PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

DISERTACIÓN PREVIA A LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN SISTEMAS Y COMPUTACIÓN

"Estudio comparativo de motores de bases de datos SQL y NoSQL para la gestión de información transaccional"

Baldassari Valencia Holger David

Director: Javier W. Condor

Quito, 2019

Contenido

Tabla de Figuras

Índice de Tablas

Resumen

La presente disertación tiene como objetivo principal realizar un estudio comparativo de motores de bases de datos SQL y NoSQL para la gestión de información transaccional. Este se centra en las pruebas realizadas en cada uno de los motores seleccionados en el presente trabajo de titulación.

Este trabajo pretende dar una mejor idea de que motor de bases de datos seleccionar con respecto al proyecto o caso que se presente. Los motores SQL se han considerado la mejor opción para la transaccionalidad puesto que contiene las características necesarias para manipular este tipo de información. Mientras que los motores NoSQL últimamente han agregado características que permitan realizar transacciones y por el momento no ha sido razón para su consideración dentro de esta área.

Las pruebas propuestas en el presente trabajo de titulación se las ejecutará en un motor SQL y NoSQL seleccionados bajo los criterios de costo, versión, sistema operativo y el soporte a transacciones. La información que se utilizará será generada en una herramienta que, de alguna manera, genere información coherente.

Al final de este proceso se realizarán las pruebas mencionadas anteriormente que permitirán analizar los resultados de desarrollo en cada uno de los motores de bases de datos, tanto como aspectos positivos, negativos, objetivos cumplidos e inconsistencias que se presentaron a lo largo del trabajo. Se culmina con las conclusiones y recomendaciones referentes a los motores de bases de datos y herramientas utilizadas durante el trabajo.

1. Generalidades

En este capítulo se revisan las generalidades del trabajo de titulación, tales como la definición del problema, justificación para el mismo, los objetivos que se desean alcanzar y antecedentes los cuales fueron la razón para realizar el mismo.

1.1 Definición del problema

Para cualquier tipo de aplicación que utilice una base de datos es importante tener en cuenta la información que se va a manejar y que procesos se realizarán con dicha información, de esta manera se puede seleccionar un DBMS[1] el cual sea mejor para ello. También se debe considerar que comúnmente se seleccionan DBMS SQL[2] ya sea para el almacenamiento de datos y para procesos transaccionales, siendo esta última la principal razón para la utilización de estos DBMS, sin embargo, en la última década se popularizó los DBMS NoSQL[3] siendo esta la razón para llevar a cabo el siguiente estudio.

Como alternativa se puede considerar los DBMS NoSQL para lo que se refiere al almacenamiento y para las operaciones comunes de inserción, eliminación, actualización y consultas de datos. Pero cuando se presenta la gestión de información transaccional, la cual se define como "un grupo de operaciones que tienen las siguientes propiedades: atómica, consistente, aislada y duradera (*ACID*[4])." (Microsoft, 2018)(traducido del inglés), se consideran los DBMS SQL puesto que las características de este tipo de motores son ACID, permitiendo asegurar la confiabilidad de las transacciones sobre las bases de datos, pero este es diferente para su contraparte la cual plantea lo siguiente: el teorema de Brewer o CAP[5] y surge el acrónimo BASE[6] describiendo de esta manera las características de un DBMS NoSQL.

1.2 Justificación del trabajo de titulación

Existen varias diferencias relacionadas a los motores de bases de datos SQL y NoSQL; las cuales se las pueden considerar como ventajas o desventajas. El siguiente estudio analizará los dos tipos de motores de bases de datos para conocer la factibilidad del cambio de un motor de base de datos SQL a uno NoSQL para la gestión de información transaccional.

---

[1] DBMS. – *Database Management System. - Sistema de Gestion de Base de Datos.*
[2] SQL. – *Structured Query Language*. - Lenguaje de consulta estructurado.
[3] NoSQL. – *Not only SQL.*
[4] ACID. – *Atomicity, Consistency, Isolation, Durability*. ACID es un acrónimo de Atomicity, Consistency …
[5] CAP. – *Consistency, Availability, Partition tolerance*. Consistencia. - al realizar una consulta o inserción …
[6] BASE. – *Basically Available, Soft state, Eventual consistency. Es una filosofía de diseño de sistemas de datos …*

1.3 Objetivos del proyecto

- **Objetivo general:**
  - Realizar un estudio comparativo de motores de bases de datos SQL y NoSQL para la gestión de información transaccional
- **Objetivos específicos:**
  - Especificar un prototipo de información transaccional.
  - Generar una matriz comparativa entre DBMS SQL y NoSQL
  - Seleccionar y verificar un DBMS SQL con el prototipo de información transaccional.
  - Seleccionar y verificar un DBMS NoSQL con el prototipo de información transaccional.

1.4 Antecedentes

Las bases de datos NoSQL empezaron a tener popularidad a mediados de abril de 2009, por lo que es el concepto ya es más conocido sin embargo su uso no ha incrementado ya que muchos de los sistemas han decidido conservar las bases de datos SQL y la mayoría de la información que se puede obtener se basa en artículos, noticias y trabajos en formato virtual.

NoSQL puede ser definido como "*Not only SQL*" y es un sistema de gestión de bases de datos la cual podría ser la siguiente generación de tecnologías que es no relacional, distribuida, escalable horizontalmente, de código abierto y más rápida, puesto que no implementa las propiedades ACID las cuales aseguran la confiabilidad de las transacciones sobre las bases de datos. NoSQL fue definida inicialmente para actualizar y, de alguna manera, modernizar las bases de datos en la nube, pero su aplicación es de uso general, ya que la cantidad de información que se maneja y almacena en ellas, pueden no responder tan rápidamente como se espera en horas de alto tráfico de datos incumpliendo los ANS[7].

Entre las características que posee NoSQL se encuentra que no presentan esquemas, tienen fácil soporte de replicación, API[8] simple, eventualmente consistente (conocido como BASE, y contrario al concepto de ACID) y contienen enormes cantidades de datos. (Información NoSQL) Cuando el concepto se lo consideraba nuevo, se incursionaba en esta modalidad y se puede ver como Amazon y Oracle han creado bases de datos NoSQL; Twitter, Netflix, Facebook, Cisco, entre otros, utilizan estos tipos de motores; CouchDB y

[7] ANS. – Acuerdo de Nivel de Servicio.
[8] API. – *Application Programming Interface*. - Interfaz de programación de aplicaciones.

SQLite crearon un lenguaje de consulta llamado UnQL[9] (Jackson, 2011) . En la actualidad la diversidad de motores que existe es amplia por esta razón se debe realizar una revisión de estos y seleccionar aquel con las funciones de mayor utilidad.

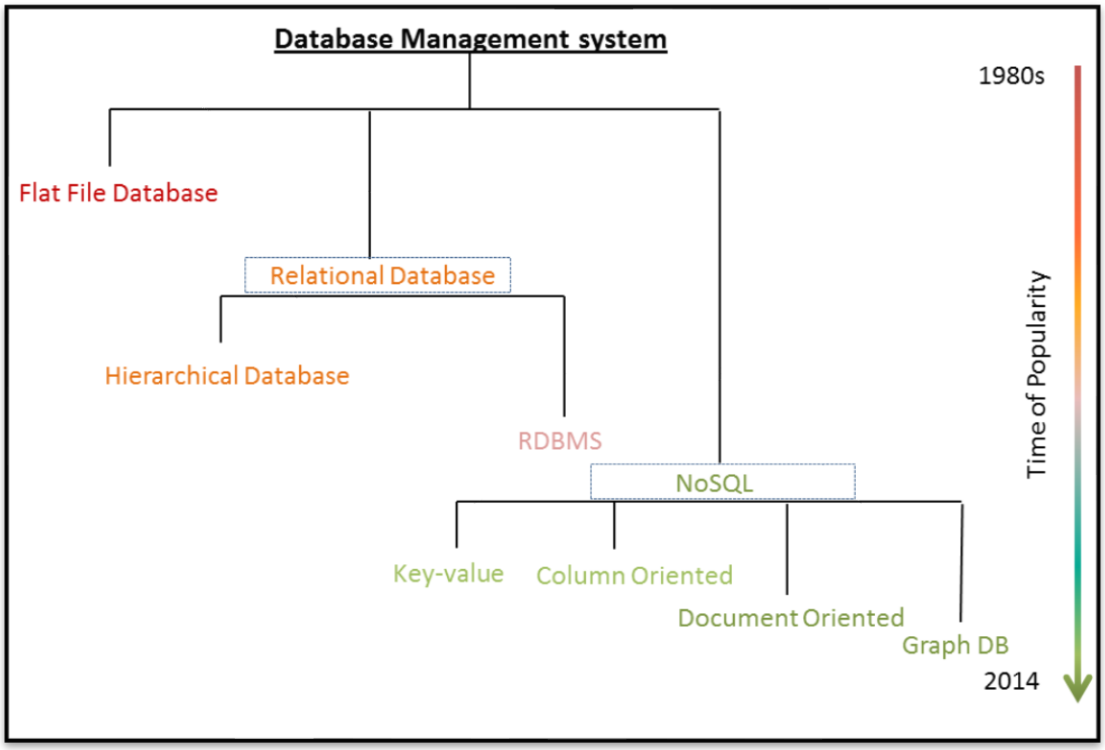


Figura 1-1 Evolución de los DBMS (Srivastava, 2014)

Como se puede observar en la figura 1-1 los DBMS empiezan como archivos planos para posteriormente evolucionar a bases de datos relacionales las cuales pasan a ser jerárquicas y posteriormente aparecen los RDBMS[10]. Al avanzar en la línea de tiempo las bases de datos evolucionan a NoSQL donde toman mayor importancia y a partir de ese momento aparecen ramificaciones como lo son clave-valor, orientada a columnas, orientadas a documentos y grafos en el 2014.

[9] UnQL. – *Unstructured Data Query Language. - Lenguaje de consulta de datos no estructurados.*
[10] RDBMS. – *Relational Database Management System. - Sistema de gestión de bases de datos relacionales.*

2. Marco teórico

En este capítulo se revisan las definiciones referentes a los motores de bases de datos, los datos, y las transacciones, también se define ciertas características tales como la concurrencia, replicación, distribución y seguridad. A partir de las diferentes definiciones se seleccionará una con la cual se trabajará durante la duración del trabajo.

2.1 Datos

Un dato se define por la RAE[11] como:

"1. m. Información sobre algo concreto que permite su conocimiento exacto o sirve para deducir las consecuencias derivadas de un hecho. A este problema le faltan datos numéricos.

2. m. Documento, testimonio, fundamento.

3. m. Inform. Información dispuesta de manera adecuada para su tratamiento por una computadora." (Real Academia Española, 2018).

La definición tomada por el diccionario de Cambridge "información, especialmente hechos o números, recopilados para ser examinados y considerados y utilizados para ayudar a la toma de decisiones, o información en forma electrónica que puede ser almacenada y utilizada por una computadora" (Cambridge Dictionary, n.d.)(traducido del inglés)

Una definición tomada del sitio "definicion.de" presenta a los datos en la informática como "expresiones generales que describen características de las entidades sobre las que operan los algoritmos. Estas expresiones deben presentarse de una cierta manera para que puedan ser tratadas por una computadora. En este caso, los datos por sí solos tampoco constituyen información, sino que ésta surge del adecuado procesamiento de los datos." (Merino, 2009)

Como se puede apreciar tanto la definición de la RAE como la de Cambridge concuerdan en que esta debe ser tratada por una computadora lo cual puede ser realizado por archivos o bien por los DBMS[12] existentes tanto SQL[13] como NoSQL[14], por esta razón la definición de Cambridge será la que se utilizará puesto que habla directamente sobre cómo se va a tratar a esos datos.

[11] RAE. - Real Academia Española.
[12] DBMS. – *Database Management System*. - *Sistema de Gestion de Base de Datos.*
[13] SQL. – *Structured Query Language*. - Lenguaje de consulta estructurado.
[14] NoSQL. – *Not only SQL*.

2.2 Información

A la información se la puede definir según el contexto en el cual se aplica, pero para el área de computación se han encontrado las siguientes definiciones.

En el sitio web de *"Tutorials Point"* define a la información como "son datos organizados o clasificados, que tiene algunos valores significativos para el receptor. La información es la data procesada en la que se basan las decisiones y las acciones." (Tutorials Point, 2018)(traducido del inglés).

Otra de las definiciones obtenidas para información es la siguiente "La información es un estímulo que tiene significado en algún contexto para su receptor. Cuando la información se ingresa y se almacena en una computadora, generalmente se la denomina data. Después del procesamiento (como el formateo y la impresión), los datos de salida pueden volver a percibirse como información." (Rouse, 2005)(traducido del inglés).

La definición tomada del sitio "*Ecomputer Notes*" es la siguiente "son datos que ha sido procesada de tal manera que sea significativa para la persona que la recibe." (Thakur, 2018)(traducido del inglés).

Como se puede observar las tres definiciones coinciden en que la información son los datos procesados de tal manera que estos sean significativos para el receptor, siendo de esta manera la definición que se utilizará durante el trabajo será la obtenida de "*Ecomputer Notes*" puesto que es la más simplificada y concreta de las tres.

2.3 DBMS

DBMS por sus siglas en inglés referentes a Database Management System, o en español Sistema de Gestión de Base de Datos (SGBD) se define como:

"Un sistema de gestión de bases de datos consiste en una colección de datos interrelacionados en un conjunto de programas para acceder a esos datos." (Kedar, 2009)(traducido del inglés).

"Un DBMS es, por lo tanto, una colección de datos interrelacionados y un conjunto de programas que administran o controlan el uso de los datos." (PAKHIRA, 2012)(traducido del inglés)

Las principales funciones de un DBMS según Malay K. Pakhira en su libro "DATABASE MANAGEMENT SYSTEM" son los siguientes:

- Define estructuras de datos para el almacenamiento de datos

- Proporciona un mecanismo adecuado para el acceso a datos y la manipulación de datos
- Mantiene la integridad del sistema
- Proporciona medidas de seguridad para datos
- Proporciona un mecanismo para el intercambio de datos entre usuarios concurrentemente
  (traducido del inglés).

En el mismo libro se puede observar la siguiente figura la cual brinda una idea general del esquema de una base de datos.

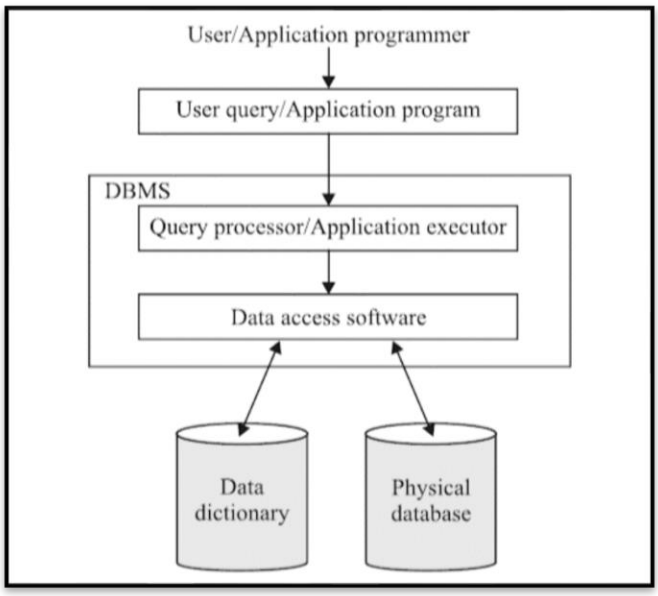


Figura 2-1 Esquema de un sistema de base de datos (PAKHIRA, 2012)

Como se puede observar en la figura 2-1 la manera de interactuar con los DBMS es a través de una aplicación para el usuario final, la misma que realiza una consulta o *query*, el cual en el DBMS se procesa y posteriormente se accede a los datos con la consulta procesada y finalmente se obtiene la información o se realiza cualquiera de los procesos que dicha consulta anteriormente solicitó. A continuación, se explicarán los componentes de un DBMS, para que de esta manera se comprenda de mejor manera el gráfico presentado.

2.3.1 Componentes de un DBMS

Los componentes de un DBMS listados por el sitio “Data Entry Outsourced” son:

- “**El software. -** Este es el conjunto de programas utilizados para controlar y administrar la base de datos general. Esto incluye el propio software DBMS, el

sistema operativo, el software de red que se utiliza para compartir los datos entre los usuarios y los programas de aplicación utilizados para acceder a los datos en el DBMS.

- **El Hardware. -** Consiste en un conjunto de dispositivos electrónicos físicos, como computadoras, dispositivos de E / S, dispositivos de almacenamiento, etc., esto proporciona la interfaz entre las computadoras y los sistemas del mundo real.
- **Datos. -** DBMS existe para recopilar, almacenar, procesar y acceder a datos, el componente más importante. La base de datos contiene los datos reales u operativos y los metadatos.
- **Procedimientos. -** Estas son las instrucciones y reglas que ayudan a utilizar DBMS, y al diseñar y ejecutar la base de datos, utilizando procedimientos documentados, para guiar a los usuarios que la operan y la administran.
- **Idioma de acceso a la base. -** Esto se usa para acceder a los datos desde y hacia la base de datos, ingresar datos nuevos, actualizar datos existentes o recuperar los datos requeridos de las bases de datos. El usuario escribe un conjunto de comandos apropiados en un lenguaje de acceso a la base de datos, los envía al DBMS, que luego procesa los datos y genera y muestra un conjunto de resultados en un formulario legible por el usuario.
- **Procesador de consultas. -** Esto transforma las consultas del usuario en una serie de instrucciones de bajo nivel. Esto lee la consulta del usuario en línea y la traduce en una serie eficiente de operaciones en una forma que se puede enviar al administrador de datos en tiempo de ejecución para su ejecución.
- **Administrador de bases de datos de ejecución. -** A veces conocido como el sistema de control de la base de datos, este es el componente central de software del DBMS que interactúa con los programas y consultas de las aplicaciones enviadas por el usuario, y maneja el acceso a la base de datos en tiempo de ejecución. Su función es convertir operaciones en consultas de usuario. Proporciona control para mantener la consistencia, integridad y seguridad de los datos.
- **Administrador de datos. -** También llamado el administrador de caché, este es responsable del manejo de los datos en la base de datos, proporcionando una recuperación al sistema que le permite recuperar los datos después de una falla.
- **Motor de base de datos. -** El servicio principal para almacenar, procesar y asegurar datos, proporciona acceso controlado y procesamiento de transacciones rápido para satisfacer los requisitos de las aplicaciones de consumo de datos más exigentes. A menudo se usa para crear bases de datos relacionales para

procesamiento de transacciones en línea o datos de procesamiento analítico en línea.

- **Diccionario de datos. -** Este es un espacio reservado dentro de una base de datos utilizada para almacenar información sobre la base de datos en sí. Un diccionario de datos es un conjunto de tablas y vistas de solo lectura, que contiene la información diferente sobre los datos utilizados en la empresa para garantizar que la representación de la base de datos siga un estándar definido en el diccionario.
- **Generador de informes. -** Es un programa que extrae información de uno o más archivos y presenta la información en un formato específico. La mayoría de los autores de informes permiten al usuario seleccionar registros que cumplan con ciertas condiciones y mostrar los campos seleccionados en filas y columnas, o también formatear los datos en diferentes gráficos."
  (Data Entry Outsourced, 2013)(traducido del inglés).

2.4 DBMS SQL

Los DBMS SQL por lo general son considerados RDBMS[15] o Sistema de gestión de bases de datos relacional (SGBDR) puesto que se define como: "un método estructurado para almacenar datos en forma de tablas, es decir, filas y columnas. La base de datos relacional también se denomina base de datos SQL, ya que está escrita utilizando el lenguaje de consulta estructurada (SQL)." (New Gen Apps, 2017)(traducido del inglés).

Para Amazon un RDBMS o DBMS SQL lo considera como: "una recopilación de elementos de datos con relaciones predefinidas entre ellos. Estos elementos se organizan como un conjunto de tablas con columnas y filas. Las tablas se utilizan para guardar información sobre los objetos que se van a representar en la base de datos. Cada columna de una tabla guarda un determinado tipo de datos y un campo almacena el valor real de un atributo. Las filas de la tabla representan una recopilación de valores relacionados de un objeto o entidad. Cada fila de una tabla podría marcarse con un identificador único denominado clave principal, mientras que filas de varias tablas pueden relacionarse con claves extranjeras. Se puede obtener acceso a estos datos de muchas formas distintas sin reorganizar las propias tablas de la base de datos." (Amazon, 2018)

Pero la definición a continuación señala que no es necesario el uso del lenguaje de consulta estructurado para que un DBMS se considere RDBMS teniendo como representantes *TurboIMAGE* que usaba un API[16] para todo el acceso a la base de datos y *pick operating*

[15] RDBMS. – *Relational Database Management System*. - *Sistema de gestión de bases de datos relacionales.*

[16] API. – *Application Programming Interface.* - Interfaz de programación de aplicaciones.

system el cual utilizaba un lenguaje similar al SQL llamado "*ENGLISH"* y el lenguaje de programación llamado "*Databasic*".

"Un sistema de gestión de bases de datos relacionales (RDBMS) es un programa que te permite crear, actualizar y administrar una base de datos relacional. La mayoría de los RDBMS comerciales utilizan el lenguaje de consultas estructuradas (SQL) para acceder a la base de datos, aunque SQL fue inventado después del desarrollo del modelo relacional y no es necesario para su uso." (TechTarget, 2018)

La idea del funcionamiento de estos DBMS se representa en la siguiente figura:

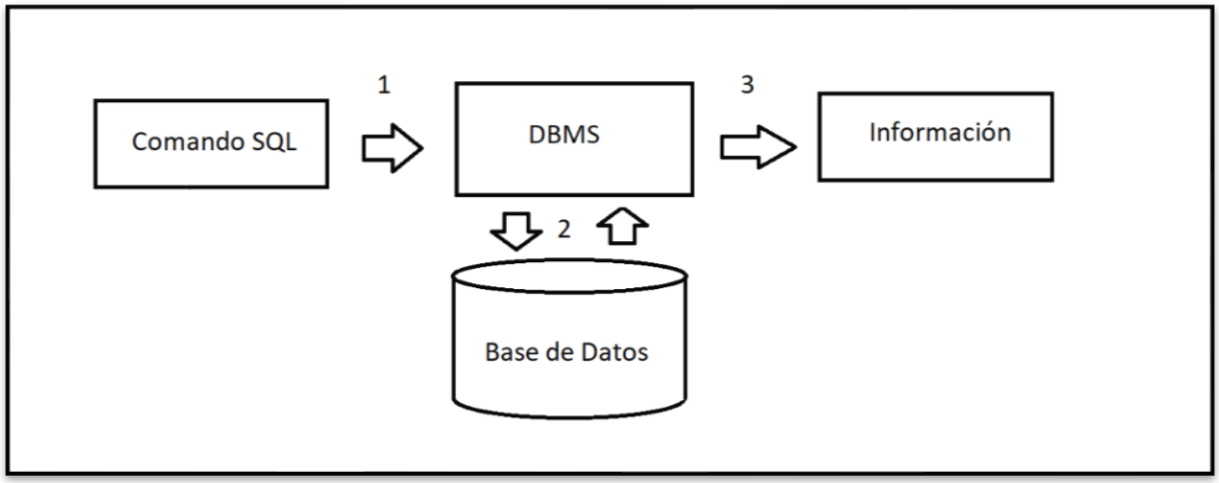


Figura 2-2 La forma en que SQL interactúa con la base de datos (Baldassari, 2018)

Como se puede observar en la figura 2-2 la consulta se realiza a través de un comando SQL o *query*, posteriormente el DBMS se conecta con la base de datos y se obtiene la información o se realizan los procesos correspondientes al comando SQL, finalmente la base de datos devuelve la respuesta al comando y el DBMS finalmente devuelve la información.

2.4.1 Dependencia funcional DBMS SQL

Cuando se da una relación muchos a muchos entre dos entidades o tablas los DBMS SQL en su modelo deben generar una tercera entidad la cual contendrá las claves primarias de las entidades relacionadas.

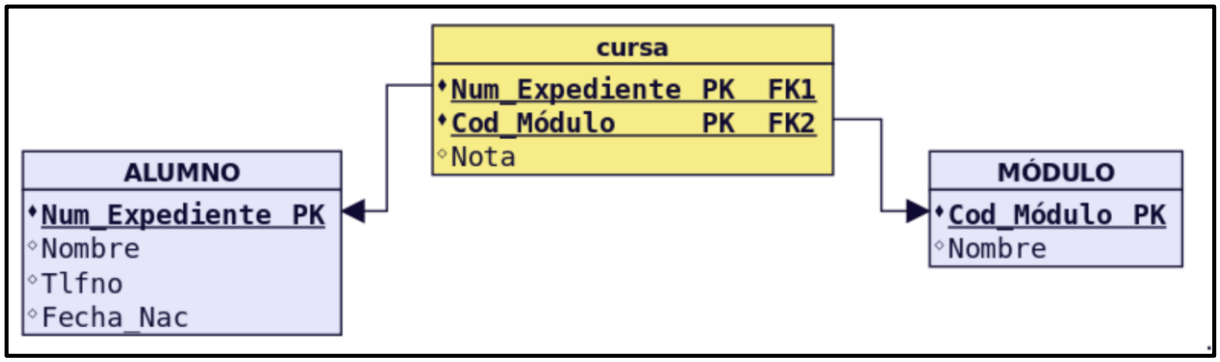


Figura 2-3 Relación muchos a muchos genera una entidad débil. (Gestión de Bases de Datos, 2018)

En la figura 2-3 se observa como la relación cursa contiene una clave primaria compuesta por dos claves foráneas correspondientes a las claves primarias de las entidades Alumno y módulo.

2.5 DBMS NoSQL

Un DBMS NoSQL (not only SQL) es definido por Oracle como "una amplia clase de sistemas de gestión de datos (mecanismos para el almacenamiento y recuperación de datos) que difieren, en aspectos importantes, del modelo clásico de relaciones entre entidades (o tablas) existente en los sistemas de gestión bases de datos relacionales, siendo el más destacado el que no usan SQL como lenguaje principal de consulta." (Oracle, 2016)

"Hoy en día, NoSQL se utiliza como un término genérico para todas las bases de datos y almacenes de datos que no siguen los principios RDBMS populares y bien establecidos y que a menudo se relacionan con grandes conjuntos de datos accedidos y manipulados en una escala web." (Tiwari, 2011)(traducido del inglés).

Existen 4 tipos DBMS NoSQL los cuales están listados en la página de MongoDB y son los siguientes:

- "**Clave valor (*Key-value stores*). -** son las bases de datos NoSQL más simples. Cada elemento de la base de datos se almacena como un nombre de atributo (o "clave") junto con su valor. Los ejemplos incluyen Riak, Voldemort y Redis.
- **Orientadas a columnas (*Wide-column stores*). -** almacena columnas de datos juntas en lugar de filas y esté optimizado para consultas sobre grandes conjuntos de datos. Cassandra y HBase son bases de datos wide-column.
- **Orientadas a documentos (*Document databases*). -** emparejar cada clave con una estructura de datos compleja conocida como documento. Los documentos pueden contener muchos pares clave-valor diferentes, pares de pares de claves o incluso documentos anidados. MongoDB es una base de datos de documentos.

- **En grafo (*Graph databases*). -** se utilizan para almacenar información sobre redes, como las conexiones sociales. Los ejemplos incluyen Neo4J y HyperGraphDB. Las bases de datos NoSQL están ganando popularidad a medida que las empresas las aplican a un número creciente de casos de uso."
  (MongoDB, 2018)(traducido del inglés).

En el sitio *GMV Innovating Solutions* se presenta la clasificación anterior, pero en esta se señala varios ejemplos para cada uno de ellos:

- "**Orientadas a documentos** (son las más versátiles, gestionan datos semi-estructurados o documentos que son almacenados en algún formato estándar como puede ser XML, JSON o BSON). Ejemplos: MongoDB (utilizada por Foursquare o eBay), CouchDB, Couchbase Server, MarkLogic, …
- **Orientadas a columnas** (pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos, funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros). Ejemplos: Accumulo, Cassandra, HBase (mantenida por Hadoop; utilizada por Facebook, Twitter o Yahoo), …
- **De clave-valor** (las más simples de entender, guardan tuplas clave-valor). Ejemplos: Aerospike, Redis, Riak, DynamoDB, …
- **En grafo** (basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados, muy útiles para guardar información en modelos con muchas relaciones como redes y conexiones sociales). Ejemplos: InfiniteGraph, Neo4j (utilizada en Infojobs), …"

(Díaz, 2014)

La principal diferencia entre las DBMS SQL y NoSQL se encuentra en que estas últimas comúnmente almacenan la data de manera no estructurada, puesto que no tiene una estructura de tabla fija como se presenta en los DBMS SQL

2.5.1 Dependencia funcional DBMS NoSQL

En el ámbito NoSQL no se presentan las relaciones como tal, y de esta manera el concepto de clave primaria y foránea no se las puede presentar, sin embargo, existe el concepto de índices los cuales son referentes al documento o al objeto.

Figura 2-4 Estructura de dependencia funcional NoSQL (Advaiya, 2015)

En la figura 2-4 se observa la estructura a manejar en MongoDB, en el ejemplo existen dos tablas User y User_Information, siendo la segunda la que contendrá como clave foránea el índice o clave primaria de la tabla *User.* Como se puede apreciar en el campo *User_Id* será el campo de la clave foránea y *User_Info_Id* será la clave primaria de esta tabla.

2.6 Teorema de Brewer o CAP

El teorema indica que **un sistema de datos distribuido pude asegurar dos de estas tres propiedades: Consistencia, Disponibilidad y Tolerancia al particionado (del inglés *Consistency, Availability, Parition Tolerance*).**

Las tres características mencionadas anteriormente se definen de la siguiente manera en el sitio "*Genbeta"*:

- "**Consistencia**. - al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición.
- **Disponibilidad**. - que todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos.
- **Tolerancia al particionado**. - Los sistemas distribuidos pueden estar divididos en particiones (generalmente de forma geográfica). Así que esta condición implica, que el sistema tiene que seguir funcionando, aunque existan fallos o caídas parciales que dividan el sistema." (Genbeta, 2018)

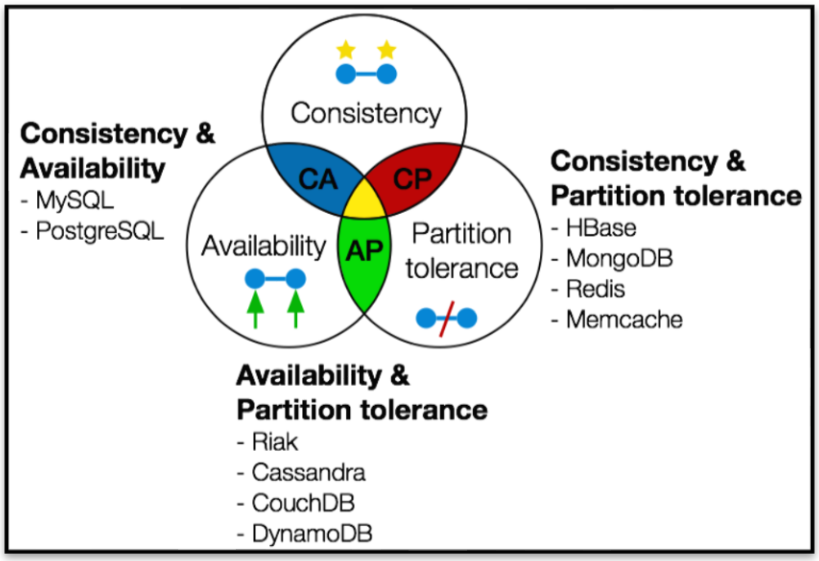


Figura 2-5 Ejemplo del Teorema de Brewer (Strappazzon, 2016)

Como se puede observar en la figura 2-5 el teorema de Brewer indica que solo se pueden presentar dos de estas características simultáneamente y en el caso de MongoDB el cual se centra en la consistencia y en la tolerancia al particionado, pero se debe considerar que está presente la opción de su configuración de tal manera que acepte la disponibilidad, pero la consistencia no será la misma.

2.7 Información Transaccional

Existen un gran número de transacciones diarias en organizaciones, el resultado de todas estas transacciones es la información transaccional. La principal función de la información transaccional es proporcionar ayuda a la realización de tareas operativas de las organizaciones.

A diferencia de la información transaccional, la información analítica se encarga de la toma de decisiones para la organización. Esto es usado por el personal que necesita una visión general de la información transaccional.

Las bases de datos son capaces de manejar todo tipo de transacciones, teniendo como principal objetivo garantizar la precisión e integridad de la información. Por esta razón es necesario que la información almacenada en la base de datos sea consistente y acertada.

Generalmente las bases de datos usadas para la transaccionalidad son SQL puesto que cumple con la definición de ACID[17] ya que como se mencionó anteriormente la definición

[17] ACID. – *Atomicity, Consistency, Isolation, Durability*. ACID es un acrónimo de Atomicity, Consistency …

de transaccionalidad indica que “un grupo de operaciones que tienen las siguientes propiedades: atómica, consistente, aislada y duradera (ACID).” (Microsoft, 2018).

Una transacción está compuesta por varios pasos:

1. Inicio de la transacción.
2. Ejecución de los *queries* o manipulación de la información.
3. Si ningún error ocurre realizar un *commit* o confirmar la transacción y finalizar.
4. Si ocurre un error deshacer la transacción y finalizar.

El *commit* de la operación es el más importante de la transacción puesto que, una vez se ejecuta este comando normalmente la transacción se ha realizado con éxito. Pero si por algún error este comando no se ejecuta entonces la información que fue manipulada en la transacción regresa a su estado anterior y no se guarda en la base ninguna modificación por esta razón se puede tener confianza que la base de datos contiene únicamente información confiable.

2.8 Concurrencia

Se define a la concurrencia como “la capacidad de una base de datos para permitir que múltiples usuarios afecten múltiples transacciones. Esta es una de las principales propiedades que separa una base de datos de otras formas de almacenamiento de datos como hojas de cálculo.” (Techopedia, 2018)(traducido del inglés)

En el libro “Introducción a los sistemas de bases de datos” se define a la concurrencia como “los DBMS permiten que muchas transacciones accedan a una misma de datos a la vez.” (Date, 2001)

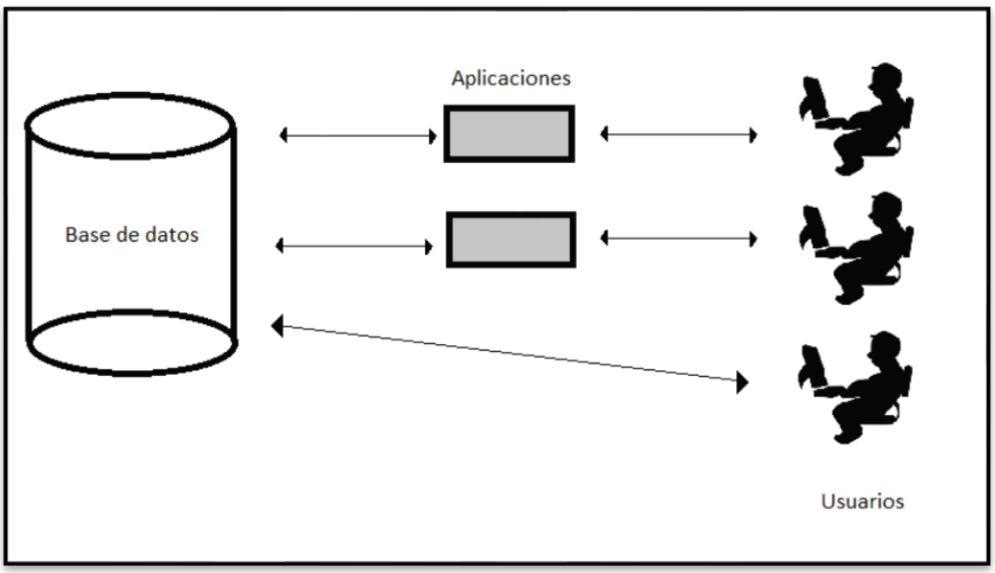


Figura 2-6 Concurrencia (Baldassari, 2018)

Como se puede observar en la figura 2-6 existe concurrencia puesto que varios usuarios acceden a la base de datos de distintas aplicaciones e incluso una de ellas accede directamente, y también existe la posibilidad de que se esté realizando distintas transacciones.

2.9 Replicación

El sitio "*Techopedia*" indica que la replicación se define como "una técnica mediante la cual una instancia de una base de datos se copia, se transfiere o se integra exactamente con otra ubicación. La replicación de la base de datos permite la copia de un archivo de base de datos desde un sistema maestro de administración de bases de datos (DBMS) y su implementación exacta en un DBMS esclavo." (Techopedia, 2018)(traducido del inglés).

La definición del sitio "Power Data" no difiere de la anterior puesto que indica que "Una replicación de base de datos es una técnica mediante la cual copiamos de forma exacta en otra ubicación una instancia de la base de datos. Se utiliza en entornos distribuidos de Sistemas de Gestión de Bases de Datos donde una sola base de datos tiene que ser utilizada y actualizada en varios lugares de forma simultánea." (Power Data, 2016)

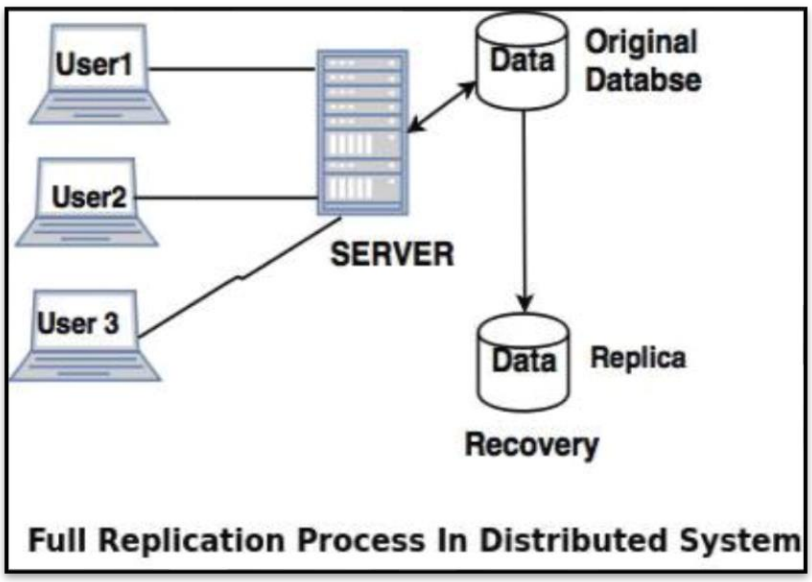


Figura 2-7 Replicación Completa (TutorialRIde, 2018)

En la figura 2-7 se puede apreciar que los usuarios acceden a la base de datos original pero esta base se conecta con una en la cual se replica la información la cual se puede utilizar como recuperación en caso de cualquier daño a la base de datos original.

2.10 Distribución

En los DBMS la distribución corresponde a separar la información que corresponde a un sistema tal y como indica la siguiente definición: “Son la que almacenan datos que pertenecen lógicamente a un sólo sistema, pero se encuentra físicamente esparcido en varios “sitios” de la red. Un sistema de base de datos distribuidos se compone de un conjunto de sitios, conectados entre sí mediante algún tipo de red de comunicaciones” (Lopez & Colmenarez, 2012)

El sitio “*Tutorials Point*” indica que “Una base de datos distribuida es una colección de múltiples bases de datos interconectadas, que se extienden físicamente a través de varias ubicaciones que se comunican a través de una red informática.” (Tutorials Point, 2018)

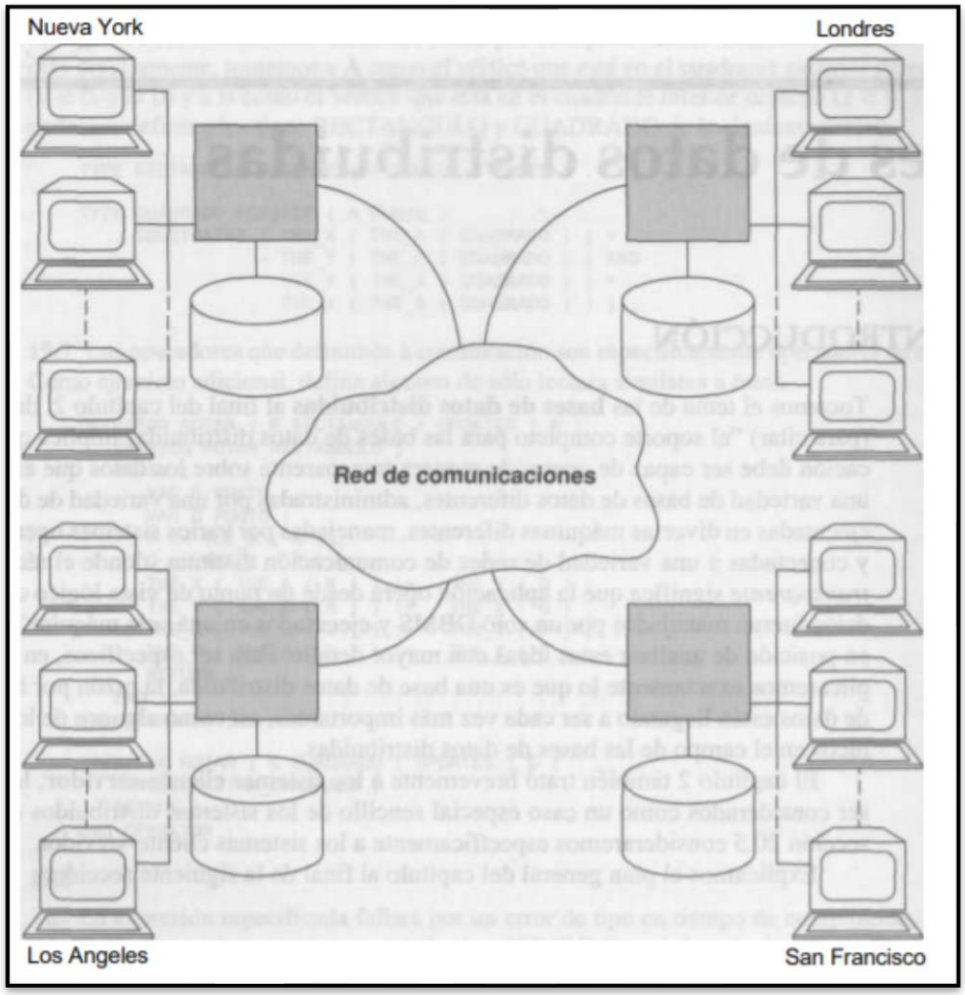


Figura 2-8 Base de datos distribuida (Date, 2001)

Como se puede ver en la figura 2-8 cada lugar tiene un sistema con una base de datos original pero estos sistemas también tienen conexión a las bases de los distintos lugares por medio de la red de comunicaciones, es decir que bien cada una de las bases de datos contienen una parte de la información de una base de datos de gran tamaño y se coopera para el procesamiento, o cada una de estas bases tiene un esquema distinto del resto y distinta información pero esta puede ser obtenida de cualquier ubicación.

2.11 Seguridad

Dado que la información que contiene una base de datos es de gran importancia se debe considerar la definición de seguridad de datos la cual indica que: “En líneas generales, seguridad de datos se refiere a medidas de protección de la privacidad digital que se aplican para evitar el acceso no autorizado a los datos, los cuales pueden encontrarse en ordenadores, bases de datos, sitios web, etc.”

A partir de la definición anterior la seguridad en una base de datos se la puede definir de la siguiente manera: “al proceso de controlar el acceso a los recursos; se basa en las credenciales y los permisos del usuario.” (Microsoft, 2018)

“La seguridad de la base de datos se refiere a las medidas colectivas utilizadas para proteger y asegurar una base de datos o un software de administración de bases de datos del uso ilegítimo y amenazas y ataques maliciosos.” (Techopedia, 2018)(traducido del inglés)

3. Análisis comparativo de DBMS SQL y DBMS NoSQL

En este capítulo se tratará los criterios de selección de los DBMS[18] SQL[19] y NoSQL[20] con los cuales se procederá a realizar una elección de cada uno de ellos para su uso en el trabajo de titulación.

3.1 Criterios de selección

Los criterios bajo los que se realizará la selección de cada uno de los DBMS serán los siguientes:

3.1.1 Costo

Este factor es importante puesto que un DBMS puede permitir usar todas sus características de manera gratuita o simplemente dar acceso a funciones básicas las cuales podrían no contener las necesarias para el presente trabajo.

3.1.2 Versión

Este factor permitirá saber con cual versión del DBMS se trabajará y permitirá revisar la documentación adecuada para su uso, también permitirá tener en cuenta las características agregadas y que podrían ser de utilidad.

3.1.3 Sistema Operativo

Permitirá conocer sobre cuales sistemas operativos son capaces de trabajar los DBMS permitiendo seleccionar aquellos que trabajen, para este caso, en Windows facilitando el uso de las características del DBMS.

3.1.4 Soporta transacciones

El factor de soporte a transacciones es necesario puesto que, si el DBMS no soporta estas no se podría realizar el presente trabajo de titulación puesto que se necesita el manejo transaccional y en la actualidad varios DBMS NoSQL añadieron soporte para ello.

En las siguientes tablas que se presenta a continuación se revisará si los DBMS tanto SQL como NoSQL cumplen con estos criterios y si estos soportan transacciones lo cual es de gran importancia para el presente trabajo de titulación.

[18]DBMS. – *Database Management System. - Sistema de Gestion de Base de Datos.*
[19]SQL. – *Structured Query Language*. - Lenguaje de consulta estructurado.
[20] NoSQL. – *Not only SQL.*

DBMS SQL	Costo	Versión (estable)	Sistema Operativo	Soporta transacciones
Oracle	Si/No	18c (18.1)	Multiplataforma	Si
SQLServer	Si/No	14.0	Linux/ Microsoft Windows Server/ Microsoft Windows	Si
MySQL	Si/No	8.0.12	Multiplataforma	Si
PostgreSQL	No	10.6	Linux/Windows	Si
Microsoft Access	Si	14.0.6123. 5001	Microsoft Windows	Si
Teradata	Si	16.20	SUSE Linux Enterprise Server	Si
IBM DB2	Si	11.1	Multiplataforma	Si
Informix	Si	12.10.xC2	Multiplataforma	Si
SAP ASE	Si	16.0	Unix/Microsoft Windows	Si
Amazon's SimpleDB	Si/No	-	Alojado	No

Tabla 3-1 Top DMBS SQL (Baldassari, 2018)

DBMS NoSQL	Costo	Versión (estable)	Sistema Operativo	Soporta transacciones
MongoDB	No	4.0.2	Windows/Linux /OS X/Solaris/Free BSD	Si
Cassandra	No	3.11.1	Multiplataforma	No
Redis	No	4.0.11	Multiplataforma	Si
Hbase	No	1.2.1	Multiplataforma	Si
Neo4j	Si/No	3.3.0	Multiplataforma	Si
Oracle NoSQL	Si/No	18.1	Multiplataforma	Si
Amazon DynamoDB	Si/No	-	Multiplataforma	Si
CouchBase	No	5.5.0	Multiplataforma	Si
Memcached		1.5.11	Multiplataforma	Si
CouchDB	No	2.2.0	Multiplataforma	No

Tabla 3-2 Top DBMS NoSQL (Baldassari, 2018)

3.2 DBMS SQL Referentes

En la siguiente lista se tiene los 10 mejores DBMS SQL establecidos por el sitio “My Tech Decisions” en la siguiente URL: https://mytechdecisions.com/it-infrastructure/10-best-database-software-systems-business-professionals/ .

1. **Oracle**

“La base de datos Oracle (Oracle DB) es un sistema de administración de base de datos relacional (RDBMS) de Oracle Corporation. Desarrollado originalmente en 1977 por Lawrence Ellison y otros desarrolladores, Oracle DB es uno de los motores de bases de datos relacionales más confiables y más utilizados.

El sistema se basa en un marco de base de datos relacional en el que los usuarios (o un extremo de la aplicación) pueden acceder directamente a los objetos de datos a través del lenguaje de consulta estructurado (SQL). Oracle es una arquitectura de base de datos relacional totalmente escalable y a menudo es utilizada por empresas globales, que administran y procesan datos en redes de área amplia y local. La base de datos Oracle tiene su propio componente de red para permitir las comunicaciones a través de redes.

Oracle DB también se conoce como Oracle RDBMS y, a veces, solo Oracle.” (Technopedia, 2018)(traducido del inglés)

URL: https://www.oracle.com/technetwork/es/documentation/index.html#database

2. **Microsoft SQL Server**

“SQL Server es el sistema de gestión de bases de datos relacionales de Microsoft (RDBMS). Es una base de datos completa diseñada principalmente para competir contra los competidores Oracle Database (DB) y MySQL.

Como todos los principales RBDMS, SQL Server es compatible con ANSI SQL, el lenguaje SQL estándar. Sin embargo, SQL Server también contiene T-SQL, su propia implementación de SQL. SQL Server Management Studio (SSMS) (anteriormente conocido como Enterprise Manager) es la herramienta de interfaz principal de SQL Server y es compatible con entornos de 32 y 64 bits.

SQL Server a veces se conoce como MSSQL y Microsoft SQL Server.” (Technopedia, 2018)(traducido del inglés)

URL: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017

3. MySQL

"MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) con todas las funciones que compite con Oracle DB y Microsoft SQL Server. MySQL está patrocinado por la compañía sueca MySQL AB, que es propiedad de Oracle Corp. Sin embargo, el código fuente de MySQL está disponible de forma gratuita porque originalmente se desarrolló como software gratuito. MySQL está escrito en C y C ++ y es compatible con todos los principales sistemas operativos." (Techopedia, 2018)(traducido del inglés)

URL: https://dev.mysql.com/doc/

4. PostgreSQL

"PostgreSQL es un sistema de gestión de bases de datos de código abierto, objeto relacional (ORDBMS) que no es propiedad ni está controlado por una empresa o persona. Debido a que el software postgresSQL es de código abierto, se administra principalmente a través de un esfuerzo coordinado en línea por una comunidad global activa de desarrolladores, entusiastas y otros voluntarios.

Lanzado por primera vez a mediados de la década de 1990, postgresSQL está escrito en C. Sus competidores principales incluyen Oracle DB, SQL Server y MySQL." (Technopedia, 2018)(traducido del inglés)

URL: https://www.postgresql.org/docs/

5. Microsoft Access

"Microsoft Access es un motor de base de datos pseudo-relacional de Microsoft. Es parte del conjunto de aplicaciones de Microsoft Office que también incluye Word, Outlook y Excel, entre otras. El acceso también está disponible para su compra como un producto independiente. Access utiliza el motor de base de datos Jet para el almacenamiento de datos.

El acceso se utiliza para implementaciones de bases de datos pequeñas y grandes. Esto se debe en parte a su interfaz gráfica fácil de usar, así como a su interoperabilidad con otras aplicaciones y plataformas, como el propio motor de base de datos SQL Server de Microsoft y Visual Basic para aplicaciones (VBA)." (Technopedia, 2018)(traducido del inglés)

URL: https://docs.microsoft.com/en-us/office/client-developer/access/access-home

6. Teradata

"Teradata es un sistema de administración de bases de datos relacionales totalmente escalable producido por Teradata Corp. Se usa ampliamente para administrar grandes operaciones de almacenamiento de datos.

El sistema de base de datos Teradata se basa en una tecnología de multiprocesamiento simétrico estándar, combinada con redes de comunicación, que conecta sistemas de multiprocesamiento simétricos para formar grandes sistemas de procesamiento paralelo." (Technopedia, 2018)(traducido del inglés)

URL: https://docs.teradata.com/landing-page/

7. IBM DB2

"DB2 es un sistema de gestión de bases de datos relacionales (RDBMS) introducido originalmente por IBM en 1983 para ejecutarse en su plataforma de mainframe MVS (almacenamiento virtual múltiple). El nombre se refiere al cambio del modelo de base de datos jerárquico prevaleciente en ese momento al nuevo modelo relacional. Aunque inicialmente se diseñó DB2 para trabajar exclusivamente en plataformas de IBM mainframe, más tarde se adaptó a otros sistemas operativos ampliamente utilizados como UNIX, Windows y actualmente en Linux. DB2 es una parte integral del portafolio de administración de información de IBM. Es un motor de base de datos de alto rendimiento con todas las funciones, capaz de manejar grandes cantidades de datos y servir simultáneamente a muchos usuarios." (Technopedia, 2018)(traducido del inglés)

URL: http://www-01.ibm.com/support/docview.wss?uid=swg27009474

8. Informix

"IBM® Informix® es un servidor de bases de datos rápido y escalable que gestiona bases de datos relacionales, relacionales de objetos y dimensionales tradicionales. Su tamaño reducido y sus capacidades de autogestión se adaptan a las soluciones integradas de gestión de datos.

El servidor de base de datos IBM Informix se ejecuta en los sistemas operativos UNIX, Linux, Mac OS X y Windows.

Todas las ediciones de Informix contienen las siguientes herramientas de cliente además del servidor de base de datos: IBM Informix Client Software Development Kit (Client SDK) incluye las interfaces de programación de aplicaciones (API) para desarrollar aplicaciones y proporcionar conectividad al cliente.

IBM OpenAdmin Tool (OAT) para Informix es una aplicación web para administrar y analizar el rendimiento de los servidores de bases de datos Informix.

Informix DataBlade Developers Kit (DBDK) contiene herramientas para desarrollar y empaquetar módulos DataBlade, que son paquetes de software que amplían la funcionalidad del servidor de bases de datos.

Otras ediciones de Informix incluyen funcionalidad adicional, como herramientas de almacenamiento o capacidades en la nube.” (IBM, 2018)(traducido del inglés)

URL: https://www.ibm.com/support/knowledgecenter/en/SSGU8G/welcomeIfxServers.html

9. SAP ASE

“ASE es la abreviatura de "Adaptive Server Enterprise", el software de administración de bases de datos relacionales fabricado y vendido por Sybase, Inc. ASE es un RDBMS versátil y de clase empresarial que es especialmente bueno en el manejo de cargas de trabajo OLTP. ASE se utiliza de forma intensiva en el mundo financiero (bancos, bolsas de valores, compañías de seguros), en comercio electrónico y en prácticamente cualquier otra área.

La versión más reciente de ASE es la versión 15.7 de ASE (publicada en septiembre de 2011); La versión anterior es la versión 15.5.

ASE 15.7 también se conoce como "la versión de SAP", ya que esta es la versión de ASE que SAP está utilizando para admitir el paquete Business Suite ERP sobre Sybase ASE.” (Sypron, 2012)(traducido del inglés)

URL:
https://help.sap.com/saphelp_nw73ehp1/helpdata/en/14/95bd14c0564c19bf1fd94c01920c32/frameset.htm

10. Amazon SimpleDB

“Amazon SimpleDB es un servicio de base de datos distribuido desarrollado por Amazon.com en el lenguaje de programación Erlang. Lanzado el 13 de diciembre de 2007, Amazon SimpleDB

es un componente de servicios web de Amazon (AWS) que funciona en conjunto con Amazon Simple Storage Service (Amazon S3) y Amazon Elastic Compute Cloud (EC2).

Amazon SimpleDB facilita el almacenamiento en la nube, el procesamiento y el procesamiento de consultas. El servicio es fácil de usar y proporciona la mayoría de las funciones de base de datos tradicionales, incluido el procesamiento en tiempo real y las consultas de datos estructuradas simplificadas.” (Technopedia, 2018)(traducido del inglés)

URL: https://docs.aws.amazon.com/simpledb/index.html

3.3 Elección DBMS SQL

En la tabla 3-1 se puede observar que varios de estos DBMS son de pago, pero también constan con una versión gratuita, pero puesto que PostgreSQL brinda todas sus características de forma gratuita este será el DBMS SQL seleccionado para la elaboración del presente trabajo de titulación.

3.3.1 PostgreSQL

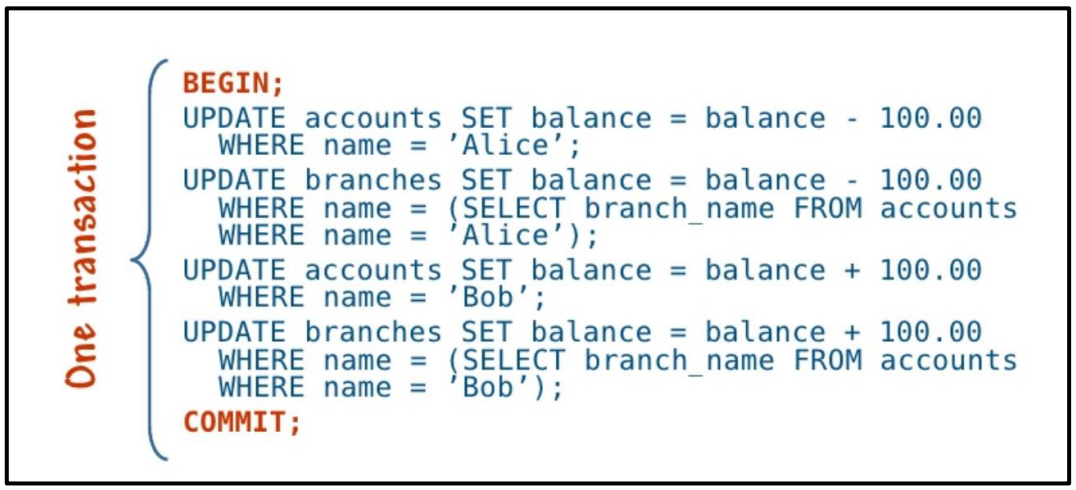


Figura 3-1 Transacciones en PostgreSQL (Steffensen, 2009)

En la figura 3-1 se puede ver como funciona una transacción en PostgreSQL, es decir en el comando SQL que se presenta, si en cualquier punto de este se presenta un error los cambios no se guardan puesto que no se ejecuta el comando *commit* el cual es el responsable de que dichos cambios sean guardados, de esta manera se asegura que la información en las tablas que anteriormente fueron alteradas no contenga información incorrecta.

A continuación, se presentan varias ventajas y desventajas del DBMS SQL PostgreSQL

Ventajas

- **Instalación ilimitada y gratuita. -** Este aspecto es importante puesto que, como se mencionó anteriormente permite tener acceso a todas las características de forma gratuita y sin el riesgo a que existan funciones que sean necesarias las cuales no

estén incluidas por el hecho de no pagar por las mismas. De igual manera el número de equipos en los que se puede realizar la instalación del DBMS no se limita con ningún tipo de restricción.

- **Gran escalabilidad. –** PostgreSQL permite configurar el DBMS en cada equipo tomando a consideración el hardware con el que se va a trabajar como lo puede ser la cantidad de memoria para hacerlo de forma óptima.
- **pgAdmin. -** Es la herramienta gráfica con la cual se administran las bases de datos de manera intuitiva y simple. Esta herramienta permite realizar comandos SQL, tareas de mantenimiento y crear copias de seguridad.
- **Estándar SQL. -** Se puede realizar consultas e incluir scripts de otros DBMS.
- **Potencia y Robustez. –** Cumple con las características ACID[21] razón por la cual las transacciones no interfieren entre sí, y garantiza la información en las bases de datos.

Desventajas

- Para pequeños volúmenes de datos la inserción y la actualización de datos es relativamente lento puesto que PostgreSQL se diseñó para ambientes de alto volumen de información.
- No cuenta con un soporte en línea o telefónico. Pero este cuenta con foros oficiales donde los usuarios presentan sus dudas y a su vez otros usuarios responden.
- La sintaxis no es intuitiva de ciertos comandos para usuarios que tienen conocimiento básico del lenguaje SQL.

3.4 DBMS NoSQL Referentes

La siguiente lista fue obtenida del sitio "Im Programmer" en la cual se encuentran los 10 mejores DBMS NoSQL del siguiente URL: https://www.improgrammer.net/most-popular-nosql-database/ .

1. **MongoDB**

"MongoDB es una base de datos multiplataforma y de código abierto orientada a documentos, una especie de base de datos NoSQL. Como una base de datos NoSQL, MongoDB rechaza la estructura basada en tablas de la base de datos relacional para adaptar documentos de tipo JSON que tienen esquemas dinámicos a los que llama BSON.

Esto hace que la integración de datos para ciertos tipos de aplicaciones sea más rápida y fácil. MongoDB está diseñado para ofrecer escalabilidad, alta disponibilidad y rendimiento

[21] ACID. – *Atomicity, Consistency, Isolation, Durability*. ACID es un acrónimo de Atomicity, Consistency …

desde una única implementación de servidor a grandes y complejas infraestructuras de múltiples sitios.” (Technopedia, 2018)(traducido del inglés)

URL: https://docs.mongodb.com/

2. **Cassandra**

“Apache Cassandra usa el sistema NoSQL en lugar del tradicional sistema de administración de bases de datos relacionales (RDBMS) porque este último no es adecuado para manejar grandes volúmenes de datos no estructurados, como los producidos por sitios web o compañías en línea. NoSQL tiene un diseño más simple y admite escala horizontal, lo que permite la adición de nuevos servidores para un mejor rendimiento.

Cassandra utiliza una arquitectura de igual a igual en lugar de la configuración maestro / esclavo utilizada en RDBMS. No hay un servidor maestro en el anterior, como en el intercambio de archivos de igual a igual. Si un servidor maestro se detiene o se descompone debido a numerosas solicitudes, los servidores esclavos se vuelven inútiles, mientras que, en una configuración de igual a igual, cada grupo de bases de datos es igual y puede aceptar solicitudes de cualquier cliente. Como resultado, Cassandra no tiene un solo punto de falla.” (Technopedia, 2018)(traducido del inglés)

URL: http://cassandra.apache.org/doc/latest/

3. **Redis**

“Redis es un almacén avanzado de valor-clave, mejor conocido como servidor de estructura de datos.

Puede considerarse como un tipo de base de datos que funciona con pares clave-valor y utiliza la memoria principal para almacenar datos. El uso de la memoria principal significa que es rápido y escalable, pero puede estar limitado por la capacidad de la memoria RAM.

También tiene una persistencia incorporada a través de instantáneas y registro en disco en el disco, por lo que puede utilizarse como una base de datos No SQL.” (Technopedia, 2018)(traducido del inglés)

URL: https://redis.io/documentation

4. **Hbase**

"Apache HBase es un tipo específico de herramienta de base de datos escrita en Java y utilizada con elementos del conjunto de herramientas de análisis de big data Hadoop de la base de software Apache. Apache HBase es un producto de código abierto, como otros elementos de Apache Hadoop. Representa una de las varias herramientas de base de datos para la entrada y salida de grandes conjuntos de datos que Hadoop y sus diversas utilidades y recursos." (Technopedia, 2018)(traducido del inglés)

URL: https://hbase.apache.org/book.html

5. **Neo4j**

"Neo4j es una base de datos de grafos nativos de código abierto, NoSQL, que proporciona un backend transaccional compatible con ACID para sus aplicaciones. El desarrollo inicial comenzó en 2003, pero ha estado disponible públicamente desde 2007. El código fuente, escrito en Java y Scala, está disponible de forma gratuita en GitHub o como una descarga de aplicaciones de escritorio fácil de usar. Neo4j tiene una edición comunitaria y una edición empresarial de la base de datos. Enterprise Edition incluye todo lo que Community Edition tiene para ofrecer, además de requisitos empresariales adicionales, como copias de seguridad, clústeres y capacidades de conmutación por error.

Neo4j se conoce como una base de datos de gráficos nativos porque implementa de manera eficiente el modelo de grafos de propiedades hasta el nivel de almacenamiento. Esto significa que los datos se almacenan exactamente a medida que los coloca en la pizarra, y la base de datos utiliza punteros para navegar y recorrer el gráfico. En contraste con el procesamiento de gráficos o las bibliotecas en memoria, Neo4j también proporciona características completas de la base de datos, incluido el cumplimiento de transacciones ACID, el soporte de clústeres y la conmutación por error en tiempo de ejecución, lo que lo hace adecuado para utilizar gráficos para datos en escenarios de producción." (neo4j, 2018)(traducido del inglés)

URL: https://neo4j.com/docs/developer-manual/current/

6. **Oracle NoSQL**

“Oracle NoSQL Database es una base de datos NoSQL escalable y distribuida, diseñada para proporcionar una administración de datos altamente confiable, flexible y disponible a través de un conjunto configurable de nodos de almacenamiento.

Los datos se pueden modelar como tablas de estilo de base de datos relacional, documentos JSON o pares clave-valor. Oracle NoSQL Database es un sistema fragmentado (no compartido) que distribuye los datos de manera uniforme entre los múltiples fragmentos del clúster, según el valor de hash de la clave principal. Dentro de cada fragmento, los nodos de almacenamiento se replican para garantizar una alta disponibilidad, una rápida conmutación por error en caso de falla de un nodo y un óptimo equilibrio de carga de las consultas. NoSQL Database proporciona controladores Java, C, Python y Node.js y una API REST para simplificar el desarrollo de aplicaciones. Oracle NoSQL Database está integrada con una amplia variedad de aplicaciones de código abierto y Oracle relacionadas para simplificar y agilizar el desarrollo y la implementación de aplicaciones modernas de big data.” (Oracle, 2018)(traducido del inglés)

URL: https://www.oracle.com/technetwork/database/database-technologies/nosqldb/documentation/index.html

7. Amazon DynamoDB

“Amazon DynamoDB es un servicio de bases de datos NoSQL totalmente administrado que ofrece un desempeño rápido y previsible, así como una escalabilidad óptima. DynamoDB le permite delegar las cargas administrativas que supone tener que utilizar y escalar bases de datos distribuidas, para que no tenga que preocuparse del provisionamiento, la instalación ni la configuración del hardware, ni tampoco de las tareas de replicación, aplicación de parches de software o escalado de clústeres. Además, DynamoDB ofrece el cifrado en reposo, que elimina la carga y la complejidad operativa que conlleva la protección de información confidencial. Para obtener más información, consulte Cifrado en reposo de Amazon DynamoDB.

Con DynamoDB, puede crear tablas de base de datos capaces de almacenar y recuperar cualquier cantidad de datos, así como de atender cualquier nivel de tráfico de solicitudes. Puede escalar la capacidad de desempeño de las tablas para aumentarla o reducirla sin tiempos de inactividad ni reducción del desempeño, así como utilizar la consola de administración de AWS para monitorizar la utilización de recursos y las métricas de desempeño.” (Amazon, 2018)

URL: https://docs.aws.amazon.com/dynamodb/index.html#lang/es_es

8. CouchBase

"Couchbase Server es una base de datos de código abierto NoSQL diseñada para soportar el desarrollo ágil y la implementación escalable de aplicaciones web, móviles e IoT empresariales

- Implementar como un caché distribuido para lecturas de baja latencia
- Implementar como un almacén de clave / valor para lecturas y escrituras de alto rendimiento
- Despliegue como una base de datos de documentos para consultas potentes y análisis livianos

Couchbase Server admite el desarrollo ágil con un modelo de datos flexible y un lenguaje de consulta potente, implementación escalable con un solo tipo de nodo y clientes conscientes de la topología." (DB-Engines, 2018)(traducido del inglés)

URL: https://developer.couchbase.com/documentation-archive

9. Memcached

"Libre y de código abierto, alto rendimiento, sistema de almacenamiento en caché de objetos de memoria distribuida, de naturaleza genérica, pero diseñado para acelerar las aplicaciones web dinámicas al aliviar la carga de la base de datos.

Memcached es un almacén de clave-valor en memoria para pequeños fragmentos de datos arbitrarios (cadenas, objetos) de resultados de llamadas a bases de datos, llamadas a API o representación de páginas.

Memcached es simple pero potente. Su diseño simple promueve la implementación rápida, la facilidad de desarrollo y resuelve muchos problemas que enfrentan los cachés de datos grandes. Su API está disponible para los idiomas más populares." (Memcached, 2018)(traducido del inglés)

URL: https://github.com/memcached/memcached/wiki

10. CouchDB

“Apache CouchDB es una base de datos no relacional o NoSQL que fue desarrollada para abarcar completamente la web. Los datos se almacenan en documentos JSON a los que se puede acceder y sus índices se pueden consultar a través de HTTP.

La indexación, transformación y combinación de documentos se realiza a través de JavaScript. Debido a que utiliza todos estos estándares y tecnologías compatibles con la web, CouchDB funciona muy bien con aplicaciones web y móviles.” (Technopedia, 2018)(traducido del inglés)

URL: http://docs.couchdb.org/en/stable/

3.5 Elección DBMS NoSQL

Como se observa en la tabla 3-2 se encuentran varias opciones para su selección, pero dado que MongoDB es el DBMS NoSQL más conocido y con mayor cantidad de documentación e información que se puede encontrar, también se debe considerar que es uno de los DMBS NoSQL que tiene soporte a transacciones, será el DBMS NoSQL seleccionado para el presente trabajo de titulación.

3.5.1 MongoDB

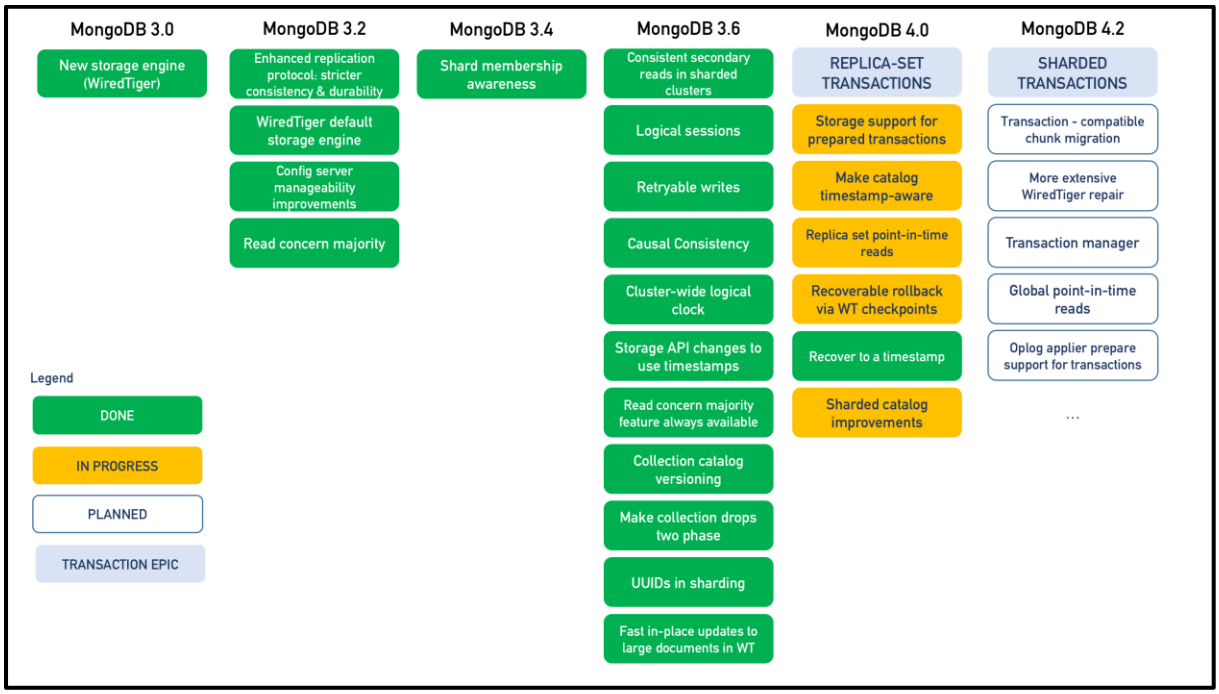


Figura 3-2 Versiones MongoDB (Melnik, 2018)

En la figura 3-2 se puede ver las versiones que MongoDB con sus características implementadas en cada una de ellas, las que están en desarrollo y las planificadas para el

futuro del DBMS NoSQL. También se puede apreciar que tanto en la versión 4.0 y 4.2 se presentan características para las transacciones.

```
with client.start_session() as s:
    s.start_transaction()
    collection_one.insert_one(doc_one, session=s)
    collection_two.insert_one(doc_two, session=s)
    s.commit_transaction()
```

Figura 3-3 Transacciones en MongoDB en Python (MongoDB, 2018)

```
try (ClientSession clientSession = client.startSession()) {
   clientSession.startTransaction();
   collection.insertOne(clientSession, docOne);
   collection.insertOne(clientSession, docTwo);
   clientSession.commitTransaction();
}
```

Figura 3-4 Transacciones en MongoDB en Java (MongoDB, 2018)

En las figuras 3-3 y 3-4 se encuentran los ejemplos propuestos por MongoDB del uso de transacciones tanto como para Python, como para java. En estos comandos las líneas pertenecientes a las transacciones en MongoDB son:

- *start_Transaction()*, *esta* reemplaza el *BEGIN* en PostgreSQL.
- *commit_Transaction(), esta* reemplaza al *COMMIT* en PostgreSQL.

Otro aspecto que tomar a consideración es que las transacciones en MongoDB se las realizan en documentos. Por ejemplo, en la figura 3-4 se puede encontrar las líneas:

*collection.insertOne(clientSession, **docOne**);*

*collection.insertOne(clientSession, **docTwo**);*

Las expresiones ***docOne*** *y* ***docTwo*** se refieren a los documentos de inserción respectivos en esta transacción.

A continuación, se presentan varias ventajas y desventajas del DBMS NoSQL MongoDB

Ventajas

- **"Plataforma de datos distribuidos. -** MongoDB se puede ejecutar en todos los centros de datos distribuidos, para garantizar nuevos niveles de disponibilidad y escalabilidad.
- **Desarrollo rápido e iterativo. -** Un modelo de datos flexible con esquema dinámico, con una poderosa interfaz gráfica de usuario y herramientas de línea de comando facilitan a los desarrolladores la creación y evolución de aplicaciones.
- **Modelo de datos flexible. -** Permite el almacenamiento de datos en documentos flexibles similares a JSON, lo que hace que la persistencia de los datos y la combinación sean fáciles.
- **TCO reducido (costo total de propiedad). -** los desarrolladores de aplicaciones pueden hacer su trabajo mucho mejor cuando se usa MongoDB. El equipo de operaciones también puede realizar bien su trabajo gracias al servicio en la nube de Atlas. Los costos se reducen significativamente ya que MongoDB se ejecuta en hardware básico.
- **Conjunto de características integrado. -** se pueden obtener diversas aplicaciones en tiempo real gracias a análisis y visualización de datos, canalizaciones de datos de transmisión por eventos, búsqueda de texto y geoespacial, procesamiento de gráficos, rendimiento en memoria y replicación global de manera confiable y segura."
  (Experto en Big Data, 2018)

Desventajas

- Al ser un DBMS NoSQL no se garantiza ACID, pero en la versión en la cual se añade el soporte a transacciones es probable que esta desventaja se vea cubierta en una gran parte, pero también se debe considerar que al empezar MongoDB no consideraba el trabajo con transacciones por lo que la implementación de estas no garantiza que sean completamente efectivas.
- Una aplicación se vuelve más compleja de desarrollar por los esquemas flexibles y dinámicos.

4. Contexto transaccional de estudio

En este capítulo se revisará el modelo a utilizar para le elaboración del presente trabajo de titulación, las tablas principales, auxiliares, sus relaciones y las transacciones que se presentan.

4.1 Modelo Banca

"El banco "BANCA"; es una institución financiera fundada en 1965 con el propósito de servir a sus clientes con préstamos en condiciones acordes a la situación del país.

Tiene algunas sucursales; cada sucursal tiene asociada una dirección con la Ciudad, la calle principal, identificación numérica y transversal próxima, monto de activos reflejada en dólares americanos, así como el nombre del responsable de la sucursal que es un funcionario del banco y su teléfono de contacto. Es importante la fecha de creación de la Sucursal.

Dentro de este contexto de banco las sucursales generan cuentas de ahorros o cuentas corrientes para los clientes, los mismos que pueden ser personas naturales o jurídicas. Para ahorros tenemos una tasa de interés del 1 al 2% dependiendo del monto de las cuentas, para corrientes la tasa del interés es de hasta el 1%.

Las sucursales también conceden préstamos, en condiciones acordadas con los clientes.; cuando generamos préstamos la tasa permitida va del 5 al 15% sobre el préstamo otorgado a clientes de los cuales es importante mantener la información actualizada de nombre, ubicación geográfica, dirección, teléfono, correo electrónico, fecha nacimiento, sexo. Un cliente puede tener una o más cuentas, así como puede deber uno o más préstamos." (CONDOR, 2018)

4.2 Tablas

4.2.1 Sucursales

Cada uno de los establecimientos del banco en los cuales se puede realizar cualquier transacción, o actualización de datos.

La tabla sucursal consta con los siguientes atributos: Nombre, Ciudad, Calle principal, Número, Calle secundaria, Responsable, Teléfono.

4.2.2 Clientes

Usuarios registrados del banco como clientes. La tabla clientes consta con los siguientes atributos: Nombre, Dirección, Teléfono, Correo electrónico, Fecha nacimiento, Sexo.

4.2.3 Préstamos

Préstamos realizados por sucursal. La tabla prestamos consta con los siguientes atributos: Numero de préstamo, Fecha emisión, Monto en dólares.

4.2.4 Cuentas

Cuentas de ahorros o corriente pertenecientes a los clientes. La tabla cuentas consta con los siguientes atributos: Número de cuenta, Fecha creación, Valor de cuenta en dólares, Tipo de cuenta.

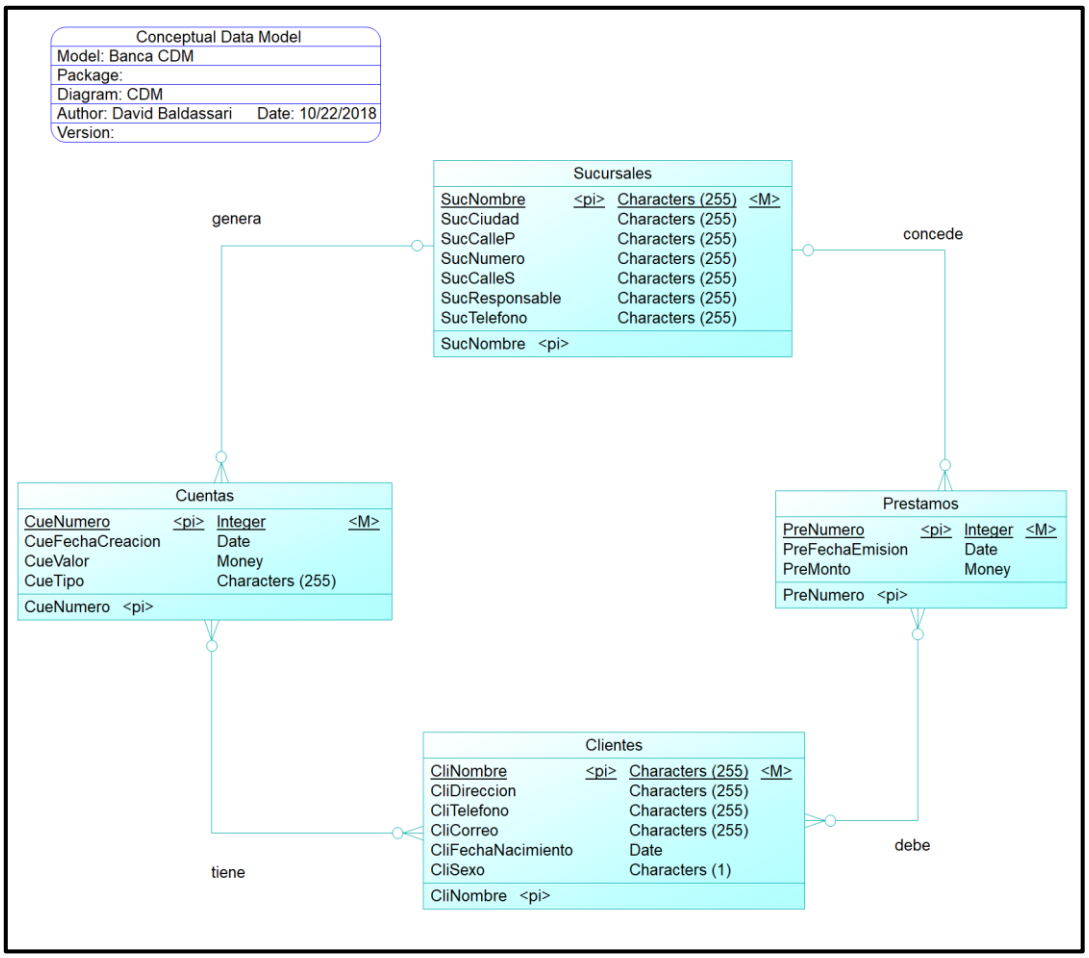


Figura 4-1 Modelo Conceptual Banca (Baldassari, 2018)

En la figura 4-1 se puede observar el Modelo conceptual de Banca con las cuatro tablas principales relacionadas.

4.2.5 Debe (Préstamos x Clientes)

Esta tabla se genera por la relación existente entre clientes y préstamos. Esta contiene como atributo las claves foráneas de las tablas relacionadas. Esta entidad débil permite observar de una mejor manera la relación muchos a muchos entre préstamos y clientes.

4.2.6 Tiene (Cuentas x Clientes)

Esta tabla se genera por la relación existente entre cuentas y clientes. Esta contiene como atributo las claves foráneas de las tablas relacionadas. De forma similar a la entidad débil debe, permite de mejor manera observar la relación muchos a muchos existente entre las entidades cuentas y clientes.

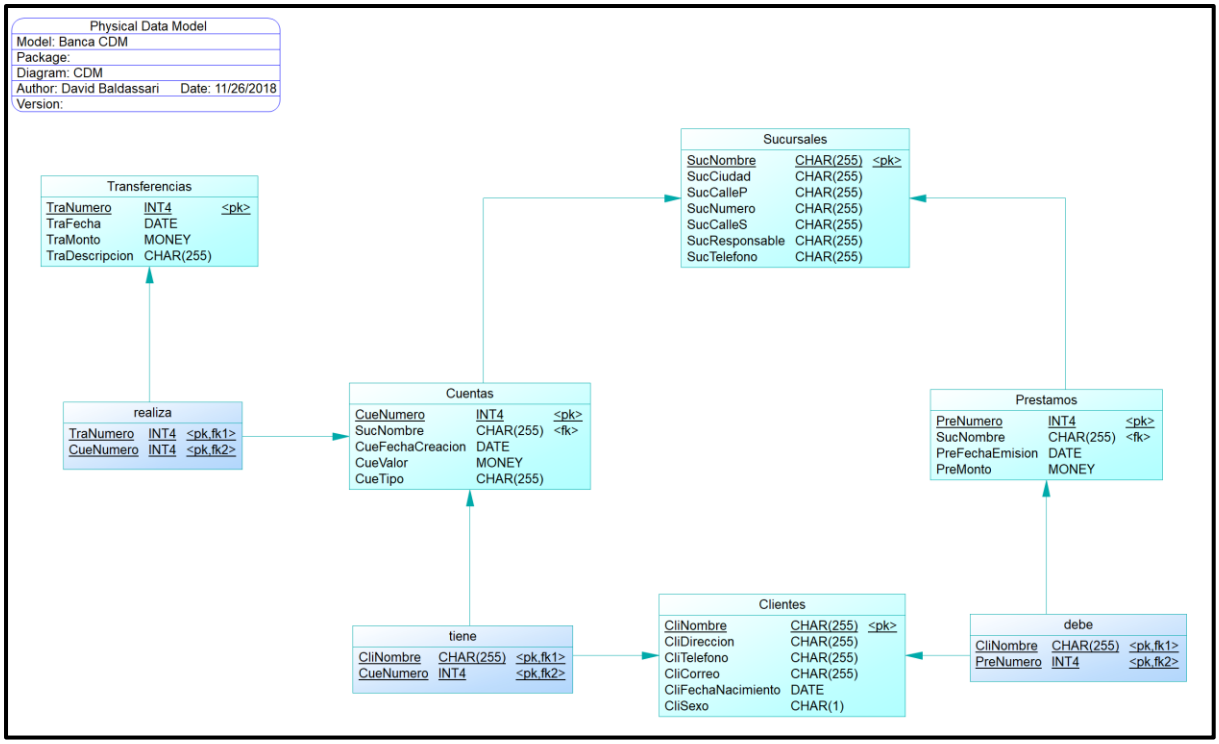


Figura 4-2 Modelo Físico Banca (Baldassari, 2018)

En la figura 4-2 se observa que las tablas tiene y debe se pueden apreciar en el modelo físico lo cual no se no se permite en el modelo conceptual en el cual solo se puede observar el modelo banca de manera general.

4.3 Relaciones

4.3.1 SUCURSALES concede varios PRESTAMOS

Esta relación permite conocer que sucursal concede ciertos préstamos a los clientes.

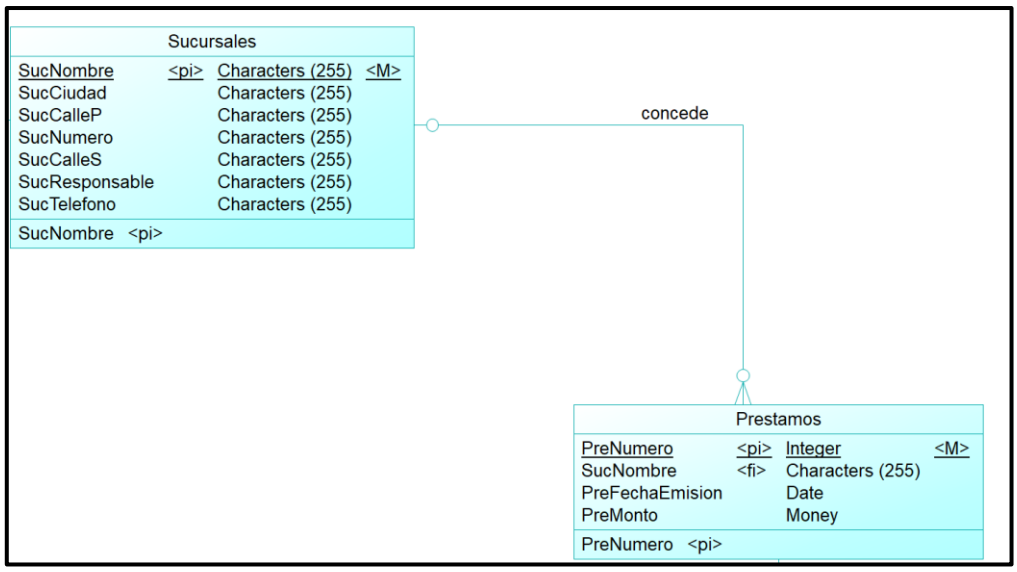


Figura 4-3 Relación SUCURSALES concede varios PRESTAMOS (Baldassari, 2018)

4.3.2 SUCURSALES genera varias CUENTAS

Esta relación permite conocer que sucursal genera las cuentas para los clientes del banco.

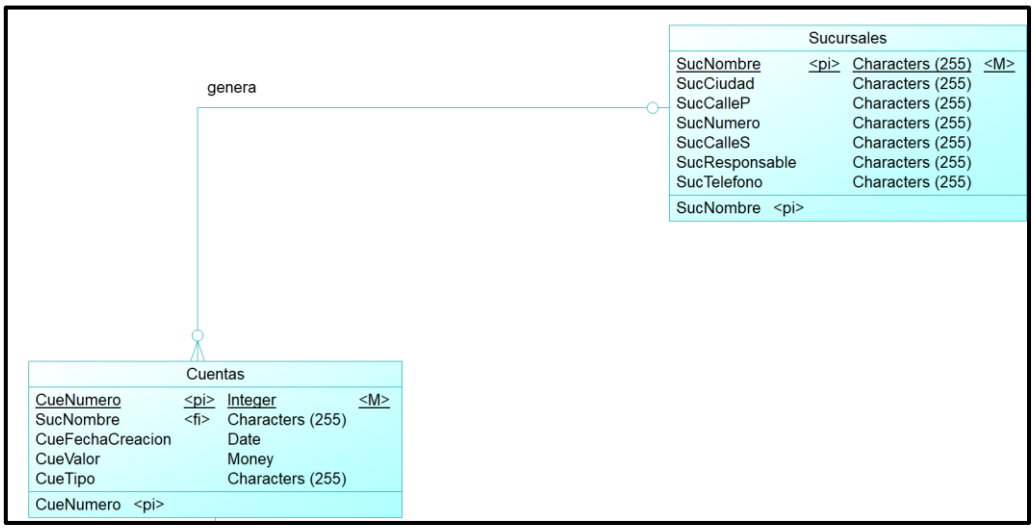


Figura 4-4 Relación SUCURSALES genera varias CUENTAS (Baldassari, 2018)

4.3.3 CLIENTES debe uno o más PRESTAMOS

Esta relación permite conocer que clientes todavía no cancelan los montos de el o los préstamos concedidos por las sucursales.

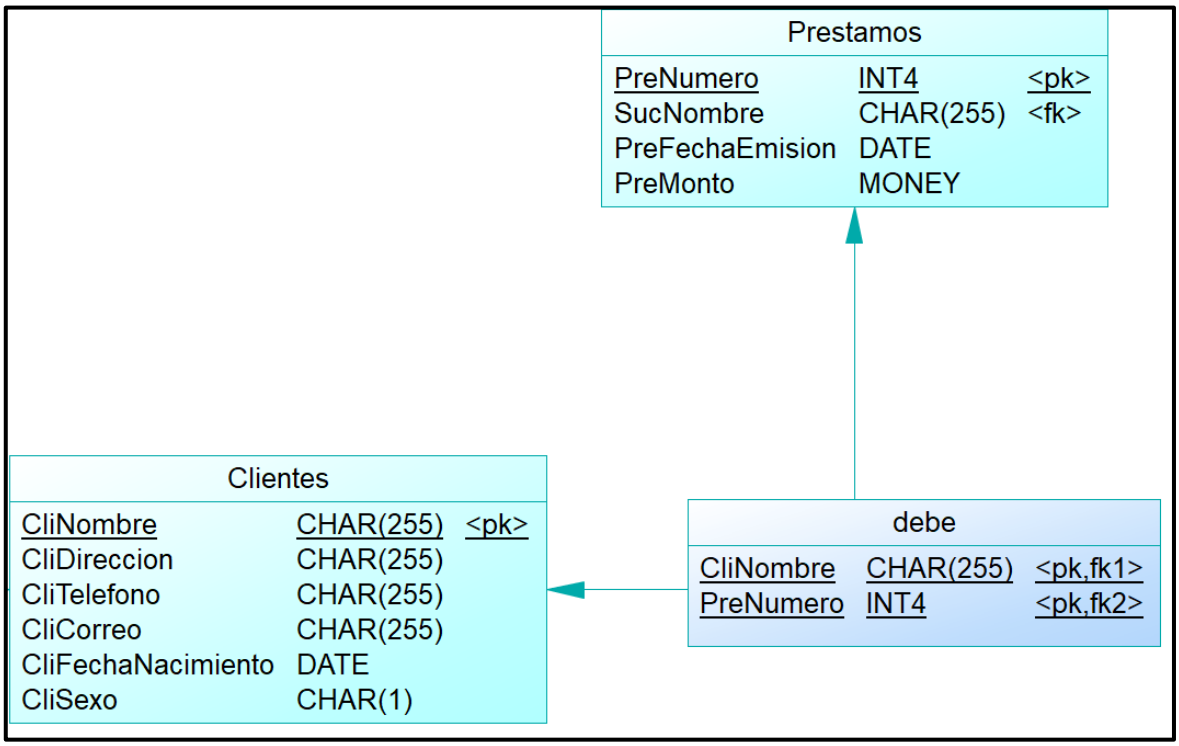


Figura 4-5 Relación CLIENTES debe uno o más PRESTAMOS, entidad débil DEBE (Baldassari, 2018)

4.3.4 CLIENTES tiene una o más CUENTAS

Esta relación permite conocer que clientes tienen una o varias cuentas asociadas.

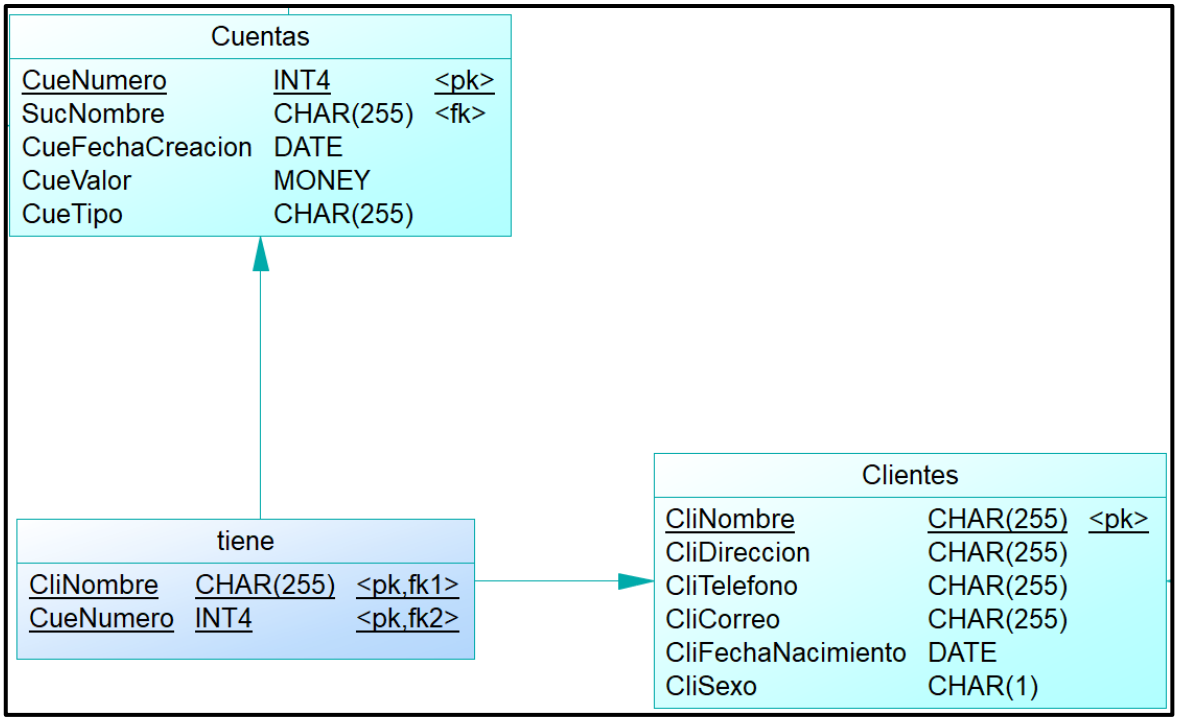


Figura 4-6 Relación CLIENTES tiene una o más CUENTAS, entidad débil TIENE (Baldassari, 2018)

4.4 Transacciones

Las transacciones existentes en el contexto bancario son de varios tipos entre los cuales se encuentran:

- **Operaciones de crédito pasivas. -** Esta transacción es el aporte del cliente a la entidad bancaria el cual debe ser devuelto al cliente, un ejemplo de esta transacción son los depósitos bancarios.
- **Operaciones de crédito activas. -** Esta transacción es opuesta a la mencionada anteriormente, es decir la entidad bancaria concede un monto al cliente, las operaciones se pueden presentar de distintas maneras, algunos ejemplos de esto son:
    - **Préstamos. -** En este caso la entidad bancaria entrega al cliente un monto, el cual el cliente debe pagar en varios plazos o de una sola vez.
    - **Apertura de crédito. -** En este caso la entidad bancaria concede crédito al cliente a cambio de una comisión o un interés calculado.
    - **Créditos y préstamos participativos. -** En este caso la entidad prestamista recibe un interés el cual varía a partir de los resultados de la empresa la cual recibe dicho préstamo.
    - **Contrato de arrendamiento. -** En este caso existe un usuario el cual adquiere un bien para ceder el uso de este a un arrendatario quien paga periódicamente un monto con el que se cancela la deuda del bien y los gastos.
    - **Contrato de facturación. -** En esta operación un banco gestiona los cobros de un cliente a cambio de un interés.

- **Operaciones de mediación. -** Estas operaciones se refiere a las transferencias bancarias, en las cuales se traspasa un monto de una cuenta a otra mediante el descuento en la primera y abono en la segunda cuenta.

La transacción para considerar en el presente trabajo de titulación es la de tipo operación de mediación, de esta forma en el modelo BANCA se debe presentar una entidad débil llamada realiza entre las tablas transferencias y cuentas como se propone en la figura 4-7.

4.4.1 Transferencias

La transacción llamada transferencia, se refiere el movimiento de montos a través de distintas cuentas, por ejemplo:

De la cuenta A que contiene un monto de 200 $ se desea transferir a las cuentas B, C y D los montos 125 $, 50 $ y 25 $ respectivamente.

Para esta transacción se propone el siguiente cambio en el modelo de la base de datos.

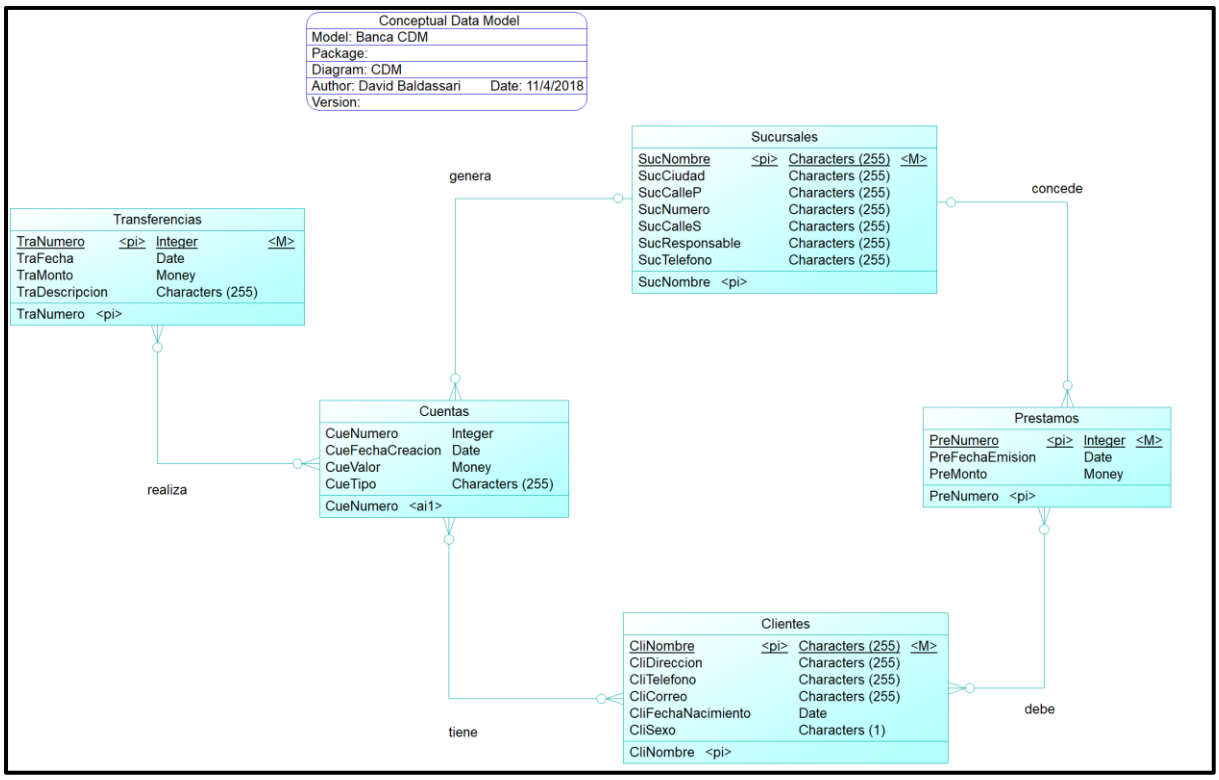


Figura 4-7 Modelo Banca propuesto para la transacción Transferencia (Baldassari, 2018)

En la figura 4-7 se encuentra la propuesta del modelo BANCA a utilizar en el presente trabajo de titulación la cual se agrega la tabla Transferencias, la misma que será de utilidad para las transacciones que se desea realizar.

5. Aplicación de información transaccional en DBMS SQL y DBMS NoSQL

En el presente capítulo se analizará los DBMS[22] SQL[23] y DBMS NoSQL[24] seleccionados con respecto a la transaccionalidad, concurrencia, replicación, distribución y seguridad.

5.1 Instalación

5.1.1 PostgreSQL

Durante el proceso de instalación de PostgreSQL se llega al punto de selección de componentes, en el cual existe una opción llamada Stack Builder, esta opción permite instalar una interfaz para la instalación de módulos que complementen PostgreSQL y siendo el encargado de resolver cualquier dependencia de software que se pueda presentar durante la instalación de estos módulos.

Es necesario tener en cuenta que en el momento de que el instalador solicite una contraseña, esta será la utilizada para el superusuario del DBMS. Y es recomendable dejar el puerto por defecto por el cual el servidor escuchará.

5.1.2 MongoDB

La instalación de MongoDB no cambia con respecto a PostgreSQL, pero en este caso una vez realizada la instalación es necesario recordar que en el disco C se debe crear una carpeta llamada *data* y dentro de esta una carpeta llamada *db* en las cuales se almacenan las bases de MongoDB. Para empezar a utilizar MongoDB se debe ingresar a la carpeta donde se instaló *MongoDB/Server/4.0/bin* aquí se encontrará dos ejecutables llamados *mongod.exe* o también denominado demonio de mongo, el cual utiliza las carpetas creadas anteriormente para que su ejecución sea correcta. y *mongo.exe* en el cual se ejecutarán todos los comandos relacionados con MongoDB.

5.2 Transaccionalidad

5.2.1 PostgreSQL

En la documentación del DBMS se puede encontrar el siguiente *query* el cual indica los cambios que se desean realizar en las tablas ‘accounts’ y ‘branches’.

[22] DBMS. – *Database Management System. - Sistema de Gestion de Base de Datos.*
[23] SQL. – *Structured Query Language*. - Lenguaje de consulta estructurado.
[24] NoSQL. – *Not only SQL.*

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
```

Figura 5-1 Query de actualización de las tablas 'accounts' y 'branches' (PostgreSQL, 2018)

Pero en el caso de la figura 5-1 si se presentase una falla durante la ejecución de estos *queries* se realizarían únicamente ciertos cambios y es probable que la información no sea consistente en un futuro uso de los mismos atributos. Por este motivo en PostgreSQL se utilizan los comandos BEGIN y COMMIT, los mismos que deben ubicarse de la siguiente manera.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
COMMIT;
```

Figura 5-2 Query de actualización de las tablas 'accounts' y 'branches' utilizando comandos BEGIN y COMMIT (PostgreSQL, 2018)

Por el hecho de utilizar los comandos presentados en la figura 5-2 se garantiza la integridad de la información.

Una vez realizados dichos cambios, existe la posibilidad de que el error no sea externo sino por parte del usuario, para este caso existen puntos de guardados que se pueden realizar con el comando que PostgreSQL brinda llamado SAVEPOINT, y el comando para regresar a uno de estos puntos de guardado es conocido como ROLLBACK TO que para este caso se emplearía de la siguiente manera.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Figura 5-3 Queries de SAVEPOINT y ROLLBACK TO (PostgreSQL, 2018)

La figura 5-3 presenta que se retira de la cuenta de nombre ‘Alice’ un monto de 100 y a continuación se realiza un punto de guardado, el monto de 100 es acreditado a la cuenta de nombre ‘Bob’ pero dado que la transacción correcta es, de la cuenta de nombre ‘Alice’ acreditar a la cuenta de nombre ‘Wally’ se ejecuta el comando ROLLBACK TO y se llama al punto de guardado generado posteriormente y se acredita correctamente a la cuenta deseada. De esta manera la información sigue siendo confiable y la transacción realizada es la correcta.

5.2.2 MongoDB

En el caso de MongoDB las transacciones se las realiza con los siguientes métodos

- *Session.startTransaction()*

Este método inicia una transacción entre varios documentos asociados con la sesión. Solo puede mantener una transacción abierta, como máximo, para una sesión. MongoDB señala, de manera importante y para tener en cuenta, que las transacciones entre varios documentos no pueden incluir operaciones de inserción que podría resultar en la creación de una nueva tabla.

```
1  function updateEmployeeInfo(session) {
2      employeesCollection = session.getDatabase("hr").employees;
3      eventsCollection = session.getDatabase("reporting").events;
4
5      session.startTransaction( { readConcern: { level: "snapshot" }, writeConcern: { w: "majority" } } );
6
7      try{
8          employeesCollection.updateOne( { employee: 3 }, { $set: { status: "Inactive" } } );
9          eventsCollection.insertOne( { employee: 3, status: { new: "Inactive", old: "Active" } } );
10     } catch (error) {
11         print("Caught exception during transaction, aborting.");
12         session.abortTransaction();
13         throw error;
14     }
15
16     commitWithRetry(session);
17 }
```

Figura 5-4 Ejemplo de uso del método session.startTransaction() (MongoDB, 2018)

En la figura 5-4 se puede encontrar el ejemplo de una función en Mongo Shell, la cual aplica el método session.startTransaction(). En la línea 5 de la figura se encuentra el inicio de la transacción dando un nivel de lectura y escritura para el archivo que se pasa a este método. Dentro del bloque *try* se realiza la actualización de un empleado en la tabla *employees* de la base 'hr' y la inserción en la tabla *events* de la base 'reporting', para finalmente realizar estos cambios con la función *commitWithRetry.*

- *Session.commitTransaction()*

Este método guarda los cambios realizados por las transacciones de múltiples documentos y finaliza con la transacción. Hasta el *commit* los cambios realizados por la transacción no son visibles fuera de esta transacción.

```
1  function commitWithRetry(session) {
2      while (true) {
3          try {
4              session.commitTransaction(); // Uses write concern set at transaction start.
5              print("Transaction committed.");
6              break;
7          } catch (error) {
8              // Can retry commit
9              if (error.hasOwnProperty("errorLabels") && error.errorLabels.includes("UnknownTransactionCommitResult") ) {
10                 print("UnknownTransactionCommitResult, retrying commit operation ...");
11                 continue;
12             } else {
13                 print("Error during commit ...");
14                 throw error;
15             }
16         }
17     }
18 }
```

Figura 5-5 Ejemplo de uso del método session.commitTransaction() (MongoDB, 2018)

En la figura 5-5 se encuentra la función en Mongo Shell correspondiente a *sesión.commitTransaction()*. Dentro del bloque *try* se encuentra el método y un mensaje a presentar al realizarse la transacción, en el bloque *catch* se encuentra un *if - else* que indica que se vuelva a intentar el *commit* o caso contrario señala un error durante el *commit*.

- *Session.abortTransaction()*

Este método finaliza la transacción de documentos múltiples y revierte los cambios a la información realizados durante la transacción. De esta manera no se guarda ningún cambio realizado durante la transacción y se mantiene la información anterior.

```
session.startTransaction( { readConcern: { level: "snapshot" }, writeConcern: { w: "majority" } } );

try{
   eventsCollection.insertOne(
      { employee: 3, status: { new: "Active", old: null },  department: { new: "XYZ", old: null } }
   );

   // Count number of events for employee 3

   var countDoc = eventsCollection.aggregate( [ { $match:  { employee: 3 } }, { $count: "eventCounts" } ] ).next();

   print( "events count (in active transaction): " + countDoc.eventCounts );

   // The following operations should fail as an employee ``3`` already exist in employees collection
   employeesCollection.insertOne(
      { employee: 3, name: { title: "Miss", name: "Terri Bachs" }, status: "Active", department: "XYZ" }
   );
} catch (error) {
   print("Caught exception during transaction, aborting.");
   session.abortTransaction();
   throw error;
}
```

Figura 5-6 Ejemplo de uso del método session.abortTransaction() (MongoDB, 2018)

En la figura 5-6 se encuentra un ejemplo de uso del método *session.abortTransaction()* el cual en el bloque *try* intenta ingresar un empleado ya existente en la tabla 'employees' de esta manera salta al bloque *catch*, en el que se aborta la transacción, así la transacción acaba y ningún cambio se realiza en la tabla 'employees'. Es importante tener en cuenta que cuando se ejecuta este comando estos cambios no son visibles, salvo para auditorias.

5.3 Concurrencia

5.3.1 PostgreSQL

PostgreSQL emplea un modelo de múltiples versiones para el control de concurrencia, es decir que al consultar en la base de datos cada transacción ve una instancia de la información o una versión de la base correspondiente a instantes previos. Todo esto permite que la transacción utilice información inconsistente que podría causarse por otras transacciones, de esta manera PostgreSQL se asegura de la existencia de aislamiento de transacciones para las sesiones de la base.

5.3.2 MongoDB

MongoDB permite que varios usuarios puedan leer y escribir la misma información, y a su vez proporciona bloqueos y otras medidas de control de concurrencia para evitar que los usuarios modifiquen un mismo bloque de información a la vez. Con estas medidas MongoDB garantiza que varias escrituras en un documento se produzcan completamente o no lo hagan, y que el usuario no le sea posible ver inconsistencia de información.

5.4 Replicación

5.4.1 PostgreSQL

PostgreSQL permite la replicación y para ello es necesario realizar ciertas configuraciones en los servidores principales y en espera. El servidor principal es aquel que envía la información de replicación a uno o más servidores en espera. Para ello existen ciertas configuraciones entre estas se encuentran:

- Servidor principal
    - *max_wal_senders (integer)*

    Especifica el número máximo de conexiones simultáneas desde los servidores en espera.

    - *wal_sender_delay (integer)*

    Especifica el retraso entre actividades en el registro de escritura anticipada o WAL (*Write-Ahead Logging*).

    - *wal_keep_segments (integer)*

    Especifica el número mínimo de segmentos de archivos de registro pasados que se guardan en el directorio *pg_xlog*, en caso de que un servidor en espera necesite recuperarlos.

    - *vacuum_defer_cleanup_age (integer)*

    Especifica el número de transacciones por las cuales las actualizaciones VACUUM y HOT diferirán la limpieza de las versiones.

    - *replication_timeout (integer)*

    Termina la conexión con aquellas conexiones de replicación inactivas en un tiempo establecido en milisegundos.

    - *synchronous_standby_names (string)*

    Especifica una lista separada por comas de nombres en espera que pueden admitir la replicación simultánea. Las transacciones en espera de su *commit* podrán continuar después de que el servidor en espera confirme la recepción de sus datos.

- Servidores en espera
    - hot_standby (boolean)

    Especifica si puede o no conectarse y ejecutar consultas durante la recuperación.

    - max_standby_archive_delay (integer)

    Si *hot_standby* está activo, este parámetro indica el tiempo de espera antes de cancelar las consultas en el servidor en espera.

    - wal_receiver_status_interval (integer)

Especifica la frecuencia mínima para el proceso del receptor registro de escritura anticipada o WAL (*Write-Ahead Logging*) en el modo de espera para enviar información sobre el progreso de la replicación al servidor primario.

- hot_standby_feedback (boolean)

Especifica si un *hot_standby* enviará información al servidor primario sobre las consultas que se están realizando en el servidor en espera.

5.4.2 MongoDB

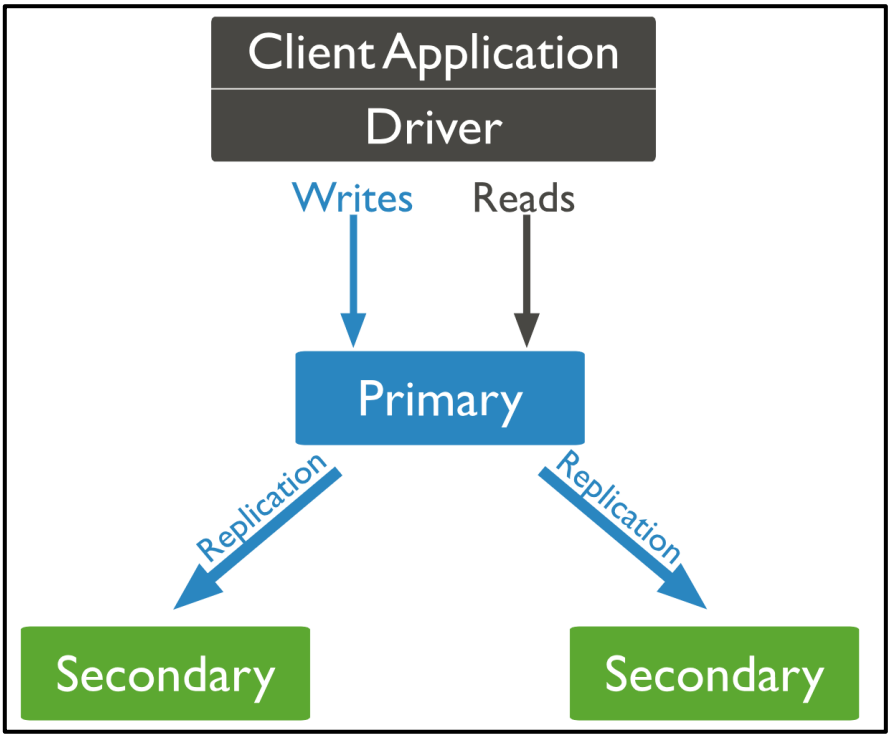


Figura 5-7 Replicación en MongoDB (MongoDB, 2018)

En la figura 5-7 se puede observar el funcionamiento de la replicación en MongoDB, donde el servidor primario recibe todas las actividades de lectura y escritura, para después ser replicados a los servidores secundarios. Si en algún momento el servidor primario no se encuentra disponible se puede configurar para que cualquiera de los servidores secundarios asuma la posición de servidor primario.

5.5 Distribución

5.5.1 PostgreSQL

En este DBMS se debe instalar las funciones de *dblink* lo que permite la comunicación entre bases de datos remotas. A partir de la versión 9.1 de PostgreSQL basta ejecutar en SQL para generar las funciones de dblink, mientras que en las anteriores se debía acceder a la dirección "C:\Program Files\PostgreSQL\8.4\share\contrib\dblink.sql" donde se encuentra el script.

```
SELECT lista_de_campos
FROM dblink('dbname=baseDatos port=puerto host=ordenadorRemoto user=usuario password=contraseña','SQL') AS Alias(campo1 tipo1,campo2 tipo2,...)
WHERE condiciones
ORDER BY criterios_de_ordenacion;
```

Figura 5-8 Ejemplo uso del comando dblink (v-espino, 2018)

En la figura 5-8 se encuentra un ejemplo de la aplicación del comando *dblink* el cual trae la base de datos deseada a través de un alias en el cual ya se puede aplicar el resto de los comandos deseados para consultas.

```
select dblink_exec('dbname=remota port=5432 host=yoquese.com user=postgres password=contrasena',
'INSERT INTO gente (codigo,nombre,sueldo) VALUES (33,\'Pepito\',1234.45));
```

Figura 5-9 Ejemplo de uso del comando dblink_exec (v-espino, 2018)

En el caso de una inserción, eliminación o actualización el comando a ejecutar es el de *dblink_exec* tal como se puede observar en la figura 5-9.

5.5.2 MongoDB

MongoDB utiliza el método sharding el cual consiste en distribuir la información en varias máquinas. Por esta razón MongoDB admite implementaciones con conjuntos de datos a gran escala y operaciones de alto rendimiento.

Con sharding MongoDB soporta el escalamiento horizontal, es decir divide la información y lo carga en varios servidores.

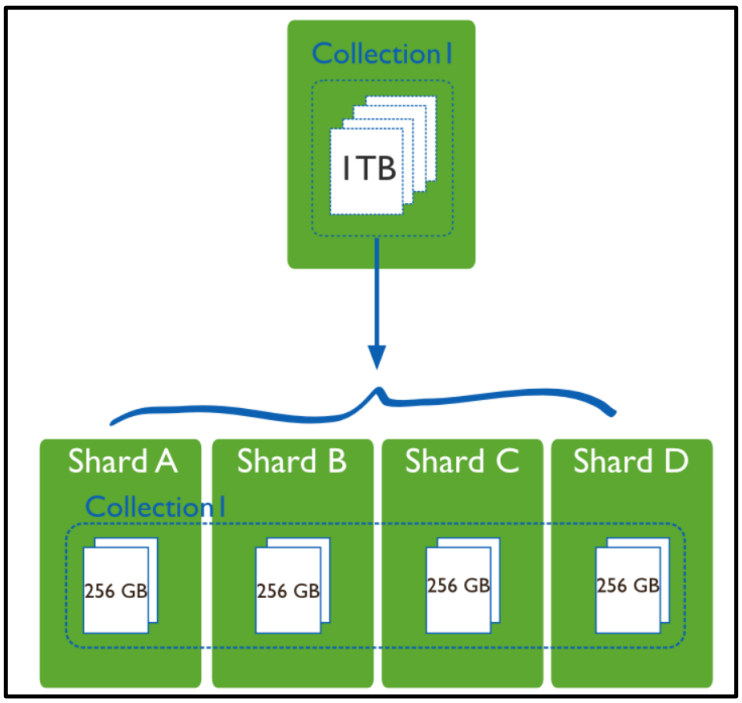


Figura 5-10 Distribución MongoDB (Grupo de Programación Declarativa, 2018)

En la figura 5-10 se observa el funcionamiento del *sharding* en MongoDB en el que la colección o tabla se reparte en diferentes particiones o servidores.

5.6 Seguridad

5.6.1 PostgreSQL

PostgreSQL contiene distintas características para tratar el tema de seguridad entre ellas:

- **Autenticación de usuario**

  Es el proceso mediante el servidor y el administrador se aseguran de que el usuario esté autorizado a realizar el comando solicitado.

- **Nombres de usuario y grupos**

  Esta característica permite crear grupos de usuarios y asignarles distintos privilegios en dicho grupo.

- **Control de acceso**

  PostgreSQL proporciona mecanismos para permitir que los usuarios limiten el acceso a la información que se proporcionan a otros usuarios.

- **Funciones y reglas**

  Esto permite a usuarios ingresar código en el backend que se ejecuten sin que los otros usuarios sepan de ello. Para que la seguridad sea visible en esta característica es necesario el control de quien puede definir estas funciones y reglas.

- **Conexión segura TCP / IP**

  Se puede usar ssh(secure shell) para cifrar la conexión de red entre los clientes y un servidor de PostgreSQL. Si se realiza correctamente, esto debería proporcionar una conexión de red segura.

5.6.2 MongoDB

MongoDB ofrece algunas características clave de seguridad que incluyen:

- Autenticación
- Control de acceso
- Cifrado

Authentication	Authorization	TLS/SSL	Enterprise Only
Authentication	Role-Based Access Control	TLS/SSL (Transport Encryption)	Kerberos Authentication
SCRAM	Enable Auth	Configure mongod and mongos for TLS/SSL	LDAP Proxy Authentication
x.509	Manage Users and Roles		Encryption at Rest
		TLS/SSL Configuration for Clients	Auditing

Figura 5-11 Características de seguridad de MongoDB (MongoDB, 2018)

En la figura 5-11 se observa las características de seguridad que se encuentran disponibles para MongoDB, en la versión empresarial existe la opción de auditoria la cual se puede configurar para generar un log que facilitará este proceso.

6. Transaccionalidad en DBMS seleccionados

En este capítulo se observa el diseño conceptual y físico de la base de datos, las transacciones de transferencias y préstamos en cada uno de los DBMS seleccionados.

6.1 Transaccionalidad en DBMS SQL

6.1.1 Modelo de datos conceptual (CDM) Banca para PostgreSQL

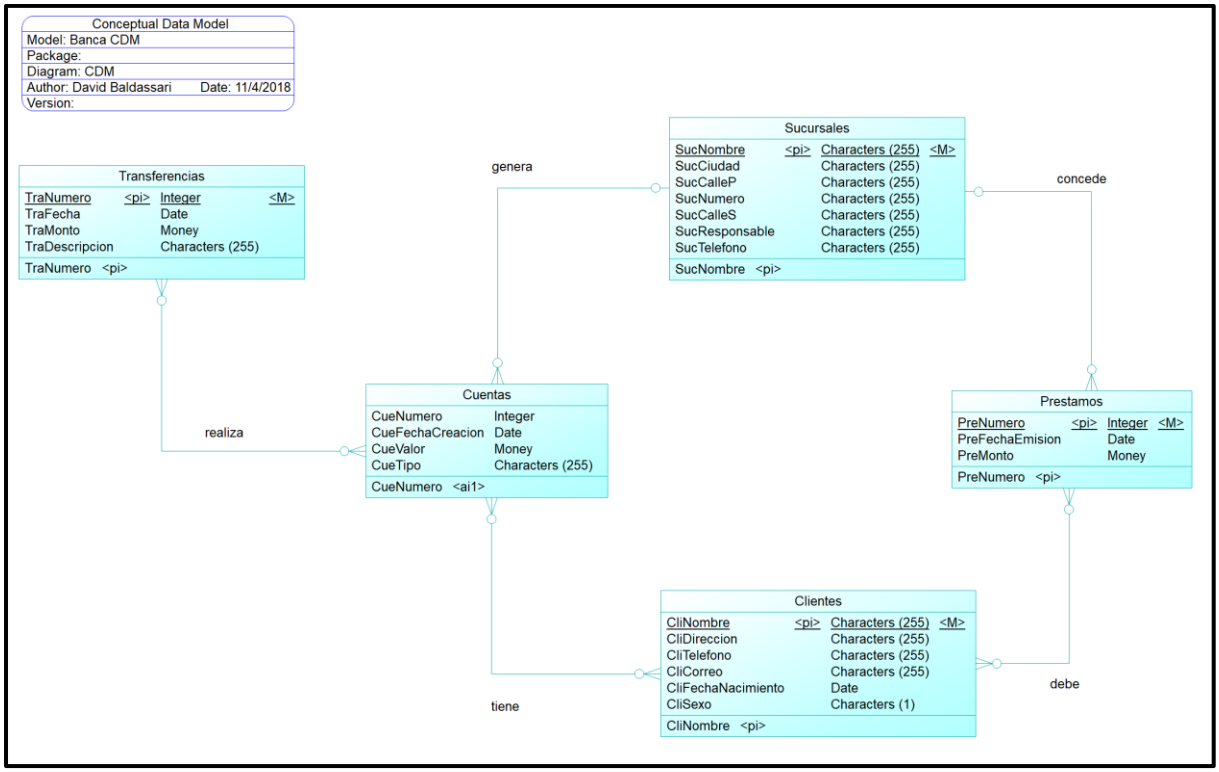


Figura 6-1 CDM Banca (Baldassari, 2018)

En la figura 6-1 se puede observar el modelo conceptual propuesto para el presente trabajo de titulación, en el cual se considera la entidad transferencias, necesaria para la operación transaccional del mismo nombre.

6.1.2 Modelo de datos físico (PDM) Banca para PostgreSQL

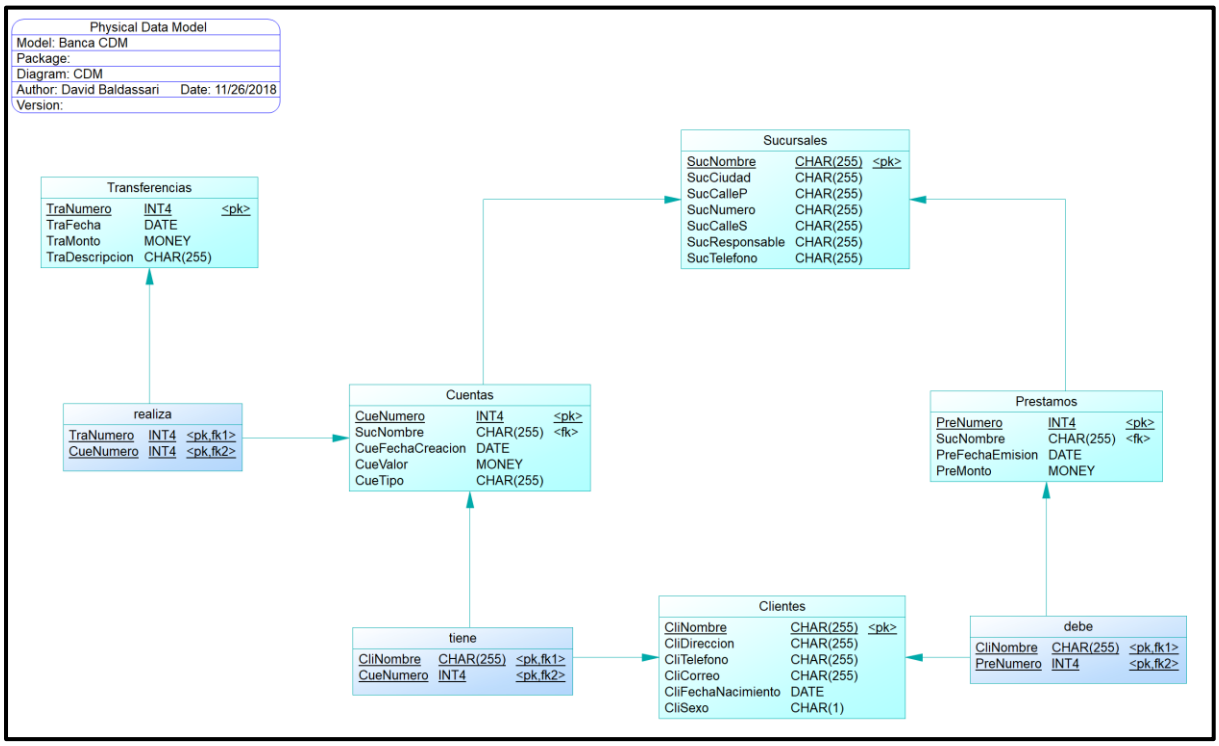


Figura 6-2 PDM Banca PostgreSQL (Baldassari, 2018)

En la figura 6-2 se aprecia el modelo físico con las entidades débiles generadas en aquellas relaciones de muchos a muchos, como lo son las entidades tiene, debe y realiza las cuales contienen las claves primarias de las tablas que se relacionan.

6.1.3 Script Base de datos PostgreSQL

En el caso de PostgreSQL la generación de la base de datos se simplifica puesto que la herramienta PowerDesigner lo permite, es decir, una vez creado el modelo conceptual se puede pedir a la herramienta que produzca el modelo físico del cual, una vez elaborado, se puede generar el script SQL en este caso para el DBMS seleccionado.

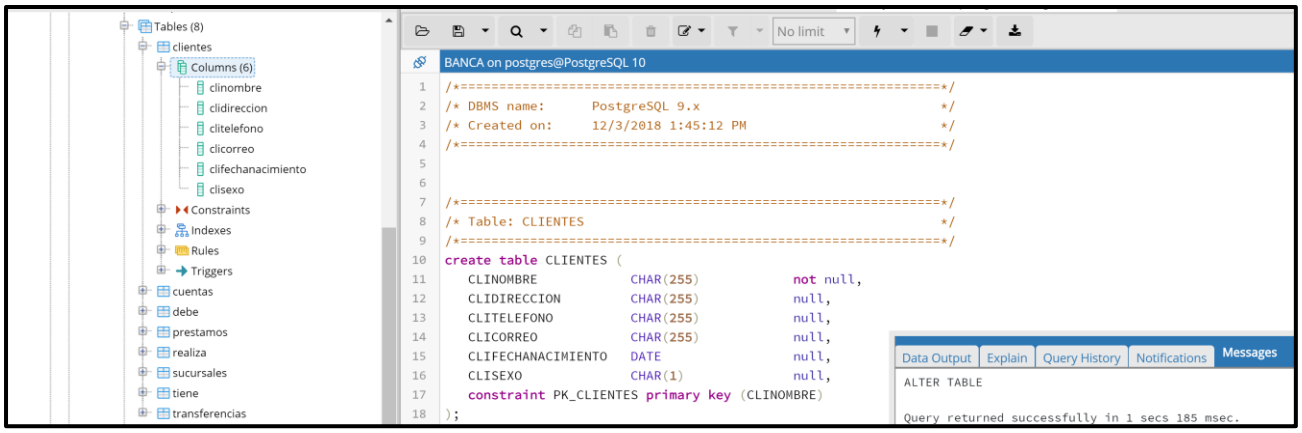


Figura 6-3 Ejecución del script generado por PowerDesigner para PostgreSQL. (Baldassari, 2018)

En la figura 6-3 se observa como el script generado por PowerDesigner se ejecuta correctamente creando las tablas del modelo con sus correspondientes atributos y lo que se puede observar con mayor facilidad gracias a la herramienta proporcionada por PostgreSQL PgAdmin.

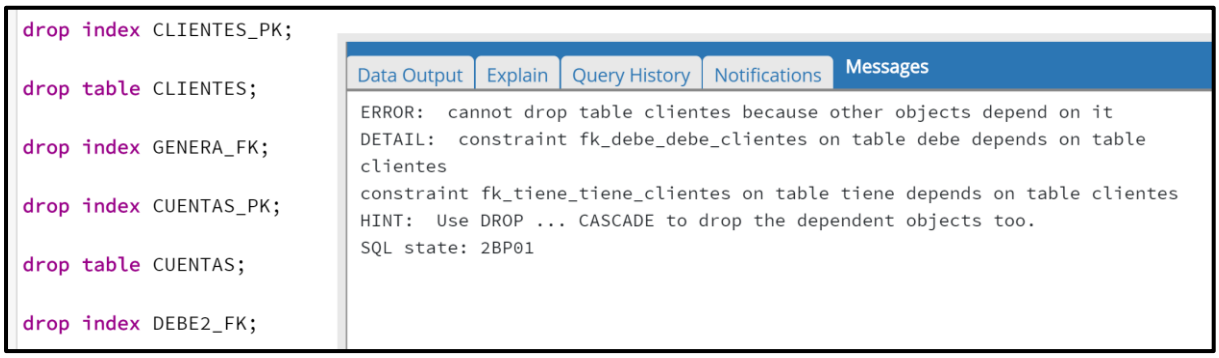


Figura 6-4 Error generado por las relaciones en PostgreSQL. (Baldassari, 2018)

La figura 6-4 presenta el siguiente error si intentamos realizar un *DROP* o eliminar la tabla clientes ya que existen otros elementos que dependen de esta, y recomienda realizar un *DROP CASCADE,* de esta manera se elimina los objetos dependientes. Con esto se puede constatar la existencia de las relaciones entre las entidades.

6.1.4 Generación data de prueba para PostgreSQL

En el presente trabajo se genera la información para las tablas sucursales, clientes, préstamos y cuentas con la herramienta mockaroo en el siguiente url: https://mockaroo.com/ .

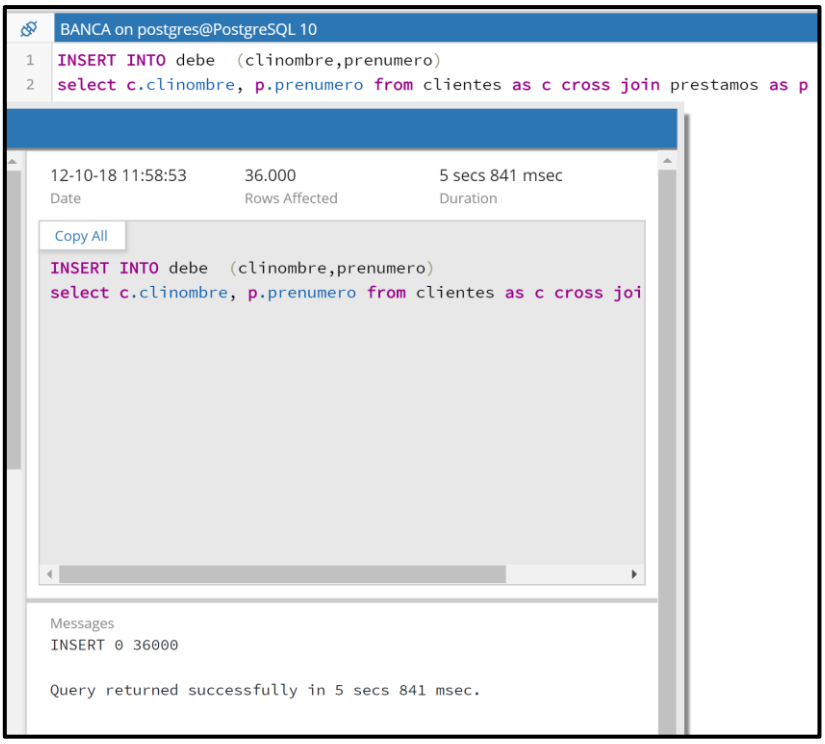


Figura 6-5 inserción de los datos en la entidad débil DEBE con CROSS JOIN en PostgreSQL. (Baldassari, 2018)

Como se puede observar en la figura 6-5 para la data de prueba de las entidades débiles se realiza un CROSS JOIN el cual permite seleccionar un campo de cada tabla relacionada y de esta manera realizar la inserción correspondiente. El query utilizado es el siguiente:

INSERT INTO debe (clinombre,prenumero)

select c.clinombre, p.prenumero from clientes as c cross join prestamos as p

6.1.5 Transferencias en PostgreSQL

Una vez generada la información para la base de datos lo que se procede a actualizar las transferencias y las cuentas existentes y a su vez se insertará en la entidad débil realiza los datos de número de transferencia y el número de las cuentas involucradas.

La primera transferencia para actualizar, en este caso, será la 103, la cual se actualizará la fecha a 2018/01/01, un monto de 30$ de la cuenta 3 a la cuenta 6 y su descripción será 'Pago'.

	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	103	2017-03-05	$100.00	Devolución ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	3	Brazil ...	2010-07-26	$747.74	Corriente ...
2	6	Czech Republic ...	2000-01-07	$264.49	Ahorros ...

	tranumero integer	cuenumero integer

Figura 6-6 Valores iniciales involucrados en la transferencia 103 PostgreSQL. (Baldassari, 2018)

En la figura 6-6 se observa los valores de las cuentas y la transferencia por actualizar, también se aprecia que en la entidad débil realiza no existe ningún registro.

Para empezar, se realiza la prueba con un error de sintaxis:

```
5  UPDATE transferencias set trafecha = '2018-01-01', tramonto = CAST(30 as  money), tradescripcion= 'Pago' where tranumero = 103;
6
7  UPDATE cuentas SET cuevalor = cuevalor - CAST(30 AS money) WHERE cuenumero=3;
8  UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=6;
9
10 INSERT INTO realizar (tranumero,cuenumero) --el error se encuentra en el nombre de la entidad débil
11 values(103,3);
12
13 INSERT INTO realiza (tranumero,cuenumero)
14 values(103,6);
```

Figura 6-7 Query de transferencia con error de sintaxis PostgreSQL. (Baldassari, 2018)

En la figura 6-7 se presenta el *query* ejecutado con el error en la línea 10, de esta manera se debe ejecutar correctamente las actualizaciones posteriores, y en la entidad débil no se debe insertar ninguno de los registros.

Data Output | Explain | Messages | Notifications | Query Histo

```
ERROR:  relation "realizar" does not exist
LINE 6: INSERT INTO realizar (tranumero,cuenumero)
```

	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	103	2014-08-09	$100.00	Devolucion ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	3	Brazil ...	2010-07-26	$747.74	Corriente ...
2	6	Czech Republic ...	2000-01-07	$264.49	Ahorros ...

	tranumero integer	cuenumero integer

Figura 6-8 Valores posteriores a la ejecución del query con error de sintaxis PostgreSQL. (Baldassari, 2018)

En la figura 6-8 se puede observar que los valores no se alteraron en ninguno de los registros, esto se debe a que la versión de PostgreSQL 10.6 ejecuta a todos los comandos como una transacción, de esta manera el DBMS se asegura que los datos no se vean afectados por errores de sintaxis que se puedan cometer en la escritura de un *script.*

Esto también es gracias a que la herramienta Pgadmin permite seleccionar las opciones de *auto commit* y *auto rollback*. Al desactivar *auto commit*, si se presenta un error el DBMS solicita un *rollback* puesto que la transacción se ha interrumpido por el error de sintaxis.

Por este motivo el error que se presenta será correspondiente al olvido por parte del usuario de incrementar el monto de la cuenta a la cual se acredita dicho monto.

```
5  UPDATE transferencias set trafecha = '2018-01-01', tramonto = CAST(30 as  money), tradescripcion= 'Pago' where tranumero = 103;
6
7  UPDATE cuentas SET cuevalor = cuevalor - CAST(30 AS money) WHERE cuenumero=3;
8  -- La linea 9 no se ejecuta
9  --UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=6;
10
11 INSERT INTO realiza (tranumero,cuenumero)
12 values(103,3);
13
14 INSERT INTO realiza (tranumero,cuenumero)
15 values(103,6);
```

Figura 6-9 Query de transferencia 103 con error de usuario PostgreSQL. (Baldassari, 2018)

La información inicial de cada registro y tabla se encuentran en la figura 6-6. En la figura 6-9 se presenta el *query*, el cual no acreditará en la cuenta número 6 el monto de 30$, sin embargo, se realizarán los demás cambios.

Data Output | Explain | Messages | Notifications | Query History

```
INSERT 0 1

Query returned successfully in 61 msec.
```

	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	103	2018-01-01	$30.00	Pago

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	3	Brazil	2010-07-26	$717.74	Corriente
2	6	Czech Republic	2000-01-07	$264.49	Ahorros

	tranumero integer	cuenumero integer
1	103	3
2	103	6

Figura 6-10 Valores posteriores a la ejecución del query de transferencia 103 con error de usuario PostgreSQL. (Baldassari, 2018)

En la figura 6-10 se puede observar cómo se actualizaron todos los valores, a excepción del monto en la cuenta número 6. Es decir, una vez realizada la transacción existe incoherencia en los datos.

A continuación, se realizará la prueba anterior, pero en esta ocasión con los comandos transaccionales *BEGIN*, *COMMIT* y *ROLLBACK*.

```
--transferencia 103. Del 2018/01/01, valor 30$ de la cuenta #3 a la cuenta #6

BEGIN;
Savepoint my_savepoint;--crea un punto de guardado
UPDATE transferencias set trafecha = '2018-01-01', tramonto = CAST(30 as  money), tradescripcion= 'Pago' where tranumero = 103;

UPDATE cuentas SET cuevalor = cuevalor - CAST(30 AS money) WHERE cuenumero=3;
-- La linea 9 no se ejecuta
--UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=6;

INSERT INTO realiza (tranumero,cuenumero)
values(103,3);

INSERT INTO realiza (tranumero,cuenumero)
values(103,6);

ROLLBACK TO SAVEPOINT my_savepoint;--regresa al punto de guardado

COMMIT;
```

Figura 6-11 Query de transferencia 103 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

Los valores iniciales para esta prueba son los presentados en la figura 6-6. El query de la figura 6-11 presenta los comandos de BEGIN, *SAVEPOINT, ROLLBACK* y *COMMIT,* en este ejemplo se crea un punto de guardado antes de realizar cualquier cambio en la data, una vez terminada la inserción en la entidad débil realiza este regresa al punto de guardado sin realizar ningún simulando que el usuario notó que faltaba la acreditación en la cuenta 6.

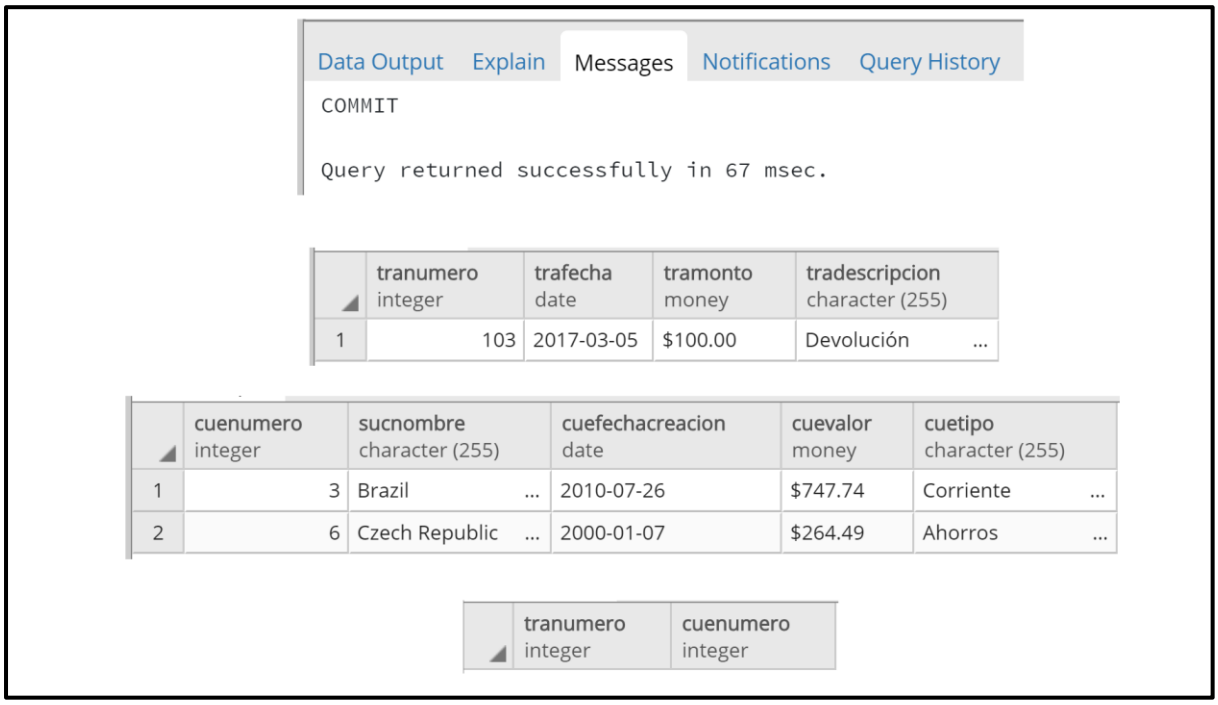
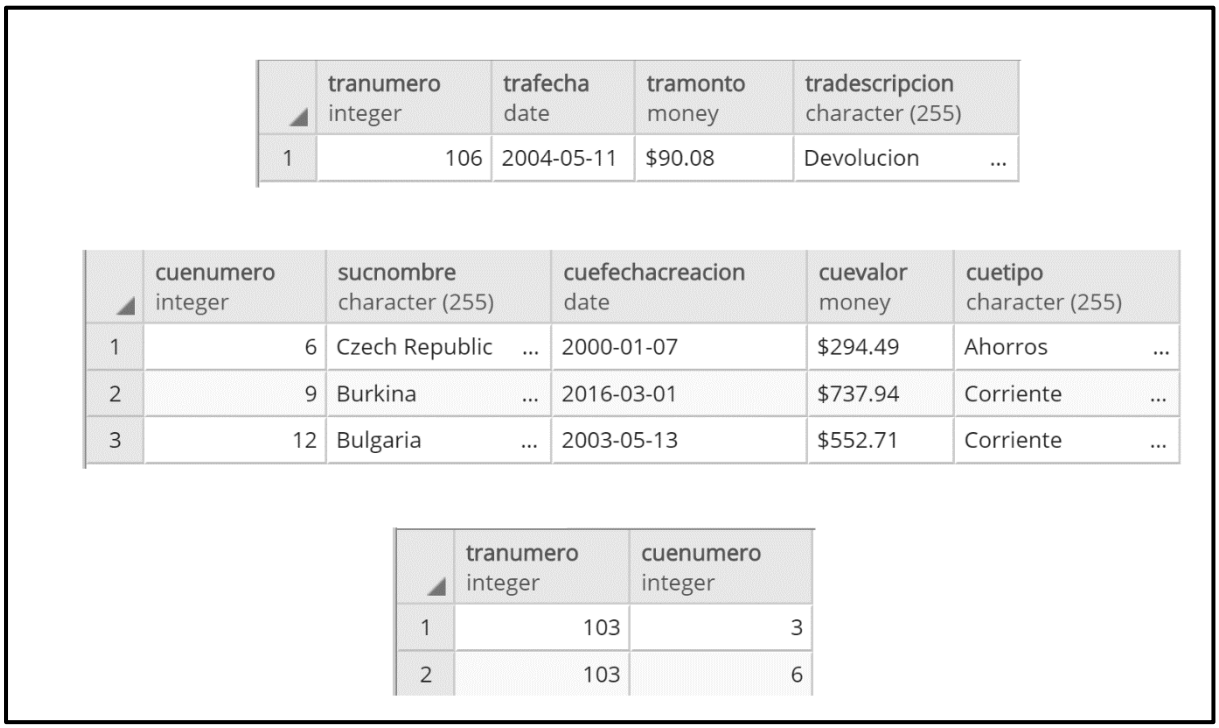


Figura 6-12 Valores posteriores a la ejecución del query de transferencia 103 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

Como se puede observar en la figura 6-12 ningún dato se altera tras la ejecución correcta del query, sin embargo, si el comando de la figura 6-11 se lo ejecuta sin los comandos de *SAVEPOINT* y *ROLLBACK,* el resultado sería el de la figura 6-10, puesto que el query no tiene ningún error que pueda interrumpir la transacción. La solución a este problema se puede presentar en el programa que se encuentre ligado a la base, que permita al usuario confirmar

la transacción o no, siendo la segunda opción el *ROLLBACK* y de esta forma evitando cualquier tipo de incoherencia en los datos.

La transferencia para actualizar a continuación será la 106, en la cual se actualizará la fecha a 2018/02/02, un monto de 60$ de la cuenta 6 y su descripción será 'Pago'. Pero en esta ocasión el monto se dividirá, acreditando 30$ a las cuentas 9 y 12.



	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	106	2004-05-11	$90.08	Devolucion ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	6	Czech Republic ...	2000-01-07	$294.49	Ahorros ...
2	9	Burkina ...	2016-03-01	$737.94	Corriente ...
3	12	Bulgaria ...	2003-05-13	$552.71	Corriente ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6

Figura 6-13 Valores iniciales involucrados en la transferencia 106 PostgreSQL. (Baldassari, 2018)

En la figura 6-13 se encuentran los valores iniciales para la transacción 106 tanto de cuentas como de la transferencia por actualizar y en este caso la entidad débil realiza ya contiene los datos de la transacción anterior.

```
6  UPDATE transferencias set trafecha = '2018-02-02', tramonto = CAST(60 as  money), tradescripcion= 'Pago' where tranumero=106;
7
8  UPDATE cuentas SET cuevalor = cuevalor - CAST(60 AS money) WHERE cuenumero=6;
9  -- La linea 10 no se ejecuta
10 --UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=9;
11 UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=12;
12
13 INSERT INTO realiza (tranumero,cuenumero)
14 values(106,6);
15
16 INSERT INTO realiza (tranumero,cuenumero)
17 values(106,9);
18
19 INSERT INTO realiza (tranumero,cuenumero)
20 values(106,12);
```

Figura 6-14 Query de transferencia 106 con error de usuario PostgreSQL. (Baldassari, 2018)

En la figura 6-14 en la décima línea del *query* se encuentra comentada, de esta manera no se ejecuta y no acreditará 30$ en la cuenta número 9 pero realizará el resto de los cambios e inserciones en las tablas generando una incoherencia de datos.

	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	106	2018-02-02	$60.00	Pago ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	6	Czech Republic ...	2000-01-07	$234.49	Ahorros ...
2	9	Burkina ...	2016-03-01	$737.94	Corriente ...
3	12	Bulgaria ...	2003-05-13	$582.71	Corriente ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6
3	106	6
4	106	9
5	106	12

Figura 6-15 Valores posteriores a ejecutar el query de transferencia 106 con error de usuario PostgreSQL. (Baldassari, 2018)

En la figura 6-15 se puede observar la inconsistencia en la cual el monto en la cuenta número 9 no ha aumentado con respecto a la transacción propuesta originalmente, pero los valores de los demás registros y tablas se han alterado.

```
--transferencia #106. Del 2018/02/02, valor total 60$ de la cuenta #6 a las cuentas #9 y #12 30$ a cada una.
BEGIN;

Savepoint my_savepoint;--crea un punto de guardado

UPDATE transferencias set trafecha = '2018-02-02', tramonto = CAST(60 as  money), tradescripcion= 'Pago' where tranumero=106;

UPDATE cuentas SET cuevalor = cuevalor - CAST(60 AS money) WHERE cuenumero=6;
-- La linea 10 no se ejecuta
--UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=9;
UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=12;

INSERT INTO realiza (tranumero,cuenumero)
values(106,6);

INSERT INTO realiza (tranumero,cuenumero)
values(106,9);

INSERT INTO realiza (tranumero,cuenumero)
values(106,12);

ROLLBACK TO SAVEPOINT my_savepoint;--regresa al punto de guardado

COMMIT;
```

Figura 6-16 Query de transferencia 106 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

En la figura 6-16 se observa la transacción con los comandos *BEGIN, SAVEPOINT, ROLLBACK* y *COMMIT* de esta manera al ejecutar el script no se realiza ningún cambio en los datos involucrados.

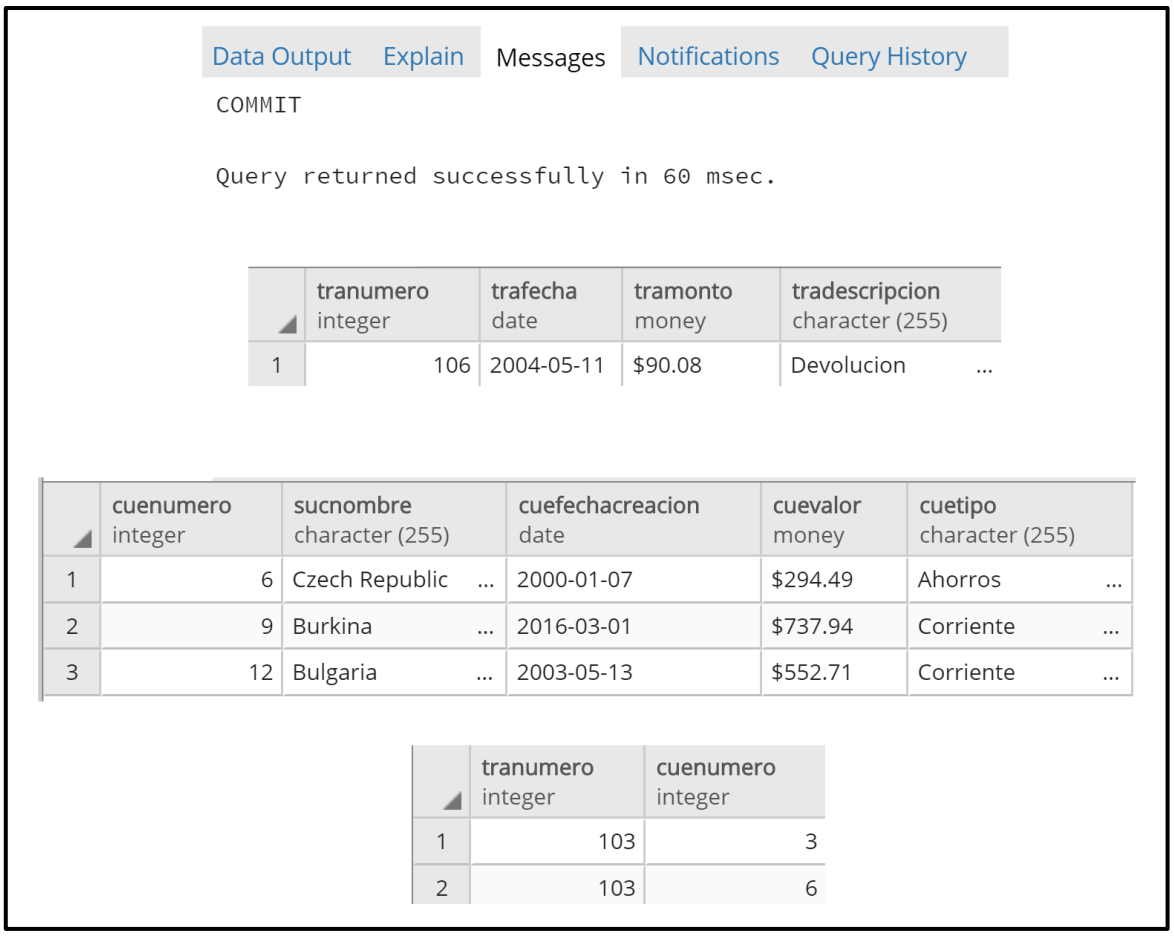


Figura 6-17 Valores posteriores a la ejecución del query de transferencia 106 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

En la figura 6-17 se observa como los valores de los registros no se han alterado y la inserción en la entidad débil no se realiza gracias al comando *ROLLBACK* que permite regresar al punto de guardado creado antes de iniciar con las actualizaciones simulando al usuario seleccionar cancelar en el caso de que se le presente una ventana de confirmación. Pero si el *query* se lo ejecuta sin el comando de ROLLBACK los datos serían los de la figura 6-15, porque no existe ningún error que cause la interrupción de la ejecución del *script*.

La última transferencia para realizar las pruebas será la 109, en la cual se actualizará la fecha a 2018/03/03, un monto de 90$ de la cuenta 9 y su descripción será 'Pago'. Pero en esta ocasión el monto se dividirá, acreditando 30$ a las cuentas 12,15 y 18.

	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	109	2012-03-21	$19.14	Devolucion ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	9	Burkina ...	2016-03-01	$767.94	Corriente ...
2	12	Bulgaria ...	2003-05-13	$582.71	Corriente ...
3	15	Brazil ...	2013-02-19	$663.82	Ahorros ...
4	18	Bahamas ...	2014-08-12	$986.14	Ahorros ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6
3	106	6
4	106	9
5	106	12

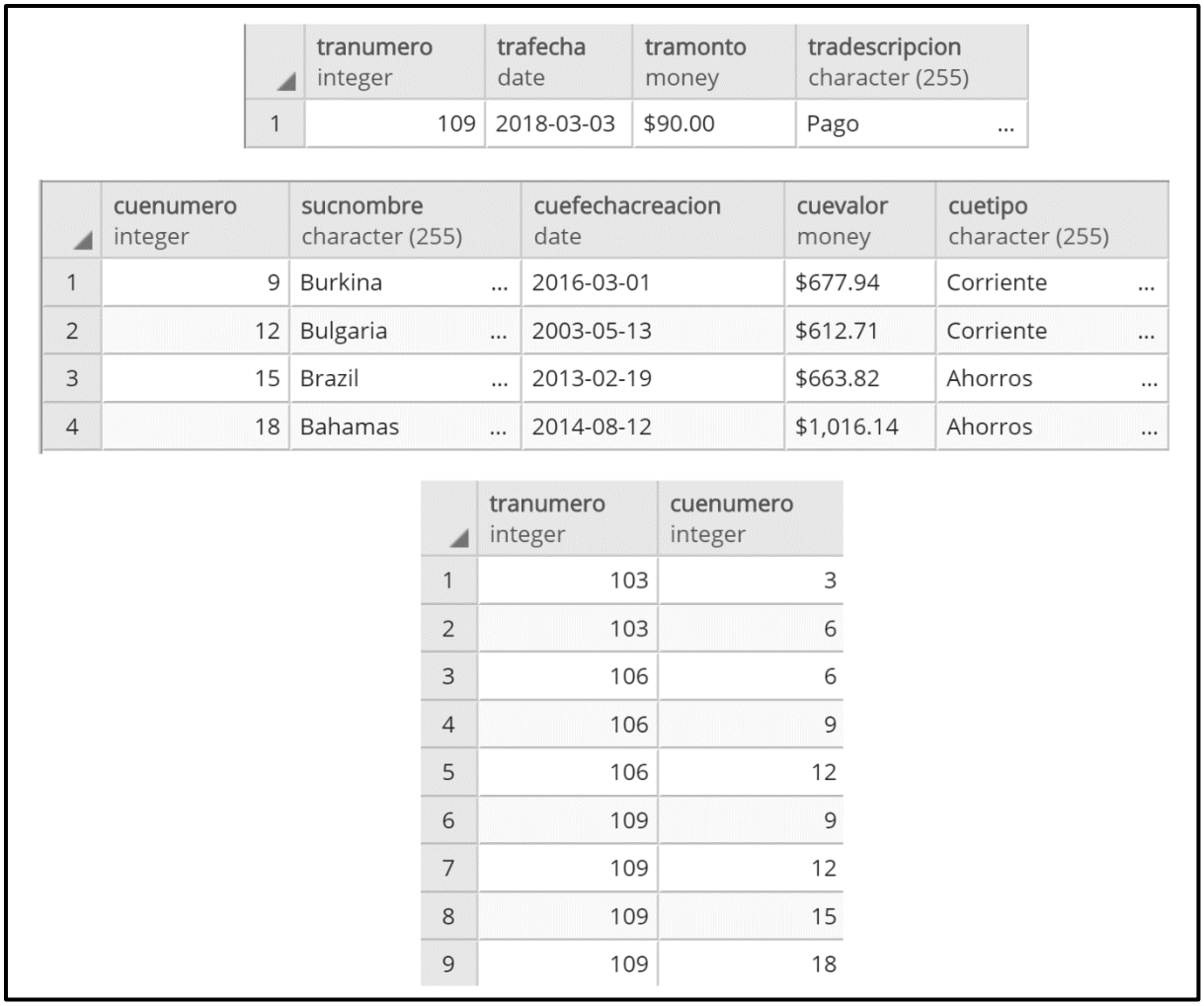
Figura 6-18 Valores iniciales involucrados en la transferencia 109 PostgreSQL. (Baldassari, 2018)

En la figura 6-18 se encuentran los valores correspondientes a la transferencia 109, en esta transacción se puede observar que la entidad débil contiene las transferencias anteriores y que las cuentas 9 y 12 contienen los valores obtenidos de la transacción anterior.

```
4  UPDATE transferencias set trafecha = '2018-03-03', tramonto = CAST(90 as  money), tradescripcion= 'Pago' where tranumero = 109;
5
6  UPDATE cuentas SET cuevalor = cuevalor - CAST(90 AS money) WHERE cuenumero=9;
7  UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=12;
8  --la linea 9 no se ejecuta
9  --UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=15;
10 UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=18;
11
12 INSERT INTO realiza (tranumero,cuenumero)
13 values(109,9);
14
15 INSERT INTO realiza (tranumero,cuenumero)
16 values(109,12);
17
18 INSERT INTO realiza (tranumero,cuenumero)
19 values(109,15);
20
21 INSERT INTO realiza (tranumero,cuenumero)
22 values(109,18);
```

Figura 6-19 Query de transferencia 109 con error de usuario PostgreSQL. (Baldassari, 2018)

En la figura 6-19 se observa que el *query* a ejecutar no acredita en la cuenta 15 los 30$ correspondientes. Pero los demás comandos si los realiza.



	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	109	2018-03-03	$90.00	Pago ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	9	Burkina ...	2016-03-01	$677.94	Corriente ...
2	12	Bulgaria ...	2003-05-13	$612.71	Corriente ...
3	15	Brazil ...	2013-02-19	$663.82	Ahorros ...
4	18	Bahamas ...	2014-08-12	$1,016.14	Ahorros ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6
3	106	6
4	106	9
5	106	12
6	109	9
7	109	12
8	109	15
9	109	18

Figura 6-20 Valores posteriores a la ejecución del query de transferencia 109 con error de usuario PostgreSQL. (Baldassari, 2018)

En la figura 6-20 se observa que todos los cambios se realizan, pero existe la incoherencia en los datos al ver que en la cuenta 15 faltan 30$ por acreditar puesto que el usuario olvidó acreditarlos y generó este error.

```
--transferencia #109. Del 2018/03/03, valor total 90$ de la cuenta #9 a las cuentas #12, #15 y #18 de 30$ a cada una.
BEGIN;
Savepoint my_savepoint;--crea un punto de guardado
UPDATE transferencias set trafecha = '2018-03-03', tramonto = CAST(90 as  money), tradescripcion= 'Pago' where tranumero = 109;

UPDATE cuentas SET cuevalor = cuevalor - CAST(90 AS money) WHERE cuenumero=9;
UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=12;
--la linea 9 no se ejecuta
--UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=15;
UPDATE cuentas SET cuevalor = cuevalor + CAST(30 AS money) WHERE cuenumero=18;

INSERT INTO realiza (tranumero,cuenumero)
values(109,9);

INSERT INTO realiza (tranumero,cuenumero)
values(109,12);

INSERT INTO realiza (tranumero,cuenumero)
values(109,15);

INSERT INTO realiza (tranumero,cuenumero)
values(109,18);
ROLLBACK TO SAVEPOINT my_savepoint;--regresa al punto de guardado
COMMIT;
```

Figura 6-21 Query de transferencia 109 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

En la figura 6-21 se encuentra el *query* de la transferencia 109 pero en esta ocasión se ejecutan los comandos *BEGIN*, *COMMIT*, *SAVEPOINT* y *ROLLBACK* los mismos que simulan que el usuario no aceptó realizar dicha transferencia por lo cual no se guarda ningún cambio.

Data Output | Explain | Messages | Notifications | Query History

```
COMMIT

Query returned successfully in 71 msec.
```

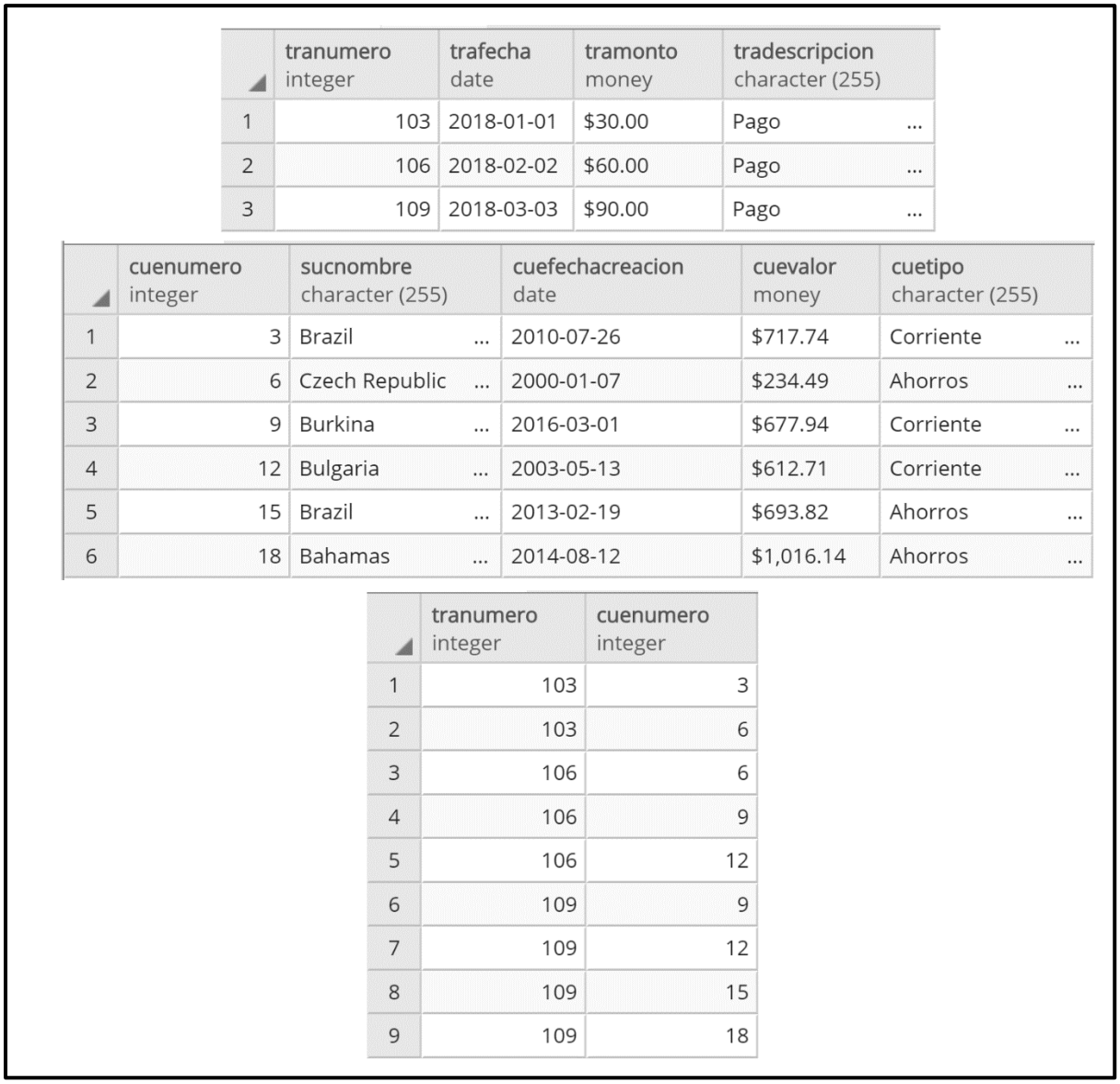
	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	109	2012-03-21	$19.14	Devolucion ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	9	Burkina ...	2016-03-01	$767.94	Corriente ...
2	12	Bulgaria ...	2003-05-13	$582.71	Corriente ...
3	15	Brazil ...	2013-02-19	$663.82	Ahorros ...
4	18	Bahamas ...	2014-08-12	$986.14	Ahorros ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6
3	106	6
4	106	9
5	106	12

Figura 6-22 Valores posteriores a la ejecución del query de transferencia 109 con error de usuario con los comandos BEGIN, COMMIT y ROLLBACK PostgreSQL. (Baldassari, 2018)

En la figura 6-22 se observa como ningún cambio se aplica a la ejecución del *query* de transferencia 109 con el error de usuario dado que el comando *rollback* regresa a la versión del punto de guardado como se puede apreciar en la figura 6-21.



	tranumero integer	trafecha date	tramonto money	tradescripcion character (255)
1	103	2018-01-01	$30.00	Pago ...
2	106	2018-02-02	$60.00	Pago ...
3	109	2018-03-03	$90.00	Pago ...

	cuenumero integer	sucnombre character (255)	cuefechacreacion date	cuevalor money	cuetipo character (255)
1	3	Brazil ...	2010-07-26	$717.74	Corriente ...
2	6	Czech Republic ...	2000-01-07	$234.49	Ahorros ...
3	9	Burkina ...	2016-03-01	$677.94	Corriente ...
4	12	Bulgaria ...	2003-05-13	$612.71	Corriente ...
5	15	Brazil ...	2013-02-19	$693.82	Ahorros ...
6	18	Bahamas ...	2014-08-12	$1,016.14	Ahorros ...

	tranumero integer	cuenumero integer
1	103	3
2	103	6
3	106	6
4	106	9
5	106	12
6	109	9
7	109	12
8	109	15
9	109	18

Figura 6-23 Valores posteriores a realizar las transferencias 103,106,109 correctamente PostgreSQL. (Baldassari, 2018)

En la figura 6-23 se encuentran los valores finales una vez ejecutadas todas las trasferencias correctamente.

6.2 Transaccionalidad en DBMS NoSQL

6.2.1 Modelo de datos conceptual (CDM) Banca para MongoDB

El modelo conceptual es el mismo que el presentado en la figura 6-2, para ambos DBMS puesto que este modelo presenta la idea de como las entidades de la base de datos están relacionadas de una forma general.

6.2.2 Modelo de datos físico (PDM) Banca para MongoDB

Para MongoDB el modelo físico no se lo puede realizar en la herramienta PowerDesigner de la manera que se lo realizó en el caso de PostgreSQL, puesto que esta herramienta no acepta este tipo DBMS para su modelamiento. Sin embargo, para el presente trabajo de titulación se lo realiza en la misma herramienta, pero esta no permitirá generar el script como en el caso anterior.

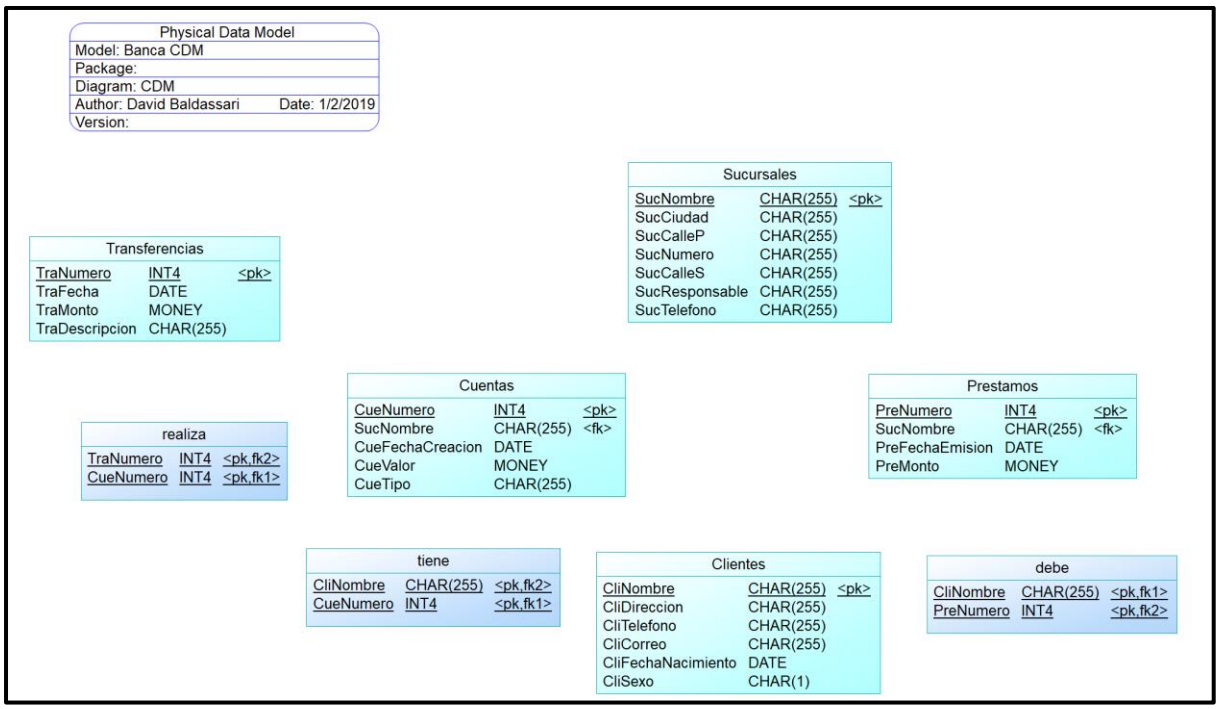


Figura 6-24 PDM Banca MongoDB (Baldassari, 2018)

En la figura 6-24 se observa el PDM en MongoDB realizado en PowerDesigner, por este motivo para representar las colecciones se presentan las tablas correspondientes, y en el caso de las claves foráneas en MongoDB se las simulará como índices de cada objeto o documento. Es importante resaltar que en este modelo las relaciones no se las incorpora dado que estas no existen como tal en un DBMS NoSQL.

6.2.3 Script Base de datos MongoDB

Para el caso de MongoDB, por el momento, no existe una herramienta que genere el script de la manera que lo realiza PowerDesigner para los DBMS SQL que soporta.

```
> use BANCA
switched to db BANCA
> {
... ... db.createCollection("transferencias")
... ... db.createCollection("realiza")
... ... db.createCollection("cuentas")
... ... db.createCollection("tiene")
... ... db.createCollection("sucursales")
... ... db.createCollection("prestamos")
... ... db.createCollection("clientes")
... ... db.createCollection("debe") }
{ "ok" : 1 }
```

Figura 6-25 Creación de las colecciones en MongoDB (Baldassari, 2018)

En la figura 6-25 se observa el comando *USE* el cual deja cambiar a una base de datos incluso si no existe, pero si no se inserta ningún registro o colección la base no se guarda. Las líneas entre las llaves crean las colecciones de la base de datos. Lo que se muestra en la última línea señala que se ejecutó correctamente el comando.

También existe la posibilidad de hacerlo mediante la herramienta de MongoDB Compass la cual es una interfaz gráfica la cual facilita el trabajo de la creación de la base de datos y las colecciones.

6.2.4 Generación data de prueba para MongoDB

En el caso de MongoDB se generará la información con la herramienta mockaroo en el siguiente url: https://mockaroo.com/ . Pero esta ocasión se selecciona un documento CSV.

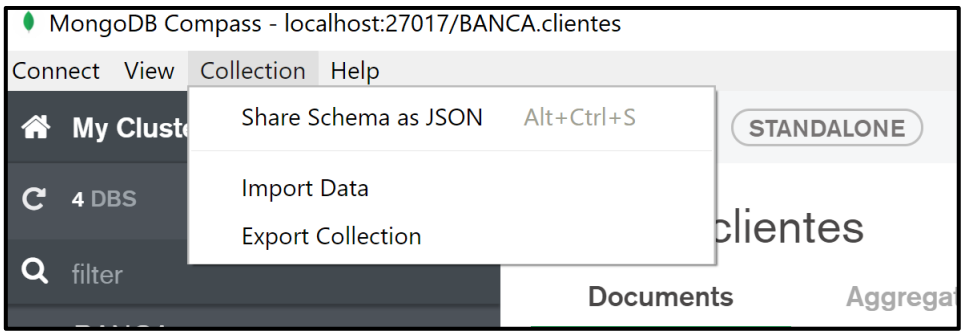


Figura 6-26 Importar archivo para la poblar la base de datos en MongoDB (Baldassari, 2018)

En la figura 6-26 se observa que la herramienta MongoDB Compass permite importar data, es decir que con el archivo CSV obtenido después de generar la información se puede poblar la base de datos.

Una vez generada la data de prueba e ingresada en la base de datos se debe ejecutar un comando el cual permite renombrar el primer campo de la colección puesto que el documento

al ser importado a la base genera un símbolo en el nombre del primer campo el cual al realizar las consultas no retorna nada.

El comando es el siguiente:

db.cuentas.updateMany({}, { $rename: { "cuenumero" : "cuenumero" } })

```
> db.cuentas.updateMany( {}, { $rename: { " cuenumero" : "cuenumero" } } )
```

Figura 6-27 Comando para renombrar un campo MongoDB. (Baldassari, 2018)

A simple vista parece que no existe ningún tipo de diferencia, pero al observarlo en el *shell* de MongoDB se aprecia como en la figura 6-27 el símbolo (*ZERO WIDTH NO-BREAK SPACE*) se presenta como un espacio. Esto se lo debe realizar para todas las colecciones puesto que fueron generadas en la misma herramienta. Lo recomendable es copiar el nombre del campo directamente del *shell* de MongoDB después de realizar un comando *find* para evitar cualquier tipo de error y facilitar la escritura del campo para las consultas o transacciones que lo involucren.

```
> db.prestamos.find().
... forEach( function(p){
... db.clientes.find().
... forEach( function(c){
... var row ={};
... row.prestamos_id=p._id;
... row.clientes_id=c._id;
... db.debe.insert(row);
... });
... });
```

Figura 6-28 Unión de las colecciones préstamos y clientes en la entidad débil debe MongoDB. (Baldassari, 2018)

Al igual que en el caso anterior es necesario realizar un *CROSS JOIN* para llenar las entidades débiles, en el caso de MongoDB se simula el comando de la forma en la que se observa en la figura 6-28, donde se recorre las colecciones de préstamos y clientes para obtener los identificadores de los documentos e insertarlos en la entidad débil debe.

6.2.5 Transferencias en MongoDB

Una vez generada la información y poblada la base de datos, se procede a realizar las transferencias.

La primera transferencia para actualizar, en este caso, será la 103, la cual se actualizará la fecha a 01-Ene-2018, un monto de 30$ de la cuenta 3 a la cuenta 6 y su descripción será 'Pago'.

Figura 6-29 Valores iniciales involucrados en la transferencia 103 MongoDB (Baldassari, 2018)

En la figura 6-29 se observa los valores iniciales de las cuentas y la transferencia por actualizar, también se puede ver que en la entidad débil realiza no contiene ningún documento.

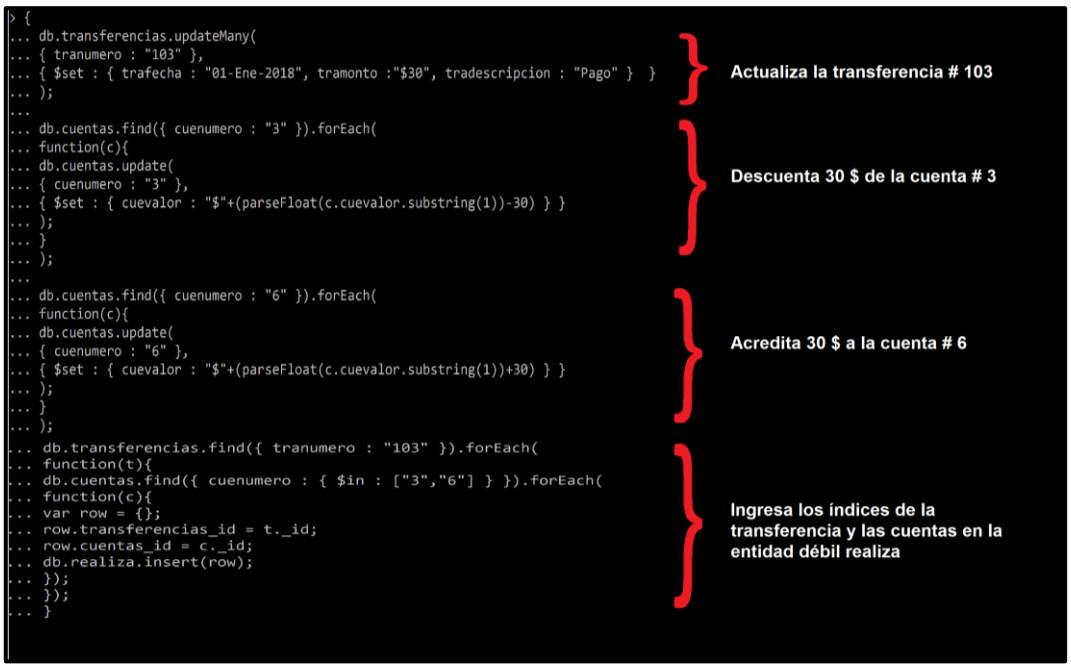


Figura 6-30 Comando de transferencia 103 en MongoDB (Baldassari, 2018)

En la figura 6-30 se puede mirar el código necesario para realizar la transferencia 103 que a comparación de la sentencia SQL utilizada para PostgreSQL es más compleja puesto que contiene funciones y ciertas sentencias correspondientes al lenguaje de programación javascript.

Para la primera prueba se cometerá un error de sintaxis en el bloque de acreditación de la figura 6-30.

```
> {
... db.transferencias.updateMany(
... { tranumero : "103" },
... { $set : { trafecha : "01-Ene-2018", tramonto : "$30", tradescripcion : "Pago" } }
... );
...
... db.cuentas.find({cuenumero:"3"}).forEach(
... function(c){
... db.cuentas.update(
... { cuenumero : "3"  },
... { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))-30) } }
... );
... }
... );
...
... db.cuenta.find({cuenumero : "6" }).forEach(
... function(c){
... db.cuentas.update(
... { cuentas : "6" },
... { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
... );
... }
... );
...
... db.transferencias.find({ tranumero : "103" }).forEach(
... function(t){
... db.cuentas.find({ cuenumero : { $in : ["3","6"] } } ).forEach(
... function(c){
... var row = {};
... row.transferencias_id = t._id;
... row.cuentas_id = c._id;
... db.realiza.insert(row);
... });
... });
... }
```

Figura 6-31 Comando de transferencia 103 con error de sintaxis (colección no existente) MongoDB. (Baldassari, 2018)

En la figura 6-31 se encuentra marcado el error de sintaxis en el que la colección en la cual se realiza la acreditación no existe en la base de datos.

Figura 6-32 Valores posteriores a la ejecución del comando de transferencia 103 con error de sintaxis (colección no existente) MongoDB. (Baldassari, 2018)

Los valores previos a la ejecución del comando se encuentran en la figura 6-29. Como se puede apreciar en la figura 6-32 se ejecutan los comandos a excepción de la acreditación en la cuenta 6.

```
> {
... db.transferencias.updateMany(
... { tranumero : "103" },
... { $set : { trafecha : "01-Ene-2018", tramonto : "$30", tradescripcion : "Pago" } }
... );
... db.cuentas.find({cuenumero:"3"}).forEach(
... function(c){
... db.cuentas.update(
... { cuenumero : "3"  },
... { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))-30) } }
... );
... }
... );
... db.cuentas.find({cuenumero : "6" }).forEach(
... function(c){
... db.cuentas.updte(
... { cuentas : "6" },
... { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
... );
... }
... );
... db.transferencias.find({ tranumero : "103" }).forEach(
... function(t){
... db.cuentas.find({ cuenumero : { $in : ["3","6"] } } ).forEach(
... function(c){
... var row = {};
... row.transferencias_id = t._id;
... row.cuentas_id = c._id;
... db.realiza.insert(row);
... });
... });
... }
```

Figura 6-33 Comando de transferencia 103 con error de sintaxis (update escrito incorrectamente) MongoDB. (Baldassari, 2018)

En la figura 6-33 se encuentra el comando de la transferencia 103 pero en esta ocasión el error se encuentra en un comando *update* escrito incorrectamente.

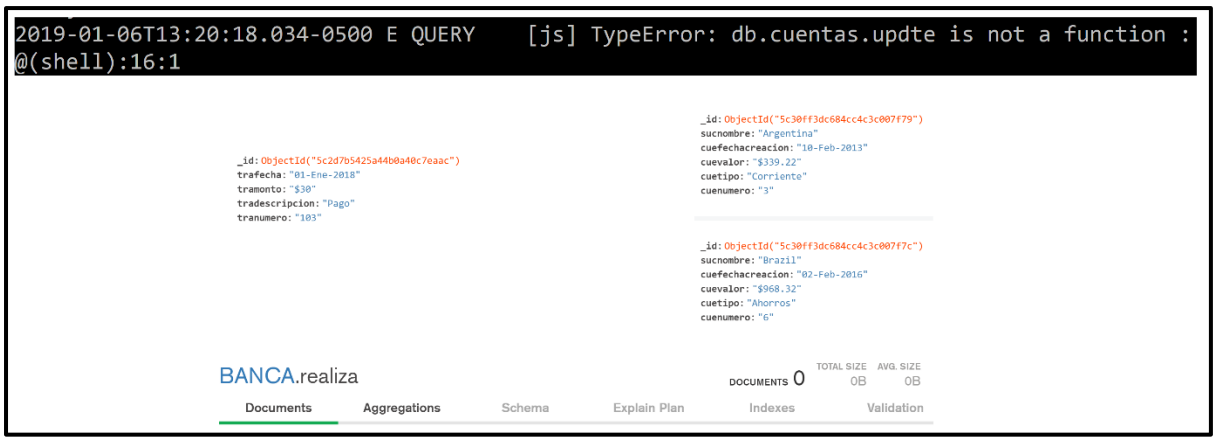


Figura 6-34 Valores posteriores a la ejecución del comando de transferencia 103 con error de sintaxis (update escrito incorrectamente) MongoDB. (Baldassari, 2018)

En la figura 6-34 se puede observar que los comandos se ejecutan hasta el bloque de descuento de los 30 $, y dado que el error se encuentra en el bloque de acreditación el mensaje de este salta y no ejecuta las líneas siguientes, por esta razón la entidad débil continúa sin documentos y generando incoherencia en los datos de la base.

Para realizar esta prueba con transaccionalidad es necesario aplicar replicación puesto que los comandos de MongoDB, actualmente, solo se encuentran disponibles para *replica sets*.

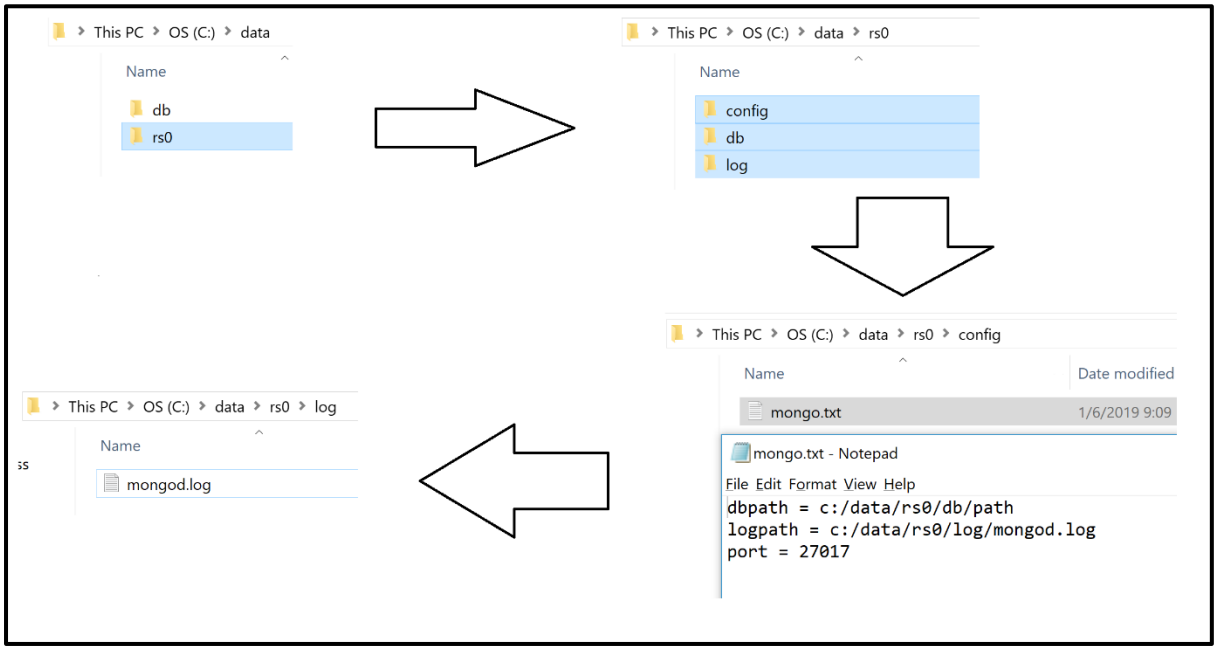


Figura 6-35 Creación de capetas para Replica Set (Baldassari, 2018)

En la figura 6-35 se encuentra la creación de las carpetas que contendrán el Replica Set. Para empezar, se crea la carpeta *rs0*, la cual contendrá las carpetas *config*, *db* y *log*. Dentro de la carpeta *config* se debe generar un archivo de texto de nombre mongo.txt con la configuración deseada para el servidor. Finalmente, dentro de la carpeta log se debe crear un archivo de texto de nombre mongod.log.

```
En la carpeta bin de MongoDB instalado

//Iniciar servidor mongo con replica

mongod --port 27017 --dbpath c:/data/rs0/db --replSet rs0

//Iniciar mongo shell

mongo --port 27017

//Ejecutar el comando para iniciar la replica

rs.initiate()
```

Figura 6-36 Comandos a ejecutar en CMD. (Baldassari, 2018)

A continuación, se abre 2 ventanas de CMD y se debe situar en el directorio donde está instalado MongoDB y se accede hasta el directorio de nombre bin. Una vez situados en este directorio se ejecuta el comando para iniciar el servidor con réplica de la figura 6-36 en uno de los CMD. Una vez iniciado el servidor con *Replica Set*, se selecciona el otro CMD y se accede en el mismo directorio, pero en esta ocasión se ejecuta el comando para iniciar *mongo shell* de la figura 6-36. Cuando *mongo shell* permita ingresar código se ejecuta el comando para iniciar la réplica de la figura 6-36.

Después de realizar este procedimiento se espera unos momentos y al ingresar comandos se puede observar un cambio, en lugar de observar únicamente el símbolo “>” se observará “rs0:PRIMARY>”.

```
    function transferencia(session){
    cuentasCollection = session.getDatabase("BANCA").cuentas;
    transferenciasCollection = session.getDatabase("BANCA").transferencias;
    realizaCollection = session.getDatabase("BANCA").realiza;
    session.startTransaction();
    try{
        transferenciasCollection.updateMany(
        { tranumero : "103" },
        { $set : { trafecha : "01-Ene-2018", tramonto :"$30", tradescripcion : "Pago" } } );
        cuentasCollection.find({ cuenumero : "3" }).forEach(
            function(c){
            cuentasCollection.update(
            { cuenumero : "3" },
            { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))-30) } } );}
        ); cuentasCollection.find({ cuenumero : "6" }).forEach(
            function(c){
//update is wrong in order to test the rollback
            cuentasCollection.updte(
            { cuenumero : "6" },
            { $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }  ); }
        ); transferenciasCollection.find({ tranumero : "103" }).forEach(
            function(t){
            cuentasCollection.find({ cuenumero : { $in : ["3","6"] } }).forEach(
                function(c){
                var row = {};
                row.transferencias_id = t._id;
                row.cuentas_id = c._id;
                realizaCollection.insert(row);
            });
            });
    } catch(error) {
    print("Caught exception during transaction, aborting.");
    session.abortTransaction();
    throw error;
    return;
    }
    session.commitTransaction();
    }

    session = db.getMongo().startSession( { w: 1 , j : true , wtimeout : 1000} );
    try{
    transferencia(session);
    } catch(error){
    throw error;
    } finally{
    session.endSession();
    }
```

Figura 6-37 Comando de transferencia 103 con error de sintaxis (update escrito incorrectamente) utilizando transaccionalidad en MongoDB. (Baldassari, 2018)

En la figura 6-37 se encuentra el comando correspondiente a la transferencia 103. Como se puede ver el comando se encuentra divido en una función llamada transferencia, la cual recibe una sesión la misma que permite inicializar las variables de las colecciones a utilizar. Posteriormente se encuentra el código de la transferencia 103 dentro de un bloque *try,* pero este cuenta con los nombres de las variables en lugar de utilizar el comando *db.collection*. Asegurando de esta manera que se pueda realizar el rollback o en el caso de MongoDB *session.abortTransaction()*. Dentro del bloque catch se tiene la impresión de un mensaje que indica que la transacción se ha abortado. Una vez finalizada la función se tiene las líneas de código las cuales inicializan la variable *session* y llamado a la función de transferencia, para finalmente terminar la sesión con la cual se trabajó dicha transferencia.

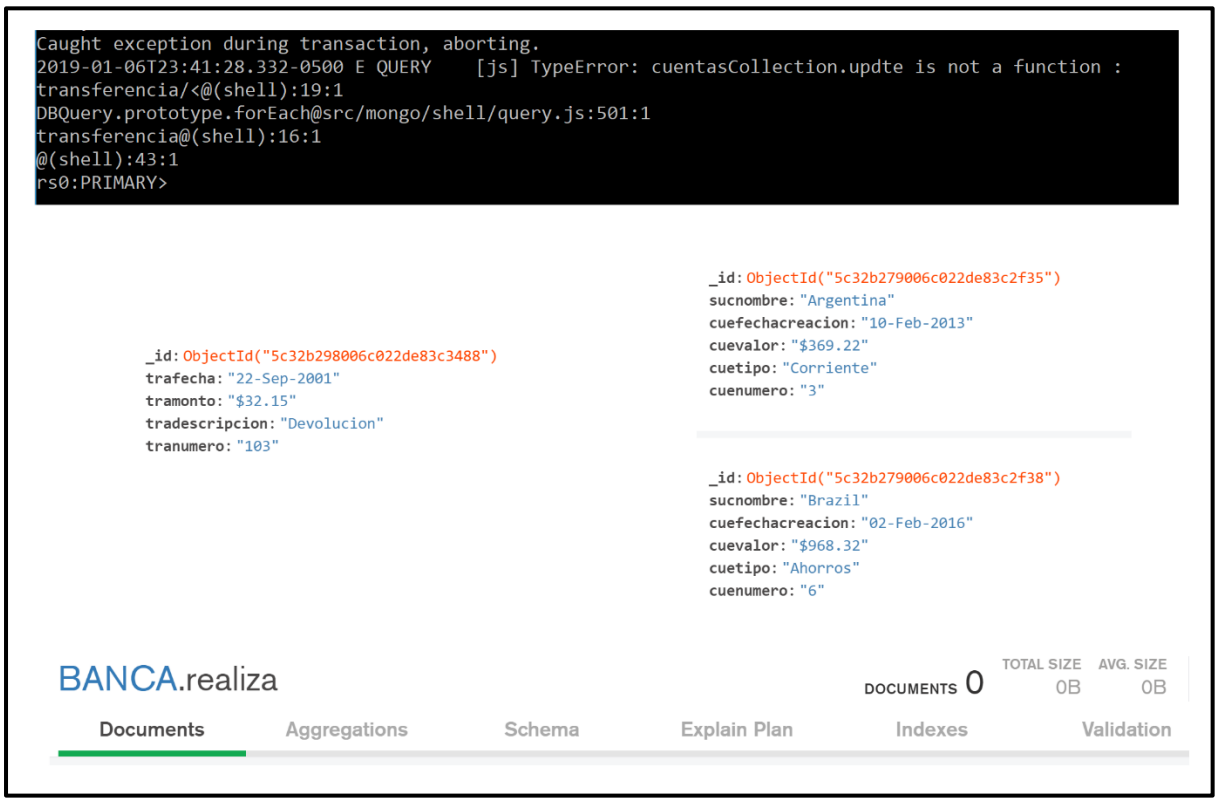


Figura 6-38 Valores posteriores a la ejecución del comando de transferencia 103 con error de sintaxis (update escrito incorrectamente) utilizando transaccionalidad en MongoDB. (Baldassari, 2018)

En la figura 6-38 se observa que se presenta el error correspondiente al *update* escrito incorrectamente y que los valores involucrados no son afectados y se mantienen como los valores originales.

En el caso de colocar una colección que no existe como en la figura 6-31 dentro del comando utilizando transaccionalidad esta lo ejecuta de todas formas, pero el resultado es igual al de la figura 6-32. Esto se da puesto que el comando no contiene errores y solamente contiene un comando correcto para una colección inexistente.

La transferencia a continuación será la 106, la cual se actualizará la fecha a 02-Feb-2018, un monto de 60$ de la cuenta 6 y su descripción será 'Pago'. Pero de igual manera que en el caso de PostgreSQL, la acreditación será 30$ a las cuentas 9 y 12.

Figura 6-39 Valores iniciales involucrados en la transferencia 106 MongoDB. (Baldassari, 2018)

En la figura 6-39 se encuentran los valores iniciales de la transferencia 106 una vez ejecutado la transferencia 103 correctamente. Por ejemplo, en el campo cuevalor de la cuenta 6 ya se encuentra la acreditación de los 30$ y en la entidad débil realiza contiene dos registros.

```
{db.transferencias.updateMany(
{ tranumero : "106" },
{ $set : { trafecha : "02-Feb-2018", tramonto :"$60", tradescripcion : "Pago" } }
);
db.cuentas.find({ cuenumero : "6" }).forEach(
function(c){
db.cuentas.update(
{ cuenumero : "6" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))-60) } }
);
}
);
db.cuentas.find({ cuenumero : "9" }).forEach(
function(c){
db.cuentas.update(
{ cuenumero : "9" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
);
}
);
db.cuentas.find({ cuenumero : "12" }).forEach(
function(c){
db.cuentas.update(
{ cuenumero : "12" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
);
}
);
db.transferencias.find({ tranumero : "106" }).forEach(
function(t){
db.cuentas.find({ cuenumero : { $in : ["6","9","12"] } }).forEach(
function(c){
var row = {};
row.transferencias_id = t._id;
row.cuentas_id = c._id;
db.realiza.insert(row);
});
});}
```

Figura 6-40 Comando de la transferencia 106 en MongoDB. (Baldassari, 2018)

La figura 6-40 contiene el comando para realizar la transferencia 106. Este comando es similar al de la transferencia 103 de la figura 6-30 sin embargo, en éste el código marcado es agregado y se cambia los datos de tal manera que estos concuerden con la descripción de la transferencia propuesta.

```
{db.transferencias.updateMany(
{ tranumero : "106" },
{ $set : { trafecha : "02-Feb-2018", tramonto :"$60", tradescripcion : "Pago" } }
);
db.cuentas.find({ cuenumero : "6" }).forEach(
function(c){
db.cuentas.update(
{ cuenumero : "6" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))-60) } }
);
}
);
db.cuentas.find({ cuenumero : "9" }).forEach(
function(c){
//update escrito incorrectamente
db.cuentas.updte(
{ cuenumero : "9" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
);
}
);
db.cuentas.find({ cuenumero : "12" }).forEach(
function(c){
db.cuentas.update(
{ cuenumero : "12" },
{ $set : { cuevalor : "$"+(parseFloat(c.cuevalor.substring(1))+30) } }
);
}
);
db.transferencias.find({ tranumero : "106" }).forEach(
function(t){
db.cuentas.find({ cuenumero : { $in : ["6","9","12"] } }).forEach(
function(c){
var row = {};
row.transferencias_id = t._id;
row.cuentas_id = c._id;
db.realiza.insert(row);
});});}
```

Figura 6-41 Comando de la transferencia 106 con error de sintaxis (update escrito incorrectamente) en MongoDB. (Baldassari, 2018)

En la figura 6-41 se observa el error de sintaxis en el *script* el cual deberá cancelar la ejecución y de esta manera se crea incoherencia de datos.

Figura 6-42 Valores posteriores a la ejecución del comando de la transferencia 106 con error de sintaxis (update escrito incorrectamente) (Baldassari, 2018)

En la figura 6-42 se puede observar cómo se altera la información en el documento de la transferencia y en el valor de la cuenta 6, generando de esta manera inconsistencia en la base de datos.

```
{
    function transferencia(session){
    cuentasCollection = session.getDatabase("BANCA").cuentas;
    transferenciasCollection = session.getDatabase("BANCA").transferencias;
    realizaCollection = session.getDatabase("BANCA").realiza;
    session.startTransaction();
    try{
    //transferencia #106 cambiar db.collection con el nombre de la variable correspondiente
    }catch(error) {
    print("Caught exception during transaction, aborting.");
    session.abortTransaction();
    throw error;
    return;
    }
    session.commitTransaction();
    }

    session = db.getMongo().startSession( { w: 1 , j : true , wtimeout : 1000} );
    try{
    transferencia(session);
    } catch(error){
    throw error;
    } finally{
    session.endSession();
    }
}
```

Figura 6-43 Comando de transferencia 106 utilizando transaccionalidad en MongoDB. (Baldassari, 2018)

En la figura 6-43 se observa el comando para la transferencia 106, tal como se puede apreciar se debe seleccionar el código de la figura 6-40 e insertarlo en el bloque *try* reemplazando los *db.collection* por las variables inicializadas anteriormente.

Y si en el bloque *try* se agrega el código de la figura 6-41 se obtiene el comando que permite probar la transaccionalidad de MongoDB.

Figura 6-44 Valores posteriores a la ejecución del comando de la transferencia 106 con error de sintaxis (update escrito incorrectamente) en MongoDB. (Baldassari, 2018)

En la figura 6-44 se observa el comando utilizando transaccionalidad, se ejecuta produciendo un error. Sin embargo, la información no se altera puesto que el comando de *session.abortTransaction()* no lo permite.

Figura 6-45 Valores posteriores a la ejecución de las transferencias 103 y 106 en MongoDB. (Baldassari, 2018)

En la figura 6-45 se encuentran los valores involucrados en las transferencias 103 y 106 después de ser ejecutadas correctamente.

7. Conclusiones y Recomendaciones

7.1 Conclusiones

- El teorema de Brewer o CAP[25] es concepto muy interesante el cual con el tiempo y la actualización de los DBMS[26] NoSQL[27] puede llegar a ser obsoleto.
- La consistencia de la información en un DBMS es de gran importancia puesto que si esta data no es confiable muchos de los trabajos los cuales están expuestos a esta información no tendrían validez y podrían afectar en muchos aspectos de una organización.
- El hecho de que las características ACID[28] sean consideradas importantes para una transacción es porque la información no se ve afectada ante las posibles causas o errores externos al DBMS con el que se trabaja.
- Tanto los DBMS SQL como NoSQL manejan concurrencia de distintas maneras, como lo son bloqueos de tablas o en el caso de PostgreSQL con diversas versiones de la base de datos.
- La instalación de PostgreSQL y MongoDB no son complicadas, pero se debe tener en cuenta ciertos puntos durante la misma son de gran importancia como las contraseñas y puertos por utilizar. También se debe tener en cuenta que posterior a la instalación es necesario realizar ciertas configuraciones o adiciones como en el caso de MongoDB.
- Las herramientas gráficas para los DBMS son de gran ayuda durante la elaboración y visualización de las bases de datos.
- La distribución en los DBMS NoSQL lo realiza para cada colección, es decir que, cada tabla se la divide en archivos de un mismo tamaño siendo completamente diferente a un DBMS SQL el cual divide el modelo por tablas, pero no divide a cada tabla.
- La seguridad en los DBMS es de gran importancia puesto que la información que muchas de las organizaciones manejan puede ser delicada, y el mal uso de esta puede perjudicar a muchas personas.
- Actualmente el modelamiento de una base de datos NoSQL no se puede realizar con tanta facilidad como en el caso de las bases de datos SQL para las cuales existen varias herramientas que facilitan su desarrollo.
- La generación de data de prueba actualmente no se la puede realizar para un modelo completo como lo es el de Banca, y se lo debe realizar por partes, es decir, se debe

[25] CAP. – Consistency, Availability, Partition tolerance. Consistencia. - al realizar una consulta o inserción …
[26] DBMS. – *Database Management System*. - *Sistema de Gestion de Base de Datos.*
[27] NoSQL. – *Not only SQL.*
[28] ACID. – *Atomicity, Consistency, Isolation, Durability*. ACID es un acrónimo de Atomicity, Consistency …

generar la información para las entidades, y posteriormente con la misma buscar una manera de generar data coherente en las entidades débiles.

- MongoDB simula la clave foránea dado que al ingresar un nuevo documento a la colección este genera un id para ese documento.
- MongoDB al ser un DBMS NoSQL requiere un mayor manejo de consola y *javascript* por esta razón es necesario utilizar otras herramientas para observar de mejor manera el código.
- La versión de PostgreSQL utilizada en el presente trabajo de titulación permite observar que la herramienta contiene un nivel extra de seguridad, en el cual todos los *queries* ingresados en la herramienta son tratados como transacciones, y en el caso de que exista un error de sintaxis presenta el mensaje correspondiente y solicita un *rollback* para que la información no se vea afectada ante el error.
- MongoDB no es el más indicado para el concepto transaccional por el momento, puesto que muchas líneas del código para MongoDB se pueden resumir en una simple línea de SQL y facilitando la lectura y entendimiento al usuario. También es importante mencionar que en MongoDB es necesaria la replicación para poder utilizar los métodos transaccionales, algo que no es imprescindible en PostgreSQL.

7.2 Recomendaciones

- Se recomendaría a las personas que están involucradas con la tecnología tener presente el teorema de Brewer o CAP como un concepto interesante, puesto que este permite tener una mejor idea de la selección de los DBMS NoSQL.
- Es recomendable que toda organización seleccione un DBMS que considere ser el indicado para el tipo de trabajo a realizar o funciones que tendrá dicho DBMS.
- En las organizaciones comúnmente se considera que la base de datos contiene información fiable por este motivo es necesario que aquellos involucrados en el manejo de la información de dicha organización realicen de la mejor manera su trabajo.
- Se recomienda realizar un trabajo enfocado a la concurrencia y manejo de bloqueos de los DBMS.
- Se recomienda leer todas las pantallas durante la instalación de los DBMS, puesto que varios de estos pueden ser de gran importancia como lo son contraseñas para el superusuario o buscar recomendaciones posteriores a la instalación dado que los usuarios de las diversas herramientas pueden ayudar a conocer soluciones a errores que se puedan presentar después de la instalación.
- Se recomienda estudiar más a fondo los DBMS NoSQL puesto que estos no siempre tienen una herramienta gráfica para el usuario y se debe trabajar con el *Shell* de la

herramienta, lo cual puede ser confuso y de mayor dificultad durante el desarrollo de una base de datos.

- Sería interesante poder observar un trabajo de titulación el cual se base en la seguridad de la información en un DBMS NoSQL, y de esta manera poder conocer si existe alguna debilidad en ellos o son igualmente confiables que un DBMS SQL.
- Se recomienda realizar un trabajo de titulación el cual de alguna manera facilite el modelado de una base de datos NoSQL.
- Se recomienda realizar un trabajo el cual se enfoque en generar data de prueba coherente para un modelo de bases de datos sin necesidad de generar la data por partes.
- Se recomienda empezar con la creación del replica set, de esta manera se omite cualquier error que se pueda presentar al tener una base creada con anterioridad a la generación de la réplica.
- Es recomendable utilizar la herramienta Notepad++ o cualquier otra para la escritura y lectura del código de MongoDB para facilitar la elaboración de los scripts.
- Es recomendable tener en cuenta que en MongoDB las claves primarias no son de gran importancia puesto que el ingreso de documentos nuevos en la colección genera automáticamente un índice hacia dicho objeto.

Glosario

A

ACID

ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

“Atomicidad

La Atomicidad requiere que cada transacción sea "todo o nada": si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios. Un sistema atómico tiene que garantizar la atomicidad en cualquier operación y situación, incluyendo fallas de alimentación eléctrica, errores y caídas del sistema.

Consistencia

La propiedad de Consistencia se asegura que cualquier transacción llevará a la base de datos de un estado válido a otro estado válido. Cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo con todas las reglas definidas, incluyendo (pero no limitado a) los constraints, los cascades, los triggers, y cualquier combinación de estos.

Aislamiento

El aislamiento ("Isolation" en inglés) se asegura que la ejecución concurrente de las transacciones resulte en un estado del sistema que se obtendría si estas transacciones fueran ejecutadas una atrás de otra. Cada transacción debe ejecutarse en aislamiento total; por ejemplo, si T1 y T2 se ejecutan concurrentemente, luego cada una debe mantenerse independiente de la otra.

Durabilidad

La durabilidad significa que una vez que se confirmó una transacción (commit), quedará persistida, incluso ante eventos como pérdida de alimentación eléctrica, errores y caídas del sistema. Por ejemplo, en las bases de datos relacionales, una vez que se ejecuta un grupo de sentencias SQL, los resultados tienen que almacenarse inmediatamente (incluso si la base de datos se cae inmediatamente luego).”

(Seta, 2013)

B

BASE

"Basically Available, Soft State, Eventual Consistency (BASE) es una filosofía de diseño de sistemas de datos que premia la disponibilidad sobre la coherencia de las operaciones. BASE se desarrolló como una alternativa para producir arquitecturas de datos más escalables y asequibles, brindando más opciones para empresas en expansión / clientes de TI y simplemente adquiriendo más hardware para expandir las operaciones de datos." (Techopedia, 2018)

C

CAP

- "**Consistencia**. - al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición.
- **Disponibilidad**. - que todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos.
- **Tolerancia al particionado**. - Los sistemas distribuidos pueden estar divididos en particiones (generalmente de forma geográfica). Así que esta condición implica, que el sistema tiene que seguir funcionando, aunque existan fallos o caídas parciales que dividan el sistema."
  (Genbeta, 2018)

Bibliografía

Advaiya. (15 de Julio de 2015). *Advaiya*. Obtenido de Advaiya: https://www.advaiya.com/blog/all-you-need-to-know-about-nosql/

Amazon. (septiembre de 2018). *Amazon*. Obtenido de Amazon: https://aws.amazon.com/es/relational-database/

Amazon. (octubre de 2018). *Amazon*. Obtenido de Amazon: https://docs.aws.amazon.com/es_es/amazondynamodb/latest/developerguide/Introduction.html

Baldassari, D. (2018). Estudio comparativo de motores de bases de datos SQL y NoSQL para la gestión de información transaccional. Ecuador.

Cambridge Dictionary. (s.f.). *Cambridge Dictionary*. Obtenido de Cambridge Dictionary: https://dictionary.cambridge.org/dictionary/english/data#dataset-cald4

CONDOR, J. (11 de Julio de 2018). BANCA modelo financiero reducido.

Data Entry Outsourced. (02 de Julio de 2013). *Data Entry Outsourced*. Obtenido de Data Entry Outsourced: https://www.dataentryoutsourced.com/blog/components-of-a-database-management-system/

Date, C. J. (2001). *Introducción a los sistemas de bases de datos.* Pearson Educación.

DB-Engines. (octubre de 2018). *DB-Engines*. Obtenido de DB-Engines: https://db-engines.com/en/system/Couchbase#a32

Díaz, J. M. (2 de octubre de 2014). *GMV*. Obtenido de GMV: https://www.gmv.com/blog_gmv/language/es/el-reto-de-las-bases-de-datos-nosql/

Experto en Big Data. (8 de marzo de 2018). *Experto en Big Data*. Obtenido de Experto en Big Data: https://expertoenbigdata.com/que-es-mongodb/#Cuales_son_las_ventajas_de_MongoDB

Genbeta. (septiembre de 2018). *Genbeta*. Obtenido de https://www.genbeta.com/desarrollo/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap

Gestión de Bases de Datos. (diciembre de 2018). *Gestión de Bases de Datos*. Obtenido de Gestión de Bases de Datos: https://gestionbasesdatos.readthedocs.io/es/latest/Tema2/Teoria.html

Grupo de Programación Declarativa. (noviembre de 2018). *Grupo de Programación Declarativa*. Obtenido de Grupo de Programación Declarativa: http://gpd.sip.ucm.es/rafa/docencia/nosql/Sharding.html

IBM. (octubre de 2018). *IBM*. Obtenido de IBM: https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.po.doc/po.htm

Jackson, J. (29 de Julio de 2011). *pcworld*. Obtenido de https://www.pcworld.com/article/236918/article.html

Kedar, S. (2009). *Database Management System.* Technical Publications.

Lopez, B., & Colmenarez, R. (8 de marzo de 2012). *Modelos de BD*. Obtenido de https://modelosbd2012t1.wordpress.com/2012/03/08/bases-de-datos-distribuidas/

Melnik, G. (15 de febrero de 2018). *mongoDB*. Obtenido de mongoDB: https://www.mongodb.com/blog/post/multi-document-transactions

Memcached. (octubre de 2018). *Memcached*. Obtenido de Memcached: https://memcached.org/

Merino, J. P. (2009). *definicion.de*. Obtenido de https://definicion.de/datos/

Microsoft. (30 de 05 de 2018). *Microsoft*. Obtenido de Microsoft: https://docs.microsoft.com/es-es/windows/desktop/Ktm/what-is-a-transaction

Microsoft. (septiembre de 2018). *Microsoft*. Obtenido de https://msdn.microsoft.com/es-es/library/cc434708%28v=vs.71%29.aspx?f=255&MSPPError=-2147217396

MongoDB. (septiembre de 2018). *MongoDB*. Obtenido de MongoDB: https://www.mongodb.com/scale/types-of-nosql-database-management-systems

MongoDB. (octubre de 2018). *MongoDB*. Obtenido de MongoDB: https://www.mongodb.com/transactions

MongoDB. (noviembre de 2018). *MongoDB*. Obtenido de MongoDB: https://docs.mongodb.com/manual/core/transactions/

MongoDB. (noviembre de 2018). *MongoDB*. Obtenido de MongoDB: https://docs.mongodb.com/manual/replication/

MongoDB. (octubre de 2018). *MongoDB*. Obtenido de MongoDB: https://docs.mongodb.com/manual/security/

neo4j. (octubre de 2018). *neo4j*. Obtenido de neo4j: https://neo4j.com/developer/graph-database/

New Gen Apps. (09 de diciembre de 2017). *New Gen Apps*. Obtenido de New Gen Apps: https://www.newgenapps.com/blog/sql-vs-nosql-finding-the-right-dbms-for-your-project

Oracle. (19 de abril de 2016). *Blog de Oracle*. Obtenido de Blog de Oracle: https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql

Oracle. (octubre de 2018). *Oracle*. Obtenido de Oracle: https://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html

PAKHIRA, M. K. (2012). *DATABASE MANAGEMENT SYSTEM.* PHI Learning Pvt. Ltd.

PostgreSQL. (noviembre de 2018). *PostgreSQL*. Obtenido de PostgreSQL: https://www.postgresql.org/docs/8.3/static/tutorial-transactions.html

Power Data. (27 de junio de 2016). *Power Data*. Obtenido de https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/beneficios-de-la-replicacion-de-base-de-datos

Real Academia Española. (septiembre de 2018). *Real Academia Española*. Obtenido de Real Academia Española: http://dle.rae.es/srv/fetch?id=Bskzsq5%7CBsnXzV1

Rouse, M. (septiembre de 2005). *Tech Target*. Obtenido de https://searchsqlserver.techtarget.com/definition/information

Seta, L. D. (19 de febrero de 2013). *Dos Ideas*. Obtenido de https://dosideas.com/noticias/base-de-datos/973-acid-en-las-bases-de-datos

Srivastava, T. (24 de noviembre de 2014). *Analytics Vidhya*. Obtenido de https://www.analyticsvidhya.com/blog/2014/11/types-databases-evolution/

Steffensen, O. (25 de febrero de 2009). *slideshare*. Obtenido de slideshare: https://www.slideshare.net/oddbjorn/Get-to-know-PostgreSQL

Strappazzon, N. (12 de junio de 2016). Obtenido de https://www.swapbytes.com/teorema-cap-base-datos/

Sypron. (29 de enero de 2012). *Sypron*. Obtenido de Sypron: http://sypron.nl/whatis_ase.html

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/8711/oracle-database

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/1243/sql-server

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/3499/postgresql

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/1218/microsoft-access

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/25987/teradata

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/24360/db2

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/26487/amazon-simpledb

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/30340/mongodb

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/30169/apache-cassandra

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/30360/redis

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/30167/apache-hbase

Technopedia. (octubre de 2018). *Technopedia*. Obtenido de Technopedia: https://www.techopedia.com/definition/30316/couchdb

Techopedia. (septiembre de 2018). *Techopedia*. Obtenido de https://www.techopedia.com/definition/27385/concurrency-databases

Techopedia. (septiembre de 2018). *Techopedia*. Obtenido de https://www.techopedia.com/definition/14720/database-replication

Techopedia. (septiembre de 2018). *Techopedia*. Obtenido de https://www.techopedia.com/definition/29841/database-security

Techopedia. (septiembre de 2018). *Techopedia*. Obtenido de https://www.techopedia.com/definition/29164/basically-available-soft-state-eventual-consistency-base

Techopedia. (octubre de 2018). *Techopedia*. Obtenido de Techopedia: https://www.techopedia.com/definition/3498/mysql

TechTarget. (abril de 2018). *TechTarget Search Data Management*. Obtenido de TechTarget Search Data Management: https://searchdatacenter.techtarget.com/es/definicion/Sistema-de-gestion-de-bases-de-datos-relacionales-RDBMS

Thakur, D. (septiembre de 2018). *Ecomputer Notes*. Obtenido de http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information

Tiwari, S. (2011). *Professional NoSQL*. John Wiley & Sons.

TutorialRIde. (septiembre de 2018). *TutorialRIde.com*. Obtenido de https://www.tutorialride.com/distributed-databases/data-replication-in-distributed-system.htm

Tutorials Point. (septiembre de 2018). *Tutorials Point*. Obtenido de https://www.tutorialspoint.com/computer_fundamentals/computer_data.htm

Tutorials Point. (septiembre de 2018). *Tutorials Point*. Obtenido de https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_databases.htm

TutorialsPoint. (noviembre de 2018). *TutorialsPoint*. Obtenido de TutorialsPoint: https://www.tutorialspoint.com/dbms/dbms_transaction.htm

v-espino. (noviembre de 2018). *v-espino*. Obtenido de v-espino: http://www.v-espino.com/~chema/daw1/tutoriales/postgres/distribuidas.htm