Ciencia de Datos

Ficha:

2795599

Aprendices:

Julián Estiven Posso

Juan Pablo Ruiz Marin

Gerónimo Trujillo Bustamante

2025

1. **¿Qué es una base de datos?**
   es un sistema organizado para almacenar, gestionar y recuperar información de manera eficiente. Está formada por una colección de datos relacionados entre sí, que se almacenan de forma estructurada en tablas o colecciones. Estos datos pueden ser accesados, actualizados, gestionados y manipulados de diversas maneras a través de un sistema de gestión de bases de datos (DBMS, por sus siglas en inglés). Las bases de datos pueden contener diferentes tipos de información, como texto, números, imágenes, o incluso datos complejos, y se utilizan en muchas aplicaciones, desde sitios web hasta sistemas empresariales, para manejar grandes volúmenes de datos de forma rápida y segura.
   El uso de bases de datos es fundamental para mantener la integridad, accesibilidad y seguridad de los datos

2. **Tipos de Bases de Datos**
   - **Relacionales (RDBMS):**
     - MySQL
     - PostgreSQL
     - Oracle
     - Microsoft SQL Server
   - **No Relacionales (NOSQL):**
     - Neo4,
     - ArangoDB
     - MongoDB
   - **Bases de Datos Jerárquicas:**
     - IBM Information Management System (IMS)
     - XML-based databases
   - **Bases de Datos de Red:**
     - Integrated Data Store (IDS)
     - IDMS (Integrated Database Management System)
   - **Bases de Datos Orientadas a Objetos (OODBMS):**
     - db4o
     - ObjectDB
     - Versant Object Database
   - **Bases de Datos en la Nube:**
     - Amazon RDS
     - Google Cloud Firestore
     - Azure Cosmos DB
   - **Bases de Datos Distribuidas:**
     - Apache Cassandra
     - Google Bigtable

    - Amazon DynamoDB
- **Bases de Datos Temporales:**
    - SAP HANA (también incluye características temporales)
    - Temporal Data Model en PostgreSQL
- **Bases de Datos Multimodales:**
    - ArangoDB
    - OrientDB
- **Bases de Datos en Memoria:**
    - Redis
    - Memcached

3. **¿Qué es Ciencia de Datos?**

   es un campo interdisciplinario que utiliza métodos, algoritmos y sistemas para extraer conocimiento y obtener insights de datos, que pueden ser estructurados o no estructurados. Combina varias disciplinas, como la estadística, la informática, la ingeniería y el análisis de datos, para interpretar grandes volúmenes de información y ayudar a tomar decisiones basadas en datos

   Aspectos claves:
    - Recopilación y Preparación de Datos
    - Análisis Exploratorio de Datos (EDA)
    - Modelado Predictivo
    - Validación y Evaluación de Modelos
    - Visualización de Datos
    - Toma de Decisiones Basada en Datos
    - Big Data y Computación en la Nube
4. **¿Qué es Python? Y su Historia**

   es un lenguaje de programación de alto nivel, interpretado y de propósito general. Es conocido por su sintaxis simple y legible, lo que lo hace accesible tanto para principiantes como para programadores experimentados

   Python es muy versátil y se utiliza en una gran variedad de aplicaciones:
    - Desarrollo web
    - Ciencia de datos y análisis
    - Automatización
    - Inteligencia artificial y aprendizaje automático
    - Desarrollo de software
    - Desarrollo de aplicaciones científicas

   Historia:

   Python comienza en la década de 1980, cuando su creador, Guido van Rossum, comenzó a desarrollar el lenguaje como una alternativa simple y poderosa a otros lenguajes de programación de la época. Aquí te doy un resumen de la historia de Python a lo largo de los años:

Orígenes (1980s):

1980: Guido van Rossum, un programador de los Países Bajos trabajaba en un proyecto llamado ABC, un lenguaje de programación diseñado para ser fácil de aprender y usar, que se empleaba en la enseñanza de informática. Aunque ABC nunca se hizo popular, inspiró mucho el diseño de Python.

1989: Durante las vacaciones de Navidad, Guido van Rossum comenzó a trabajar en un nuevo lenguaje de programación que sería más accesible, legible y adecuado para resolver tareas cotidianas. Lo nombró Python como un homenaje al programa de comedia británico "Monty Python's Flying Circus", un show que le gustaba mucho. Esto no tiene relación con las serpientes, aunque el logo de Python a menudo se asocia con ellas.

Primeras versiones (1991 - 2000):

1991: Van Rossum publicó la primera versión de Python (versión 0.9.0). Esta versión ya incluía características esenciales, como las excepciones, funciones y la sintaxis basada en espacios en blanco, lo que la hacía más legible que otros lenguajes populares de la época.

1994: Se lanza la versión 1.0 de Python. A medida que el lenguaje crecía, fue ganando popularidad entre los programadores que apreciaban su simplicidad, flexibilidad y elegancia. Python comenzó a ser utilizado en proyectos pequeños y medianos, como herramientas de automatización, desarrollo de aplicaciones simples y scripting.

Expansión y mejoras (2000 - 2010):

2000: Python pasó por una reestructuración importante con la versión 2.0. Durante este tiempo, se introdujeron muchas características nuevas, como la recolección de basura, la gestión de memoria y las nuevas funciones de cadena.

2008: Se lanzó Python 3.0, una revisión importante que no fue completamente compatible con versiones anteriores de Python (es decir, código escrito para Python 2.x no necesariamente funcionaba en Python 3). Esta versión buscaba corregir limitaciones y mejorar el diseño del lenguaje, pero también causó algunos desafíos, ya que muchos desarrolladores se vieron obligados a actualizar su código para adaptarse a los nuevos cambios.

La transición a Python 3 y el auge (2010 - 2020):

A medida que Python 3 ganaba terreno, muchos desarrolladores y proyectos comenzaron a adoptar la nueva versión. Python pasó a ser el lenguaje más popular para ciencia de datos y aprendizaje automático, en gran parte gracias a bibliotecas como NumPy, Pandas, TensorFlow, Keras, Matplotlib, y otras,

que facilitaron el análisis de grandes volúmenes de datos y la implementación de modelos de machine learning.

2010: Python se consolidó como uno de los lenguajes más populares y comenzó a ser ampliamente adoptado por grandes empresas y desarrolladores independientes. El ecosistema de bibliotecas creció rápidamente, especialmente en áreas como el análisis de datos, la inteligencia artificial y el desarrollo web (gracias a frameworks como Django y Flask).

Python hoy (2020 - presente):

A partir de 2020, Python se ha consolidado como uno de los lenguajes de programación más populares y utilizados a nivel mundial. Su comunidad es enorme y activa, y el lenguaje sigue evolucionando con mejoras continuas en rendimiento y características.

El Python Software Foundation (PSF), una organización sin fines de lucro, es responsable de gestionar el desarrollo del lenguaje, y su trabajo ha sido clave en su expansión a nivel global.

Python sigue siendo el lenguaje predilecto en áreas como:

- Ciencia de datos y análisis (gracias a bibliotecas como Pandas, NumPy, SciPy, etc.).
- Inteligencia artificial y aprendizaje automático (gracias a TensorFlow, Keras, PyTorch).
- Automatización y scripting.
- Desarrollo web (con frameworks como Django, Flask).
- Desarrollo de software en general (incluyendo aplicaciones de escritorio y juegos).

La comunidad sigue creciendo, con muchos recursos educativos, conferencias y eventos dedicados a Python (por ejemplo, PyCon), lo que contribuye a que cada vez más personas se adentren en el mundo de la programación con Python.

Python 2 y el fin de su soporte (2020):

En 2020, el soporte oficial para Python 2 llegó a su fin. La versión 2.x ya no recibe actualizaciones, por lo que los desarrolladores fueron animados a migrar sus proyectos a Python 3, lo que resultó en una transición importante para la comunidad.

5. **Diferencias Base de datos Relaciones y NOSQL**
    - **Modelo de Datos**

SQL:

- Utiliza un modelo relacional, donde los datos se organizan en tablas con filas y columnas.
- Cada fila representa un registro y cada columna un atributo de ese registro.
- Los datos deben seguir una estructura predefinida con esquemas rígidos, lo que significa que los tipos de datos y las relaciones entre las tablas deben ser definidos antes de almacenar los datos.

NOSQL:

Utiliza varios modelos de datos, no necesariamente relacionales, como:
Clave-valor (por ejemplo, Redis).
Documental (por ejemplo, MongoDB).
Columnares (por ejemplo, Cassandra).
Grafos (por ejemplo, Neo4j).

- **Escalabilidad**

  SQL:

  - Las bases de datos SQL son más escalables verticalmente, es decir, para mejorar el rendimiento, generalmente se mejora el hardware del servidor (más memoria, procesadores más rápidos, etc.).
  - Escalar horizontalmente (distribuir datos entre múltiples servidores) suele ser más difícil debido a la estructura rígida y las relaciones entre tablas.

  NOSQL:

  - Las bases de datos NoSQL están diseñadas para ser escalables horizontalmente, es decir, puedes agregar más servidores para manejar grandes volúmenes de datos y tráfico.
  - Son ideales para grandes volúmenes de datos y aplicaciones distribuidas, ya que permiten repartir la carga de trabajo entre diferentes máquinas sin perder rendimiento.

- **Flexibilidad del Esquema**

  SQL:

  - Esquema fijo: Se requiere definir previamente las tablas, las columnas y los tipos de datos. Si deseas cambiar el esquema (por ejemplo, agregar una nueva columna), debes hacerlo explícitamente mediante migraciones o alteraciones de la tabla.
  - Esto puede ser restrictivo cuando los requisitos de datos cambian con el tiempo.

  NOSQL:

  - Esquema flexible: No es necesario definir un esquema estricto antes de almacenar los datos. Los datos pueden variar de un

documento a otro o de un registro a otro, lo que hace que NoSQL sea adecuado para datos que cambian con frecuencia o que no se ajustan bien a una estructura rígida.
    - Es ideal para aplicaciones que necesitan adaptarse rápidamente a nuevos requisitos.
- **Consultas y Lenguaje de Consulta**
  SQL:
    - Utiliza SQL (Structured Query Language), un lenguaje estandarizado para realizar consultas. Las consultas son estructuradas y se utilizan para hacer operaciones complejas, como join (unión de tablas), filtros, agregaciones, etc.
    - Las consultas SQL son potentes y permiten manejar relaciones complejas entre datos.

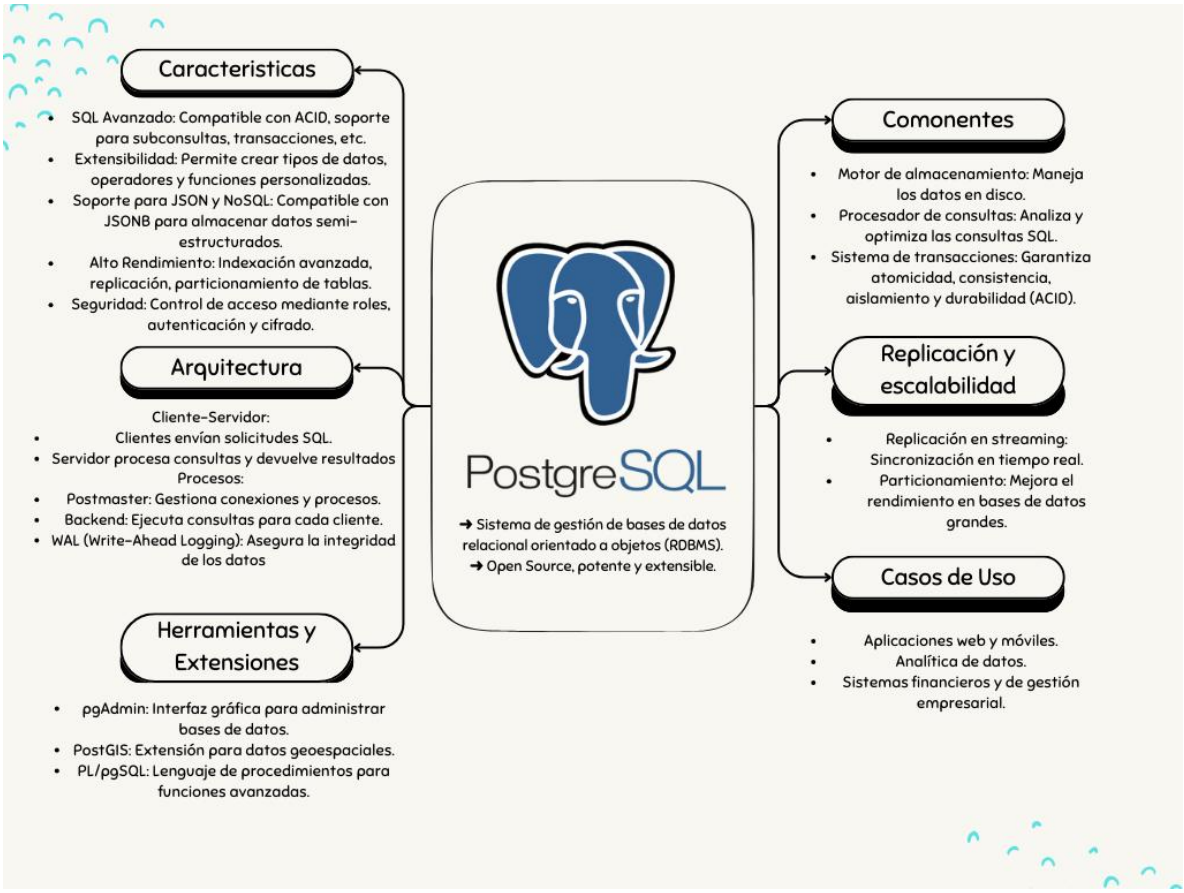
  NOSQL:
    - No existe un lenguaje de consulta único o estandarizado para NoSQL, ya que depende del tipo de base de datos. Cada tipo de base de datos NoSQL tiene su propia forma de hacer consultas.
    - Por ejemplo, en MongoDB (documental) se usa una sintaxis basada en JSON, y en Cassandra (columnares), se usa un lenguaje específico llamado CQL (Cassandra Query Language).
    - Aunque algunas bases de datos NoSQL permiten consultas complejas, en general, no están tan optimizadas para consultas relacionales complejas como SQL.
- **Transacciones y Consistencia**
  SQL:
    - Ofrecen una fuerte consistencia y cumplen con las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), lo que garantiza que las transacciones sean seguras y que los datos sean consistentes incluso en caso de fallos.
    - Ideal para aplicaciones donde la integridad de los datos y la consistencia son críticas (por ejemplo, sistemas bancarios).

  NOSQL:
    - No todas las bases de datos NoSQL siguen el modelo ACID. Muchos adoptan el principio BASE (Basically Available, Soft state, Eventual consistency), lo que significa que la consistencia se logra de manera eventual y no siempre inmediata.
    - Esto permite una mayor disponibilidad y rendimiento en aplicaciones distribuidas, pero con la posibilidad de que los datos no estén completamente sincronizados en todos los nodos de manera instantánea.

- **Rendimiento**
  SQL:
  - Las bases de datos SQL son muy eficientes cuando se manejan datos estructurados y consultas complejas que requieren relaciones entre diferentes tablas.
  - Sin embargo, el rendimiento puede degradarse cuando se trata de grandes volúmenes de datos no estructurados o cuando es necesario escalar horizontalmente.

  NOSQL:
  - Las bases de datos NoSQL son más rápidas para operaciones de lectura/escritura en grandes volúmenes de datos no estructurados o semi-estructurados.
  - Son muy buenas para aplicaciones de alto rendimiento, donde la latencia y la rapidez de las consultas son cruciales, pero sin la necesidad de relaciones complejas.
- **Casos de Uso**
  SQL:
  Son ideales para aplicaciones donde los datos son estructurados y las relaciones entre ellos son claras y estables. Ejemplos incluyen sistemas bancarios, sistemas de gestión empresarial, aplicaciones de comercio electrónico, y otras aplicaciones que requieren transacciones complejas y consistentes
  NOSQL:
  Son más adecuadas para aplicaciones que manejan grandes volúmenes de datos no estructurados o semi-estructurados, como redes sociales, análisis de grandes datos, aplicaciones móviles, servicios web en tiempo real, o sistemas que requieren escalabilidad horizontal.

6. Mapa Conceptual



7. **Replit + GitHub:**

¿Qué son?

Replit:

es una plataforma en línea que permite a los usuarios escribir, ejecutar y colaborar en código de manera rápida y sencilla, directamente desde su navegador. Es un entorno de desarrollo integrado (IDE) basado en la web que admite muchos lenguajes de programación y proporciona una interfaz fácil de usar para escribir, ejecutar y compartir proyectos de programación

GitHub:

es una plataforma de desarrollo colaborativo que permite a los programadores y equipos de desarrollo almacenar, gestionar y compartir proyectos de software utilizando Git, un sistema de control de versiones distribuido. GitHub facilita el trabajo en equipo y la colaboración, permitiendo que varias personas trabajen en el mismo proyecto de manera eficiente, con herramientas para la gestión de versiones, la revisión de código y la integración continua

¿Cómo se integran?

1. **Crear una cuenta en Replit:**

Si no tienes una cuenta en Replit, crea una visitando Replit.com y registrándote.

2. **Vincular tu cuenta de GitHub en Replit:**

En Replit, ve a la configuración de tu cuenta. Para esto, haz clic en tu avatar en la esquina superior derecha y selecciona "Settings" (Configuraciones).
En la sección de "GitHub Integration", selecciona "Connect GitHub".
Se te pedirá que autorices a Replit para acceder a tu cuenta de GitHub. Después de autorizar, tu cuenta de GitHub estará vinculada a Replit.

3. **Crear un proyecto en Replit:**

Si ya tienes un proyecto en GitHub, puedes importarlo a Replit, o bien puedes crear un nuevo proyecto en Replit (un "Repl").
Si estás creando un proyecto nuevo, selecciona el lenguaje de programación que prefieras y haz clic en "Create" para comenzar a trabajar.

4. **Importar un repositorio de GitHub a Replit:**

Si ya tienes un repositorio en GitHub y quieres importarlo a Replit, sigue estos pasos:
En Replit, en la pantalla principal, haz clic en "Import from GitHub".
Replit te mostrará una lista de tus repositorios disponibles en GitHub. Selecciona el repositorio que deseas importar y haz clic en "Import".
Replit clonará el repositorio y abrirá el proyecto, permitiéndote trabajar directamente en el entorno de Replit.

5. **Trabajar en el proyecto en Replit:**

Ahora puedes hacer cambios en tu código dentro de Replit. Replit proporciona un editor de código en línea donde puedes escribir, ejecutar y depurar tu proyecto.

6. **Realizar cambios y hacer commits a GitHub desde Replit:**

Cuando hayas realizado cambios en tu proyecto en Replit, puedes guardar esos cambios en tu repositorio de GitHub. Para hacerlo, ve a la pestaña "Version control" (Control de versiones) en el panel lateral izquierdo. Aquí verás las opciones para hacer commits de tus cambios, agregar comentarios y sincronizar el repositorio con Git Hub.

Haz clic en "Commit" para guardar tus cambios en tu repositorio de GitHub.
Si deseas subir tus cambios al repositorio de GitHub, puedes hacer clic en "Push". Esto subirá tus cambios a GitHub, manteniendo tu repositorio actualizado.

7. **Trabajar con ramas:**

Si trabajas con ramas en GitHub, puedes gestionar esas ramas directamente desde Replit. Puedes crear nuevas ramas, hacer cambios en ellas y luego fusionarlas con la rama principal (usualmente main o master).
Asegúrate de usar el sistema de control de versiones de Replit para gestionar las ramas y realizar los merge cuando sea necesario.

8. Sincronización automática:

Si haces un commit en GitHub desde otro lugar (por ejemplo, desde una computadora local o un cliente de Git), puedes sincronizar esos cambios con Replit, simplemente haciendo clic en el botón de "Pull" en la interfaz de Replit. Esto descargará los últimos cambios desde GitHub a tu proyecto de Replit.