como entero, cadena, booleano, flotante, tipo enumerado, subrango, etcétera.

También se emplean otros tipos de datos adicionales para representar la fecha, la hora y otros.

Atributos de los tipos de relación Los tipos de relación también pueden tener atributos; los atributos de los tipos de relación 1:1 o 1:N se pueden trasladar a uno de los tipos de entidad participantes.

En el caso de un tipo de relación 1:N, un atributo de relación solo se puede migrar al tipo de entidad que se encuentra en el lado N de la relación.

Para los tipos de relación M:N, algunos atributos pueden determinarse mediante la combinación de entidades participantes en una instancia de relación, no mediante una sola relación. Dichos atributos deben especificarse como atributos de relación.

3.1.1.3. Relaciones

Un tipo de relación R entre n tipos de entidades E_1, E_2, \dots, E_n define un conjunto de asociaciones (o un conjunto de relaciones) entre las entidades de esos tipos de entidades.

Como en el caso de los tipos de entidades y los conjuntos de entidades, normalmente se hace referencia a un tipo de relación y su correspondiente conjunto de relaciones con el mismo nombre R .

Matemáticamente, el conjunto de relaciones R es un conjunto de instancias de relación r_i , donde cada r_i asocia n entidades individuales (e_1, e_2, \dots, e_n) y cada entidad e_j de r_i es un miembro del tipo de entidad E_j , $1 \leq j \leq n$. Por tanto, un tipo de relación es una relación matemática en E_1, E_2, \dots, E_n .

De forma alternativa, se puede definir como un subconjunto del producto cartesiano $E_1 \times E_2 \times \dots \times E_n$. Se dice que cada uno de los tipos de entidad E_1, E_2, \dots, E_n participa en el tipo de relación R ; de forma parecida, cada una de las entidades individuales e_1, e_2, \dots, e_n se dice que participa en la instancia de relación $r_i = (e_1, e_2, \dots, e_n)$.

En los diagramas ER, los tipos de relaciones se muestran mediante rombos, conectados a su vez mediante líneas a los rectángulos que representan los tipos de entidad participantes y el nombre de la relación se muestra dentro del rombo.

Grado de relación, nombres de rol y relaciones recursivas

Grado de un tipo de relación El grado de un tipo de relación es el número de tipos de entidades participantes. Un tipo de relación de grado dos se denomina binario y uno de grado tres ternario. Las relaciones pueden ser generalmente de cualquier grado, pero las más comunes son las relaciones binarias.

Nombres de rol y relaciones recursivas Cada tipo de entidad que participa en un tipo de relación juega un papel o rol particular en la relación.

El nombre de rol hace referencia al papel que una entidad participante del tipo de entidad juega en cada instancia de relación y ayuda a explicar el significado de la relación.

Los nombres de rol no son técnicamente necesarios en los tipos de relación donde todos los tipos de entidad participantes son distintos, puesto que cada nombre de tipo de entidad participante se puede utilizar como participación.

Cuando un tipo de entidad se relaciona consigo misma, se tiene una relación recursiva y es necesario indicar los roles que juegan los miembros en la relación.

Restricciones en los tipos de relaciones Los tipos de relaciones normalmente tienen ciertas restricciones que limitan las posibles combinaciones entre las entidades que pueden participar en el conjunto de relaciones correspondiente.

Estas restricciones están determinadas por la situación del minimundo representado por las relaciones.

Podemos distinguir dos tipos principales de restricciones de relación: razón de cardinalidad y participación.

Razones de cardinalidad para las relaciones binarias La razón de cardinalidad de una relación binaria especifica el número máximo de instancias de relación en las que una entidad puede participar.

Las posibles razones de cardinalidad para los tipos de relación binaria son 1:1, 1:N, N:1 y M:N.

1. uno a uno: una relación R de X a Y es uno a uno si cada entidad en X se asocia con cuando mucho una entidad en Y e, inversamente, cada entidad en Y se asocia con cuando mucho una entidad en X .
2. uno a muchos: una relación R de X a Y es uno a muchos si cada entidad en X se puede asociar con muchas entidades en Y , pero cada entidad en Y se asocia con cuando mucho una entidad en X .
3. muchos a uno: una relación R de X a Y es muchos a uno si cada entidad en X se asocia con cuando mucho una entidad en Y , pero cada entidad en Y se puede asociar con muchas entidades en X .

4. muchos a muchos: una relación R de X a Y es muchos a muchos si cada entidad en X se puede asociar con muchas entidades en Y y cada entidad en Y se puede asociar con muchas entidades en X .

Restricciones de participación y dependencias de existencia La restricción de participación especifica si la existencia de una entidad depende de si está relacionada con otra entidad a través de un tipo de relación.

Esta restricción especifica el número mínimo de instancias de relación en las que puede participar cada entidad y en ocasiones recibe el nombre de restricción de cardinalidad mínima.


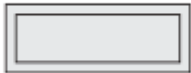




Hay dos tipos de restricciones de participación, total y parcial; la participación total también se conoce como dependencia de existencia.


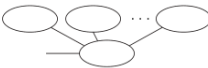




1. Participación total: si todo miembro de un conjunto de entidades debe participar en una relación, es una participación total del conjunto de entidades en la relación. Esto se denota al dibujar una línea doble desde el rectángulo de entidades hasta el rombo de relación.
2. Participación parcial: una línea sencilla indica que algunos miembros del conjunto de entidades no deben participar en la relación.

Nos referiremos a la razón de cardinalidad y a las restricciones de participación, en conjunto, como restricciones estructurales de un tipo de relación.

3.1.1.4. Resumen de la notación para los diagramas ER

Tabla 3.1: Notación para los diagramas ER

Notación para los diagramas ER	
Símbolo	Significado
	Entidad
	Entidad débil
	Relación
	Relación de identificación
	Atributo
	Atributo clave

Continuación de Tabla 3.2	
Continuación la notación para los diagramas ER	
	Atributo multivalor
	Atributo compuesto
	Atributo derivado
	Participación total
	Razón de cardinalidad
	Restricción estructural
Fin de Tabla	

3.1.2. El modelo entidad-relación extendido

De acuerdo a la bibliografía de Catherine[8], el modelo entidad-relación extendido (EE-R) extiende el modelo ER para permitir la inclusión de varios tipos de abstracción, y para expresar restricciones más claramente. A los diagramas ER estándar se agregan símbolos adicionales para crear diagramas EE-R que expresen estos conceptos.

3.1.2.1. Especialización

Con frecuencia, un conjunto de entidades contiene uno o más subconjuntos que tienen atributos especiales o que participan en relaciones que otros miembros del mismo conjunto de entidades no tiene.

El método de identificar subconjuntos de conjuntos de entidades existentes, llamado especialización, corresponde a la noción de herencia de subclase y clase en el diseño orientado a objetos, donde se representa mediante jerarquías de clase.

El círculo que conecta a la superclase con las subclases se llama círculo de especialización. Cada subclase se conecta al círculo mediante una línea que tiene un símbolo de herencia, un símbolo de subconjunto o copa, con el lado abierto de frente a la superclase. Las subclases heredan los atributos de la superclase y opcionalmente pueden tener atributos locales distintos.

Dado que cada miembro de una subclase es miembro de la superclase, al círculo de especialización a veces se le conoce como relación isa.

En ocasiones una entidad tiene solo un subconjunto con propiedades o relaciones especiales de las que quiere tener información. Solo contiene una subclase para una especialización. En este caso, en el diagrama EE-R se omite el círculo y simplemente se muestra la subclase conectada mediante una línea de subconjunto a la superclase.

Las subclases también pueden participar en relaciones locales que no se apliquen a la superclase o a otras subclases en la misma jerarquía.

3.1.2.2. Generalización

Además de la especialización, también se pueden crear jerarquías de clase al reconocer que dos o más clases tienen propiedades comunes e identificar una superclase común para ellas, un proceso llamado generalización. Estos dos procesos son inversos uno de otro, pero ambos resultan en el mismo tipo de diagrama jerárquico.

3.1.2.3. Restricciones

Las subclases pueden ser traslapantes (*overlapping*), lo que significa que la misma instancia de entidad puede pertenecer a más de una de las subclases, o desarticuladas (*disjoint*), lo que significa que no tienen miembros en común. A esto se le refiere como restricción de desarticulación (*disjointness*) y se expresa al colocar una letra adecuada, *d* u *o*, en el círculo de especialización. Una *d* indica subclases de desarticulación y una *o* indica subclases de traslapamiento.

Una especialización también tiene una restricción de completud (*completeness*), que muestra si todo miembro del conjunto de entidades debe participar en ella.

Si todo miembro de la superclase debe pertenecer a alguna subclase, se tiene una especialización total. Si a algunos miembros de la superclase no se les puede permitir pertenecer a alguna subclase, la especialización es parcial.

En algunas jerarquías de especialización es posible identificar la subclase a la que pertenece una entidad al examinar una condición o predicado específico para cada subclase, es decir, es una especialización definida por predicado, pues la membresía a la subclase está determinada por un predicado.

Algunas especializaciones definidas por predicado usan el valor del mismo atributo en el predicado definatorio para todas las subclases. Estas se llaman especializaciones definidas por atributo.

Las especializaciones que no están definidas por predicado se dice que son definidas por el usuario, pues el usuario es el responsable de colocar la instancia de entidad en la subclase correcta.

3.1.2.4. Jerarquías múltiples y herencia

Cuando el mismo conjunto de entidades puede ser una subclase de dos o más superclases, se dice que tal clase es una subclase compartida y tiene herencia múltiple de sus superclases.

3.1.2.5. Unión

Mientras que una subclase compartida representa un miembro de todas sus superclases y hereda atributos de todas ellas, una subclase se puede relacionar con la de una colección, llamada unión o categoría de superclases, en vez de pertenecer a todas ellas. En este caso, una instancia de la subclase hereda solo los atributos de una de las superclases, dependiendo de a cuál miembro de la unión pertenece.

Las categorías pueden ser parciales o totales, dependiendo de si cada miembro de los conjuntos que constituyen la unión participan en ella.

3.1.3. Modelo relacional

De acuerdo a la bibliografía de Elmasri[6], el modelo relacional introducido por Ted Codd en 1970[1] utiliza el concepto de una relación matemática como bloque de construcción básico y tiene su base teórica en la teoría de conjuntos y la lógica del predicado.

La lógica de predicado, utilizada ampliamente en matemáticas, proporciona un marco en el que una afirmación (declaración de hecho) puede verificarse como verdadera o falsa.

La teoría de conjuntos es una ciencia matemática que trata con conjuntos o grupos de cosas y se utiliza como base para la manipulación de datos en el modelo relacional.

El modelo relacional representa la base de datos como una colección de relaciones. Cuando una relación está pensada como una tabla de valores, cada fila representa una colección de valores relacionados.

Asimismo, cada fila de la tabla representa un hecho que, por lo general, corresponde con una relación o entidad real. El nombre de la tabla y de las columnas se utiliza para ayudar a interpretar el significado de cada uno de los valores de las filas.

En terminología formal, una fila recibe el nombre de tupla, una cabecera de columna es un atributo y el nombre de la tabla una relación. El tipo de dato que describe los valores que pueden aparecer en cada columna está representado por un dominio de posibles valores.

Basado en estos conceptos, el modelo relacional tiene tres componentes bien definidos:

1. Una estructura de datos lógica representada por relaciones.
2. Un conjunto de reglas de integridad para garantizar que los datos sean consistentes.
3. Un conjunto de operaciones que define cómo se manipulan los datos.

3.1.3.1. Relaciones

Estructuras de datos relacionales Su usan tablas con relación entre ellas.

Tablas En este modelo, las tablas se usan para contener información acerca de los objetos a representar en la base de datos. Al usar los términos del modelo Entidad-Relación, los conjuntos de entidades y de relaciones se muestran usando tablas.

Una relación o esquema de relación se representa como una tabla bidimensional en la que las filas de la tabla corresponden a registros individuales y las columnas corresponden a atributos.

Formalmente, un esquema de relación R , denotado por $R(A_1, A_2, \dots, A_n)$ está constituido por un nombre de relación R y una lista de atributos A_1, A_2, \dots, A_n .

Cada atributo A_i es el nombre de un papel jugado por algún dominio D en el esquema de relación R . Se dice que D es el dominio de A_i y se especifica como $dom(A_i)$.

Un esquema de relación se utiliza para describir una relación; se dice que R es el nombre de la misma. El grado de una relación es el número de atributos n de la misma.

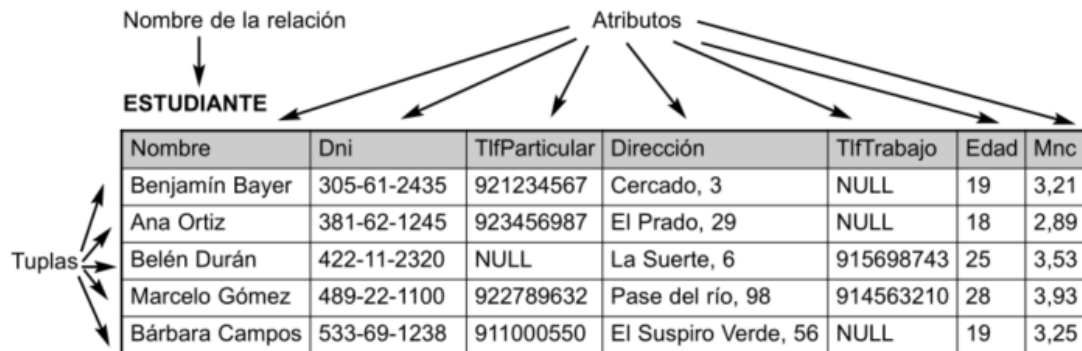


Figura 3.1: Tabla en el modelo Relacional

Cada fila de la tabla corresponde a un registro individual o instancia de entidad. En el modelo relacional cada fila se llama tupla y la tabla que representa una relación tiene las siguientes características:

- Cada celda de la tabla contiene solo un valor.
- Cada columna tiene un nombre distinto, que es el nombre del atributo que representa.
- Todos los valores en una columna provienen del mismo dominio, pues todos son valores del atributo correspondiente.
- Cada tupla o fila es distinta; no hay tuplas duplicadas.
- El orden de las tuplas o filas es irrelevante.

Relaciones y tablas de bases de datos Una relación (o estado de relación) r del esquema $R(A_1, A_2, \dots, A_n)$, también especificado como $r(R)$, es un conjunto de n -tuplas $r = t_1, t_2, \dots, t_m$.

Cada tupla t es una lista ordenada de n valores $t = \langle v_1, v_2, \dots, v_n \rangle$, donde v_i , $1 \leq i \leq n$, es un elemento de $dom(A_i)$ o un valor especial NULL.

El i -ésimo valor de la tupla t , que se corresponde con el atributo A_i , se referencia como $t[A_i]$ o $t[i]$ si utilizamos una notación posicional.

3.1.3.2. Claves

En el modelo relacional, las claves son importantes porque aseguran que cada fila en una tabla sea unívocamente identificable.

También son usadas para establecer relaciones entre tablas y asegurar la integridad de los datos.

Una clave es un atributo o grupo de atributos que pueden identificar los valores de otros atributos.

Clave compuesta Una clave compuesta es una clave que se compone de más de un atributo. Un atributo que forma parte de una clave se denomina atributo clave.

Superclave Un atributo o atributos que identifican de manera única cualquier fila de una tabla

3.1.3.3. Restricciones de integridad

Integridad de dominio La integridad de dominio es la validez de las restricciones que debe cumplir una determinada columna de la tabla.

Integridad de entidad Todas las claves principales son únicas, y ninguna clave primaria puede ser nula.

Integridad referencial Una clave externa puede ser nula siempre que no sea parte de la clave principal de su tabla, o puede tener el valor que coincida con el valor de la clave primaria en una tabla con la que está relacionada (cada valor de clave externa no nula debe hacer referencia a un valor de clave primaria existente).

3.1.3.4. Propiedades de las relaciones

Grado El número de columnas en una tabla se llama grado de la relación. Una relación con una sola columna es de grado uno y se llama relación unaria. Una relación con dos columnas se llama binaria, una con tres columnas se llama ternaria y, después de ella, por lo general se usa el término n-aria. El grado de una relación es parte de la intensión de la relación y nunca cambia.

Cardinalidad La cardinalidad de una relación es el número de entidades a las que otra entidad puede mapear bajo dicha relación.

3.1.3.5. ACID

El modelo relacional en las transacciones cumple con las propiedades de ACID, que es el acrónimo de *Atomicity* (atomicidad), *Consistency* (consistencia), *Isolation* (aislamiento) y *Durability* (durabilidad).

Atomicidad Requiere que se completen todas las operaciones (solicitudes SQL) de una transacción; si no, la transacción se cancela.

Si una transacción T_1 tiene cuatro solicitudes SQL, las cuatro solicitudes deben completarse con éxito; de lo contrario, se anula toda la transacción.

En otras palabras, una transacción se trata como una unidad de trabajo única, indivisible y lógica.

Consistencia Indica la permanencia del estado consistente de la base de datos. Una transacción lleva una base de datos de un estado consistente a otro.

Cuando se completa una transacción, la base de datos debe estar en un estado coherente. Si alguna de las partes de la transacción viola una restricción de integridad, se anula la transacción completa.

Aislamiento Significa que los datos utilizados durante la ejecución de una transacción no pueden ser utilizados por una segunda transacción hasta que se complete la primera.

En otras palabras, si la transacción T_1 se está ejecutando y está utilizando el elemento de datos X , ninguna otra transacción puede acceder a ese elemento de datos ($T_2...T_n$) hasta que finalice T_1 .

Esta propiedad es particularmente útil en entornos de bases de datos multiusuario porque varios usuarios pueden acceder y actualizar la base de datos al mismo tiempo.

Durabilidad Garantiza que una vez que se realizan y confirman los cambios en la transacción, no se pueden deshacer ni perder, incluso en el caso de una falla del sistema.

3.1.3.6. Structured Query Language

Structured Query Language o SQL está basado en el álgebra relacional, en el cálculo relacional y es un lenguaje de manipulación de datos, un lenguaje de definición de datos, un lenguaje de control de transacciones y un lenguaje de control de datos.

Lenguaje de manipulación de datos (DML) Un *Data Manipulation Language* o DML incluye comandos para insertar, actualizar, eliminar y recuperar datos dentro de las tablas de la base de datos.

Lenguaje de definición de datos (DDL) Un *Data Definition Language* o DDL incluye comandos para crear objetos de base de datos como tablas, índices y vistas, así como comandos para definir accesos a objetos de la base de datos.

Lenguaje de control de transacciones (TCL) Los comandos de un *Transaction Control Language* o TCL se ejecutan dentro del contexto de una transacción, que es una unidad lógica de trabajo compuesta por una o más instrucciones SQL.

SQL proporciona comandos para controlar el procesamiento de estas transacciones atómicas.

Lenguaje de control de datos (DCL) Los comandos de un *Data Control Language* o DCL se utilizan para controlar el acceso a los objetos de datos, como otorgar a un usuario permiso para ver solo una tabla y otorgar a otro usuario permiso para cambiar los datos de la misma tabla.

Tabla 3.2: Comandos de SQL.

Comandos de manipulación de datos	
Comando	Descripción
SELECT	Selecciona atributos de filas en una o más tablas o vistas
FROM	Especifica las tablas de las que se deben recuperar los datos
WHERE	Restringe la selección de filas en función de una expresión condicional
GROUP BY	Agrupar las filas seleccionadas en función de uno o más atributos
HAVING	Restringe la selección de filas agrupadas en función de una condición
ORDER BY	Ordena las filas seleccionadas en función de uno o más atributos
INSERT	Inserta filas en una tabla
UPDATE	Modifica los valores de un atributo en una o más filas de la tabla
DELETE	Elimina una o más filas de una tabla
Operadores de comparación =, <, >, ≤, ≥, <>, !=	Usados en expresiones condicionales
Operadores lógicos AND, OR, NOT	Usados en expresiones condicionales
Operadores especiales BETWEEN	Comprueba si un valor de atributo está dentro de un rango
IN	Comprueba si un valor de atributo coincide con algún valor dentro de una lista de valores
LIKE	Comprueba si un valor de atributo coincide con un patrón de cadena dado
IS NULL	Comprueba si un valor de atributo es nulo

Continuación de Tabla [3.2](#)

Continuación de comandos SQL

EXIST	Comprueba si una subconsulta devuelve alguna fila
DISTINCT	Limita que los valores sean únicos
Comandos de definición de datos	
CREATE SCHEMA	Crea el esquema de la base de datos
CREATE TABLE	Crea una nueva tabla en el esquema de la base de datos del usuario
NOT NULL	Asegura que una columna no tendrá valores nulos
UNIQUE	Asegura que una columna no tendrá valores duplicados
PRIMARY KEY	Define una clave primaria para una tabla
FOREIGN KEY	Define una clave externa para una tabla
DEFAULT	Define un valor predeterminado para una columna (cuando no se proporciona ningún valor)
CHECK	Valida datos en un atributo
CREATE INDEX	Crea un índice para una tabla
CREATE VIEW	Crea un subconjunto dinámico de filas y columnas a partir de una o más tablas
ALTER TABLE	Modifica la definición de una tabla (agrega, modifica o elimina atributos o restricciones)
CREATE TABLE AS	Crea una nueva tabla basada en una consulta en el esquema de la base de datos del usuario
DROP TABLE	Elimina permanentemente una tabla (y sus datos)
DROP INDEX	Elimina permanentemente un índice
DROP VIEW	Elimina permanentemente una vista
Lenguaje de control de transacciones	
COMMIT	Guarda de manera permanente los cambios en los datos
ROLLBACK	Restaura los datos a sus valores anteriores
Lenguaje de control de datos	
GRANT	Le da un usuario permiso de hacer una acción de sistema o de acceder a datos de un objeto
REVOKE	Le quita el privilegio a un usuario de hacer algunas operaciones

Fin de Tabla

3.1.4. Modelos NoSQL

De acuerdo a la bibliografía de Catherine [9], el término NoSQL significa *not only SQL* y se usa para agrupar sistemas de bases de datos diferentes a los relacionales.

Por los nuevos requerimientos en la época actual como disponibilidad total, tolerancia a fallos, almacenamiento de penta bytes de información distribuida en miles de servidores, la necesidad de nodos con escalabilidad horizontal, entre otros, surge la necesidad de sistemas de bases de datos no relacionales.

Estos tipos de sistemas no requieren esquemas fijos, son fáciles y rápidos en la instalación, usan lenguajes no declarativos, ofrecen alto rendimiento y disponibilidad, evitan operaciones de juntas, soportan paralelismo y escalan principalmente en forma horizontal soportando estructuras distribuidas que no necesariamente cumplen las propiedades ACID[9], sino que se enfocan en el modelo de consistencia de datos BASE.

3.1.4.1. Teorema CAP

En el Simposio de Principios de Computación Distribuida (PODC, en inglés) en el año 2000[10], el Dr. Eric Brewer declaró en su presentación que “en cualquier sistema de datos altamente distribuido hay tres propiedades comúnmente deseables: *Consistency* (consistencia), *Availability* (disponibilidad) y *Partition tolerance* (tolerancia al particionado). Sin embargo, es imposible que un sistema proporcione las tres propiedades al mismo tiempo”.

El acrónimo CAP representa las tres propiedades deseables:

Consistencia En una base de datos distribuida, la consistencia tiene el papel más importante. Todos los nodos deben ver los mismos datos al mismo tiempo, lo que significa que las réplicas deben actualizarse inmediatamente. Sin embargo, esto implica lidiar con la latencia y los atrasos de la red.

No hay que confundir la consistencia en la gestión de transacciones con la consistencia del teorema CAP. La consistencia de la gestión de transacciones se refiere al resultado cuando la ejecución de una transacción en una base de datos cumple con todas las restricciones de integridad.

La consistencia en CAP se basa en la suposición de que todas las transacciones tienen lugar al mismo tiempo en todos los nodos, como si se estuvieran ejecutando en una base de datos de un solo nodo (todos los nodos ven los mismos datos al mismo tiempo).

Disponibilidad En términos simples, el sistema siempre cumple una solicitud. Ninguna solicitud recibida se pierde y este es un requisito fundamental para todas las organizaciones centradas en la web.

Tolerancia al particionado El sistema continúa funcionando incluso en caso de falla de un nodo. Esto es equivalente a la transparencia de fallas en bases de datos distribuidas. El sistema fallará solo si fallan todos los nodos.

Aunque el teorema CAP se centra en sistemas basados en la web altamente distribuidos, sus implicaciones están muy extendidas para todos los sistemas distribuidos, incluidas las bases de datos.

En los sistemas de bases de datos, las propiedades ACID aseguran que todas las transacciones exitosas den como resultado un estado de base de datos consistente, uno en el que todas las operaciones de datos siempre devuelven los mismos resultados.

Para bases de datos distribuidas centralizadas y pequeñas, la latencia no es un problema, pero para una base de datos altamente distribuida el garantizar transacciones ACID sin pagar un alto precio en latencia de red o en conflictos de datos.

La relación entre consistencia y disponibilidad ha generado un nuevo tipo de sistemas de datos distribuidos, diferente al ACID, denominados BASE, *Basically Available* (básicamente disponibles), *Soft state* (estado suave), *Eventually consistent* (eventualmente consistente).

BASE BASE se refiere a un modelo de consistencia de datos en el que los cambios de datos no son inmediatos, sino que se propagan lentamente a través del sistema hasta que todas las réplicas sean consistentes.

En la práctica, la aparición de bases de datos distribuidas NoSQL proporciona un espectro de consistencia que va desde lo altamente consistente (ACID) hasta lo eventualmente consistente (BASE).

3.1.4.2. Clave-Valor

De acuerdo a la bibliografía de Coronel[11] Una base de datos de clave-valor, o almacén de clave-valor, es un paradigma de almacenamiento de datos diseñado para almacenar, recuperar y administrar arreglos asociativos.

Comúnmente se usa un diccionario o tabla hash que contienen una colección de registros anidados, secuenciaa de bits, que se almacenan y se recuperan utilizando una clave que identifica de manera única el registro y se utiliza para encontrar rápidamente los datos dentro de la base de datos.

No obstante, es responsabilidad de las aplicaciones que hagan uso de este tipo de base de datos interpretar el significado de los datos. No hay claves foráneas y las relaciones no pueden rastrearse entre las claves. Esta simplificación en el RDBMS de clave-valor permite que sean rápidas y escalables.

Los pares de clave-valor generalmente se organizan en *buckets*. Todas las claves dentro un *bucket* deben ser únicas, pero pueden repetirse en otros *buckets* y todas las operaciones, incluidas las consultas, se basan en el *bucket* + la clave.

Respecto a las operaciones de este tipo de bases de datos, se usan las operaciones de *get*, *store* y *delete*. La operación *get* o *fetch* es usada para obtener el valor de un

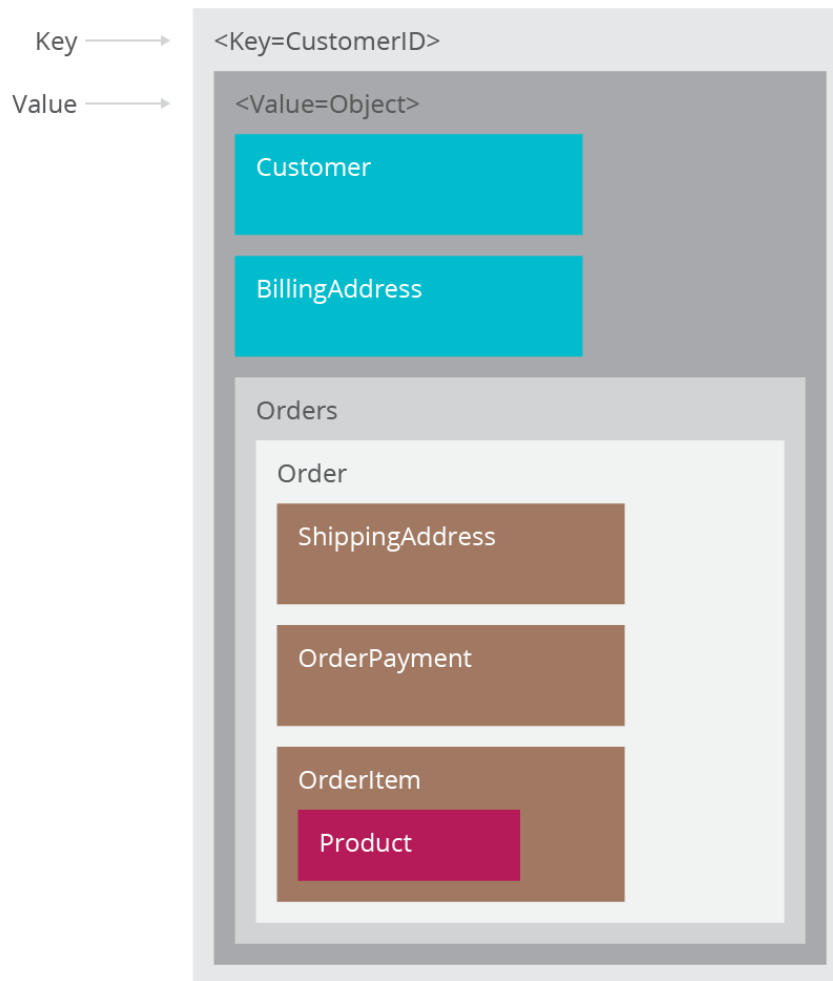


Figura 3.2: *Bucket* en el almacenamiento clave-valor

par. El operador de *store* almacena datos en una clave. Si la combinación de *bucket* + clave no existe, se añade como un nuevo par de clave-valor. En cambio, en caso de existir la combinación de *bucket* + clave, el valor es reemplazado por el nuevo. El operador de *delete* es para eliminar un par de clave-valor.

Algunas de las bases de datos de clave-valor populares son Riak, Redis, Memcached DB, Berkeley DB, HamsterDB, Amazon DynamoDB y Project Voldemort (una implementación de código abierto de Amazon DynamoDB)[12].

3.1.4.3. Orientado a documentos

Una base de datos de documentos es una base de datos NoSQL que almacena datos en documentos etiquetados en pares clave-valor.

A diferencia de una base de datos clave-valor donde el componente de valor puede contener cualquier tipo de datos, una base de datos de documentos siempre almacena un documento en el componente de valor. El documento puede estar en cualquier formato codificado, como XML, JSON o BSON.

Otra diferencia importante es que, si bien las bases de datos clave-valor no intentan comprender el contenido del componente de valor, las bases de datos de

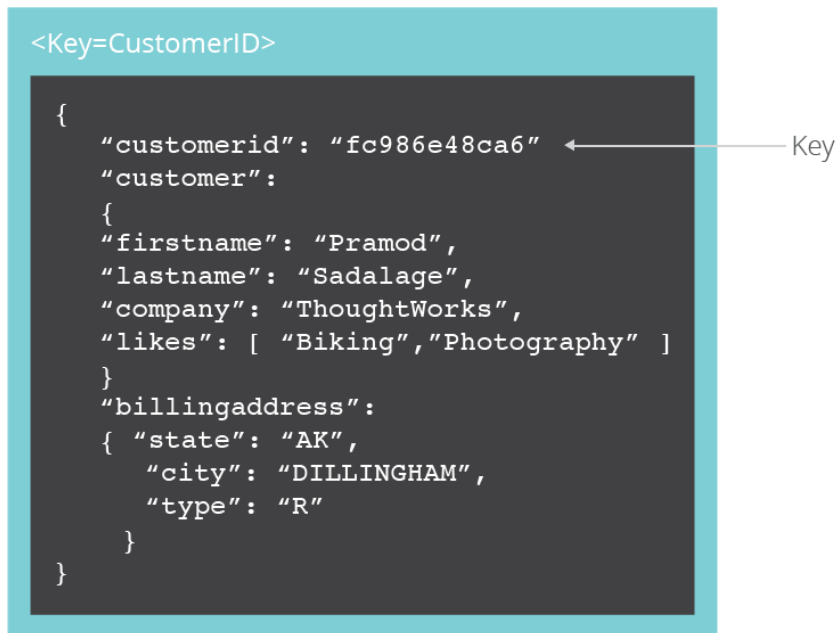


Figura 3.3: Documento en el almacenamiento de documentos

documentos sí lo hacen. Las etiquetas representan porciones de un documento.

Por ejemplo, un documento puede tener etiquetas para identificar qué texto en el documento representa el título, el autor y el cuerpo del documento.

Dentro del cuerpo del documento, puede haber etiquetas adicionales para indicar capítulos y secciones. A pesar del uso de etiquetas en los documentos, las bases de datos de documentos se consideran sin esquema, es decir, no imponen una estructura predefinida en los datos almacenados.

Para una base de datos de documentos, no tener esquemas significa que aunque todos los documentos tienen etiquetas, no todos los documentos deben tener las mismas etiquetas, por lo que cada documento puede tener su propia estructura.

Las etiquetas en una base de datos de documentos son extremadamente importantes porque son la base de la mayoría de las capacidades adicionales que tienen las bases de datos de documentos sobre las bases de datos clave-valor.

Las etiquetas dentro del documento son accesibles para el SGBD, lo que hace posible consultas complejas. Al igual que las bases de datos clave-valor agrupan pares clave-valor en grupos lógicos llamados *buckets*, las bases de datos de documentos agrupan documentos en grupos lógicos llamados colecciones.

Si bien se puede recuperar un documento especificando la colección y la clave, también es posible realizar consultas en función del contenido de las etiquetas.

Las bases de datos de documentos tienden a funcionar bajo el supuesto de que un documento es independiente, que no está en diferentes tablas como en una base de datos relacional.

Una base de datos de documentos asume que todos los datos relacionados de una orden estén en un solo documento.

Por lo tanto, cada orden en una colección contendría datos sobre el cliente, el

pedido en sí y los productos comprados en esa orden.

Las bases de datos de documentos no almacenan relaciones como se hace en el modelo relacional y generalmente no tienen soporte para operaciones como la unión.

3.1.4.4. Orientado a columnas

Este modelo de base de datos se originó con el BigTable de Google. Otras bases de datos orientados a columnas incluyen HBase, Hypertable y Cassandra.

Cassandra comenzó como un proyecto en Facebook, pero Facebook lo lanzó a la comunidad de código abierto, que ha seguido desarrollando a Cassandra en una de las bases de datos orientadas a columnas más populares.

Afortunadamente, las bases de datos de la familia de columnas son conceptualmente simples y conceptualmente lo suficientemente cercanas al modelo relacional para que su comprensión del modelo relacional pueda ayudarlo a comprender el modelo familiar de columnas.

El término base de datos orientada a columnas puede referirse a dos conjuntos diferentes de tecnologías que a menudo se confunden entre sí.

En cierto sentido, el término de base de datos orientada a columnas puede usarse para tecnologías de bases de datos relacionales tradicionales que usan almacenamiento enfocado en columnas en lugar de almacenamiento en filas.

Por otro lado, en un sistema de base de datos NoSQL describe un tipo de base de datos que organiza datos en pares nombre-valor donde el nombre actúa también como la clave.

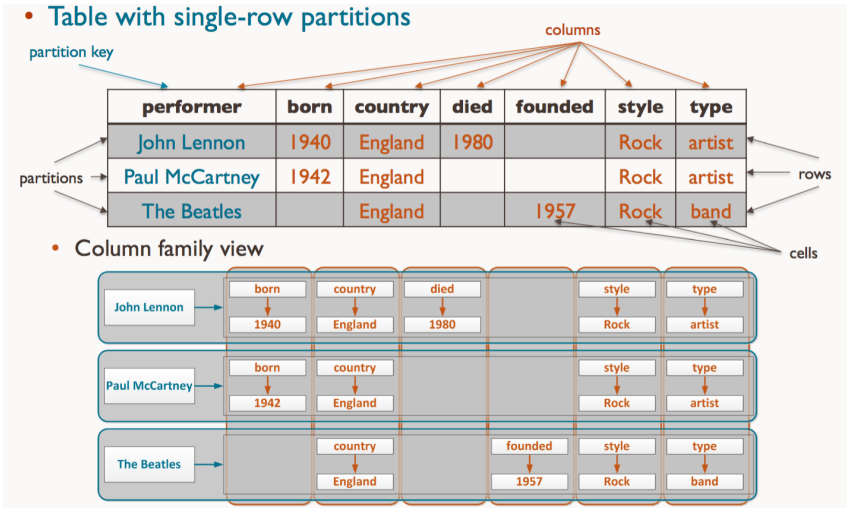


Figura 3.4: Familia de columnas

Cada par de clave-valor representa una columna y siempre contiene una fecha que sirve para resolver conflictos de escritura, datos expirados entre otras cosas.

Al ser una bases de datos NoSQL, no requiere que los datos se ajusten a estructuras predefinidas ni admite SQL para consultas.

3.1.4.5. Orientado a grafos

Una base de datos de grafos es una base de datos NoSQL basada en la teoría de grafos para almacenar datos sobre entornos ricos en relaciones.

La teoría de grafos es un campo matemático y de ciencias de la computación que modela relaciones, o aristas, entre objetos llamados nodos.

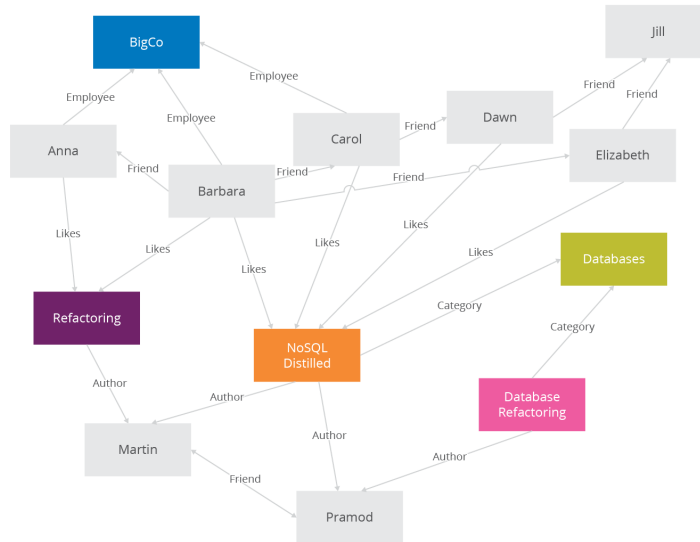


Figura 3.5: modelo conceptual orientado a grafos

Modelar y almacenar datos sobre relaciones es el enfoque de las bases de datos de grafos y la teoría en la que se basa es un campo de estudio bien establecido que se remonta a cientos de años.

Como resultado, la creación de un modelo de base de datos basado en la teoría de grafos proporciona de inmediato una fuente rica de algoritmos y aplicaciones que han ayudado a las bases de datos de grafos a ganar terreno.

Los componentes principales de las bases de datos de grafos son nodos, aristas y propiedades, como se muestra en la figura 3.6. Un nodo corresponde a la idea de una instancia de entidad relacional.

El nodo es una instancia específica de algo sobre lo que queremos mantener datos. Cada nodo (círculo) en la figura 3.6 representa un solo agente.

Las propiedades son como atributos; son los datos que necesitamos almacenar sobre el nodo. Todos los nodos de agente pueden tener propiedades como nombre y apellido, pero no todos los nodos deben tener las mismas propiedades.

Un borde es una relación entre nodos. Los bordes (mostrados como flechas en la figura 3.6) pueden estar en una dirección, o pueden ser bidireccionales.

Para las consultas, la terminología correcta sería atravesar el grafo. Las bases de datos de grafos los recorridos se enfocan en las relaciones entre nodos, como la ruta más corta y el grado de conexión.

La base de datos de grafos comparte algunas características con otras bases de datos NoSQL en que las bases de datos de grafos no obligan a los datos a ajustarse a estructuras predefinidas, no admiten SQL y están optimizados para proporcionar

velocidad de procesamiento, al menos para datos intensivos en relaciones.

Sin embargo, otras características clave no se aplican a las bases de datos de grafos. Las bases de datos de grafos no se escalan muy bien a los clústeres debido a las diferencias en los datos agregados.

3.1.4.6. Orientado a grafos

Una base de datos de grafos es una base de datos NoSQL basada en la teoría de grafos para almacenar datos sobre entornos ricos en relaciones.

La teoría de grafos es un campo matemático y de ciencias de la computación que modela relaciones, o aristas, entre objetos llamados nodos.

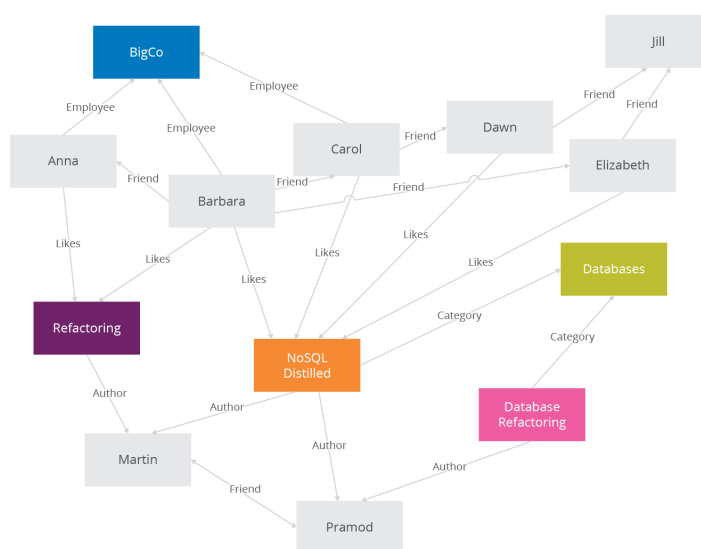


Figura 3.6: modelo conceptual orientado a grafos

Modelar y almacenar datos sobre relaciones es el enfoque de las bases de datos de grafos y la teoría en la que se basa es un campo de estudio bien establecido que se remonta a cientos de años.

Como resultado, la creación de un modelo de base de datos basado en la teoría de grafos proporciona de inmediato una fuente rica de algoritmos y aplicaciones que han ayudado a las bases de datos de grafos a ganar terreno.

Los componentes principales de las bases de datos de grafos son nodos, aristas y propiedades, como se muestra en la figura 3.6. Un nodo corresponde a la idea de una instancia de entidad relacional.

El nodo es una instancia específica de algo sobre lo que queremos mantener datos. Cada nodo (círculo) en la figura 3.6 representa un solo agente.

Las propiedades son como atributos; son los datos que necesitamos almacenar sobre el nodo. Todos los nodos de agente pueden tener propiedades como nombre y apellido, pero no todos los nodos deben tener las mismas propiedades.

Un borde es una relación entre nodos. Los bordes (mostrados como flechas en la figura 3.6) pueden estar en una dirección, o pueden ser bidireccionales.

Para las consultas, la terminología correcta sería atravesar el grafo. Las bases de datos de grafos los recorridos se enfocan en las relaciones entre nodos, como la ruta más corta y el grado de conexión.

La base de datos de grafos comparte algunas características con otras bases de datos NoSQL en que las bases de datos de grafos no obligan a los datos a ajustarse a estructuras predefinidas, no admiten SQL y están optimizados para proporcionar velocidad de procesamiento, al menos para datos intensivos en relaciones.

Sin embargo, otras características clave no se aplican a las bases de datos de grafos. Las bases de datos de grafos no se escalan muy bien a los clústeres debido a las diferencias en los datos agregados.

3.2. Tecnologías a usar

Para seleccionar las tecnologías a usar para el desarrollo de la aplicación web, se ha optado por hacer un estudio y comparación de tecnologías similares.

3.2.1. Hypertext Transfer Protocol

De acuerdo a la W3C[13], Hypertext Transfer Protocol, HTTP, o protocolo de transferencia de hipertexto, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de una serie de RFC, siendo el más importante de ellos el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente se le suele llamar “agente de usuario.” *user agent*, que realiza una petición enviando un mensaje con cierto formato al servidor. Al servidor se le suele llamar servidor web y envía un mensaje de respuesta.

Los mensajes HTTP son en texto plano lo que lo hace más legible y fácil de depurar. Esto tiene el inconveniente de hacer los mensajes más largos.

Los mensajes tienen la siguiente estructura:

1. Línea inicial (termina con retorno de carro y un salto de línea).
 - a) Para las peticiones: la acción requerida por el servidor (método de petición) seguido de la URL del recurso y la versión HTTP que soporta el cliente.

-
- b) Para respuestas: La versión del HTTP usado seguido del código de respuesta (que indica que ha pasado con la petición seguido de la URL del recurso) y de la frase asociada a dicho retorno.
 2. Las cabeceras del mensaje que terminan con una línea en blanco. Son metadatos. Estas cabeceras le dan gran flexibilidad al protocolo.
 3. Cuerpo del mensaje: es opcional. Su presencia depende de la línea anterior del mensaje y del tipo de recurso al que hace referencia la URL. Típicamente tiene los datos que se intercambian cliente y servidor. Por ejemplo para una petición podría contener ciertos datos que se quieren enviar al servidor para que los procese. Para una respuesta podría incluir los datos que el cliente ha solicitado.

HTTP define una serie predefinida de métodos de petición que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se ha ido aumentando según se avanzaba en las versiones.

Cada método indica la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa depende de la aplicación del servidor. Por ejemplo, el recurso puede corresponderse con un archivo que reside en el servidor.

1. GET: el método GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben recuperar datos y no deben tener ningún otro efecto.
2. HEAD: pide una respuesta idéntica a la que correspondería a una petición GET, pero en la respuesta no se devuelve el cuerpo. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.
3. POST: envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.
4. PUT: sube, carga o realiza un upload de un recurso especificado (archivo o fichero) y es un camino más eficiente ya que POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor. En contraste, el método PUT permite escribir un archivo en una conexión socket establecida con el servidor. La desventaja del método PUT es que los servidores de alojamiento compartido no lo tienen habilitado.

3.2.2. HTML 5

De acuerdo a la documentación de Mozilla[14], HyperText Markup Language, abreviado HTML, o lenguaje de marcado de hipertextos, es la pieza más básica para la construcción de la web y se usa para definir el sentido y estructura del contenido en una página web.

Es un estándar a cargo del World Wide Web Consortium (W3C) o Consorcio WWW, organización dedicada a la estandarización de casi todas las tecnologías ligadas a la web.

HTML hace uso de enlaces que conectan las páginas web entre sí, ya sea dentro de un mismo sitio web o entre diferentes sitios web.

Un elemento HTML se separa de otro texto en un documento por medio de “etiquetas”, las cuales consisten en elementos rodeados por “<,>”.

Ejemplos de estas etiquetas son <head>, <title>, <body>, <header>, <p>, , , , y otros más.

HTML 5 (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML.

HTML5 establece elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas <div> y , pero tienen un significado semántico, como por ejemplo <nav> (bloque de navegación del sitio web) y <footer>.

Características:

1. Incorpora etiquetas: *canvas* 2D y 3D, audio, vídeo con codecs para mostrar los contenidos multimedia. Actualmente hay una lucha entre imponer codecs libres (WebM + VP8) o privados (H.264/MPEG-4 AVC).
2. Etiquetas para manejar grandes conjuntos de datos: Datagrid, Details, Menu y Command. Permiten generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en cliente.
3. Mejoras en los formularios: Nuevos tipos de datos (*email*, *number*, *url*, *datetime*) y facilidades para validar el contenido sin JavaScript.
4. Visores: MathML (fórmulas matemáticas) y SVG (gráficos vectoriales); en general se deja abierto a poder interpretar otros lenguajes XML.
5. Drag & Drop: nueva funcionalidad para arrastrar objetos como imágenes.

Respecto a la compatibilidad con los navegadores, la mayoría de elementos de HTML5 son compatibles con Firefox 19, Chrome 25, Safari 6 y Opera 12 en adelante.

3.2.3. CSS vs SASS

CSS

De acuerdo a la documentación de la W3C[15], Cascading Style Sheets, CSS, o hojas de estilo en cascada, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.

CSS está diseñado principalmente para marcar la separación del contenido del documento y la presentación del mismo, con características como las capas o layouts, los colores y las fuentes.

La separación entre el contenido del documento y la presentación busca mejorar la accesibilidad, proveer más flexibilidad y control, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css y reducir la complejidad y la repetición de código en la estructura del documento.

La especificación CSS describe un esquema prioritario para determinar qué reglas de estilo se aplican si más de una regla coincide para un elemento en particular. Estas reglas son aplicadas con un sistema llamado de cascada, de modo que las prioridades son calculadas y asignadas a las reglas, así que los resultados son predecibles.

La especificación CSS es mantenida por el World Wide Web Consortium (W3C). El MIME type text/css está registrado para su uso por CSS descrito en el RFC 23185. El W3C proporciona una herramienta de validación de CSS gratuita para los documentos CSS.

CSS se ha creado en varios niveles y perfiles. Cada nivel de CSS se construye sobre el anterior, generalmente añadiendo funciones al nivel previo.

Los perfiles son, generalmente, parte de uno o varios niveles de CSS definidos para un dispositivo o interfaz particular. Actualmente, pueden usarse perfiles para dispositivos móviles, impresoras o televisiones.

La última versión del estándar, CSS3.1, está dividida en varios documentos separados, llamados “módulos”.

Cada módulo añade nuevas funcionalidades a las definidas en CSS2, de manera que se preservan las anteriores para mantener la compatibilidad.

Los trabajos en el CSS3.1 comenzaron a la vez que se publicó la recomendación oficial de CSS2 y los primeros borradores de CSS3.1 fueron liberados en junio de 1999.

Debido a la modularización del CSS3.1, diferentes módulos pueden encontrarse en diferentes estados de su desarrollo, de forma que hay alrededor de cincuenta módulos publicados, tres de ellos se convirtieron en recomendaciones oficiales de la W3C en 2011: “Selectores”, “Espacios de nombres”, y “Color”.

Respecto al soporte de los navegadores web, cada navegador web usa un motor de renderizado para renderizar páginas web, y el soporte de CSS no es exactamente igual en ninguno de los motores de renderizado. Ya que los navegadores no aplican el CSS correctamente, muchas técnicas de programación han sido desarrolladas para ser aplicadas por un navegador específico (comúnmente conocida esta práctica como *CSS hacks* o *CSS filters*).

Además, CSS3 definen muchas queries, entre las cuales se provee la directiva @supports que permite a los desarrolladores especificar navegadores con soporte para alguna función en específico directamente en el CSS3.1.

Sass

De acuerdo a la documentación de Sass^[16], Sass es un lenguaje de preprocesado que genera hojas de estilo en cascada (CSS) y consta de dos sintaxis.

Características

Variables Las variables comienzan con un signo de dólar y la asignación de valor se realiza con dos puntos.

SassScript permite 4 tipos de datos:

1. Números (incluyendo las unidades).
2. Strings (con comillas o sin ellas).
3. Colores (código, o nombre).
4. Booleanos.

Las variables pueden ser resultados o argumentos de varias funciones disponibles. Durante el proceso de traducción, los valores de las variables son insertados en el documento CSS de salida.

Anidamiento CSS soporta anidamiento lógico, pero los bloques de código no son anidados. Sass permite que el código anidado sea insertado dentro de cualquier otro bloque.³

Mixins Como CSS no soporta *mixins*, cualquier código duplicado debe ser repetido en cada lugar. Un *mixin* en Sass es una sección de código que contiene código Sass.

Cada vez que se llama un *mixin* en el proceso de conversión el contenido del mismo es insertado en el lugar de la llamada. Los *mixin* permiten una solución limpia a las repeticiones de código, así como una forma fácil de alterar el mismo.

Elección

Tomando en cuenta la experiencia del equipo con CSS, además de que el proyecto no se enfocará en hacer muchas hojas de estilo para cada componente, página o vista de la aplicación web, se usará CSS en lugar de Sass.

3.2.4. JavaScript vs TypeScript

JavaScript

De acuerdo con la documentación de Mozilla^[17], JavaScript es una marca registrada con licencia de Sun Microsystems (ahora Oracle) que se usa para describir la implementación del lenguaje de programación JavaScript.

Debido a problemas de registro de marcas en la Asociación Europea de Fabricantes de Computadoras, la versión estandarizada del lenguaje tiene el nombre de ECMAScript, sin embargo, en la práctica se conoce como lenguaje JavaScript.

Abreviado como JS, es un lenguaje ligero e interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico; es usado en node.js, Apache CouchDB y Adobe Acrobat.

EL núcleo del lenguaje JavaScript está estandarizado por el Comité ECMA TC39 como un lenguaje llamado ECMAScript. La última versión de la especificación es ECMAScript 6.0.

El estándar ECMAScript define

1. Sintaxis: reglas de análisis, palabras clave, flujos de control, inicialización literal de objetos.
2. Mecanismos de control de errores: *throw*, *try/catch*, habilidad para crear tipos de Errores definidos por el usuario.
3. Tipos: *boolean*, *number*, *string*, *function*, *object*.
4. Objetos globales: en un navegador, los objetos globales son los objetos de la ventana, pero ECMAScript solo define API no específicas para navegadores, como *parseInt*, *parseFloat*, *decodeURI*, *encodeURI*.
5. Mecanismo de herencia basada en prototipos.
6. Objetos y funciones incorporadas.
7. Modo estricto.

La sintaxis básica es similar a Java y C++ con la intención de reducir el número de nuevos conceptos necesarios para aprender el lenguaje. Las construcciones del lenguaje, tales como sentencias *if*, bucles *for* y *while*, bloques *switch* y *try catch* funcionan de la misma manera que en estos lenguajes (o casi).

JavaScript puede funcionar como lenguaje procedimental y como lenguaje orientado a objetos. Los objetos se crean programáticamente añadiendo métodos y propiedades a lo que de otra forma serían objetos vacíos en tiempo de ejecución, en contraposición a las definiciones sintácticas de clases comunes en los lenguajes compilados como C++ y Java. Una vez se ha construido un objeto, puede usarse como modelo (o prototipo) para crear objetos similares.

Las capacidades dinámicas de JavaScript incluyen construcción de objetos en tiempo de ejecución, listas variables de parámetros, variables que pueden contener funciones, creación de scripts dinámicos (mediante *eval*), introspección de objetos (mediante *for ... in*), y recuperación de código fuente (los programas de JavaScript pueden decompilar el cuerpo de funciones a su código fuente original).

Desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores más antiguos soportan por lo menos ECMAScript 3.

El 17 de Julio de 2015, ECMA International publicó la sexta versión de ECMAScript, la cual es oficialmente llamada ECMAScript 2015 y fue inicialmente nombrada como ECMAScript 6 o ES6. Desde entonces, los estándares ECMAScript están en ciclos de lanzamiento anuales.

TypeScript

De acuerdo con TypeScript Publishing[18], TypeScript es por definición es JavaScript para el desarrollo de aplicaciones, siendo también un superconjunto del mismo.

TypeScript es un lenguaje compilado orientado a objetos. Fue diseñado por Anders Hejlsberg (diseñador de C#) en Microsoft. TypeScript es tanto un lenguaje como un conjunto de herramientas. TypeScript es un superconjunto de JavaScript que genera código JavaScript.

Características

- **Compilación:** cuenta con un transpilador para la verificación de errores si hay errores de compilación, cosa que no es posible con JavaScript.
- **Tipeo estático fuerte:** provee un sistema opcional de tipeo estático y de inferencia de tipos a través del TypeScript Language Service, lo que permite inferir el tipo de una variable declara sin tipo en función de su valor.
- **Definiciones de tipo:** permite la extensión del lenguaje con bibliotecas externas JavaScript.
- **Programación orientada a objetos:** admite conceptos como clases, interfaces, herencia, etc.

Elección

De acuerdo con el sitio stack overflow[19], JavaScript es el lenguaje más popular de 2019 y aunque TypeScript es de los lenguajes que tienen un mayor nivel de aceptación, se usará JavaScript no solo por ser el lenguaje más popular y, en consecuencia, con más compatibilidad y material de ayuda, sino también porque el equipo está acostumbrado a este lenguaje.

3.2.5. JavaScript Frameworks: Vue/Nuxt vs React vs Angular

De acuerdo con Wikipedia[20], un web *framework* es un conjunto de *software* que permiten el desarrollo de una aplicación web y en el lenguaje JavaScript hay varias opciones, incluidas las más populares Vue, React y Angular.

Vue/Nuxt

De acuerdo con la documentación de Vue.js[21], Vue es un *framework* progresivo para desarrollar interfaces de usuario. A diferencia de otros *frameworks* monolíticos, Vue está diseñado desde cero para ser utilizado incrementalmente.

La librería central está enfocada solo en la capa de visualización y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar sofisticadas *single-page applications* cuando se utiliza en combinación con librerías de apoyo.

Comparación con React React y Vue comparten muchas similitudes; ambos utilizan un DOM virtual, proporcionan componentes de vista reactivos y componentes, mantienen el enfoque en la librería central, con temas como el enrutamiento y la gestión global del estado manejadas por librerías asociadas.

Tanto React como Vue ofrecen un rendimiento comparable en los casos de uso más comunes, con Vue normalmente un poco por delante debido a su implementación más ligera del DOM virtual.

En Vue, las dependencias de un componente se rastrean automáticamente durante su renderizado, por lo que el sistema sabe con precisión qué componentes deben volver a renderizarse cuando cambia el estado. Se puede considerar que cada componente tiene un *shouldComponentUpdate* automáticamente implementado.

En general, esto elimina la necesidad de toda una clase de optimizaciones de rendimiento a los desarrolladores y les permite centrarse más en la construcción de la aplicación en sí misma a medida que va escalando.

Comparación con Angular En términos de rendimiento, ambos *frameworks* son excepcionalmente rápidos y no hay suficientes datos de casos de uso en el mundo real para hacer un veredicto.

Vue es mucho menos intrusivo en las decisiones del desarrollador que Angular, ofreciendo soporte oficial para una variedad de sistemas de desarrollo, sin restricciones sobre cómo estructurar su aplicación. Muchos desarrolladores disfrutan de esta libertad, mientras que algunos prefieren tener solo una forma correcta de desarrollar cualquier aplicación.

Para empezar con Vue, todo lo que se necesita es familiarizarse con HTML y ES5 JavaScript; con estas habilidades básicas, se puede empezar a desarrollar aplicaciones no triviales.

La curva de aprendizaje de Angular es mucho más pronunciada. La superficie de la API del framework es enorme y como usuario necesitará familiarizarse con muchos más conceptos antes de ser productivo. La complejidad de Angular se debe en gran medida a su objetivo de diseño de apuntar solo a aplicaciones grandes y complejas - pero eso hace que el *framework* sea mucho más difícil de entender para los desarrolladores con menos experiencia.

Nuxt.js

De acuerdo a la documentación de Nuxt[22], Nuxt.js es un *framework* progresivo basado en Vue.js para crear aplicaciones web. Se basa en Vue.js y herramientas de desarrollo como webpack, Babel y PostCSS. El objetivo de Nuxt es hacer que el desarrollo web en Vue.js sea eficaz.

Características

1. Escribir archivos Vue (*.vue)
2. División automática de código
3. Representación del lado del servidor
4. Potente sistema de enrutamiento con datos asincrónicos
5. Servicio de archivos estáticos
6. Soporte sintaxis ES2015+ (Javascript ES6)
7. Gestión del elemento <head> (<title>, <meta>, etc.)
8. Preprocesador: Sass, Less, Stylus, etc.

React

Angular

Elección

Se usará Vue/Nuxt por ser el *framework* con el que el equipo está más acostumbrado, además de ser el más flexible de las opciones expuestas.

3.2.6. CSS Frameworks: Vuetify vs Bootstrap

Un CSS framework es

Vuetify

De acuerdo a la documentación de Google[23], Material Design es un lenguaje visual que sintetiza los principios clásicos del buen diseño respecto a las ideas de Google y en estos principios está basado Vuetify.

El objetivo de Material Design es crear un lenguaje visual que sintetice los principios clásicos del buen diseño, unificar el desarrollo de un único sistema subyacente que unifique la experiencia del usuario en plataformas y dispositivos, así como personalizar el lenguaje visual de Material Design.

Asimismo, de acuerdo con la documentación de Vuetify[24], este CSS *framework* está integrado para ser usado en los componentes de Vue/Nuxt, desde botones, barras de navegación, *layouts* y demás.

Bootstrap

De acuerdo a la documentación de Bootstrap[25], es un CSS *framework* orientado al diseño responsivo de una aplicación web.

Tiene *templates* para botones, barras de navegación, estilos de tipografía entre otros. Es de fácil integración con React, AngularJS o Vue y tiene una comunidad extensa por los años y popularidad que tiene.

Elección

Se ha elegido usar en una primera instancia Vuetify porque es un CSS *framework* que está integrado en las tecnologías asociadas de Vue, como Vue Router, Vue Meta. Asimismo, sus componentes son simples de entender y de implementar.

3.2.7. MongoDB vs Apache CouchDB

MongoDB

De acuerdo a la documentación de Wikipedia[26], MongoDB (del inglés humongous, “enorme”) es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.

En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Características

1. Consultas ad hoc: MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares.
2. Indexación: cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios.
3. Replicación: MongoDB soporta el tipo de replicación primario-secundario.
4. Balanceo de carga; MongoDB puede escalar de forma horizontal usando el concepto de *sharding*.
5. Almacenamiento de archivos: MongoDB puede ser utilizado como un sistema de archivos, aprovechando la capacidad de MongoDB para el balanceo de carga y la replicación de datos en múltiples servidores.
6. Agregación: MongoDB proporciona un framework de agregación que permite realizar operaciones similares al GROUP BY de SQL.

3.2.8. GoJS vs mxGraph vs D3.js

De acuerdo a la documentación de GoJS[27], GoJS es una biblioteca de JavaScript y TypeScript para crear diagramas y gráficos interactivos.

GoJS le permite crear todo tipo de diagramas y gráficos para sus usuarios, desde simples diagramas de flujo y organigramas hasta diagramas industriales altamente específicos, diagramas SCADA y BPMN, diagramas médicos como genogramas y diagramas de modelos de brotes, y más.

GoJS facilita la construcción de diagramas JavaScript de nodos complejos, enlaces y grupos con plantillas y diseños personalizables.

GoJS ofrece muchas funciones avanzadas para la interactividad del usuario, como arrastrar y soltar, copiar y pegar, edición de texto en el lugar, información sobre herramientas, menús contextuales, diseños automáticos, plantillas, enlace de datos y modelos, gestión de estado y deshacer transaccional, paletas, descripciones generales, controladores de eventos, comandos, herramientas extensibles para operaciones personalizadas y animaciones personalizables.

GoJS se implementa en TypeScript y puede usarse como una biblioteca de JavaScript o incorporarse a su proyecto desde fuentes de TypeScript.

GoJS normalmente se ejecuta completamente en el navegador, renderizando a un elemento HTML Canvas o SVG sin ningún requisito del lado del servidor.

3.2.9. Python 3 vs Java

De acuerdo con Mark Lutz[28], Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipo dinámico, tipos de datos dinámicos y clases.

Tiene sintaxis clara, interfaces para muchas llamadas de sistema y bibliotecas, así como para varios sistemas de ventanas. Además, es extensible en C o C++.

También se puede usar como un lenguaje de extensión para aplicaciones que necesitan una interfaz programable y es portátil: se ejecuta en muchas variantes de Unix, en Mac y en Windows 2000 y versiones posteriores.

La biblioteca estándar del lenguaje, cubre áreas como el procesamiento de cadenas (expresiones regulares, Unicode, cálculo de diferencias entre archivos), protocolos de Internet (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, programación CGI), ingeniería de software (pruebas unitarias, registro, creación de perfiles, análisis del código Python) e interfaces del sistema operativo (llamadas al sistema, sistemas de archivos, sockets TCP/IP).

Fortalezas

Es orientado a objetos y funcional Python es un lenguaje orientado a objetos; su modelo de clase admite nociones avanzadas como el polimorfismo, la sobrecarga del operador y la herencia múltiple; sin embargo, en el contexto de la simple sintaxis y escritura de Python, la programación orientada a objetos es fácil de aplicar.

Además de servir para la estructuración y reutilización de código, la naturaleza orientada a objetos de Python lo hace ideal como herramienta de secuencias de comandos para otros lenguajes de sistemas orientados a objetos. Por ejemplo, con el código apropiado, los programas Python pueden especializar clases implementadas en C++, Java y C#.

No obstante, la programación orientada a objetos es una opción en Python. Al igual que C++, Python admite modos de programación tanto procedimentales como orientados a objetos. Las herramientas orientadas a objetos se pueden aplicar siempre que las restricciones lo permitan.

Además de sus paradigmas originales de procedimientos (basados en declaraciones) y orientados a objetos (basados en clases), Python en los últimos años ha adquirido soporte incorporado para la programación funcional, un conjunto que incluye generadores, comprensiones, cerraduras, mapas, decoradores, funciones anónimas lambdas.

Es extensible Su conjunto de herramientas lo ubica entre los lenguajes de *scripting* tradicionales como Tcl, Scheme y Perl; y los lenguajes de desarrollo de sistemas como C, C++ y Java.

Python proporciona toda la simplicidad y facilidad de uso de un lenguaje de programación, junto con herramientas de ingeniería de software más avanzadas que normalmente se encuentran en lenguajes compilados.

A diferencia de algunos lenguajes de secuencias de comandos, esta combinación hace que Python sea útil para proyectos de desarrollo a gran escala. Algunas de las herramientas de Python son:

Escritura dinámica Python realiza un seguimiento de los tipos de objetos que utiliza su programa cuando se ejecuta; eso no requiere declaraciones complicadas de tipo y tamaño en su código. De hecho, no existe una declaración de tipo o variable en Python.

Debido a que el código Python no restringe los tipos de datos, también se aplica automáticamente a toda una gama de objetos.

Gestión automática de la memoria Python asigna automáticamente objetos y los reclama el recolector de basura cuando ya no se usan y la mayoría puede crecer y reducirse según la demanda. Es decir, Python realiza un seguimiento de los detalles de la memoria de bajo nivel.

Tipos de objetos incorporados Python proporciona estructuras de datos de uso común como listas, diccionarios y cadenas como partes intrínsecas del lenguaje. Son flexibles y fáciles de usar. Por ejemplo, los objetos integrados pueden crecer y reducirse según demanda, pueden anidarse arbitrariamente para representar información compleja, y más.

Herramientas incorporadas Para procesar todos esos tipos de objetos, Python viene con operadores potentes y estándar, que incluyen concatenación (unir colecciones), segmentar (extraer secciones), ordenar, mapear y más.

Utilidades de biblioteca Para tareas más específicas, Python también viene con una gran colección de herramientas de biblioteca precodificadas que admiten todo, desde la coincidencia de expresiones regulares hasta la creación de redes. Una vez que aprende el lenguaje en sí, las herramientas de la biblioteca de Python son donde ocurre gran parte de la acción a nivel de aplicación.

Utilidades de terceros Debido a que Python es de código abierto, los desarrolladores pueden contribuir con herramientas precodificadas que admitan tareas que aún no son herramientas estándar; en la Web, encontrará soporte gratuito para COM, imágenes, programación numérica, XML y acceso a bases de datos.

3.2.10. Django vs Flask

Flask es un micro *web framework* escrito en Python. Se clasifica como micro porque no requiere herramientas o bibliotecas particulares.

No tiene capa de abstracción de base de datos, validación de formularios ni ningún otro componente donde las bibliotecas de terceros preexistentes brinden funciones comunes. Sin embargo, Flask admite extensiones que pueden agregar características de la aplicación como si se implementaran en el propio Flask.

Existen extensiones para mapeadores relacionales de objetos, validación de formularios, manejo de carga, varias tecnologías de autenticación abiertas y varias herramientas relacionadas con el marco común. Las extensiones se actualizan con mucha más frecuencia que el programa central Flask.

Ventajas y desventajas de Flask

Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2; además, tiene una licencia BSD.

Entre las ventajas y desventajas, destacamos:

Ventajas

1. Es un framework que se destaca en instalar extensiones o complementos de acuerdo al tipo de proyecto que se va a desarrollar, es decir, es perfecto para el prototipado rápido de proyectos.
2. Incluye un servidor web, así podemos evitamos instalar uno como Apache o Nginx. Además, nos ofrece soporte para pruebas unitarias y para Cookies de seguridad (sesiones del lado del cliente), apoyándose en el motor de plantillas Jinja2.

-
3. Su velocidad es mejor a comparación de Django. Generalmente el desempeño que tiene Flask es superior debido a su diseño minimalista que tiene en su estructura.
 4. Flask permite combinarse con herramientas para potenciar su funcionamiento, por ejemplo: Jinja2, SQLAlchemy, Mako y Peewee entre otras.

Desventajas

1. Su sistema de autenticación de usuarios es muy básico, a comparación del potente sistema de autenticación que utiliza Django, este puede crear un sistema de Login API sencillo para aplicaciones más pequeñas.
2. Su representación de Plugins no es tan extensa como la tiene Django.
3. Es complicado en las pruebas unitarias o migraciones.
4. El ORM (Mapeo objeto relacional) para conectar con las bases de datos, SQLAlchemy es externo.

Capítulo 4

Análisis y Diseño del Sistema

4.1. Metodología

La metodología, es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. En un proyecto de desarrollo de software la metodología ayuda a definir: Quién debe hacer Qué Cuándo y Cómo debe hacerlo. La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con altas posibilidades de éxito. Una metodología para el desarrollo de software comprende actividades a seguir para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado.[<https://www.uladech.edu.pe/images/stories/universidad/documentos/2018/metodologia-desarrollo-software-v001.pdf>]

Dentro del campo del desarrollo de software, nos encontramos con dos grupos de metodologías, las tradicionales y las ágiles. Las tradicionales, exigen una documentación exhaustiva y se centran en cumplir con el plan del proyecto definido totalmente en la fase inicial del desarrollo del mismo.

Dado a que cualquier cambio en el proceso generaba la necesidad de una reconstrucción en la metodología, surgieron las llamadas metodologías ágiles, las cuales permiten realizar cambios en los requerimientos conforme se va desarrollando el proyecto basada en las habilidades del equipo y una buena relación con el usuario mostrando avances funcionales en cortos periodos de tiempo con la posibilidad de que este realice una evaluación y de ser necesario, sugiera cambios.

Se han propuesto muchos modelos ágiles de proceso y están en uso en toda la industria. Entre ellos se encuentran los siguientes:

- Desarrollo adaptativo de software (DAS)
- Scrum
- Método de desarrollo de sistemas dinámicos (MDSD)
- Cristal
- Desarrollo impulsado por las características (DIC)

-
- Desarrollo esbelto de software (DES)
 - Modelado ágil (MA)
 - Proceso unificado ágil (PUA)

4.1.1. SCRUM

Para la factibilidad técnica se realiza una evaluación de las herramientas de hardware y software que el equipo de trabajo tiene disponible.

Scrum se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo.

Los tres pilares que soportan toda la implementación del control de procesos:

- Transparencia
- Inspección
- Adaptación

Scrum prescribe cuatro eventos formales, contenidos dentro del Sprint, para la inspección y adaptación.[K. Schwaber and J. Sutherland, The Scrum Guide, Julio 2016,]

- Planificación del Sprint (*Sprint Planning*)
- Scrum Diario (*Daily Scrum*)
- Revisión del Sprint (*Sprint Review*)
- Retrospectiva del Sprint (*Sprint Retrospective*)

El Equipo Scrum consiste en los siguientes roles:

- El Dueño de Producto (*Product Owner*):
Responsable de maximizar el valor del producto y el trabajo del Equipo de Desarrollo. Es la única persona responsable de gestionar la Lista del Producto (*Product Backlog*).
- El Equipo de Desarrollo (*Development Team*):
Profesionales que realizan el trabajo de entregar un Incremento de producto “Terminado” que potencialmente se pueda poner en producción al final de cada Sprint. Solo los miembros del Equipo de Desarrollo participan en la creación del Incremento.
- El Scrum Master:
Responsable de asegurar que Scrum se entienda y se adopte, asegurándose de que el Equipo Scrum trabaja ajustándose a la teoría, prácticas y reglas de Scrum. Ayuda a las personas externas al Equipo Scrum a entender qué interacciones con el Equipo Scrum pueden ser útiles y cuáles no.

La metodología Scrum cuenta con los elementos descritos a continuación:

- **Lista de Producto (*Product Backlog*):**
Es el listado de todas las tareas que necesita el proyecto para alcanzar su realización. Al iniciar el desarrollo del proyecto esta lista no se encuentra completa y conforme avanzan los *sprints* se le añaden tareas para solventar las necesidades que van surgiendo gracias a la retroalimentación del cliente.
- **Lista de Pendientes del Sprint (*Sprint Backlog*) :**
Es la lista de tareas seleccionadas del product backlog que se planifica realizar durante el periodo del *sprints* y se definen a los responsables de cada tarea.
- **Sprint:**
Es un periodo de tiempo determinado donde el equipo completa conjuntos de tareas incluidas en el *backlog*.
- **Incremento :**
El Incremento es la suma de todos los elementos de la Lista de Producto completados durante un *sprints* y el valor de los incrementos de todos los Sprints anteriores

La metodología nos permite definir un periodo de hasta un mes para cada sprint y se ha optado por un periodo de 30 días, contemplándose un total de ocho sprints, donde al término de cada uno se tendrá un avance del sistema.

4.1.1.1. Lista de Producto (*Product Backlog*)

La Lista de Producto es una lista ordenada de todo lo que podría ser necesario en el producto y es la única fuente de requisitos para cualquier cambio a realizarse en el producto, enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a realizarse sobre el producto para entregas futuras.[<http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.p>

Id	Descripción	Estimación	Valor	Prioridad
A	Investigación y selección de las tecnologías del modelo de base de datos no relacional a utilizar.	2	1	Media
B	Interfaz gráfica de la plataforma web.	3	3	Media
C	Edición y validación del esquema entidad-relación	3	4	Media
D	Transformación del esquema entidad-relación al modelo relacional.	5	4	Alta
E	Visualización de la transformación de modelo entidad-relación al modelo relacional.	3	5	Media
F	Transformación del modelo relacional al modelo NoSQL.	5	5	Alta
G	Visualización del modelo relacional al modelo NoSQL	3	4	Media

Continuación de Tabla 3.2

Continuación de tabla				
H	Pruebas de caso de estudio para verificar la correcta transformación y coherencia en los datos.	2	2	Baja

4.1.1.2. Historias de usuario

Historia Id	Rol	Evento	Sub tarea	Funcionalidad	Criterio de aceptación	Prioridad
A - 01	Scrum team	Investigación Bases de datos relacionales.	1			Baja
A - 02	Scrum team	Redacción y selección de las tecnologías a utilizar para el desarrollo de la plataforma.	1			Baja
A - 03	Scrum team	Investigación Bases de datos relacionales.	1			Baja
A - 04	Scrum team	Redacción bases de datos relacionales en el documento técnico.	1			Baja
A - 05	Scrum team	Investigación Bases de datos no relacionales	1			Baja
A - 06	Scrum team	Redacción bases de datos no relacionales en el documento técnico.	1			Baja
A - 07	Scrum team	Investigación y selección del modelo de base de datos no relacional a utilizar junto a las tecnologías a utilizar	1			Media
B - 01	Scrum team	Análisis y diseño de la arquitectura Web	1			Alta
B - 02	Scrum team	Integración de la primera versión de la plataforma web.	1			Media

Continuación de Tabla 3.2

B - 03	Scrum team	Desarrollo de la estructura básica del backend.	1			Media
B - 04	Scrum team	Planteamiento de escenarios de los esquemas entidad-relación.	1			Baja
B - 05	Scrum team	Desarrollo de la estructura básica del frontend.	1			Media
B - 06	Scrum team	Primera implementación del maquetado frontend.	1			Media
B - 07	Scrum team	Interfaz gráfica de la plataforma web.	1			Media
C - 01	Scrum team	Definición de las reglas del modelo entidad-relación (etapa 1).	1			Alta
C - 02	Scrum team	Desarrollo para la edición de diagramas en la plataforma web.	1			Media
C - 03	Scrum team	Habilitar la carga de un modelo entidad-relación en la plataforma web.	1			Media
C - 04	Scrum team	Agregar elementos básicos para la edición del diagrama E-R.	1			Media
C - 05	Scrum team	Edición y validación del esquema entidad-relación.	1			Alta
D - 01	Scrum team	Pruebas de captura de distintos diagramas entidad-relación.	1			Baja
D - 02	Scrum team	Transformación del esquema entidad-relación al modelo relacional.	1			Alta
D - 03	Scrum team	Guardado del esquema entidad-relación.	1			Media
D - 04	Scrum team	Visualización de la primera etapa de transformación en la plataforma web.	1			Alta

Continuación de Tabla 3.2

D - 05	Scrum team	Adición de elementos a la paleta de edición del diagrama E-R.	1			Media
D - 06	Scrum team	Documento reporte técnico para presentación TT1	1			Alta
E - 01	Scrum team	Planteamiento de escenarios del modelo relacional al modelos noSQL	1			Media
E - 02	Scrum team	Definición de las reglas de transformación al modelo noSQL	1			Alta
E - 03	Scrum team	Prueba de transformación de distintos diagramas E-R.	1			Baja
E - 04	Scrum team	Conexión de la plataforma con un SGBD.	1			Alta
E - 05	Scrum team	Generación de distintos diagramas E-R	1			Media
E - 06	Scrum team	Lectura del modelo E-R para visualizar en la plataforma.	1			Media
E - 07	Scrum team	Visualización de la transformación de modelo entidad-relación al modelo relacional.	1			Alta
F - 01	Scrum team	Generar pruebas de concepto del esquema noSQL	1			Baja
F - 02	Scrum team	Agregación de la función de transformación a la plataforma.	1			Media
F - 03	Scrum team	Visualización de la segunda etapa de transformación en la plataforma web.	1			Alta
F - 04	Scrum team	Pruebas de distintos escenarios del modelo relacional al modelo noSQL.	1			Baja
F - 05	Scrum team	Agregar nuevas funciones de visualización a la interfaz gráfica.	1			Media

Continuación de Tabla 3.2

F - 06	Scrum team	Comprobación de la coherencia de la transformación entre bases de datos.	1			Media
F - 07	Scrum team	Transformación del modelo relacional al modelo noSQL seleccionado.	1			Alta
G - 01	Scrum team	Agregar nuevas funciones de visualización a la interfaz gráfica.	1			Media
G - 02	Scrum team	Comprobación de la coherencia de la transformación entre bases de datos.	1			Alta
G - 03	Scrum team	Pruebas de persistencia y coherencia de las bases SQL a noSQL(etapa 3)	1			Media
G - 04	Scrum team	Integración de la visualización a la plataforma.	1			Alta
G - 05	Scrum team	Visualización del modelo relacional al modelo noSQL seleccionado.	1			Alta
H - 01	Scrum team	Documento reporte técnico para presentación TT2	1			Alta
H - 02	Scrum team	Pruebas de caso de estudio para verificar la correcta transformación y coherencia en los datos.	1			Alta

4.2. Factibilidad técnica

Para la factibilidad técnica se realiza una evaluación de las herramientas de hardware y software que el equipo de trabajo tiene disponible.

Recursos técnicos de software

Equipo	Tipo	Sistema operativo	Procesador	RAM
Toshiba Satellite Radius P55W-B	Laptop	Windows 10	Intel Core i7-10U	8GB
Dell Laptop Latitude 3400	Laptop	Windows 10	Intel Core I5 8265	8GB

Continuación de Tabla 3.2

Continuación de tabla				
HP Pavilion15” DeskJet Ink Advantage 3775	Laptop Impresora Multifuncional	Windows 10 -	Core i5-9300H -	8GB -

Recursos técnicos de hardware

Nombre	Tipo
Windows 10	Sistema operativo
Ubuntu	Sistema operativo
MongoDB Atlas	Base de datos en la nube
Visual Studio Code	IDE
Flask	Framework
Vue.js	Framework
Nuxt	Framework

Con todo esto, se puede concluir que se cuenta con todo el software y hardware necesario para la realización del proyecto.

4.3. Factibilidad económica

Estimación

Para la mayoría de los proyectos, el mayor costo es el primer rubro. Debe estimarse el esfuerzo total (en meses-hombre) que es probable se requiera para completar el trabajo de un proyecto. Desde luego, se cuenta con datos limitados para realizar tal valoración, de manera que habrá que hacer la mejor evaluación posible y a continuación agregar contingencia significativa (tiempo y esfuerzo adicionales) en caso de que la estimación inicial sea optimista.

4.3.1. COCOMO II

El modelo COCOMO (por sus acrónimo en inglés COnstructive COst MOdel) original se convirtió en uno de los modelos de estimación de costo más ampliamente utilizados y estudiados en la industria. Evolucionó hacia un modelo de estimación más exhaustivo, llamado COCOMO II. Como su predecesor, COCOMO II en realidad es una jerarquía de modelos de estimación que aborda las áreas siguientes: [Roger S. Pressman, Ph.D. **Ingeniería del software UN ENFOQUE PRÁCTICO 7** edición, ed. MCGRAW-HILL]

1. **Modelo de composición de aplicación.** Se usa durante las primeras etapas de la ingeniería de software, cuando son primordiales la elaboración de prototipos de las interfaces de usuario, la consideración de la interacción del software

y el sistema, la valoración del rendimiento y la evaluación de la madurez de la tecnología.

2. **Modelo de etapa temprana de diseño.** Se usa una vez estabilizados los requisitos y establecida la arquitectura básica del software.
3. **Modelo de etapa postarquitectónica.** Se usa durante la construcción del software.

Como todos los modelos de estimación para software, los modelos COCOMO II requieren información sobre dimensionamiento. Como parte de la jerarquía del modelo, están disponibles tres diferentes opciones de dimensionamiento: puntos objeto, puntos de función y líneas de código fuente.

4.3.1.1. El modelo de composición de aplicación COCOMO II

El esfuerzo necesario para concretar un proyecto de desarrollo de software, cualquiera sea el modelo empleado, se expresa en meses/persona (PM) y representa los meses de trabajo de una persona fulltime, requeridos para desarrollar el proyecto. Para derivar una estimación del esfuerzo debemos utilizar la siguiente relación.

$$Esfuerzo_{estimado} = \frac{NOP}{PROD}$$

donde:

- Esfuerzo estimado (PM) es la estimación del esfuerzo en meses-hombre.
- NOP es el número de puntos de aplicación en el sistema entregado.
- PROD es la productividad del punto de aplicación

Para determinar NOP se necesita utilizar el punto de objeto el cual una medida de software indirecta que se calcula usando conteos del número de:

1. pantallas (en la interfaz de usuario)
2. reportes
3. componentes que probablemente se requieran para construir la aplicación.

Cada instancia de objeto (por ejemplo, una pantalla o reporte) se clasifica en uno de tres niveles de complejidad (simple, medio o difícil), usando criterios sugeridos por Boehm [Boehm, B., "Anchoring the Software Process", IEEE Software, vol. 13, núm. 4, julio 1996, pp. 73-82]

Tipo de objeto	Cantidad	Peso de complejidad			Subtotal
		Simple	Media	Difícil	
Pantalla	4	1	2	<u>3</u>	12
Reporte	9	10	<u>5</u>	8	45
Componente	10			10	100
Total OP					157

Determinado el conteo de puntos de objeto se multiplican el número original de instancias de objeto por el factor de ponderación y se suman para obtener un conteo total de puntos de objeto.

$$NOP = (Puntos_de_objeto) \left(\frac{100 - \%reutilizacion}{100} \right)$$

Dentro de este punto, no se encuentra código el cual reutilizar, por lo que:

$$NOP = (Puntos_de_objeto)$$

$$NOP = (157)$$

Para la tasa de productividad se utiliza la figura [referenciar tabla], para diferentes niveles de experiencia del desarrollador y de madurez del entorno de desarrollo.

Experiencia del desarrollador	Muy baja	Baja	Nominal	Alta	Muy alta
Madurez/ capacidad del entorno	Muy baja	Baja	Nominal	Alta	Muy alta
PROD	4	7	10	25	50

Una vez determinados los nuevos puntos de objeto y la tasa de productividad, se puede calcular una estimación del esfuerzo del proyecto

$$Esfuerzo_{estimado} = \frac{NOP}{PROD}$$

$$Esfuerzo_{estimado} = \frac{157}{7}$$

$$Esfuerzo_{estimado} = 22.42$$

$$Esfuerzo_{estimado} = 23 \text{ Meses} - \text{hombre}$$

Una vez obtenidos estos valores, se pueden sumar los costos derivados del desarrollo de software

Factibilidad económica

Dentro de la factibilidad económica se encuentran los costos necesarios que llevará consigo el desarrollo de la aplicación.

Costo de hardware

Costo	Equipo
\$10,000.00	Toshiba Satellite Radius P55W-B, Windows 10, Intel Core i7-10U, 8GB RAM
\$13,999.00	Dell Laptop Latitude 3400, Ubuntu, Intel Core I5 8265, 8GB RAM.

Continuación de Tabla 3.2

Continuación de tabla	
\$16,499.00	HP Pavilion15", Windows 10, Core i5-9300H, 8GB RAM.
\$1,599.00	DeskJet Ink Advantage 3775

Nombre	Costo
Windows 10	\$3,599.00
Ubuntu	Gratuito
MongoDB Atlas	
Visual Studio Code	Gratuito
Flask	Software libre
Vue.js	Software libre
Nuxt	Software libre

Es necesario detallar que el costo de la herramienta no será determinado puramente con los costos de software y hardware descritos dentro de esta sección, ya que son requeridos valores tales como el tiempo de desarrollo y el esfuerzo requeridos que serán especificados gracias al Modelo Constructivo de Costos (COCOMO II)

Gastos de operación

Gasto de operación	Gasto mensual (MXN)	Gasto total (MXN)
Luz	200	
Agua	200	
Internet	499	
Plan de datos	299	
Papelería	150	
Recursos humanos		

4.4. Factibilidad operativa

Factibilidad operativa

La herramienta va dirigida hacia cualquier estudiante que se encuentre trabajando con modelos de bases de datos desde un enfoque conceptual como lo requieren las primeras etapas del aprendizaje sobre el tema, buscando principalmente lograr un impacto en los estudiantes que comienzan a tener una aproximación a los modelos no relacionales de bases de datos.

Se cuenta con una interfaz intuitiva para que el estudiante pueda visualizar la información de una manera ya conocida, como lo son los diagramas E-R y las opciones que ésta les brinde de manera comprensible, no requiriendo de un usuario o personal especializado para su utilización. El usuario contará con registro ligado a su cuenta de correo electrónico que le permitirá acceder a la herramienta de forma

única brindándole un almacenamiento de los proyectos que con esta puedan haberse realizado.

Dentro de la herramienta se encuentran tres funciones principales, en un desarrollo primario se cuenta con un diagramador entidad-relación el cual permite una visualización abstracta de la construcción de una base de datos que apoya en su correcta descripción gracias a mensajes sobre la conexión de elementos por medio de la validación del mismo.

El segundo bloque podrá recibir el diagrama entidad-relación anteriormente tratado, o bien, trabajar sobre uno cuyo desarrollo y carga se haya realizado con antelación. Posteriormente el diagrama podrá ser transformado al modelo relacional con la posibilidad de obtener el esquema de la base de datos en sentencias SQL. Finalmente si es requerido, se obtendrá el esquema de base de datos en un modelo de datos no relacional y tendrá la posibilidad de implementar el modelo no relacional a un SGBDNR (Sistema Gestor de Base de Datos No Relacional).

Basándonos en las necesidades y tecnologías mencionadas a utilizar para el desarrollo de la herramienta, dentro del personal se deberá contar con un perfil como se describe a continuación:

Analista de sistemas:

Se entiende por analista a la persona que trabaja con los requerimientos, en el diseño global y en el diseño detallado. Los principales atributos que deberían considerarse en un analista son:

- Liderazgo
- Dedicación
- Habilidad para el diseño,
- Análisis de requerimientos
- Correcta comunicación
- Cooperación entre sus pares.

El salario promedio para un puesto de Analista de sistema en México es de \$11,708 al mes. Las estimaciones de salarios se basan en 564 salarios que empleados y usuarios que trabajan de Analista de sistema enviaron a Indeed de forma anónima, y en los salarios que recopilamos de los anuncios de empleo que se publicaron en Indeed en los últimos 36 meses. [<https://www.indeed.com.mx/salaries/analista-de-sistema-Salaries>]

Desarrollador

Las tendencias actuales siguen enfatizando la importancia de la capacidad de los analistas. Sin embargo, debido a que la productividad se ve afectada notablemente por la habilidad del programador en el uso de las herramientas actuales, existe una tendencia a darle mayor importancia a la capacidad del programador.

- Front end.

El desarrollador front-end es un especialista encargado de diseñar la interfaz de usuario de los sitios web, es decir, es el encargado de traducir las definiciones de diseño y estilo visual realizadas en etapas previas a código semántico. Diseña la estructura, la tipografía, la colorimetría, imágenes, banners, etc.

El salario promedio de Front End en México es de \$27.500-Mes Basado en 777 salarios [<https://neuvo.com.mx/salario/?job=front+end>]

- Back end.

El desarrollador back-end es el encargado de implementar la interacción entre el usuario y el sitio web. Se trata del responsable de la programación de un sitio web y todos sus componentes, coordinando páginas, formularios, funcionalidades, bases de datos y servidores web, evitando problemas en las capas más profundas del proyecto.

El salario promedio de Back End en México es de \$29.000-Mes Basado en 429 salarios [<https://neuvo.com.mx/salario/?job=back+end>]

Bibliografía

- [1] E. F. Codd, «A relational model of data for large shared data banks», *Communications of the ACM*, vol. 13, DOI: [10.1145/362384.362685](https://doi.org/10.1145/362384.362685).
- [2] C. Li, «Transforming relational database into HBase: A case study», en *2010 IEEE International Conference on Software Engineering and Service Sciences*, jul. de 2010, págs. 683-687. DOI: [10.1109/ICSESS.2010.5552465](https://doi.org/10.1109/ICSESS.2010.5552465).
- [3] A. Chebotko, A. Kashlev y S. Lu, «A Big Data Modeling Methodology for Apache Cassandra», en *2015 IEEE International Congress on Big Data*, jun. de 2015, págs. 238-245. DOI: [10.1109/BigDataCongress.2015.41](https://doi.org/10.1109/BigDataCongress.2015.41).
- [4] M. J. Mior, K. Salem, A. Aboulmaga y R. Liu, «NoSE: Schema Design for NoSQL Applications», *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, n.º 10, págs. 2275-2289, oct. de 2017, ISSN: 2326-3865. DOI: [10.1109/TKDE.2017.2722412](https://doi.org/10.1109/TKDE.2017.2722412).
- [5] D. Martínez-Mosquera, R. Navarrete y S. Luján-Mora, «Modeling and Management Big Data in Databases—A Systematic Literature Review», *Sustainability*, 2020. DOI: <https://doi.org/10.3390/su12020634>.
- [6] Ramez Elmasri, *Fundamentos de Sistemas de Bases de Datos*. Pearson, ISBN: 84-7829-085-0.
- [7] P. Chen, *The entity-relationship model: Toward a unified View of data*. dirección: <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.523.6679>.
- [8] Catherine M. Ricardo, *Bases de datos*. McGraw-Hill, ISBN: 978-970-10-7275-2.
- [9] Cristina Marta Bender, Claudia Deco, Juan Sebastián González Sanabria, María Hallo y Julio César Ponce Gallegos, *Tópicos Avanzados de Bases de Datos*, 1.ª ed. Iniciativa Latinoamericana de Libros de Texto Abiertos.
- [10] E. Brewer, «Towards robust distributed systems», ene. de 2000. DOI: [10.1145/343477.343502](https://doi.org/10.1145/343477.343502).
- [11] C. Coronel y S. Morris, *Database Systems: Design, Implementation, and Management*, 13.ª ed. Cengage, ISBN: 978-1-337-62790-0.
- [12] P. J. Sadalage y M. Fowler, *Nosql Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, 1.ª ed. Pearson Education, ISBN: 978-0-321-82662-6.
- [13] *HTTP - Hypertext Transfer Protocol*. dirección: <https://www.w3.org/Protocols/>.
- [14] *HTML*. dirección: <https://developer.mozilla.org/es/docs/Web/HTML>.

-
- [15] *What is CSS?* Dirección: <https://www.w3.org/standards/webdesign/htmlcss#whatcss>.
 - [16] *Documentation Sass*. dirección: <https://sass-lang.com/documentation>.
 - [17] *JavaScript*. dirección: <https://developer.mozilla.org/es/docs/Web/JavaScript>.
 - [18] T. Publishing, *TypeScript Programming Language*, 1.^a ed. Independently published, nov. de 2019, ISBN: 1-70883-980-1.
 - [19] *stack overflow - Developer Survey Results 2019 - Most Popular Technologies*. dirección: <https://insights.stackoverflow.com/survey/2019#most-popular-technologies>.
 - [20] *Web framework*. dirección: https://en.wikipedia.org/wiki/Web_framework.
 - [21] *¿Qué es Vue.js?* Dirección: <https://es.vuejs.org/v2/guide/>.
 - [22] *What is NuxtJS?* Dirección: <https://nuxtjs.org/guide>.
 - [23] *Introduction Material Design*. dirección: <https://material.io/design/introduction#goals>.
 - [24] *Vuetify Component API Overview*. dirección: <https://vuetifyjs.com/en/components/api-explorer/>.
 - [25] *Documentation Bootstrap*. dirección: <https://getbootstrap.com/docs/4.5/getting-started/introduction/>.
 - [26] *MongoDB*. dirección: <https://es.wikipedia.org/wiki/MongoDB>.
 - [27] *GoJs*. dirección: <https://gojs.net/latest/index.html>.
 - [28] M. Lutz, *Learning Python*, 5.^a ed. O'Really, jun. de 2013, ISBN: 978-1-4493-5573-9.