ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

TECNICATURA SUPERIOR EN

Telecomunicaciones

Proyecto Integrador I

Unidad 2: Capa de almacenamiento

Las Base de Datos en el IoT

Cristian Gonzalo Vera

Dirección General de EDUCACIÓN TÉCNICA Y FORMACIÓN PROFESIONAL
Ministerio de EDUCACIÓN
CÓRDOBA Seguimos haciendo

ÍNDICE

1. Modelo Relacional vs. Modelo de Negocio
2. Proceso de Diseño del Modelo Relacional
3. Implementación y Normalización de Bases de Datos

- Aplicaciones Prácticas y Escenarios de Uso
  1. Uso de Bases de Datos en IoT
  2. Ejemplos de Proyectos IoT Integrando Bases de Datos
- Mejores Prácticas y Estrategias para BDR
  1. Manejo de Transacciones y Seguridad de Datos
  2. Documentación y Versionado de APIs
  3. Pruebas y Validación de Bases de Datos

3. Las Bases de Datos No Relacionales (BDNR)
   - Fundamentos de Bases de Datos No Relacionales (BDNR)
     1. Definición y Características de BDNR
     2. Tipos de Bases de Datos No Relacionales
     3. Introducción a MongoDB
   - Modelo de Datos en MongoDB
     1. Diferencias entre el Modelo Relacional y el Modelo de Documentos
     2. Estructura de Datos en MongoDB
     3. Esquema Dinámico en MongoDB
   - Operaciones CRUD en MongoDB
     1. Crear, Leer, Actualizar, Eliminar (CRUD)
     2. Uso de la Shell de MongoDB para operaciones CRUD
     3. Integración de MongoDB con Lenguajes de Programación
   - Consultas Avanzadas y Índices en MongoDB
     1. Realización de Consultas Complejas
     2. Optimización de Consultas con Índices
     3. Ejemplos de Consultas en MongoDB
   - Integración y Automatización con MongoDB

Prologo

Introducción

El Internet de las Cosas (IoT) involucra una interconexión creciente de dispositivos que comunican y procesan datos a través de internet. Esta área tecnológica tiene aplicaciones prácticas que benefician a la automatización y eficiencia en múltiples campos. Para su desarrollo es importante contar con una estructura teórica y práctica clara para el diseño y desarrollo de proyectos relacionados con IoT.

Para estructurar estos aprendizajes, la materia "Proyecto Integrador I" utiliza un modelo de 7 capas, distribuido de la siguiente manera:

- **Edge**:
    1. **Percepción**
    2. **Conectividad**
    3. **Preprocesamiento**

- **Fog** == Edge--→ Cloud

- **Cloud**:
    4. **Almacenamiento**
    5. **Procesamiento**
    6. **Análisis**
    7. **Presentación**

La Capa de Almacenamiento

Objetivo

Este documento se centra en la capa de **Almacenamiento** del modelo de IoT, detallando el uso y la implementación de bases de datos tanto relacionales como no

relacionales. Exploraremos cómo estas tecnologías facilitan la gestión eficaz de los datos en proyectos de IoT y su integración con otras capas a través de APIs RESTful.

Importancia de la Capa de Almacenamiento

La capa de Almacenamiento es crucial por varias razones:

- **Centralización de Datos**: Consolida los datos recogidos de múltiples fuentes para su posterior procesamiento y análisis.
- **Integridad y Seguridad**: Asegura que los datos estén almacenados de manera segura y sean consistentemente accesibles.
- **Escalabilidad**: Permite manejar incrementos en el volumen de datos sin degradar el rendimiento del sistema.

Tecnologías Involucradas

- **Bases de Datos Relacionales (MySQL)**
  - **Estructuradas**: Ideal para datos que requieren una organización rigurosa y relaciones complejas.
  - **Transacciones**: Soporta operaciones de transacciones complejas necesarias para garantizar la consistencia de los datos.
- **Bases de Datos No Relacionales (MongoDB)**
  - **Flexibilidad**: Adecuadas para manejar grandes volúmenes de datos semi-estructurados o no estructurados.
  - **Escalabilidad Horizontal**: Facilita la expansión de la capacidad de almacenamiento a través de la distribución de datos.

Integración con Otras Capas

La integración se realiza principalmente a través de APIs RESTful, que permiten:

- **Acceso y Manipulación de Datos**: Las APIs facilitan las operaciones CRUD (crear, leer, actualizar, eliminar) desde y hacia la base de datos.

- **Comunicación entre Capas**: Sirven como el puente para enviar datos procesados desde el Almacenamiento al Análisis y la Presentación, así como la captura de los mismos desde el Edge posterior carga en la BD o bien al Procesamiento.

Conclusión

Al finalizar esta Unidad, estarán familiarizados con el conocimiento necesario para implementar bases de datos en proyectos de IoT, comprendiendo cómo estas se integran dentro de un sistema multicapa para facilitar el flujo y análisis de información.

Las Base de Datos Relacionales

Requerimientos del ABP

1. ¿Qué es una base de datos relacional?
2. ¿Qué es una entidad, una tabla, campo y registro en el contexto de BDR?
3. ¿Qué es el lenguaje SQL y cómo está estructurado?

Fundamentos de Bases de Datos Relacionales

Definición y Características de Bases de Datos Relacionales

Una base de datos relacional organiza datos en tablas compuestas de filas y columnas. Cada tabla representa una entidad del mundo real y las tablas se pueden relacionar entre sí a través de claves, permitiendo consultas sofisticadas y garantizando la integridad de los datos. Estas bases de datos son gestionadas por sistemas conocidos como Sistemas de Gestión de Bases de Datos Relacionales (RDBMS), tales como MySQL, Oracle, o PostgreSQL, los cuales utilizan SQL (Structured Query Language) para administrar y manipular los datos.

Estructura de las Bases de Datos Relacionales

- **Tablas**: Conjuntos organizados de datos donde cada fila representa un registro y cada columna un campo.
- **Campos**: Cada columna en una tabla, que define un tipo de dato específico como texto, número, fecha, etc., que todos los registros en esa columna deben seguir.
- **Registros**: Cada fila en una tabla, que es un conjunto de campos que juntos representan una entidad única o una instancia de la entidad.

Lenguaje SQL y sus Clasificaciones

SQL es el lenguaje estándar utilizado para consultar y manipular bases de datos relacionales. Se clasifica en varias categorías que facilitan diferentes aspectos de la gestión de la base de datos:

1. **DDL (Data Definition Language)**: Incluye comandos para definir o modificar la estructura de la base de datos, como **CREATE**, **ALTER**, y **DROP**.
2. **DML (Data Manipulation Language)**: Utilizado para el manejo de los datos dentro de las bases de datos. Incluye comandos como **INSERT**, **UPDATE**, y **DELETE**.
3. **DCL (Data Control Language)**: Se refiere a los comandos que controlan los aspectos de seguridad en la base de datos. Incluye comandos como **GRANT** y **REVOKE**.
4. **TCL (Transaction Control Language)**: Gestiona las transacciones en la base de datos mediante comandos como **COMMIT** y **ROLLBACK** que ayudan a mantener la integridad de los datos.

Importancia de SQL en las BDR

El uso de SQL en las bases de datos relacionales es fundamental porque permite realizar consultas complejas eficientemente, manejar la seguridad y la integridad de los datos, y asegurar la escalabilidad y la interoperabilidad entre diferentes sistemas y aplicaciones. SQL ayuda a los administradores y desarrolladores a crear sistemas robustos que pueden manejar grandes volúmenes de datos de manera segura y eficaz.

Funciones y Herramientas de SQL

1) Procedimientos Almacenados y Triggers

- **Procedimientos Almacenados**: Son bloques de código SQL que se guardan en la base de datos y se pueden ejecutar repetidamente. Estos procedimientos son útiles para automatizar tareas comunes y mejorar la seguridad y la eficiencia del sistema.
- **Triggers**: Son respuestas automáticas a eventos específicos dentro de la base de datos, como cambios en una tabla. Los triggers ayudan a mantener la integridad de la base de datos ejecutando automáticamente código en respuesta a eventos.

2) Índices y Optimización de Consultas

- **Índices**: Estructuras que mejoran el rendimiento de las operaciones de búsqueda en la base de datos. Funcionan creando un acceso rápido a los datos, lo que puede reducir significativamente el tiempo de respuesta de las consultas.
- **Optimización de Consultas**: Técnicas utilizadas para mejorar la eficiencia de las consultas SQL. Esto incluye la elección adecuada de índices, la redacción eficiente de consultas y la optimización del rendimiento del servidor.

Integración y Automatización de BD

Requerimientos del ABP

1. ¿Cómo se cargan automáticamente los datos, cómo se borran, modifican o actualizan a nivel operativo?

Procesos Automáticos en Bases de Datos Relacionales

Carga y Modificación de Datos

La integración y automatización en las bases de datos relacionales implican procesos que facilitan la manipulación eficiente y automática de los datos. Estos procesos incluyen la carga, modificación, actualización y eliminación de datos sin intervención manual continua.

- **Carga de Datos**: Se realiza típicamente a través de herramientas ETL (Extract, Transform, Load) que automatizan la extracción de datos de diversas fuentes, transforman los datos en un formato adecuado y luego los cargan en la base de datos. Este proceso es crucial en entornos empresariales donde los datos deben ser consolidados de múltiples sistemas.
- **Modificación y Actualización de Datos**: Los triggers en las bases de datos pueden ser configurados para ejecutar acciones automáticas en respuesta a eventos específicos, como la actualización de un registro. Esto asegura que los cambios necesarios en los datos se realicen de manera oportuna y precisa.
- **Eliminación de Datos**: La eliminación programada de datos puede gestionarse mediante scripts que se ejecutan a intervalos regulares, cumpliendo con políticas de retención de datos o necesidades operativas.

Herramientas y Métodos para la Automatización

ETL, Triggers, APIs Las herramientas y métodos de automatización son esenciales para el manejo eficiente de las bases de datos, permitiendo no solo la simplificación de procesos sino también asegurando la integridad y la actualidad de los datos.

- **Herramientas ETL (Extract, Transform, Load)**: Estas herramientas son fundamentales para procesar grandes volúmenes de datos y cargarlos en la base de datos. Herramientas como Informatica PowerCenter, Talend y SSIS de Microsoft son ampliamente utilizadas en la industria.
- **Triggers**: Son procedimientos almacenados que se activan automáticamente en respuesta a eventos de la base de datos. Por ejemplo, un trigger puede ser configurado para actualizar automáticamente un campo de "última modificación" cada vez que un registro es modificado.
- **APIs**: Las interfaces de programación de aplicaciones permiten la interacción programática con la base de datos. En entornos modernos, las APIs RESTful se utilizan para facilitar la integración de la base de datos con otras aplicaciones y servicios web, permitiendo operaciones CRUD remotas y en tiempo real.

Seguridad y Conformidad en la Gestión de Bases de Datos

Políticas y Tecnologías de Seguridad

La seguridad y la conformidad son aspectos críticos en la gestión de bases de datos, especialmente cuando se automatizan procesos y se manejan datos sensibles.

- **Control de Acceso**: Implementar controles de acceso robustos es fundamental para asegurar que solo usuarios autorizados puedan realizar operaciones en la base de datos. Esto incluye la autenticación de usuarios, la asignación de roles y la definición de permisos.
- **Auditoría y Conformidad**: Las herramientas de auditoría son utilizadas para registrar y monitorear todas las acciones realizadas en la base de datos. Esto no solo ayuda a garantizar la conformidad con las regulaciones de protección de datos, como GDPR o HIPAA, sino que también permite identificar y responder a actividades sospechosas o malintencionadas.
- **Cifrado de Datos**: El cifrado de datos en reposo y en tránsito es crucial para proteger la información sensible de accesos no autorizados y violaciones de seguridad.

Gestión Remota y Despliegue de BD

Requerimientos del ABP

1. ¿Cómo se aloja una base de datos en un servidor?
2. ¿Cómo interactuar con ella remotamente para el mantenimiento?
3. ¿Cómo integrarla con servicios de consulta sobre mi base de datos por terceros u otros software?

Alojamiento de Bases de Datos en Servidores

Configuración y Despliegue

Alojar una base de datos en un servidor implica seleccionar el entorno adecuado y configurar el servidor para garantizar la disponibilidad, seguridad y rendimiento óptimos. Los pasos incluyen:

- **Selección del Tipo de Servidor**: Dependiendo de las necesidades, se puede elegir entre servidores físicos, servidores virtuales privados (VPS) o servicios de bases de datos gestionadas en la nube (como Amazon RDS, Azure SQL Database o Google Cloud SQL).
- **Instalación del SGBD**: Instalar el sistema de gestión de bases de datos que se adapte a las necesidades del proyecto, como MySQL, PostgreSQL, SQL Server, entre otros.
- **Configuración de Red**: Configurar adecuadamente la red y los firewalls para asegurar el acceso seguro al servidor de bases de datos, permitiendo solo conexiones autorizadas y necesarias.

Acceso y Mantenimiento Remoto de Bases de Datos

Herramientas y Procedimientos

El acceso y mantenimiento remotos son esenciales para la gestión eficaz de bases de datos, especialmente en entornos distribuidos o en la nube.

- **Herramientas de Acceso Remoto**: Utilizar herramientas como VPNs para acceder de manera segura al servidor de bases de datos. Herramientas de administración de bases de datos basadas en la web, como phpMyAdmin para MySQL, o PgAdmin para PostgreSQL, permiten realizar tareas administrativas a través del navegador.
- **Automatización del Mantenimiento**: Implementar scripts automatizados que se ejecutan a través de cron jobs o tareas programadas para realizar respaldos, actualizaciones y monitorización del rendimiento.

Integración de Bases de Datos con Servicios Externos

Creación e Implementación de APIs

La integración con servicios externos se realiza eficazmente mediante la implementación de APIs RESTful, que proporcionan una interfaz limpia y eficiente para interactuar con la base de datos desde aplicaciones externas.

- **Desarrollo de la API**: Crear APIs utilizando frameworks como Express para Node.js o Django para Python, que permitan operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la base de datos.
- **Seguridad de la API**: Implementar autenticación y autorización en las APIs, usando estándares como OAuth o JWT (JSON Web Tokens), para asegurar que solo los usuarios y aplicaciones autorizados puedan acceder o modificar los datos.
- **Documentación y Versionado**: Mantener documentación clara para las APIs, utilizando herramientas como Swagger o OpenAPI, y gestionar versiones de la API para asegurar compatibilidad y facilitar actualizaciones.

Modelado y Diseño de Bases de Datos

Requerimientos del ABP

1. ¿Cómo se relacionan el modelo relacional y el modelo de negocio?
2. ¿Cuál sería el esquema de trabajo desde el modelo de negocio hasta la implementación de una base de datos relacional y su normalización?

Modelo Relacional vs. Modelo de Negocio

Interrelación de Modelos

El modelo de negocio y el modelo relacional son esenciales en la arquitectura de sistemas de información, pero sirven a diferentes aspectos del desarrollo de software y gestión empresarial.

- **Modelo de Negocio**: Define cómo la empresa crea y entrega valor a sus clientes. Incluye la identificación de ingresos, costos, clientes, ofertas de productos o servicios, y estrategias para alcanzar los objetivos empresariales. Este modelo es crucial para entender las necesidades operativas y de información de la empresa.
- **Modelo Relacional**: Especifica cómo se organiza, almacena y gestiona la información dentro de una base de datos. Utiliza tablas, filas y columnas para estructurar los datos. El diseño de un modelo relacional se basa en los requisitos identificados en el modelo de negocio, asegurando que la base de datos soporte eficazmente las operaciones empresariales.

Proceso de Diseño del Modelo Relacional

Etapas del Diseño

El diseño de un modelo relacional se desarrolla en varias etapas, desde la conceptualización hasta la implementación física, basándose en los requisitos detallados por el modelo de negocio.

1. **Análisis de Requisitos**: Comprensión de las necesidades de información del negocio, incluyendo los tipos de datos que se deben almacenar y cómo se accederá a estos datos. Esto a menudo involucra entrevistas con stakeholders y la revisión de procesos empresariales.
2. **Diseño Conceptual**: Utilización de herramientas de modelado de datos como diagramas entidad-relación (ER) para crear una representación visual de la base de datos, definiendo entidades, atributos y relaciones.
3. **Diseño Lógico**: Traducción del modelo ER a un modelo relacional concreto, definiendo tablas, claves primarias y foráneas, y otras restricciones. Este paso transforma el modelo conceptual en un plan que puede implementarse utilizando un sistema de gestión de bases de datos relacional.

Implementación y Normalización de Bases de Datos

Desarrollo y Optimización

Una vez diseñado el modelo relacional, el siguiente paso es la implementación y la normalización para garantizar la eficiencia y la integridad de la base de datos.

1. **Implementación**: Creación física de la base de datos en un sistema de gestión de bases de datos como MySQL, Oracle, o SQL Server. Esto incluye la definición de tablas, la creación de índices y la implementación de restricciones.
2. **Normalización**: Proceso de estructuración de las tablas para minimizar la redundancia de datos y mejorar la integridad. La normalización se realiza aplicando una serie de "formas normales", que son reglas para evaluar y ajustar el diseño de la base de datos. Estas incluyen:
    - **Primera Forma Normal (1NF)**: Asegura que las tablas contienen solo valores atómicos y que cada campo contiene datos de un solo tipo.
    - **Segunda Forma Normal (2NF)** y **Tercera Forma Normal (3NF)**: Eliminan las dependencias funcionales y transversales para reducir la redundancia y las anomalías en la inserción, actualización o eliminación.

Aplicaciones y Escenarios de Uso

Requerimientos del ABP

1. ¿Cómo pueden las bases de datos gestionar eficientemente la gran cantidad de datos generados por dispositivos IoT?
2. ¿Qué tipos de proyectos IoT pueden beneficiarse significativamente de la integración con bases de datos?
3. ¿Cómo se distribuye el manejo de datos entre el edge y el cloud en un sistema IoT?

Uso de Bases de Datos en IoT

Gestión de Grandes Volúmenes de Datos

En el ámbito del Internet de las Cosas (IoT), las bases de datos no solo sirven para almacenar datos, sino que también son fundamentales para procesar y analizar la información recopilada por innumerables dispositivos distribuidos. La capacidad para manejar grandes volúmenes de datos en tiempo real y con alta disponibilidad es crucial.

- **Almacenamiento Eficiente**: Las bases de datos en un entorno IoT deben ser capaces de manejar y almacenar eficientemente grandes volúmenes de datos que fluyen continuamente desde dispositivos IoT.
- **Procesamiento en Tiempo Real**: Implementar bases de datos que soporten el procesamiento en tiempo real para permitir acciones inmediatas y decisiones basadas en los datos recibidos.
- **Análisis y Obtención de Insights**: Las bases de datos deben soportar análisis avanzados para convertir los datos crudos en información útil, que puede influir en la toma de decisiones y en la mejora de procesos.

Ejemplos de Proyectos IoT Integrando Bases de Datos

Escenarios Diversificados en IoT

Los proyectos IoT que integran bases de datos son increíblemente variados, abarcando desde la automatización del hogar hasta soluciones industriales complejas. A continuación, se presentan ejemplos donde la integración de bases de datos es fundamental:

1. **Smart Cities (Ciudades Inteligentes)**
    - **Gestión de Tráfico y Transporte**: Bases de datos que recogen y analizan datos de sensores viales para optimizar semáforos y rutas de transporte público.
    - **Gestión de Recursos Energéticos**: Análisis de patrones de consumo energético para mejorar la eficiencia y reducir costos en iluminación pública y edificios municipales.
2. **Agricultura Inteligente**
    - **Monitoreo de Condiciones Climáticas**: Uso de bases de datos para analizar datos ambientales y mejorar las decisiones agrícolas, como la irrigación y el tratamiento de cultivos.
    - **Gestión de Recursos**: Optimización del uso de fertilizantes y agua basada en datos históricos y actuales almacenados y procesados en bases de datos.
3. **Salud Conectada**
    - **Monitoreo de Pacientes**: Bases de datos que recopilan datos de dispositivos de monitoreo de salud para seguir la evolución de enfermedades crónicas o la recuperación post-operatoria.
    - **Investigación Médica**: Agregación y análisis de grandes conjuntos de datos de investigación para acelerar descubrimientos médicos y la personalización del tratamiento.

Distribución del Manejo de Datos: Edge vs. Cloud

Optimización de la Arquitectura de Datos

En un sistema IoT, la distribución del manejo de datos entre el edge y el cloud es vital para la eficiencia y la respuesta rápida.

- **Edge Computing**: El procesamiento de datos en el edge minimiza la latencia al procesar datos localmente en el dispositivo o cerca de él, lo que es crucial para aplicaciones en tiempo real.
- **Cloud Computing**: El cloud proporciona una potente capacidad de procesamiento y almacenamiento a gran escala, ideal para análisis profundos y almacenamiento a largo plazo.

Mejores Prácticas y Estrategias para BDR

Requerimientos del ABP

1. ¿Cuáles son las mejores prácticas para el manejo de transacciones y la seguridad de datos en BDR?
2. ¿Cómo se debe gestionar la documentación y el versionado de APIs que interactúan con BDR?
3. ¿Qué métodos se recomiendan para las pruebas y validación de bases de datos?

Manejo de Transacciones y Seguridad de Datos

Estrategias de Integridad y Confidencialidad

El manejo eficiente de las transacciones y la seguridad de los datos son fundamentales para mantener la integridad y la confiabilidad de las bases de datos relacionales.

- **Transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)**: Garantizar que cada transacción se procese de manera fiable es esencial. Las propiedades ACID aseguran que las transacciones sean procesadas de manera integral, manteniendo la consistencia de la base de datos incluso en casos de fallo.
- **Mecanismos de Control de Acceso**: Implementar políticas de seguridad robustas mediante controles de acceso basados en roles y permisos detallados que regulen quién puede ver o modificar los datos.
- **Encriptación y Masking**: Proteger los datos sensibles usando encriptación para datos en reposo y en tránsito, y masking para ocultar datos sensibles sin alterar el formato original.

Documentación y Versionado de APIs

Mantenimiento de Interfaces de Programación de Aplicaciones

La documentación y el versionado adecuados de las APIs son esenciales para la gestión eficaz de las interfaces que interactúan con las bases de datos.

- **Documentación Exhaustiva**: Utilizar herramientas como Swagger o OpenAPI para documentar todas las APIs de manera clara y detallada. Esto incluye descripciones de endpoints, parámetros, métodos de respuesta y modelos de datos.
- **Gestión de Versiones**: Implementar un sistema de versionado para las APIs para gestionar cambios sin interrumpir las aplicaciones existentes. Esto ayuda a los desarrolladores a adaptarse a nuevas funciones o cambios sin comprometer la funcionalidad existente.

Pruebas y Validación de Bases de Datos

Aseguramiento de la Calidad y Optimización

Las pruebas y la validación son críticas para asegurar que la base de datos y sus aplicaciones asociadas funcionen como se espera y cumplan con los requisitos especificados.

- **Pruebas Unitarias y de Integración**: Realizar pruebas unitarias para verificar cada componente o módulo y pruebas de integración para asegurar que todos los módulos trabajen juntos de manera adecuada.
- **Pruebas de Regresión**: Asegurar que las nuevas modificaciones no afecten las funcionalidades existentes.
- **Pruebas de Estrés y Rendimiento**: Evaluar cómo la base de datos maneja grandes volúmenes de datos o un alto número de solicitudes simultáneas para identificar y mitigar posibles cuellos de botella.

- **Auditorías de Seguridad**: Realizar auditorías regulares para detectar y rectificar vulnerabilidades de seguridad, garantizando que la base de datos esté protegida contra amenazas internas y externas.

Las Base de Datos No Relacionales

Requerimientos del ABP

1. ¿Qué son las bases de datos NoSQL y cuáles son sus tipos?
2. ¿Qué es MongoDB y cuáles son sus características principales?

Definición y Características de BDNR

NoSQL: Flexibilidad y Escalabilidad

Las bases de datos NoSQL, o "Not Only SQL", son sistemas de gestión de bases de datos que proporcionan un mecanismo para el almacenamiento y la recuperación de datos que es modelado de formas distintas a las relaciones tabulares utilizadas en las bases de datos relacionales. Estas bases de datos son especialmente útiles para manejar grandes volúmenes de datos desestructurados o semi-estructurados y son conocidas por su flexibilidad, rendimiento escalable y alta capacidad para manejar grandes volúmenes de datos rápidamente.

- **Flexibilidad en los Esquemas**: Permiten modificar los esquemas sin interrumpir las aplicaciones existentes.
- **Escalabilidad Horizontal**: Facilitan la distribución de datos a través de múltiples servidores.
- **Alto Rendimiento con Grandes Volumen de Datos**: Optimizadas para velocidades de lectura y escritura rápidas y para manejar grandes volúmenes de datos.

Tipos de Bases de Datos No Relacionales

Variedad de Modelos de Datos

Las bases de datos NoSQL se clasifican en varios tipos, cada uno optimizado para diferentes tipos de necesidades de datos y aplicaciones:

- **Documentales**: Como MongoDB y CouchDB. Estas bases de datos almacenan datos en documentos JSON, XML o BSON, lo que los hace ideales para aplicaciones que recopilan y manejan diferentes tipos de datos estructurados y semi-estructurados.
- **Clave-valor**: Como Redis y DynamoDB. Están diseñadas para almacenar, recuperar y administrar arrays asociativos. Son extremadamente rápidas para consultas de recuperación y útiles para aplicaciones que necesitan acceso rápido a grandes volúmenes de datos.
- **Columnares**: Como Cassandra y HBase. Optimizadas para lecturas y escrituras rápidas en grandes volúmenes de datos distribuidos, almacenando datos en columnas en lugar de filas, lo que es ideal para análisis en tiempo real.
- **Grafos**: Como Neo4j y ArangoDB. Diseñadas para almacenar y navegar relaciones entre datos. Son especialmente útiles para aplicaciones donde las relaciones entre los datos son tan importantes como los datos mismos, como las redes sociales o sistemas de recomendación.

Introducción a MongoDB

Características y Casos de Uso

MongoDB es un sistema de base de datos NoSQL orientado a documentos que almacena datos en formatos JSON-like. Es uno de los sistemas de bases de datos NoSQL más populares debido a su naturaleza de alto rendimiento, alta disponibilidad y fácil escalabilidad.

- **Características Principales**:
    1. **Esquema Dinámico**: No requiere un esquema fijo, permitiendo que los documentos en una colección tengan diferentes campos y estructuras.
    2. **Indexación**: Soporta índices secundarios, lo que permite realizar consultas eficientes en cualquier campo del documento.
    3. **Replicación**: Soporta la replicación automática, proporcionando alta disponibilidad a través de conjuntos de réplicas.

4. **Sharding**: Distribuye datos a través de múltiples máquinas automáticamente, lo que permite escalar horizontalmente.

- **Casos de Uso Apropiados**:
    1. **Aplicaciones Web y Móviles**: Donde la flexibilidad del esquema y la escalabilidad horizontal son críticas.
    2. **Big Data y Análisis en Tiempo Real**: Capacidad para manejar grandes volúmenes de datos con variabilidad en la estructura.
    3. **Gestión de Contenidos y Catálogos**: Donde cada registro puede ser único y cambiar con el tiempo.

Modelo de Datos en MongoDB

Requerimientos del ABP

1. ¿Cómo se comparan los modelos de datos relacional y NoSQL?
2. ¿Cómo se estructuran los datos en MongoDB?

Diferencias entre el Modelo Relacional y el Modelo de Documentos

Comparación de Paradigmas

La elección entre un modelo relacional y un modelo basado en documentos como MongoDB depende de las necesidades específicas de la aplicación, la naturaleza de los datos y los requisitos de rendimiento.

- **Modelo Relacional**:
  - **Estructura**: Datos organizados en tablas con filas y columnas. Cada tabla representa un tipo de entidad y las relaciones entre entidades se gestionan mediante claves primarias y foráneas.
  - **Esquema Rígido**: Requiere definiciones de esquema precisas y cambios de esquema pueden ser disruptivos.
  - **Consultas**: Usa SQL para realizar consultas, que es poderoso para operaciones complejas de unión y transacción.
- **Modelo de Documentos (MongoDB)**:
  - **Estructura**: Datos almacenados en documentos, que son entidades autocontenidas y fácilmente representables en formatos como JSON o BSON.
  - **Esquema Dinámico**: Permite que cada documento tenga una estructura diferente. Ideal para datos que varían en estructura o donde la definición de esquema es flexible.

- **Consultas**: Consultas basadas en la estructura del documento sin necesidad de unirse a operaciones, simplificando el desarrollo y mejorando el rendimiento en operaciones de datos masivos.

Estas diferencias subrayan que mientras los modelos relacionales son ideales para aplicaciones que requieren integridad de transacciones y relaciones complejas, los modelos de documentos ofrecen flexibilidad y escalabilidad, especialmente útiles en aplicaciones web modernas, big data y sistemas que requieren un alto rendimiento en operaciones de escritura y lectura.

Estructura de Datos en MongoDB

Documentos y Colecciones

MongoDB utiliza documentos para almacenar registros y colecciones para agrupar estos documentos, ofreciendo una estructura más intuitiva para desarrolladores acostumbrados a trabajar con lenguajes de programación orientados a objetos.

- **Documentos**: En MongoDB, un documento es una estructura de datos compuesta de pares campo-valor. Los campos pueden incluir otros documentos, arrays y arrays de documentos, lo que es una diferencia fundamental respecto a los modelos relacionales que requieren que cada "campo" (columna) contenga un solo dato.
- **Colecciones**: Las colecciones en MongoDB son equivalentes a las tablas en las bases de datos relacionales, pero sin un esquema fijo. Esto significa que los documentos dentro de una misma colección pueden tener diferentes campos, lo que permite una gran flexibilidad al manejar datos heterogéneos.

Esquema Dinámico en MongoDB

Flexibilidad y Agilidad en el Desarrollo

Uno de los mayores beneficios de MongoDB es su esquema dinámico, que permite a los desarrolladores crear y modificar la estructura de datos sin tener que preocuparse por las restricciones de un esquema fijo.

- **Adaptabilidad**: Ideal para aplicaciones que evolucionan rápidamente y donde nuevos tipos de datos pueden aparecer conforme se desarrolla y expande la aplicación.
- **Desarrollo Rápido**: Reduce la necesidad de realizar mantenimiento de base de datos relacionado con la modificación del esquema, lo cual puede ser costoso y propenso a errores en modelos relacionales.

Operaciones CRUD en MongoDB

Requerimientos del ABP

1. ¿Cómo se realizan las operaciones CRUD en MongoDB?
2. ¿Cómo se integra MongoDB con otros lenguajes de programación para realizar estas operaciones?

Crear, Leer, Actualizar, Eliminar (CRUD)

Fundamentos de las Operaciones CRUD en MongoDB

Las operaciones CRUD son esenciales para la interacción con cualquier base de datos. En MongoDB, estas operaciones se manejan de manera un poco diferente debido a su naturaleza basada en documentos.

- **Crear (Create)**: En MongoDB, la creación de datos se realiza mediante la inserción de uno o más documentos en una colección. Se utiliza el comando **insertOne()** para un documento o **insertMany()** para múltiples documentos.
- **Leer (Read)**: Para leer documentos, MongoDB ofrece métodos como **find()** y **findOne()**. Estos métodos permiten buscar documentos dentro de una colección que cumplan con ciertos criterios de búsqueda especificados.
- **Actualizar (Update)**: Para actualizar documentos, se utilizan métodos como **updateOne()**, **updateMany()**, o **replaceOne()**. Estos comandos permiten modificar documentos existentes en la base de datos basándose en criterios específicos.
- **Eliminar (Delete)**: Para eliminar documentos, MongoDB utiliza los métodos **deleteOne()** o **deleteMany()**, que eliminan documentos que cumplen con los criterios especificados.

Uso de la Shell de MongoDB para operaciones CRUD

Interacción Directa con la Base de Datos

La shell de MongoDB es una interfaz de línea de comandos que permite a los usuarios interactuar directamente con la base de datos y realizar operaciones CRUD.

- **Ejemplos de Comandos**:
  - **Crear**: db.collection.insertOne({name: "John", age: 30})
  - **Leer**: db.collection.find({age: {$gt: 20}})
  - **Actualizar**: db.collection.updateOne({name: "John"}, {$set: {age: 31}})
  - **Eliminar**: db.collection.deleteOne({name: "John"})

Estos comandos muestran cómo se pueden realizar operaciones básicas en la shell de MongoDB, proporcionando una herramienta poderosa para desarrolladores y administradores de bases de datos para manipular y gestionar datos de manera eficiente.

Integración de MongoDB con Lenguajes de Programación

Conectividad con Aplicaciones Modernas

MongoDB se puede integrar fácilmente con varios lenguajes de programación modernos, lo que permite desarrollar aplicaciones robustas que aprovechan las capacidades de MongoDB.

- **Node.js**: Utilizando el driver oficial de MongoDB para Node.js, los desarrolladores pueden conectar sus aplicaciones Node.js con una base de datos MongoDB utilizando promesas o callbacks para manejar las operaciones CRUD.
- **Python**: Usando PyMongo, el driver de MongoDB para Python, los desarrolladores pueden realizar operaciones CRUD, manejar la autenticación y configurar la conexión a la base de datos de forma segura y eficiente.

Ejemplo de Integración con Node.js

```
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const dbName = 'myproject';
const client = new MongoClient(url, { useUnifiedTopology: true });

async function run() {
    try {
        await client.connect();
        console.log("Connected correctly to server");
        const db = client.db(dbName);
        const col = db.collection('documents');

        // Insert a single document
        const p = await col.insertOne({a:1});
        // Find one document
        const myDoc = await col.findOne();
        // Print to the console
        console.log(myDoc);
    } catch (err) {
        console.log(err.stack);
    }
    finally {
     await client.close();
    }
}
run().catch(console.dir);
```

Consultas Avanzadas y Índices (MongoDB)

Requerimientos del ABP

1. ¿Cómo se realizan consultas avanzadas en MongoDB?
2. ¿Qué son los índices en MongoDB y cómo se utilizan para optimizar las consultas?

Realización de Consultas Complejas

Consultas Avanzadas en MongoDB

MongoDB ofrece una variedad de operadores y funciones que permiten la realización de consultas complejas, facilitando el manejo y análisis de grandes volúmenes de datos.

- **Consultas con Filtros**: Utilizando operadores como **$eq**, **$ne**, **$gt**, **$lt**, **$in**, y muchos otros, es posible filtrar documentos basados en condiciones específicas.
- **Consultas de Agregación**: MongoDB proporciona un poderoso framework de agregación que permite realizar operaciones complejas de procesamiento de datos. Esto incluye tuberías de agregación (**$match**, **$group**, **$sort**, **$limit**), que procesan datos y devuelven resultados calculados.
- **Consultas Geoespaciales**: Soporta consultas basadas en la localización geográfica, útiles para aplicaciones que manejan datos de ubicación.

Ejemplo de una Consulta de Agregación:

db.users.aggregate([{ $match: { status: "active" } }, { $group: { _id: "$age", total: { $sum: 1 } } }, { $sort: { total: -1 } }]);

Este ejemplo muestra cómo agrupar documentos por edad, contar cuántos usuarios activos hay en cada grupo de edad y luego ordenar los resultados en orden descendente.

Optimización de Consultas con Índices

Índices en MongoDB

Los índices son estructuras de datos especiales que almacenan una pequeña porción de los datos de la colección de forma fácilmente accesible. Los índices son particularmente útiles para mejorar la velocidad de las operaciones de búsqueda.

- **Creación de Índices**: MongoDB permite la creación de índices en cualquier campo de un documento, incluyendo índices en campos dentro de documentos incrustados y en arrays.
- **Tipos de Índices**: Incluyen índices simples, compuestos, de texto completo, geoespaciales y otros tipos que soportan consultas específicas.
- **Manejo de Índices**: La administración adecuada de índices incluye entender cuándo crearlos y cómo afectan al rendimiento tanto en consultas como en operaciones de inserción y actualización.

Ejemplo de Creación de un Índice:

db.users.createIndex({ name: 1 });

Este comando crea un índice ascendente en el campo **name** de la colección **users**, lo cual acelerará las consultas que buscan documentos basados en el nombre.

Ejemplos de Consultas en MongoDB

Consultas Dinámicas y Flexibles

Aquí algunos ejemplos que ilustran cómo se pueden realizar diferentes tipos de consultas en MongoDB:

- **Consulta Básica**:
  db.users.find({ age: { $gt: 20 } });

  Esta consulta encuentra todos los usuarios que tienen más de 20 años.

- **Consulta de Texto Completo**:
  db.users.find({ $text: { $search: "engineer" } });

  Busca usuarios cuyos documentos contienen la palabra "engineer".

- **Consulta Geoespacial**:
  db.stores.find({ location: { $near: { $geometry: { type: "Point", coordinates: [-122.34, 47.61] }, $maxDistance: 5000 } } });

  Esta consulta busca tiendas dentro de 5000 metros de un punto dado.

Integración y Automatización con MongoDB

Requerimientos del ABP

1. ¿Cómo se automatiza la carga de datos en MongoDB?
2. ¿Cómo se puede integrar MongoDB con APIs RESTful?

Automatización de la Carga de Datos en MongoDB

Procesos Eficientes para la Gestión de Datos

Automatizar la carga de datos en MongoDB es crucial para sistemas que manejan grandes volúmenes de información, permitiendo una integración fluida y eficiente de datos en tiempo real o a través de cargas por lotes.

- **Herramientas ETL (Extract, Transform, Load)**: Existen varias herramientas y plataformas que pueden conectar con MongoDB para automatizar el proceso de extracción, transformación y carga de datos. Herramientas como Apache NiFi, Talend, y Informatica PowerCenter están equipadas para trabajar con MongoDB, facilitando la integración de datos desde múltiples fuentes.
- **Scripts de Automatización**: Se pueden desarrollar scripts personalizados en lenguajes como Python o JavaScript para automatizar la importación de datos. Estos scripts pueden ejecutarse a intervalos regulares o en respuesta a eventos específicos.

APIs RESTful con MongoDB

Integración de Aplicaciones Modernas

La creación de APIs RESTful que interactúan con MongoDB es una práctica común para aplicaciones web y móviles, facilitando el acceso y la manipulación de datos a través de HTTP.

Uso de Python en el Desarrollo de APIs RESTful

Frameworks Populares

1. **Flask**: Flask es un micro-framework para Python que es especialmente popular por su simplicidad y flexibilidad. No requiere herramientas o bibliotecas particulares y es muy ligero, lo que lo hace ideal para proyectos más pequeños o para desarrolladores que prefieren un mayor control sobre los componentes que utilizan.
    - **Ventajas**:
        - Simple y fácil de aprender.
        - Extensible con numerosas extensiones disponibles para agregar funcionalidades como la manipulación de bases de datos, validación de formularios, manejo de autenticaciones, y más.
        - Adaptable a las necesidades del proyecto, permitiendo tanto aplicaciones simples como complejas.
2. **Django**: Django es un framework de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático. Viene con muchas funcionalidades "listas para usar", como un ORM (Object-Relational Mapping) que facilita la interacción con bases de datos.
    - **Ventajas**:
        - Incluye un servidor de desarrollo y un sistema de administración por defecto, lo que acelera el desarrollo inicial.
        - Altamente escalable y versátil.
        - Excelente documentación y gran comunidad de soporte.
3. **FastAPI**: Un framework moderno y rápido (de alto rendimiento) para construir APIs con Python 3.7+ basado en estándares Python type hints. FastAPI es conocido por su velocidad y es uno de los frameworks más rápidos disponibles para Python.
    - **Ventajas**:
        - Rápido para codificar: aumenta la velocidad para desarrollar características gracias a su sistema de autocompletado en todos los editores de código.

- Menos errores: reduce alrededor del 40% de los errores humanos debido al sistema de tipos.
- Automáticamente interactivo: genera documentación interactiva y explorable de la API.

Integración de Python con MongoDB

Para integrar Python con MongoDB, PyMongo es el driver recomendado y oficial para interactuar con una base de datos MongoDB desde Python. Proporciona herramientas y funcionalidades necesarias para realizar todas las operaciones CRUD y más. Además, PyMongo es compatible con Flask, Django y FastAPI, lo que facilita la integración en cualquier tipo de aplicación Python.

Uso de Node.js y Express para el Desarrollo de APIs RESTful

Características y Beneficios

1. **Node.js**: Es un entorno de ejecución para JavaScript construido sobre el motor V8 de Chrome. Permite ejecutar código JavaScript en el servidor, lo que lo convierte en una solución ideal para desarrollar aplicaciones web y APIs en el mismo lenguaje utilizado en el front-end.
    - **Ventajas**:
        - **Asíncrono y Basado en Eventos**: Todas las API de Node.js son no bloqueantes (asíncronas). Esto significa que el servidor basado en Node.js nunca espera a que una API devuelva datos, lo que lo hace extremadamente eficiente para I/O.
        - **Unificado**: Utiliza JavaScript tanto en el cliente como en el servidor, lo que simplifica el desarrollo y reduce el esfuerzo para mantener coherencia entre ambos.
        - **Comunidad Extensa**: Tiene una de las mayores bibliotecas de módulos de código abierto que ayudan a acelerar el desarrollo de aplicaciones.

2. **Express**: Es un framework para Node.js diseñado para facilitar la construcción de aplicaciones web y APIs. Es minimalista, flexible y cuenta con un robusto conjunto de características para aplicaciones web y móviles.
    - **Ventajas**:
        - **Rápido**: Framework minimalista que ayuda a organizar su aplicación web del lado del servidor en MVC (Model View Controller) y otras configuraciones populares.
        - **Middleware**: Permite a los desarrolladores escribir handlers de peticiones que se pueden ejecutar en cualquier punto del ciclo de vida de una petición HTTP.
        - **Ruteo**: Gestiona las rutas de las aplicaciones de una manera muy eficiente.

Integración de Node.js y Express con MongoDB

La integración de Node.js y Express con MongoDB se realiza comúnmente a través de Mongoose, que es una biblioteca que proporciona mapeo de objetos MongoDB (ODM) para Node.js. Mongoose facilita la construcción de modelos y la realización de consultas.

Escalabilidad y Disponibilidad en MongoDB

Requerimientos del ABP

¿Cómo maneja MongoDB la escalabilidad y la alta disponibilidad mediante sharding y replicación?

Sharding y Replicación en MongoDB

Mecanismos Claves

- **Sharding**: Sharding en MongoDB es el proceso de almacenar datos en múltiples máquinas para soportar despliegues de bases de datos de muy gran tamaño. MongoDB usa el sharding para distribuir los datos a través de varios servidores, haciendo que el conjunto de datos y la carga de trabajo sean distribuidos y gestionados de manera horizontal. Esto es crucial para aplicaciones que no pueden servirse desde un solo servidor debido a problemas de tamaño de la base de datos o demandas de tráfico.
    - **Tipos de Shards**: MongoDB permite varios tipos de sharding, típicamente basados en un campo clave de sharding que determina cómo se distribuyen los datos.
    - **Balanceador**: MongoDB automáticamente equilibra los datos a través de los shards para asegurar que todos los shards tengan aproximadamente la misma cantidad de datos y carga de trabajo.
- **Replicación**: MongoDB utiliza la replicación para aumentar la disponibilidad de datos y la resistencia del sistema. A través de los conjuntos de réplicas, MongoDB permite que múltiples servidores mantengan copias idénticas del conjunto de datos. Esto no solo ayuda en la tolerancia a fallos, sino también en la alta disponibilidad, asegurando que el sistema pueda seguir funcionando incluso si algunos nodos fallan.
    - **Conjuntos de Réplicas**: Un conjunto de réplicas en MongoDB consiste en dos o más copias del mismo conjunto de datos.

- **Rol de Primario y Secundarios**: Dentro de un conjunto de réplicas, un nodo es designado como primario y maneja todas las escrituras y lecturas; los nodos secundarios mantienen copias del conjunto de datos del primario y pueden servir lecturas si están configurados para hacerlo.

Estrategias para Manejar Grandes Volúmenes de Datos

Optimización del Almacenamiento y la Recuperación

- **Indexación Adecuada**: Asegurar que todos los datos relevantes estén indexados de manera eficiente para optimizar las consultas y reducir la carga en el sistema.
- **Optimización de Consultas**: Continuamente monitorizar y optimizar las consultas para garantizar que se ejecuten con la máxima eficiencia, especialmente en sistemas con grandes volúmenes de datos.

Consistencia y la Partición de Datos

Balance Entre Rendimiento y Consistencia

- **Consistencia Eventual**: En un modelo de consistencia eventual, como el utilizado por los conjuntos de réplicas de MongoDB, se garantiza que todas las escrituras se replicarán eventualmente, pero los clientes pueden ver datos desactualizados si leen de un nodo secundario antes de que se haya propagado la replicación.
- **Estrategias de Partición**: La elección del campo de sharding es crucial porque una mala elección puede llevar a un "sharding poco balanceado", donde algunos shards manejan mucha más carga que otros.

Implementar sharding y replicación efectivamente en MongoDB requiere una planificación cuidadosa y una comprensión profunda de los patrones de acceso a los datos y las cargas de trabajo de la aplicación. Con las configuraciones adecuadas, MongoDB puede escalar horizontalmente para soportar aplicaciones extremadamente grandes y ofrecer alta disponibilidad y resistencia contra fallos.

Seguridad y Mantenimiento en MongoDB

Requerimientos del ABP

1. ¿Qué medidas de seguridad se deben considerar en MongoDB?
2. ¿Cómo se realiza el mantenimiento y la monitorización en MongoDB?

Configuración de Seguridad en MongoDB

Medidas Clave para Proteger la Base de Datos

- **Autenticación y Autorización**: MongoDB soporta una amplia gama de mecanismos de autenticación, incluyendo SCRAM, x.509, LDAP, y Kerberos. Es vital configurar la autenticación para asegurar que solo los usuarios autorizados puedan acceder a la base de datos. La autorización a través de roles específicos controla el nivel de acceso que tiene cada usuario, restringiendo las operaciones que pueden realizar.
- **Control de Acceso a la Red**: Configurar firewalls y listas de control de acceso a la red (ACL) para limitar quién puede conectarse a la base de datos. Esto incluye permitir conexiones solo desde aplicaciones y servidores confiables.
- **Cifrado**: Utilizar el cifrado en reposo y en tránsito para proteger los datos sensibles. MongoDB ofrece cifrado en tránsito mediante el uso de TLS/SSL y cifrado en reposo utilizando el cifrado de datos transparente.
- **Auditoría**: Configurar la auditoría para rastrear y registrar actividades críticas que afecten a los datos o a la configuración de MongoDB. Esto es crucial para cumplir con las regulaciones de cumplimiento y para detectar actividades sospechosas o malintencionadas.

Backup y Recuperación de Datos

Estrategias para Protección de Datos

- **Backups Regulares**: Implementar una estrategia de backups regulares para proteger los datos contra pérdidas accidentales o maliciosas. MongoDB ofrece

herramientas como **mongodump** para crear backups de los datos y **mongorestore** para restaurarlos.

- **Punto de Recuperación**: Definir y mantener un punto de recuperación objetivo (RPO) adecuado y pruebas regulares de recuperación para asegurar que los backups funcionen correctamente en un escenario de desastre.
- **Almacenamiento de Backups**: Almacenar los backups en una ubicación segura, preferiblemente off-site o en un servicio de almacenamiento en la nube que ofrezca alta disponibilidad y durabilidad.

Monitorización y Ajustes de Rendimiento

Mantenimiento Proactivo para Optimización

- **Herramientas de Monitorización**: Utilizar herramientas integradas como MongoDB Atlas o soluciones de terceros como New Relic o Prometheus para monitorizar constantemente el estado y el rendimiento de la base de datos. Estas herramientas pueden alertar a los administradores sobre condiciones anormales como uso elevado de CPU, memoria o espacio en disco.
- **Análisis de Rendimiento**: Analizar regularmente el rendimiento de las consultas y los índices. Utilizar el **explain** de MongoDB para entender cómo se ejecutan las consultas y hacer ajustes en los índices o en las propias consultas para mejorar el rendimiento.
- **Actualizaciones y Parches**: Mantener la base de datos actualizada con las últimas versiones y parches de seguridad. Esto no solo mejora la seguridad sino que también asegura que se aprovechen las últimas mejoras y características de rendimiento.

Implementar estas prácticas de seguridad y mantenimiento en MongoDB no solo ayuda a proteger los datos contra amenazas externas e internas, sino que también garantiza que la base de datos funcione de manera eficiente y confiable, soportando las operaciones críticas del negocio y proporcionando un servicio ininterrumpido a los usuarios.

Referencias Bibliográficas

1. **Título**: "Bases de Datos: Desde Chen hasta Codd con ORACLE" **Autor**: Silberschatz, Abraham; Korth, Henry F.; Sudarshan, S. **Edición**: 2da Edición (Versión en español)
2. **Título**: "Fundamentos de Bases de Datos" **Autor**: Silberschatz, Abraham; Korth, Henry F.; Sudarshan, S. **Edición**: 4ta Edición en Español
3. **Título**: "MongoDB: Diseño y Programación" **Autor**: Alberto Casero Ávila **Edición**: 1ra Edición
4. **Título**: "MySQL y mSQL" **Autor**: Randy Jay Yarger, George Reese, Tim King **Edición**: Edición en español
5. **Título**: "Bases de datos NoSQL y Big Data" **Autor**: Rafael Camps Paré, Ricard Mayer **Edición**: 1ra Edición