DISEÑO, CREACIÓN Y GESTIÓN DE BASES DE DATOS RELACIONALES: UN ENFOQUE PRÁCTICO CON SQL Y PYTHON

MARÍA CATALINA BALLESTEROS VELASCO
ADRIANA LUCÍA GUERRERO MERCHÁN
ÉMERZON STEVEN MENDOZA QUINTERO
ANGÉLICA LILIANA PORRAS BARÓN

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2025

DISEÑO, CREACIÓN Y GESTIÓN DE BASES DE DATOS RELACIONALES: UN ENFOQUE PRÁCTICO CON SQL Y PYTHON

MARÍA CATALINA BALLESTEROS VELASCO
ADRIANA LUCÍA GUERRERO MERCHÁN
ÉMERZON STEVEN MENDOZA QUINTERO
ANGÉLICA LILIANA PORRAS BARÓN

Trabajo de grado para optar al título de
Matemático

Director
Andrés Sebastián Ríos Gutiérrez
Candidato a Doctor

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE CIENCIAS
ESCUELA DE MATEMÁTICAS
BUCARAMANGA
2025

DEDICATORIA

Dedico este trabajo con profundo cariño a mi tía Leo y a mis padres, quienes me han dado un hogar lleno de amor, apoyo y valores. Todo lo que he logrado y lo que aún me propongo alcanzar es por ellos y para ellos.

María Catalina Ballesteros Velasco

Dedico este trabajo a mis padres y hermano, cuyo esfuerzo y apoyo incondicional han sido fundamentales a lo largo de mi carrera.

Adriana Lucía Guerrero Merchán

Este trabajo va dedicado a mi familia, quienes me ofrecieron el amor y todo lo necesario para culminar mi pregrado. También va dedicado a mi primo Felipe, que se alegra con cualquier cosa buena que me sucede.

Émerzon Steven Mendoza Quintero

Este trabajo es dedicado a mi mamá y a mi hermana mayor, que siempre me apoyaron desde que tengo memoria.

Angélica Liliana Porras Barón

AGRADECIMIENTOS

Agradezco profundamente a las personas que han sido el pilar fundamental en mi vida y en este camino académico.

A mi mamá Alicia, mi papá Félix y mi tía Leo, mis tres pilares, mi motor y mi inspiración. Su amor, apoyo incondicional y confianza me han sostenido en cada paso. Gracias por ser mi refugio seguro, por motivarme a alcanzar mis metas y por recordarme con su ejemplo lo que significa estar protegida y profundamente amada.

A mis hermanos Juan Félix, Silvia, Saray, Juan Diego y a mi primo Santiago, quienes no solo han sido mis compañeros de vida, sino también mis mejores amigos. Ser su hermana mayor es un honor inmenso, y saber que me miran con admiración me impulsa a ser mejor cada día. Gracias por creer siempre en mí y por hacerme sentir capaz de lograr lo que me proponga.

A mi papá Juan Félix y a Nelly, gracias por su guía, sus consejos sinceros y por brindarme amor, compañía y todas las herramientas necesarias para volar alto.

A mi mamá Rocío, gracias por tu ejemplo de fortaleza. En ti he aprendido lo que significa ser una mujer fuerte, valiente y decidida.

A mis tíos Mauricio, Ofelia y Marina, quienes han sido como padres para mí. Gracias por su amor incondicional y su apoyo constante. Me llena de alegría y gratitud tenerlos en mi vida.

A mis tutores, quienes me han acompañado desde el primer semestre, gracias por enseñarme a no rendirme, por su orientación académica y por su apoyo humano en cada etapa de esta formación.

A mi profesor Jorge Villamizar, gracias por ser guía, maestro y luz en mi camino. Su apoyo ha sido clave en mi desarrollo profesional y personal. A mi director Andrés Ríos, gracias por su acompañamiento y sus valiosos consejos a lo largo de este proceso.

A los amigos que me regaló la universidad, en especial a Émerzon y Laura. Gracias por hacer que mis últimos semestres fueran más llevaderos, por su amistad sincera, por enseñarme el valor del trabajo en equipo, por compartir tiempo, risas, aprendizajes y por todo el amor que me han brindado. Ustedes han sido un regalo de vida.

A todos, gracias por creer en mí.

María Catalina Ballesteros Velasco

Agradezco primeramente a Dios por ser mi guía constante, por iluminar el camino en los momentos oscuros y enviarme las señales justas en los momentos precisos.

A mis padres, quienes, aún en la distancia, han estado siempre presentes con su amor, sus sabios consejos y su incansable esfuerzo. Sé que están orgullosos de mí, y su apoyo ha sido fundamental para hacer este camino más llevadero. A mi hermanito, Yoelito, cuya alegría y cariño, a través de sus mensajes y llamadas, han sido un motor para seguir adelante y nunca rendirme.

A Iván por su paciencia, su amor incondicional y por sostenerme en los momentos que quise renunciar.

A mis amigos y compañeros por su ánimo constante, por su compañía y sus palabras de aliento. Porque muchas veces fueron la luz en medio de la oscuridad, en especial a Anita, Cata y Pao por su apoyo emocional y su inquebrantable amistad.

A Juliana, mi terapeuta, por brindarme las herramientas necesarias para afrontar este proceso, por su orientación y sus sabios consejos, que me han ayudado a sanar y crecer.

Y, por último, pero no menos importante, al profesor Andrés, por sus valiosos consejos, y por su dedicación inquebrantable a sus estudiantes. Su compromiso y pasión por la enseñanza lo han convertido en uno de mis principales referentes laborales.

En general, agradezco a cada persona que, con su granito de arena, ha contribuido en mi formación tanto académica como personal, ayudándome a encontrar el camino que debo seguir.

Adriana Lucía Guerrero Merchán

Sé que las palabras y las páginas de este documento no me alcanzarán para mostrar cuán agradecido estoy por este gran logro, pero haré lo posible por reducir mi sincero escrito.

Agradezco a Dios por bendecirme cada día y por darme la oportunidad de vivir esta vida tan hermosa. También estoy agradecido con el universo, que me permitió aprender y entender un poco de él.

A Pepita, que gracias a su amor de madre estoy donde estoy; a Mechito, que ha sido mi inspiración desde que tengo memoria; a mi hermanito, que siempre ha estado para mí, y a todos mis familiares, que se sentían orgullosos de que yo estuviera en la universidad. Y por supuesto, a mi primo Felipe, con el que quiero celebrar este logro comiéndonos una hamburguesa.

A María, por ser la mejor compañera y amiga que tuve. Gracias a su compañerismo y amor, mis días en la universidad fueron los mejores.

A mis amistades, que fueron tan importantes en esta carrera como un libro, un lapicero y un papel. Especialmente a Jimmy, Sergio, Luis, Laura, Óscar, Santiago y Jhónnatan que le agregaron un poco de diversión a esta etapa de mi vida.

A todos los profesores que compartieron conmigo un poco de su vida, de su conocimiento y de su valioso tiempo. Especialmente al profe Élder, que no solamente me enseñó un poco de Optimización y Topología, sino a ser una mejor persona y a superarme cada día; a Jhean, que siempre me apoyó y ayudó incluso sin ser su alumno; a Diego, que nuestras conversaciones fueron más humorísticas que matemáticas; a Félix, cuya pedagogía me mostró de una manera clara y hermosa el mundo de las EDO; y por último y no menos importante, a nuestro director Andrés, que sin su dedicación este trabajo no hubiera sido posible.

En general, agradezco a cualquier persona que estuvo conmigo en alguna parte de mi vida universitaria, incluso, a las personas que me apoyaron y por sus razones, ya no comparten conmigo. Sé que aprendí algo de cada una y eso ayudó a que todo esto se diera.

Y finalmente, agradezco a Steven por no abandonar a Émerzon en ningún momento.

Émerzon Steven Mendoza Quintero

En primer lugar, agradezco a Dios, que me ha dado fortaleza.

A mi padre y a mi madre, que me amaron, apoyaron y estuvieron conmigo a lo largo de este proceso.

A mis tres hermanos, porque siempre he querido ser mejor por ellos.

En general, a toda mi familia, que siempre ha estado orgullosa de mí y de lo que estudio.

A todos mis compañeros de cuatro patas, los que están conmigo y los que ya partieron de este mundo, por acompañarme hasta tarde en la noche cuando tenía que estudiar o terminar algún trabajo.

A Josué, por hacerme feliz, quererme y ayudarme cuando lo necesito.

A los amigos que he hecho a lo largo de la carrera, gracias porque siempre nos hemos apoyado mutuamente.

Al profesor Andrés Ríos, por darnos palabras de ánimo cuando dudábamos.

A mí misma, por no rendirme y siempre seguir adelante.

Angélica Liliana Porras Barón

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE TABLAS

pág.

RESUMEN

TÍTULO: DISEÑO, CREACIÓN Y GESTIÓN DE BASES DE DATOS RELACIONALES: UN ENFOQUE PRÁCTICO CON SQL Y PYTHON. *

AUTORES: MARÍA CATALINA BALLESTEROS VELASCO - ADRIANA LUCÍA GUERRERO MERCHÁN - ÉMERZON STEVEN MENDOZA QUINTERO - ANGÉLICA LILIANA PORRAS BARÓN **

PALABRAS CLAVE: BASES DE DATOS, BIG DATA, SQL, MYSQL Y PYTHON.

DESCRIPCIÓN:

Este trabajo corresponde a un manual introductorio sobre la creación, diseño y gestión de bases de datos utilizando MySQL y Python, proporcionando una guía clara y estructurada para comprender los conceptos básicos, la implementación y la aplicación práctica de estos programas para la gestión de bases de datos relacionales. A lo largo del manual se presenta, desde cómo hacer una base de datos, cómo administrarla y sobre la realización de consultas. Todo esto se realiza mediante ejemplos, facilitando el aprendizaje de manera visual e interactiva. Se incluyen ejemplos aplicados a la bases de datos del Departamento Administrativo Nacional de Estadística (DANE)[1], lo que permite que los conocimientos adquiridos se apliquen en un contexto real.

---

* Trabajo de grado

** Facultad de Ciencias. Escuela de Matemáticas. Director: Andrés Sebastián Ríos Gutiérrez, Candidato a Doctor.

[1] Enlace de consulta `https://www.dane.gov.co/`

ABSTRACT

TITLE: DESIGN, CREATION, AND MANAGEMENT OF RELATIONAL DATABASES: A PRACTICAL APPROACH WITH SQL AND PYTHON. *

AUTORs: MARÍA CATALINA BALLESTEROS VELASCO - ADRIANA LUCÍA GUERRERO MERCHÁN - ÉMERZON STEVEN MENDOZA QUINTERO - ANGÉLICA LILIANA PORRAS BARÓN **

kEYWORDS: DATABASES, BIG DATA, SQL, MYSQLQ AND PYTHON.

DESCRIPTION:

This work is an introductory manual on the creation, design, and management of databases using MySQL and Python, providing a clear and structured guide to understanding the basic concepts, implementation, and practical application of these programs for managing relational databases. Throughout the manual, we cover everything from how to create a database, manage it, and perform queries. All of this is done through examples, facilitating learning in a visual and interactive manner. Examples applied to the databases of the National Administrative Department of Statistics (DANE)[1] are included, allowing the acquired knowledge to be applied in a real-world context.

---

* Bachelor Thesis

** Faculty of Science. School of Mathematics. Advisor: Andrés Sebastián Ríos Gutiérrez, Ph.D. Candidate.

1 Consultation link `https://www.dane.gov.co/`

INTRODUCCIÓN

El avance tecnológico y la disponibilidad de grandes volúmenes de datos han transformado radicalmente la práctica de las ciencias matemáticas en las últimas décadas, proporcionando las bases necesarias para realizar análisis y modelos. En este contexto, el manejo eficiente y la manipulación adecuada de datos se han convertido en habilidades esenciales para los investigadores y profesionales en el campo de las matemáticas.

En este vasto universo matemático, el uso de tecnologías específicas ha transformado la forma en que se almacenan, acceden y transforman los datos. Entre las tecnologías que destacan están las bases de datos relacionales [2]. Estas se distinguen porque los datos almacenados son de tipo estructurado, es decir, los datos están organizados en tablas, en filas y columnas. Las bases de datos relacionales presentan una estructura rigurosa y un soporte completo para operaciones como transacciones, aislamiento y durabilidad. Esto las convierte en tecnologías adecuadas para las aplicaciones que requieren una integridad de los datos estricta, como la gestión de inventarios, sistemas de gestión de clientes y sistemas de reservas.

En este contexto, se seguirán explorando tecnologías y herramientas que son ampliamente utilizadas en el análisis de datos. Una de las herramientas más destacadas en este ámbito es MySQL [3], cuya sintaxis se basa en el lenguaje SQL. MySQL es una solución de gestión de bases de datos relacionales desarrollada por Oracle, que permite almacenar, organizar y acceder a datos de manera eficiente. Su capacidad para manejar y analizar datos provenientes de diversas fuentes, incluidas las bases de datos relacionales, la convierte en una herramienta poderosa para los matemáticos que buscan transformar datos

---

[2] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. En: Communications of the ACM 13.6 [1970], págs. 377-387. DOI: 10.1145/362384.362685

[3] Oracle Corporation. MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/. 2020

en información valiosa y procesable.

La importancia de las bases de datos relacionales en el manejo de grandes volúmenes de datos radica en su capacidad para mantener la integridad y la consistencia de la información. Estas bases de datos utilizan el lenguaje SQL (Structured Query Language) para realizar consultas y manipular los datos, permitiendo a los usuarios recuperar información de manera rápida y precisa. Además, las bases de datos relacionales admiten operaciones complejas y transacciones que aseguran la fiabilidad de los datos, lo cual es fundamental en aplicaciones críticas donde la precisión es crucial.

El desarrollo de este manual se fundamenta en la creciente necesidad de contar con recursos educativos prácticos que faciliten el aprendizaje y la implementación de bases de datos relacionales en entornos académicos y profesionales. A lo largo de este trabajo, se abordarán los principios teóricos fundamentales de las bases de datos relacionales, así como las técnicas prácticas para su diseño, creación y gestión. Se hará especial énfasis en el lenguaje SQL como herramienta principal para la interacción con las bases de datos, y en Python como lenguaje de programación complementario que permite la automatización y posterior implementación para el análisis estadístico. Se ilustrará el uso de Jupyter Notebook como entorno de desarrollo integrado, y de MySQL Workbench como herramienta de administración y modelado de bases de datos. El manual resultante no solo proporcionará una guía paso a paso para la configuración y uso de estas herramientas, sino que también incluirá ejemplos con bases de datos reales y consultas diseñadas para reforzar el aprendizaje y la comprensión de los conceptos tratados.

1. Planteamiento y justificación del problema

A lo largo del tiempo, la forma de almacenar datos ha evolucionado, pasando del papel a los sistemas computacionales, lo que ha llevado a un crecimiento inmenso en la cantidad de información que se requiere almacenar. Esta expansión ha hecho que la búsqueda y la gestión de datos sean cada vez más complejas, generando la necesidad de herramientas eficientes como las bases de datos, que permiten realizar consultas de manera rápida y precisa.

En la actualidad, las empresas dependen de bases de datos para almacenar y gestionar información de todo tipo. Sin embargo, el avance tecnológico exige un conocimiento actualizado en el uso de bases de datos.

En este sentido, se está planificando la apertura de la asignatura de Big Data y Analítica de Datos en la Universidad Industrial de Santander para los pregrados de Matemáticas y Licenciatura en Matemáticas, con el fin de brindar algunas de las herramientas más necesarias en el campo laboral de los futuros profesionales.

La elaboración de un manual detallado sobre bases de datos se presenta como una herramienta indispensable para complementar la formación académica y preparar a los profesionales para los desafíos tecnológicos actuales.

2. Objetivos

Objetivo general

Diseñar un manual para la creación y gestión de bases de datos con datos reales y simulados, utilizando MySQL y Python.

Objetivos específicos

- Construir una base de datos simulada y a partir de esta, explicar el manejo de las funciones más utilizadas en el lenguaje SQL mediante MySQL.
- Implementar consultas a la base de datos del Censo realizado por el DANE en 2018, utilizando MySQL.
- Implementar consultas SQL del objetivo anterior, desde Jupyter Notebook haciendo uso de bibliotecas como Pandas, Numpy y Pyspark a la base de datos del Censo Nacional de Población y Vivienda realizado por el DANE en el 2018.

3. Fundamentos y trabajos previos

Los fundamentos teóricos del presente proyecto se basan en recursos digitales que se recopilan durante todo el desarrollo de este mismo; dichos recursos son:

- "SQL Tutorial" ofrecido por el sitio web W3Schools, cuyo propósito principal es enseñar de una manera sencilla y práctica a programar y desarrollar páginas web [4].
- "Curso de SQL y BASES DE DATOS Desde Cero para PRINCIPIANTES" publicado en la plataforma Youtube por el canal "MoureDev by Brais Moure" [5].
- Manuales previos y proyectos afines cuyo contenido es una serie de guías de inicio en SQL, uso de SQL con Python y desarrollo de proyectos en SQL y/o Python.

A continuación, se muestran los manuales y libros consultados para el desarrollo de este proyecto.

- MANUAL DE PRÁCTICA BÁSICA CON SQL. Este manual contiene una serie de prácticas utilizando el sistema gestor de base de datos de Oracle Database con el fin de hacer uso del lenguaje de consultas estructurado SQL a través de la línea de comandos sin pretender que funcionen como formularios exclusivos [6].

---

[4] W3schools. SQL Tutorial. Recuperado el 6 de abril del 2025. URL: https://www.w3schools.com/sql/default.asp

[5] Canal MoureDev by Brais Moure. Curso de SQL y BASES DE DATOS Desde Cero para PRINCIPIANTES. Enlace web: https://www.youtube.com/watch?v=OuJerKzV5T0. 2024

[6] Evangelista Nava Elizabeth. MANUAL DE PRÁCTICA BÁSICA CON SQL. En: Centro Universitario UAEM Atlacomulco [2015]

- Juez de consultas SQL basado en Postgresql. Este juez de consultas recoge el proceso de desarrollo de una aplicación de escritorio para la corrección de consultas SQL cuyo objetivo principal es proporcionar una herramienta para el aprendizaje y la enseñanza en los fundamentos de bases de datos relacionales creando una aplicación mediante Python[7].

- Learning SQL: Master SQL Fundamentals. Un libro con quince capítulos y tres apéndices que muestra que las sentencias usadas en el lenguaje SQL se pueden reducir a dos tipos: las sentencias utilizadas para crear objetos de base de datos (tablas, índices, restricciones, etc.) que se conocen colectivamente como sentencias de esquema SQL y las sentencias utilizadas para crear, manipular y recuperar los datos almacenados en una base de datos, que se conocen como sentencias de datos SQL. A partir de lo anterior, el contenido del libro inicia desde la historia de las bases de datos, cómo crearlas y llenarlas de datos, hasta el uso de los Metadatos. Aunque este libro muestra muchas de las sentencias de esquema SQL, su enfoque principal está en las características de programación y uso del sistema gestor MySQL [8]

- The Fundamentals of People Analytics With Applications in R. Un capítulo dedicado exclusivamente a SQL llamado “Introduction to SQL” donde muestran los comandos y las funciones más básicas con ejemplos sencillos. Este libro enseña estadística aplicada y análisis de datos mediante el uso de R Studio y SQL. [9].

- Practical SQL, 2nd Edition: A Beginner’s Guide to Storytelling with Data. Esta guía

---

[7] Raúl Medina González. Juez de consultas SQL basado en Postgresql. En: [2020]

[8] Alan Beaulieu. Learning SQL: Master SQL Fundamentals. 2nd. O’Reilly, 2009.

[9] Craig Starbuck. The Fundamentals of People Analytics With Applications in R. St. Louis, MO, USA: Springer, 2023.

para principiantes comienza con los fundamentos de bases de datos, consultas, tablas y datos que son comunes a SQL en múltiples sistemas de bases de datos. Los capítulos 13 al 17 cubren temas más específicos de PostgreSQL, como la búsqueda de texto completo y los sistemas de información geográfica (GIS) [10].

- SQL Pocket Guide: A Guide to SQL Usage. Este libro guía no pretende cubrir todos los conceptos de SQL en profundidad, sino ser una simple referencia para cuando se ha olvidado alguna sintaxis y se necesita consultarla rápidamente o cuando se ha trabajando en otro lenguaje de programación durante un tiempo y es necesario un repaso rápido de cómo funciona SQL. Este libro se divide en tres secciones. Conceptos básicos, que inicia en el capítulo uno y termina en el cuatro; Objetos de base de datos, tipos de datos y funciones, que recorren el capítulo cinco hasta el siete y finaliza con los conceptos avanzados que se mencionan en los capítulos ocho, nueve y diez [11].

- C and Python Applications: Embedding Python Code in C Programs, SQL Methods, and Python Sockets. Este libro enfocado en la programación muestra cómo usar C y Python para escribir aplicaciones en SQL. Los dos primeros capítulos repasan los fundamentos de C y Python y los capítulos siguientes se agrupan en técnicas SQL, Python incrustado y aplicaciones de sockets[12].

- The Language of SQL. La organización de los veinte capítulos de este libro consiste en presentar sus temas en una secuencia única. Comenzando con la recuperación de datos y en los capítulos finales, abordando el diseño de bases de datos. El autor

---

[10] Anthony DeBarros. Practical SQL, 2nd Edition: A Beginner's Guide to Storytelling with Data. 2nd. 2022.

[11] Alice Zhao. SQL Pocket Guide: A Guide to SQL Usage. 4th. O'Reilly, 2021.

[12] Philip Joyce. C and Python Applications: Embedding Python Code in C Programs, SQL Methods, and Python Sockets. Apress, 2022

lo presenta de dicha forma como una táctica motivacional, permitiéndole al usuario adentrarse rápidamente en temas interesantes relacionados con la recuperación de datos, antes de abordar los temas más complejos de los índices y las claves foráneas [13].

- STRUCTURED QUERY LANGUAGE (SQL): A Practical Introduction. Cada capítulo de los once presentes en este libro contiene un ligero resumen del contenido de este. El libro comienza con dos capítulos introductorios a SQL. El primero, que describe algunos términos utilizados en el procesamiento de datos y su relación con SQL; y el segundo, una guía básica de SQL que presenta una descripción general de los comandos y sentencias usadas en SQL. A lo largo de los capítulos, se muestra la creación, el mantenimiento y las consultas de tablas; la adición, actualización e integridad de los datos; seguridad en las bases de datos y finaliza con el uso de SQL con un lenguaje anfitrión. [14].

[13] Larry Rockoff. The Language of SQL. 3rd. Addison-Wesley, 2022.

[14] Akeel Din. STRUCTURED QUERY LANGUAGE (SQL): A Practical Introduction. NCC BLACKWELL

4. Descripción del caso de estudio

Para este trabajo se toman los datos del Censo Nacional de Población del 2018 (recuento detallado de las personas que viven en el territorio colombiano en el año 2018), que están disponibles en la página del Departamento Administrativo Nacional de Estadística (DANE) [15], en específico, se trabaja con los datos de los departamentos de Antioquia, donde se toma la tabla referente a los fallecidos; La Guajira, donde se estudia la tabla de personas; Magdalena, donde se toma la tabla de hogares; y por último, en el departamento de Santander se analiza la tabla de viviendas. Con estos datos se realizan diferentes consultas mediante Python en conjunto con el lenguaje SQL, realizando algunas gráficas y consultas a los datos con base en la información que brinda cada una de las tablas de los distintos departamentos.

4.1. Zonas de estudio

La zona de estudio se centra en algunos departamentos de Colombia, que se detallan a continuación. Para cada departamento se estudia una tabla distinta. Este estudio consiste en hacer diferentes consultas de cada tabla a través de Python.

4.1.1. Antioquia

Antioquia es un departamento de Colombia ubicado al noroccidente del país. Limita al norte, con el mar Caribe; al oriente, con los departamentos de Córdoba, Sucre, Bolívar, Santander y Boyacá; al sur, con Caldas y Risaralda; y al occidente, con Chocó. Su territo-

[15] Enlace de consulta: `https://microdatos.dane.gov.co/index.php/catalog/643/study-description#metadata-scope`

rio se caracteriza por una geografía montañosa atravesada por la cordillera de los Andes, lo que influye en su clima y distribución poblacional. Su capital es Medellín.

Para este departamento, se tomó la tabla de fallecidos, en la cual se analizan diversas variables relacionadas con la mortalidad en el departamento. Entre ellas, se incluyen el municipio de residencia o fallecimiento, la edad de la persona, el sexo y la causa de la muerte, entre otros factores.

4.1.2. La Guajira

La Guajira es un departamento de Colombia ubicado en el extremo norte del país que limita al norte y al oriente con el mar Caribe y Venezuela; en el sur, con Cesar; y por el occidente, con Magdalena. Su territorio está compuesto por una combinación de zonas desérticas, serranías y llanuras costeras, lo que influye en su clima seco y en las condiciones de vida de sus habitantes. La capital del departamento es Riohacha, una ciudad con una fuerte presencia de comunidades indígenas, principalmente el pueblo Wayúu, que constituye gran parte de la población.

Para este departamento, se tomó la tabla de datos de personas, en la cual se analizan diversas variables demográficas y socioeconómicas. Entre ellas, se incluyen aspectos como la distribución de la población por municipios, edad, sexo, etnia, nivel educativo, alfabetización, entre otros.

4.1.3. Magdalena

El Magdalena es un departamento de Colombia ubicado al noreste del país, en la región Caribe. Este departamento limita al norte con el mar Caribe; al oriente, con La Guajira y Cesar; al sur, con Bolívar; y al occidente, con Atlántico. Su geografía es diversa,

con zonas costeras, valles y la imponente Sierra Nevada de Santa Marta. La capital del departamento es Santa Marta, una de las ciudades más antiguas de América.

Para este departamento, se tomó la tabla de datos de los hogares, en la cual se analizan diversas características relacionadas con la composición y condiciones de los hogares. Entre ellas, se incluyen el número total de cuartos por hogar, el acceso al agua, el lugar donde se preparan los alimentos, el número total de personas en el hogar y otras variables que permiten evaluar la calidad de vida de los habitantes del Magdalena.

4.1.4. Santander

Santander es un departamento de Colombia ubicado al noreste del país, en la región Andina. Dicho departamento limita al norte, con Cesar y Norte de Santander; al oriente, con Boyacá, en el sur; con Boyacá y Antioquia; y al occidente, con Bolívar. Su geografía está marcada por la presencia de la cordillera Oriental, con extensos valles, cañones y montañas que influyen en su clima variado. Su capital es Bucaramanga.

Para este departamento, se tomó la tabla de datos de as viviendas, en la cual se analizan diversas características relacionadas con las condiciones habitacionales. Entre ellas, se incluyen aspectos como el tipo de vivienda (casa, apartamento, tipo de cuarto, entre otros), los materiales de construcción, el acceso a servicios públicos (agua, energía, alcantarillado, recolección de residuos), total de hogares en la vivienda, el estrato, y otras variables que permiten evaluar las condiciones de las viviendas en las que viven los santandereanos.

5. Metodología

5.1. Seminario

La metodología de este proyecto se basa en un seminario, cuyo desarrollo fue el trabajo grupal en reuniones semanales presenciales o virtuales, en las cuales se trataron y discutieron diversos temas relacionados con la creación de un manual para el manejo de bases de datos relacionales en SQL. Durante estas sesiones del seminario, también se abordaron temas adicionales, como el uso de Python y otros lenguajes de programación en caso de ser necesarios para la gestión y manipulación de datos. En cada reunión se debatió acerca de cuáles serían los objetivos específicos; dichos objetivos guiaron el seminario y permitieron un avance para la construcción del manual. En las sesiones virtuales, se usaron herramientas en línea para compartir recursos digitales, escribir en conjunto y hacer revisiones en tiempo real del proyecto. Una de las características más importantes del seminario fue la constante discusión semanal sobre los enfoques más pertinentes para explicar los conceptos y fundamentos dentro del manual. En varias ocasiones, surgieron diferencias de opinión respecto a la mejor manera de organizar la información, seleccionar ejemplos y definir el nivel técnico y práctico del contenido. Estos debates se convirtieron en parte fundamental del proceso que se llevó a cabo, ya que impulsaron a analizar más profundamente cada decisión a tomar, asegurando que el manual fuera claro, preciso y accesible para todo tipo de usuario. Además, las reuniones permitieron detectar posibles errores conceptuales o técnicos, lo que generó una investigación más a fondo en ciertas áreas del proyecto. En particular, discutimos la integración de Python con SQL, la optimización de consultas y la comparación entre distintas herramientas de programación para la gestión de bases de datos. Gracias a este proceso continuo de revisión y mejora entre estudiantes y el profesor, se logró definir la estructura final del manual y garantizar que incluyera ejemplos prácticos y explicaciones bien fundamentadas. En conclusión, la me-

todología basada en un seminario de reuniones semanales, combinada con un enfoque de discusión activa y revisión continua por parte de estudiantes y profesor, no solo facilitó la creación del manual, sino que también fortaleció el conocimiento en bases de datos y programación. Este proceso permitió que el producto final fuera más preciso, estructurado y útil para los usuarios a los que está dirigido.

6. PRODUCTO

Presentación

Este manual surge a partir de la observación, el uso y la recopilación de diversos recursos disponibles en línea, así como de textos especializados en el manejo de Python y SQL. Su elaboración ha sido el fruto de constantes reuniones de revisión y discusión, desarrolladas en el marco del seminario bajo la orientación y dirección del profesor Andrés Sebastián Ríos Gutiérrez.

El objetivo principal de este trabajo es brindar una introducción práctica y clara al manejo de bases de datos para cualquier persona interesada —ya sean profesionales del campo de la computación, ciencia de datos, estadística, matemáticas, ingeniería de sistemas, software o industrial— facilitando un primer acercamiento al apasionante mundo de la programación orientada a datos.

A lo largo del manual, se abordan desde los aspectos más esenciales, como la presentación de distintos sistemas de gestión de bases de datos, con especial énfasis en MySQL, el cual fue escogido como el principal sistema de gestión para llevar a cabo el manual. Esta elección se debe a su buen rendimiento al ejecutar consultas complejas sobre conjuntos de datos extensos, además de su alta compatibilidad con herramientas de análisis como Python.

También se ofrece una explicación clara sobre los tipos de datos utilizados en MySQL, explicando de forma detallada los pasos necesarios para su descarga e instalación. Además, el manual proporciona información importante sobre la administración de bases de datos y tablas, el uso de modificadores, la inserción de datos, la relación entre tablas, la creación de tablas relacionadas y la consulta de datos relacionados, lo cual facilita su comprensión incluso para quienes se inician en este campo.

De igual forma, se brinda información detallada para una correcta instalación de Python,

Jupyter Notebook y Miniconda, así como las indicaciones necesarias para descargar los archivos CSV desde la página oficial del Departamento Administrativo Nacional de Estadística (DANE) donde se encuentran disponibles los datos del Censo Nacional de Población del año 2018 en Colombia.
El manual también explica, paso a paso, cómo establecer la conexión entre SQL y Python, precisando el proceso de integración de estos dos entornos con el fin de automatizar consultas y análisis de datos.
Siguiendo esta línea, se presentan los códigos y pasos necesarios para realizar distintos tipos de consultas y análisis sobre la base de datos del censo anteriormente mencionado.
Finalmente, este manual se constituye como herramienta útil para quienes desean adentrarse en el mundo del análisis de datos, aprovechando sus conocimientos previos en Python y ampliándolos hacia el manejo de bases de datos. Asimismo, busca establecer lazos entre la matemática, la programación y la inteligencia artificial, por tratarse de un posible primer texto de consulta para un curso relacionado con asignaturas como Big Data, Bases de Datos, Minería de Datos e Inteligencia de Negocios.

6.1. Introducción a las bases de datos y SQL

Una **base de datos** es un conjunto organizado y estructurado de información, diseñado específicamente para almacenar, gestionar y facilitar el acceso eficiente a datos diversos. Las bases de datos permiten guardar grandes cantidades de información de forma segura y ordenada, lo que facilita la recuperación rápida y precisa de estos datos cuando se necesitan.
Los datos almacenados en una base de datos pueden ser muy variados, desde textos y números, hasta imágenes, archivos multimedia o cualquier otro tipo de información digital. Debido a su capacidad para manejar distintos tipos de datos, las bases de datos se utilizan ampliamente en múltiples áreas, tales como negocios, educación, salud, finanzas,

comercio electrónico y administración pública, entre otros sectores. La importancia de las bases de datos radica en que permiten organizar la información de manera coherente y accesible, asegurando que los datos estén disponibles cuando y cómo se requieren, apoyando así la toma de decisiones informadas y eficientes. Además, las bases de datos desempeñan un papel fundamental en el análisis de datos, ya que permiten almacenar grandes volúmenes de información, facilitando su posterior consulta, procesamiento y análisis para descubrir tendencias, patrones y tomar decisiones estratégicas.

Entre los diferentes tipos de bases de datos, una de las más utilizadas es la **base de datos relacional**. Este modelo almacena los datos en estructuras denominadas tablas, que están organizadas en filas y columnas. Las filas representan conjuntos específicos de información, mientras que las columnas representan características o atributos que describen dicha información. Una característica esencial de las bases de datos relacionales es que permiten relacionar tablas entre sí, facilitando la organización y el manejo de grandes volúmenes de información interconectada.

Para interactuar con las bases de datos relacionales se utiliza el lenguaje **SQL** (Structured Query Language, o Lenguaje de Consulta Estructurado). SQL es un lenguaje estándar diseñado específicamente para gestionar información en bases de datos relacionales. Gracias a su sintaxis sencilla y clara, SQL permite a los usuarios realizar consultas específicas para recuperar datos, además de insertar, actualizar y eliminar información almacenada. La versatilidad de SQL lo ha convertido en la herramienta esencial para cualquier actividad que implique gestionar datos almacenados en bases de datos relacionales.

6.2. Sistemas de gestión de base de datos

Un **Sistema de Gestión de Bases de Datos (SGBD)** es un software especializado diseñado para crear, administrar, almacenar y gestionar bases de datos de manera eficiente, segura y estructurada.

Los sistemas de gestión de bases de datos cumplen diversas funciones esenciales para

el manejo efectivo de la información, tales como:

- **Definir y crear bases de datos**: permiten establecer la estructura en la que se organizarán los datos, incluyendo sus relaciones y restricciones.
- **Almacenar y gestionar datos**: proporcionan métodos eficientes y seguros para almacenar grandes volúmenes de información, garantizando su disponibilidad y accesibilidad.
- **Garantizar la integridad de los datos**: aplican reglas y restricciones que evitan redundancias e inconsistencias, asegurando la calidad y precisión de la información.
- **Consultar y actualizar información**: facilitan la recuperación rápida y precisa de datos, así como su inserción, modificación y eliminación a través de lenguajes específicos como SQL.
- **Controlar el acceso y la seguridad**: permiten establecer y gestionar permisos para definir quién puede acceder a la información y bajo qué condiciones, garantizando su confidencialidad y protección.
- **Administrar múltiples usuarios**: permiten que varios usuarios interactúen simultáneamente con la base de datos, asegurando que no se produzcan conflictos en las operaciones realizadas.
- **Respaldar y restaurar datos**: posibilitan la creación de copias de seguridad y permiten recuperar la información en caso de fallas, errores o situaciones de pérdida accidental.

La importancia de los sistemas de gestión de bases de datos radica en que simplifican significativamente el manejo de grandes volúmenes de información, proporcionando herramientas avanzadas para su análisis y garantizando la calidad, integridad y seguridad de los datos. Algunos de estos sistemas son:

6.2.1. Oracle DB

Oracle Database (conocido comúnmente como Oracle DB) es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation. Es uno de los sistemas más utilizados a nivel empresarial debido a su capacidad para manejar grandes volúmenes de datos con un alto rendimiento, escalabilidad y seguridad.Oracle DB proporciona herramientas avanzadas para la gestión de datos, ofreciendo soporte completo para transacciones complejas, recuperación ante fallos y procesamiento eficiente de consultas complejas sobre grandes volúmenes de datos. Además, su flexibilidad y robustez lo hacen idóneo para aplicaciones críticas en sectores como finanzas, telecomunicaciones, comercio electrónico y administración pública.[16]

6.2.2. SQL Server

SQL Server es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. Es ampliamente utilizado en entornos corporativos por su alto rendimiento, seguridad avanzada y capacidad para gestionar eficientemente grandes volúmenes de información. SQL Server proporciona funcionalidades destacadas para análisis avanzado de datos, inteligencia empresarial (Business Intelligence) y almacenamiento en la nube, convirtiéndolo en una herramienta esencial para aplicaciones que requieren capacidades analíticas complejas. Además, su integración nativa con otras aplicaciones de Microsoft facilita considerablemente su implementación y uso en diferentes contextos empresariales[17].

6.2.3. PostgreSQL

---

[16] Oracle Corporation. Oracle Database Documentation. Recuperado el 26 de marzo de 2025. 2021. URL: https://docs.oracle.com/en/database/

[17] Microsoft Corporation. Microsoft SQL Server. Recuperado el 26 de marzo de 2025. 2021. URL: https://www.microsoft.com/en-us/sql-server

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto, reconocido especialmente por su capacidad de manejar datos complejos, su cumplimiento riguroso de los estándares SQL y su comunidad activa y colaborativa. Al ser un sistema de **código abierto**, PostgreSQL permite a cualquier usuario acceder a su código fuente, lo que permite modificar, mejorar y distribuir dicho código libremente. Esta característica ha generado una comunidad robusta de desarrolladores que constantemente contribuye con mejoras, nuevas funcionalidades y soluciones a problemas emergentes, asegurando así la constante evolución y calidad del software. PostgreSQL destaca por su estabilidad, flexibilidad y alto rendimiento, siendo ampliamente utilizado en aplicaciones avanzadas como análisis de datos complejos, aplicaciones geoespaciales, inteligencia empresarial, sistemas financieros y aplicaciones científicas [18].

6.2.4. MySQL

MySQL es un sistema de gestión de bases de datos relacionales ampliamente utilizado a nivel mundial, reconocido especialmente por su rapidez, facilidad de uso y fiabilidad. Originalmente desarrollado como un proyecto de código abierto, MySQL se ha consolidado como uno de los sistemas preferidos para aplicaciones web, sitios de comercio electrónico y otros entornos que requieren gestionar grandes volúmenes de datos de forma eficiente y segura. Su popularidad radica en la combinación de velocidad, escalabilidad y facilidad de administración. Además, ofrece soporte para múltiples usuarios, transacciones y funciones avanzadas que permiten realizar operaciones complejas, y esto lo convierte en una opción adecuada tanto para proyectos pequeños como para sistemas empresariales[19].

---

[18] PostgreSQL Global Development Group. PostgreSQL Features. Recuperado el 26 de marzo de 2025. 2023. URL: https://www.postgresql.org/about/

[19] Oracle Corporation. MySQL Overview. Recuperado el 26 de marzo de 2025. 2023. URL: https://www.mysql.com/about/

MySQL como herramienta para el análisis de datos

En el desarrollo de este manual se ha optado por utilizar `MySQL` como sistema de gestión de bases de datos debido a sus características de eficiencia, fiabilidad y compatibilidad con entornos de análisis de datos. MySQL es una solución sólida y ampliamente adoptada que permite gestionar grandes volúmenes de información, lo cual es fundamental en contextos donde se requiere realizar análisis estadísticos, visualizaciones gráficas y procesamiento avanzado de datos.

Una de las principales razones para elegir MySQL sobre otros sistemas es su buen rendimiento al ejecutar consultas complejas sobre conjuntos de datos extensos, sin comprometer la velocidad de respuesta ni la consistencia de los resultados. Además, su compatibilidad con herramientas de análisis como `Python` y su integración fluida con entornos interactivos como `Jupyter Notebook` permiten extraer, transformar, analizar y visualizar los datos de manera eficiente.

Entre los beneficios clave del uso de MySQL en este contexto se destacan los siguientes:

- Permite almacenar y consultar datos de gran tamaño de forma estructurada y optimizada.
- Ofrece compatibilidad con bibliotecas de análisis y visualización en Python, como `pandas`, `matplotlib` y `seaborn`.
- Facilita la conexión desde Jupyter Notebook utilizando conectores como `mysql-connector-python`, lo que permite automatizar el flujo de análisis.
- Soporta operaciones complejas necesarias para el análisis estadístico, generación de gráficos, mapas de calor y mapas autoorganizados.

- Es un sistema ampliamente documentado, con una comunidad activa y soporte continuo, lo cual facilita el aprendizaje y resolución de problemas.

Por estas razones, MySQL representa una solución adecuada y eficiente para los propósitos de este manual, permitiendo integrar de forma efectiva la gestión de bases de datos con el análisis estadístico avanzado en un entorno práctico y accesible para personas que estén iniciando en el trabajo con bases de datos.

6.3. Tipos de datos

En este capítulo se explican los tipos de datos que se usan en SQL con el fin de proporcionar una base sólida para entender cómo representar y usar adecuadamente los distintos tipos de información en una base de datos. Escoger bien el tipo de datos no solamente mejora el rendimiento y el análisis, sino que garantiza la integridad de los datos. En MySQL, los tipos de datos se clasifican en tres grandes categorías: **cadena**, **numéricos** y de **fecha y hora**[20]. Escoger el tipo de dato adecuado para cada columna es fundamental para asegurar el buen funcionamiento y rendimiento de una base de datos.

6.3.1. Tipos de datos de cadena

Estos tipos permiten almacenar información textual, como nombres, direcciones, descripciones o cualquier otro tipo de dato que use caracteres.

- `CHAR(tamaño)`: almacena cadenas de longitud fija. Si el contenido tiene menos caracteres, se completa con espacios. El tamaño puede ir de 0 a 255 caracteres. El valor por defecto es 1.

[20] MySQL Documentation Team. MySQL 8.0 Reference Manual: Data Types. Recuperado el 26 de marzo de 2025. 2023. URL: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

- `VARCHAR(tamaño)`: guarda cadenas de longitud variable, hasta un máximo de 65.535 caracteres. No se rellenan los espacios.
- `BINARY(tamaño)` y `VARBINARY(tamaño)`: similares a `CHAR` y `VARCHAR`, pero almacenan datos binarios en lugar de texto.
- `TINYTEXT`, `TEXT`, `MEDIUMTEXT` y `LONGTEXT`: permiten guardar cadenas de texto largas. Cada uno tiene un límite máximo diferente: 255, 65.535, 16.777.215 y 4.294.967.295 caracteres, respectivamente.
- `TINYBLOB`, `BLOB`, `MEDIUMBLOB` y `LONGBLOB`: almacenan datos binarios (como imágenes o archivos), con las mismas capacidades que los tipos `TEXT`.
- `ENUM(val1, val2, ...)`: permite que una columna almacene solo uno de los valores definidos en una lista. Si se inserta un valor no permitido, se guarda una cadena vacía.
- `SET(val1, val2, ...)`: permite seleccionar varios valores de una lista predefinida, hasta un máximo de 64.

6.3.2. Tipos de datos numéricos

Estos tipos se utilizan para representar valores enteros o decimales, ya sea para conteos, identificadores, precios, cantidades, etc.

- `BIT(tamaño)`: almacena valores en formato binario. El tamaño puede estar entre 1 y 64 bits. El valor predeterminado es 1.
- `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` o `INTEGER`, y `BIGINT`: representan números enteros de diferentes tamaños. Por ejemplo, `TINYINT` puede ir de -128 a 127 (con signo), y `BIGINT` puede alcanzar valores de hasta ±9 cuatrillones.

- `BOOL` y `BOOLEAN`: representan valores lógicos. El valor `0` se interpreta como falso, y cualquier valor diferente de cero como verdadero.
- `FLOAT(p)` y `DOUBLE(p)`: permiten almacenar números reales (con decimales) con precisión de punto flotante. El parámetro `p` determina la precisión: si es de 0 a 24 se usa `FLOAT`, y de 25 a 53 se usa `DOUBLE`.
- `DECIMAL(tamaño, d)` o `DEC(tamaño, d)`: usados para almacenar valores numéricos exactos con punto fijo, como valores monetarios. `tamaño` indica el número total de dígitos, y `d` cuántos de ellos van después del punto decimal.

Nota: en versiones recientes de MySQL (desde la 8.0.17), la sintaxis `FLOAT(tamaño, d)` está obsoleta y se recomienda usar `FLOAT(p)` o `DOUBLE(p)`.

6.3.3. Tipos de datos de fecha y hora

Estos tipos permiten trabajar con información cronológica, como fechas de registro, horas de ingreso o marcas de tiempo para auditoría.

- `DATE`: guarda una fecha en el formato `AAAA-MM-DD`. El rango válido va del 1 de enero del año 1000 hasta el 31 de diciembre del año 9999.
- `DATETIME(fsp)`: combina fecha y hora. Formato `AAAA-MM-DD hh:mm:ss`. Puede incluir fracciones de segundo (`fsp`). Admite inicialización automática con la fecha y hora actuales usando `DEFAULT` y `ON UPDATE`.
- `TIMESTAMP(fsp)`: similar a `DATETIME`, pero almacena el valor como una marca de tiempo basada en segundos desde `1970-01-01 00:00:00` UTC. También admite actualización automática con `CURRENT_TIMESTAMP`.

- `TIME(fsp)`: almacena únicamente la hora, en formato `hh:mm:ss`. El rango va de `-838:59:59` a `838:59:59`.
- `YEAR`: guarda un año con cuatro dígitos, entre 1901 y 2155, o `0000`. A partir de MySQL 8.0, el formato de año con dos dígitos ha sido descontinuado.

6.4. Descarga e instalación de MySQL

En este capítulo se muestra el proceso de instalación de un sistema de gestión de bases de datos relacionales (mencionado en la Sección 6.2) que usa SQL como lenguaje fuente, en este caso, MySQL. El procedimiento inicia desde el enlace de descarga y concluye con la creación de un usuario y servidor para la administración de bases de datos.
El paso a paso de la instalación se muestra a continuación:

- Ingrese al siguiente enlace: `https://dev.mysql.com/downloads/mysql/` [21]. Una vez en la página, seleccione el sistema operativo correspondiente y haga clic en el botón `Download` resaltado en color verde.

---

[21] MySQL Documentation Team. MySQL Downloads. Recuperado el 26 de marzo de 2025. 2023. URL: https://dev.mysql.com/downloads/mysql/

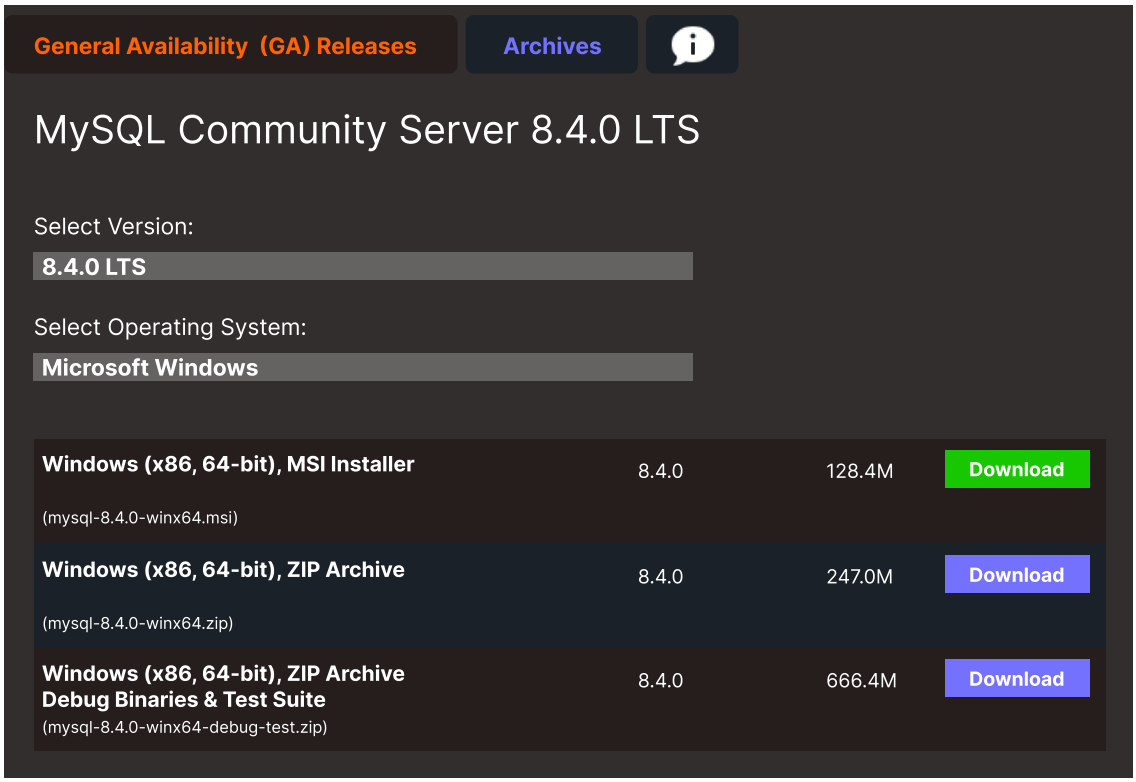

Figura 1. Elección de versión y de sistema operativo.

- El sistema ofrece la opción de registrarse, pero este paso no es obligatorio. Se puede omitir haciendo clic en el botón `No thanks, just start my download`.

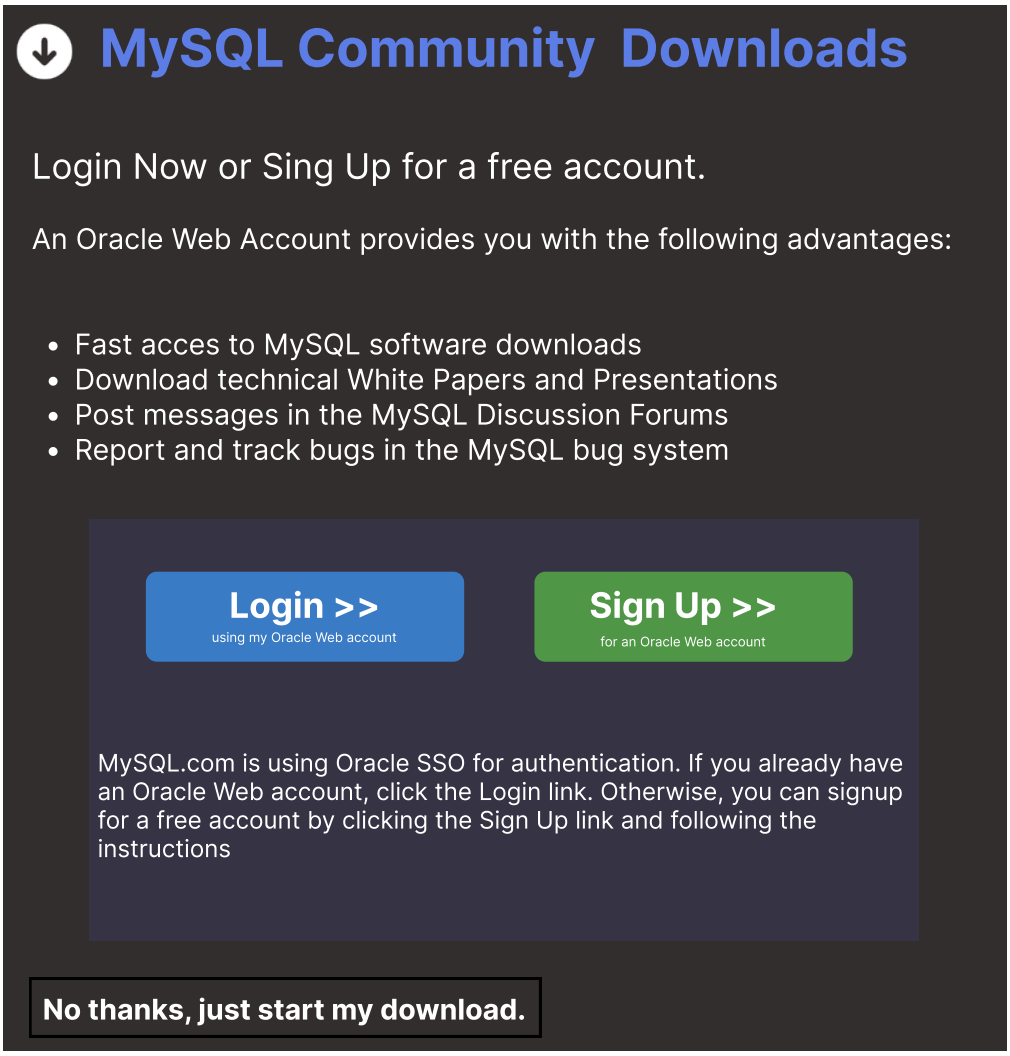

Figura 2. Inicio de sesión o registro.

- Una vez finalizada la descarga, ejecute el instalador y haga clic en el botón `Next` para continuar con la instalación.

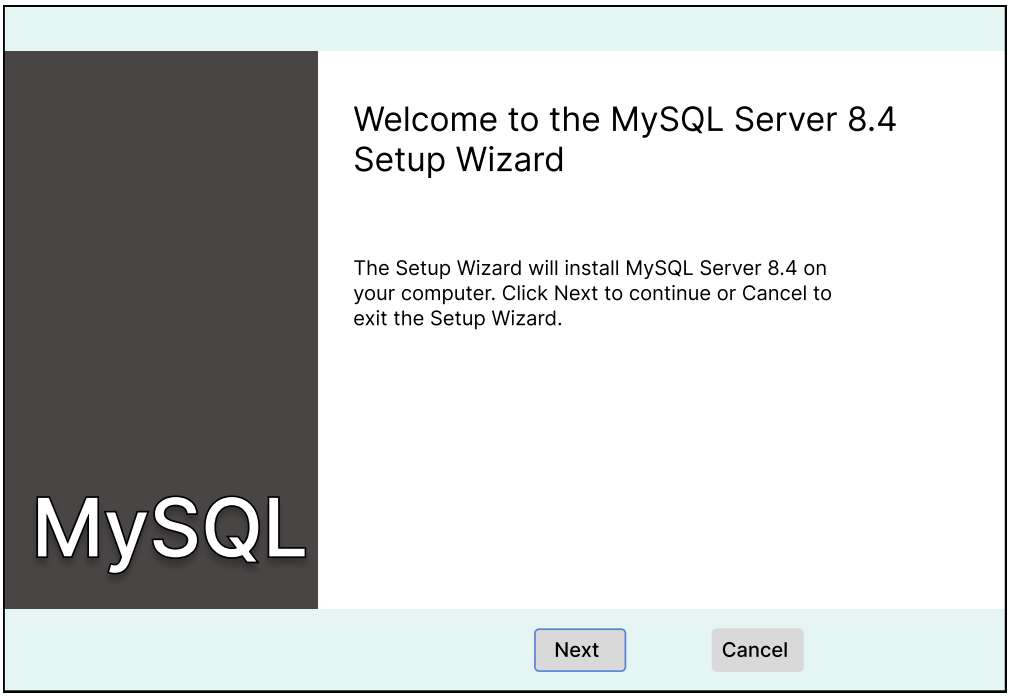

Figura 3. Interfaz de bienvenida

- Seleccione la casilla `I accept the terms in the License Agreement` para aceptar los términos de la licencia.

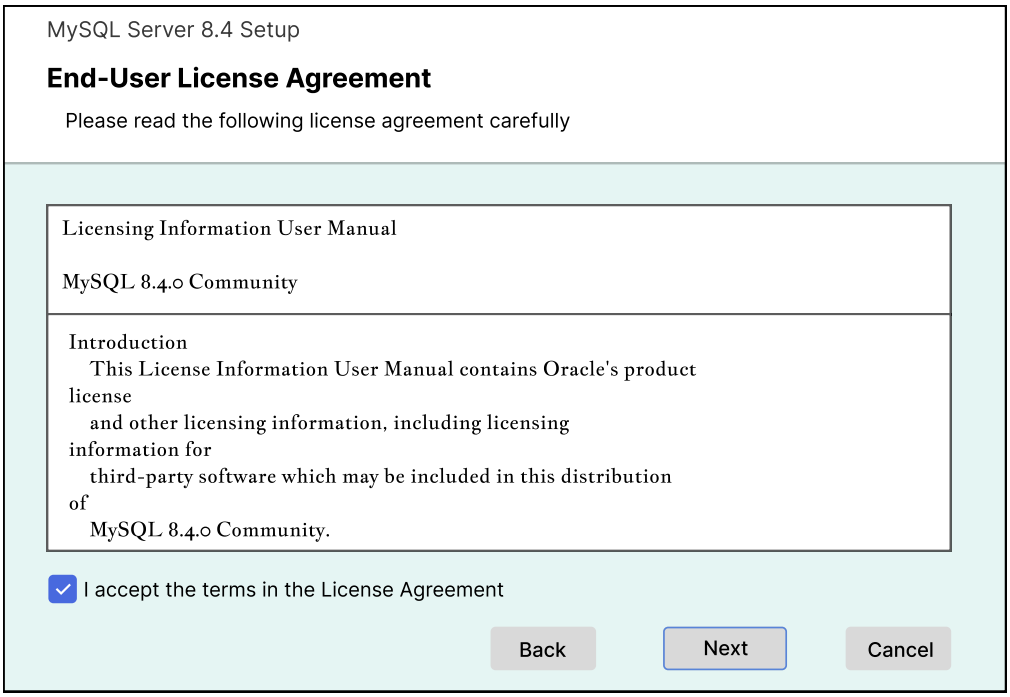

Figura 4. Términos y condiciones.

- Se recomienda seleccionar la opción `Typical`, ya que corresponde a la instalación predeterminada para la mayoría de los usuarios.

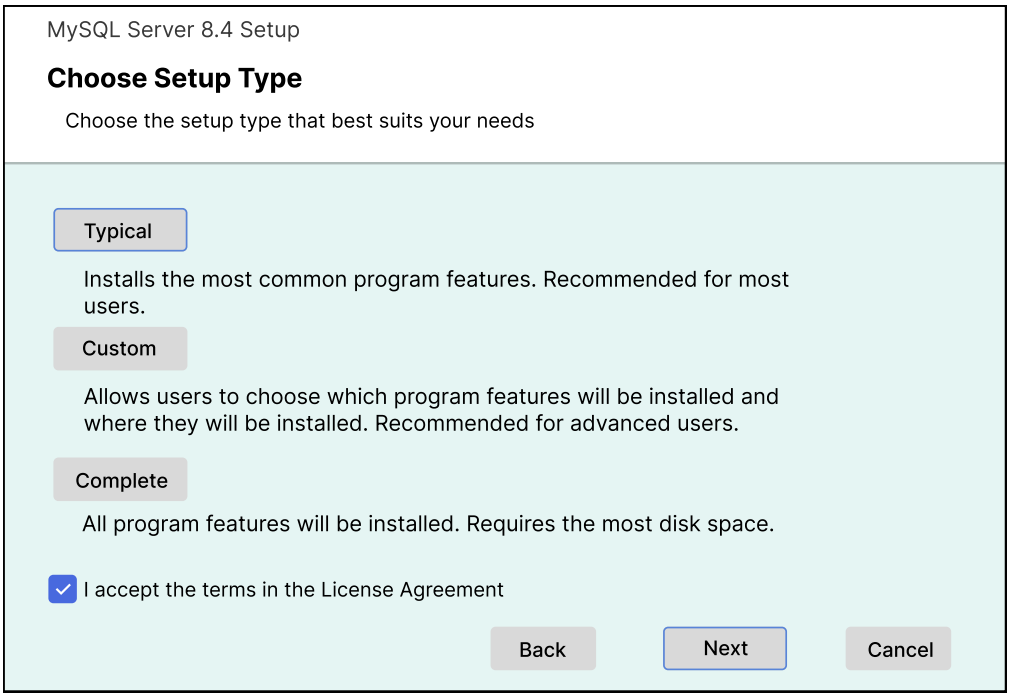

Figura 5. Elección de ajustes.

- Una vez finalizada la instalación, haga clic en la opción `Finish` para cerrar el asistente.

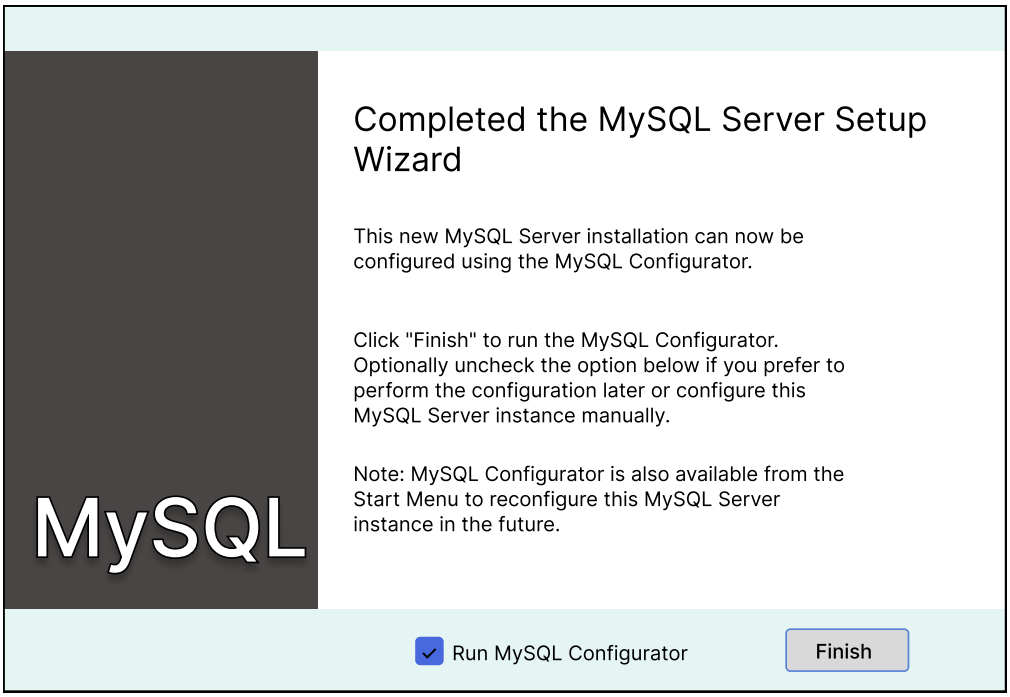

Figura 6. Pestaña de finalización de instalación.

- A continuación, se mostrará la pantalla de configuración inicial. Para continuar, haga clic en la opción `Next`.

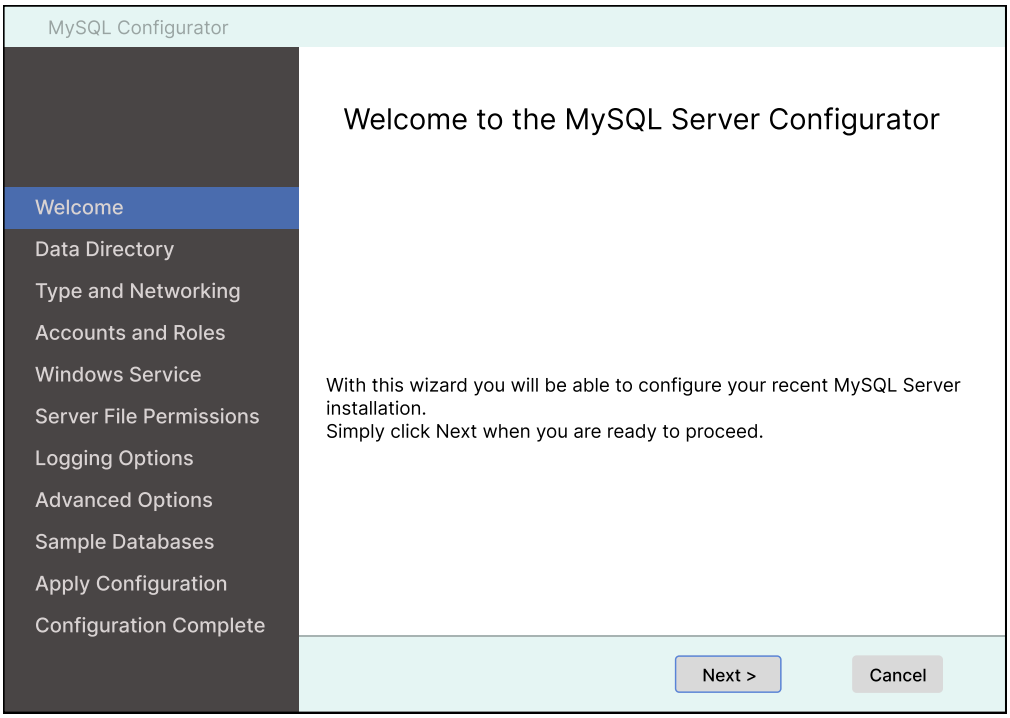

Figura 7. Interfaz de bienvenida a MySQL.

- El instalador indica que el directorio predeterminado para almacenar la información del programa es: `C:\ProgramData\MySQL\MySQL Server 8.4\`.

  Si se desea modificar esta ubicación, haga clic en el ícono de los tres puntos ubicado al lado de la dirección y seleccione la nueva ruta. En este caso, no se realizarán cambios, por lo que se debe hacer clic en la opción `Next`.

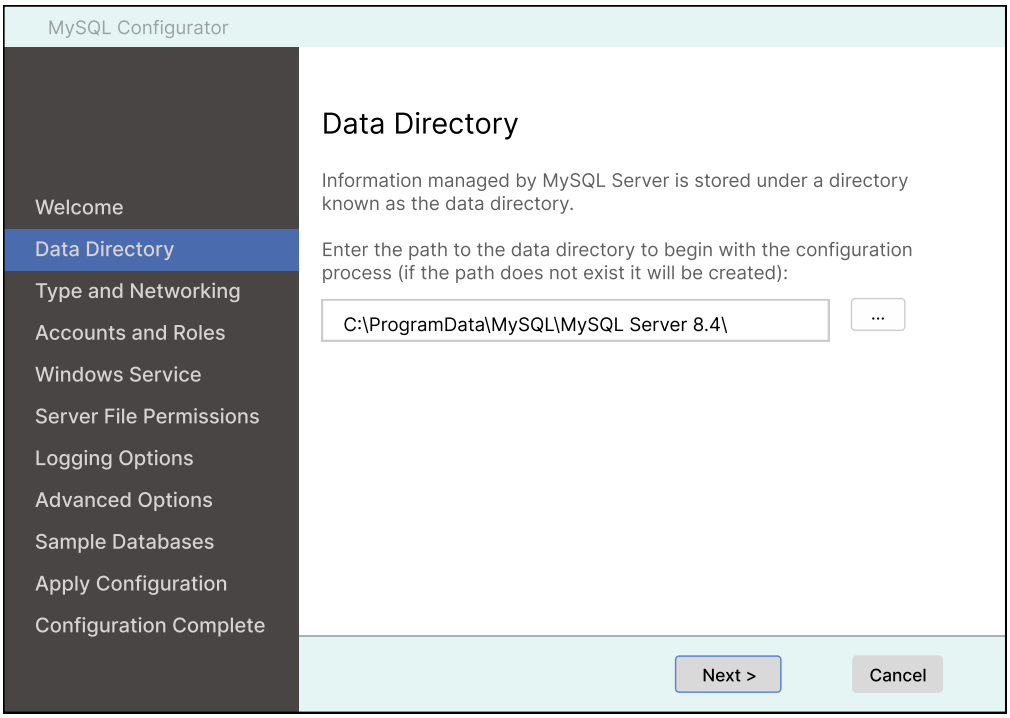

Figura 8. Destino de almacenamiento.

- En esta etapa de la configuración, se solicita seleccionar el tipo de equipo al que se conectará MySQL. Las opciones disponibles son: `Development Computer`, `Server` y `Dedicated Server`. La selección dependerá del caso de uso.

  Para este ejemplo, se utilizará la opción `Development Computer` junto con la configuración predeterminada. A continuación, haga clic en `Next` para continuar con el procedimiento.

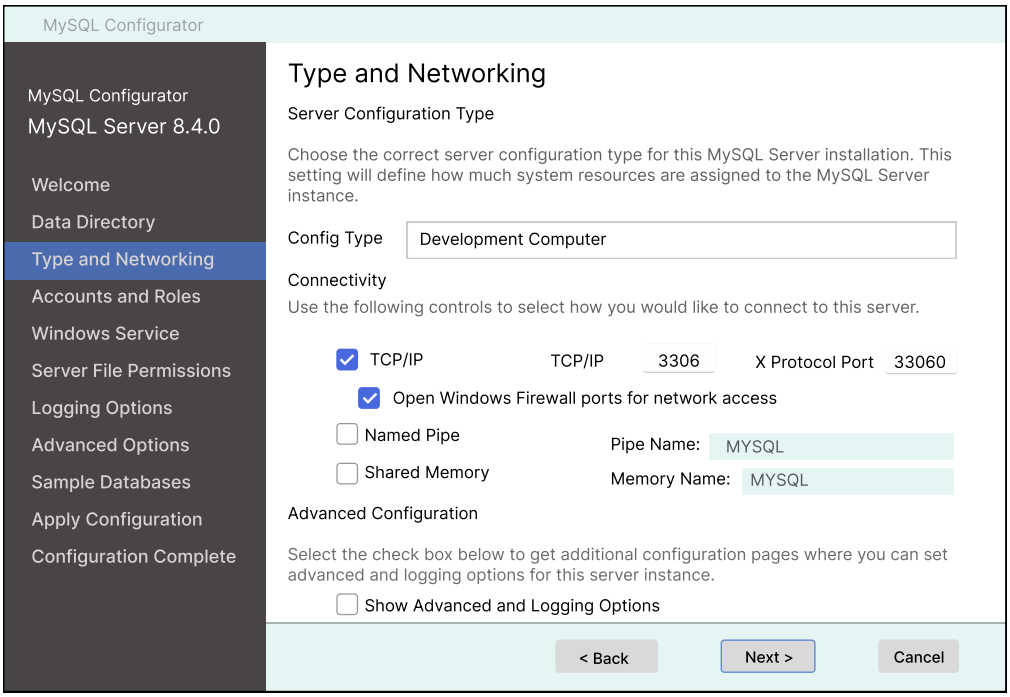

Figura 9. Selección del tipo de conexión.

- En la siguiente etapa de la configuración, se debe establecer una contraseña para el usuario `root`. Ingrese una contraseña segura y fácil de recordar según su criterio. Una vez definida, haga clic en `Next` para continuar.

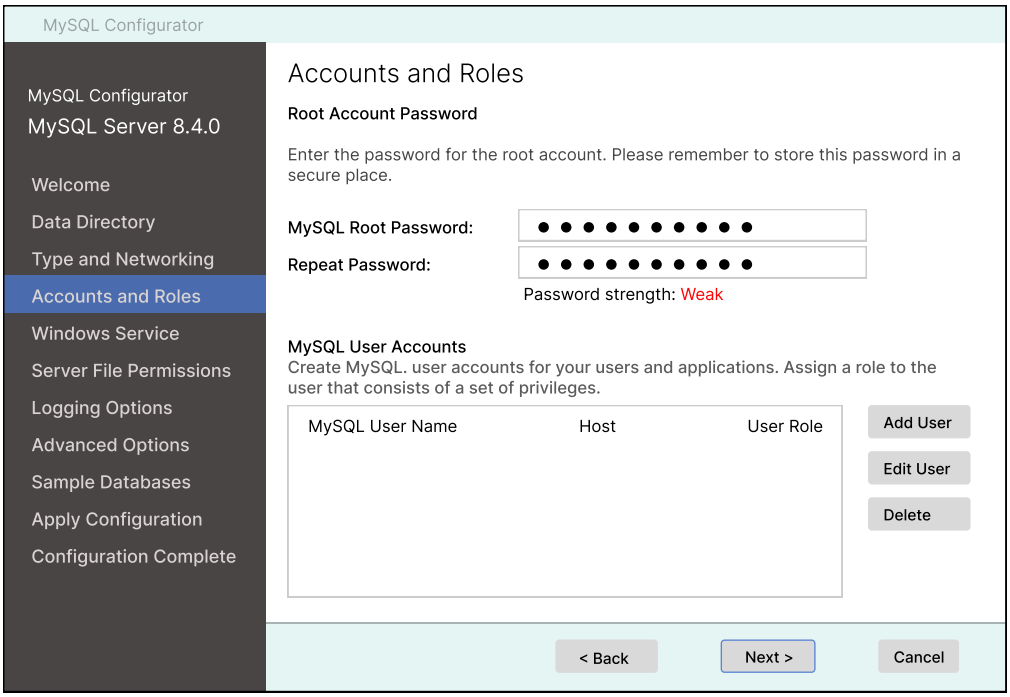

Figura 10. Creación de cuenta y contraseña.

La contraseña creada durante el proceso de instalación de MySQL es fundamental, ya que será la misma que se utilizará posteriormente al momento de establecer la conexión con la base de datos en la Subsección 6.16.2.

- Mantenga la configuración predeterminada y haga clic en `Next` para continuar con el proceso de instalación.

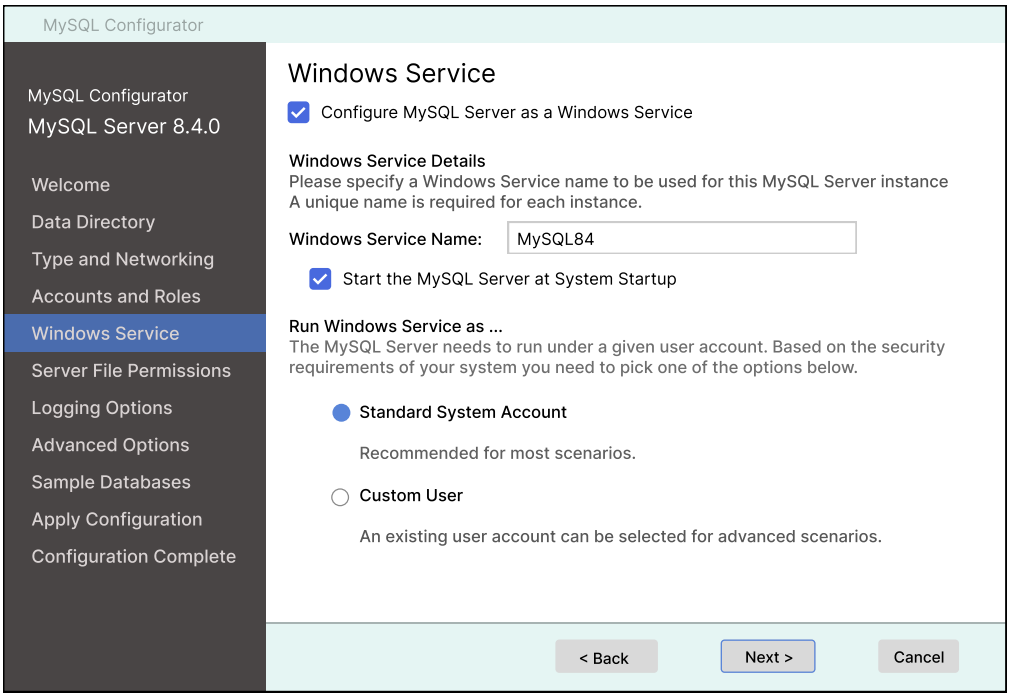

Figura 11. Configuración de MySQL como servicio de Windows.

- Al igual que en el paso anterior, mantenga la configuración predeterminada y haga clic en `Next` para continuar con el procedimiento.

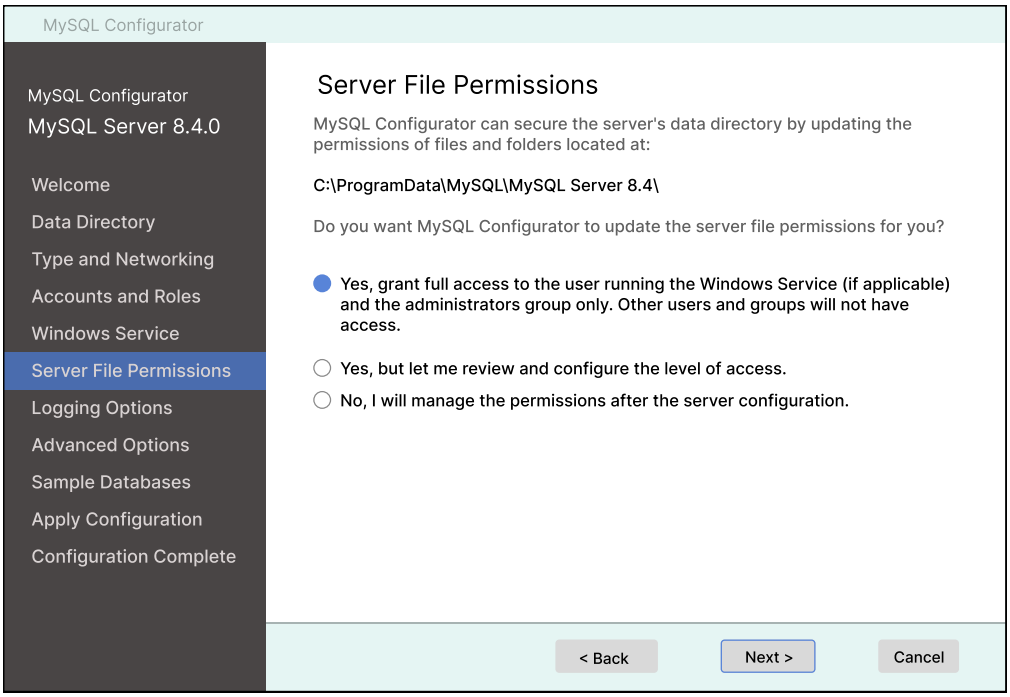

Figura 12. Permisos del servidor

- En esta etapa, se debe crear la base de datos. Para ello, seleccione la opción `Create World database` y haga clic en `Next` para continuar.

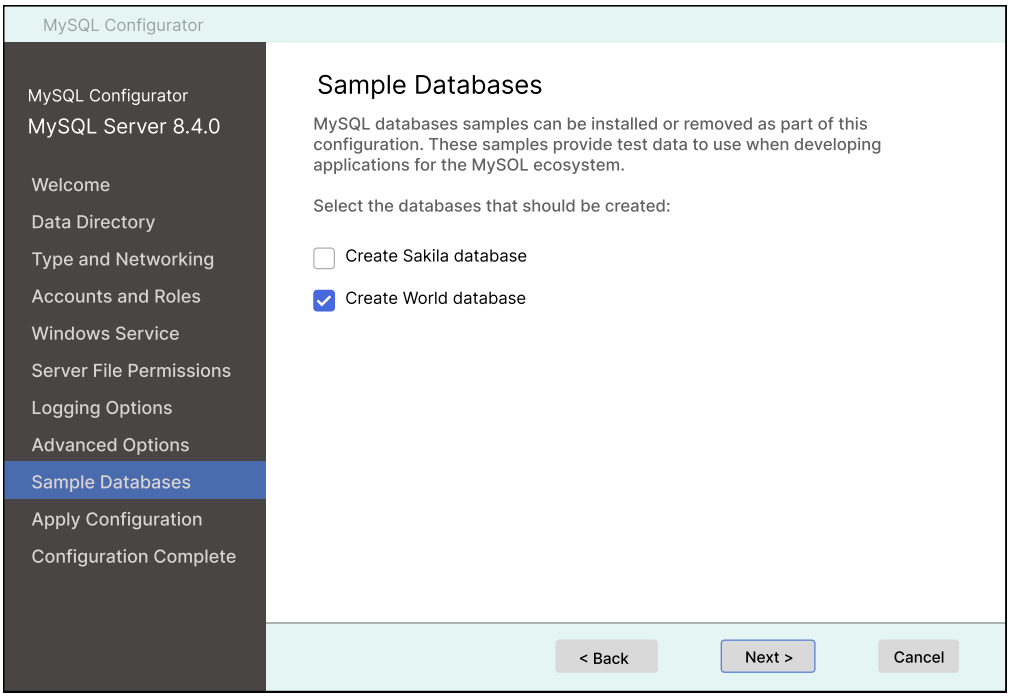

Figura 13. Bases de datos de muestra.

- Para que la instalación y configuración continúen correctamente, simplemente haga clic en el botón `Next`.

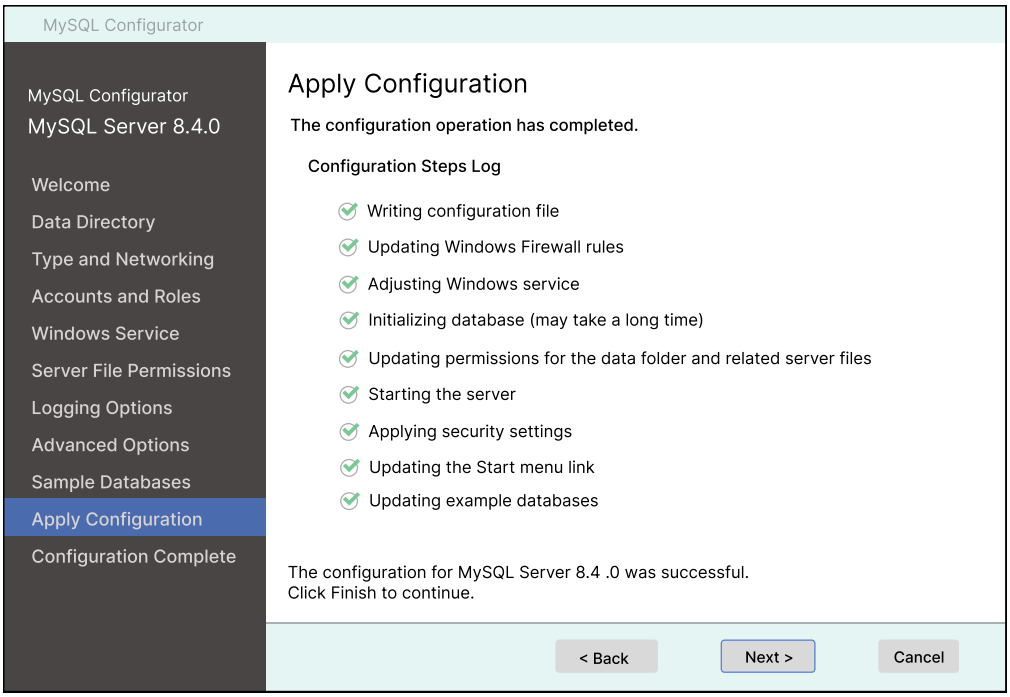

Figura 14. Aplicar configuración.

- Como último paso, haga clic en la opción `Next` para finalizar el proceso de instalación.

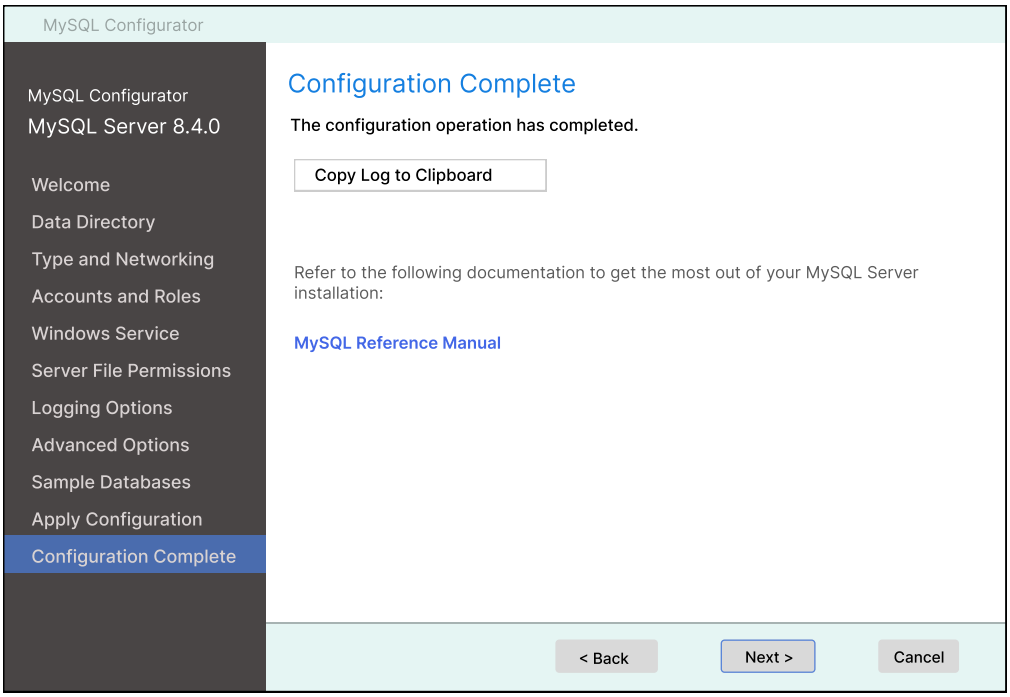

Figura 15. Configuración completa.

Con este procedimiento, se ha completado la instalación del servidor de bases de datos MySQL. A partir de este momento, el servidor está habilitado para aceptar conexiones desde la línea de comandos mediante el cliente de MySQL.

6.5. Administración de la base de datos

En este capítulo se explica el proceso de creación y eliminación de bases de datos en MySQL mediante el uso de sentencias SQL. Además, se presenta la sintaxis correspondiente, junto con ejemplos prácticos y consideraciones clave para garantizar una gestión adecuada de las bases de datos dentro del sistema[22].

6.5.1. Creación de la base de datos

[22] (W3schools.: *SQL Tutorial*. Recuperado el 6 de abril del 2025. `https://www.w3schools.com/sql/default.asp`)

El comando `CREATE DATABASE` se utiliza para crear una base de datos. Esta base de datos puede almacenar varias tablas, que contienen los datos relacionados.

La estructura básica para crear una base de datos es la siguiente:

```
CREATE DATABASE nombre_de_la_base_de_datos;
```

Este comando creará una nueva base de datos con el nombre especificado. Es importante tener en cuenta que el nombre de la base de datos debe ser único dentro del sistema de bases de datos.

Ejemplo. 1. Suponga que desea crear una base de datos para almacenar información de una empresa. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE DATABASE empresa;
```

El resultado de este comando es la creación de una nueva base de datos llamada `empresa`.

6.5.2. Eliminación de la base de datos

El comando `DROP DATABASE` se utiliza para eliminar por completo una base de datos existente, junto con todas las tablas, datos y objetos que contiene.

La estructura básica de esta sentencia es la siguiente:

```
DROP DATABASE nombre_de_la_base_de_datos;
```

Ejemplo. 2. Suponga que desea eliminar una base de datos llamada `empresa`. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
DROP DATABASE empresa;
```

El resultado de este comando es la eliminación definitiva de la base de datos llamada `empresa`.

6.6. Administración de tablas

En este capítulo se explica la creación y la administración de tablas en MySQL. Las tablas son uno de los elementos más importantes en una base de datos relacional; ya que en ellas se almacenan y se organizan los datos en forma de filas y columnas. A continuación, se muestran las sentencias requeridas para la creación, administración, modificación y eliminación de tablas, filas y columnas seguida de ejemplos[23].

6.6.1. Creación de tablas

El comando `CREATE TABLE` se utiliza para definir una nueva tabla dentro de una base de datos.

La estructura básica de esta sentencia es la siguiente:

```
CREATE TABLE nombre_de_la_tabla ();
```

Este comando crea una tabla sin columnas definidas.

Si se desea crear una tabla con columnas definidas, la estructura básica de la sentencia es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    nombre_columna1 tipo_de_dato,
    nombre_columna2 tipo_de_dato,
```

[23] (MySQL Documentation Team: *MySQL 8.0 Reference Manual: CREATE TABLE Statement*. Recuperado el 26 de marzo de 2025. 2023. `https://dev.mysql.com/doc/refman/8.0/en/create-table.html`)

```
    ...
);
```

En este caso, dentro de los paréntesis se especifica una lista de columnas. Cada columna incluye un nombre identificador y el tipo de dato que almacenará. Es posible añadir tantas columnas como se necesiten, separadas por comas.

Ejemplo. 3. Suponga que desea crear una tabla completamente vacía llamada `empleados`, sin definir inicialmente las columnas. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE TABLE empleados ();
```

El resultado de este comando es:

Tabla `empleados` **vacía**
(Sin columnas)

Tabla 1. Tabla `empleados` vacía sin columnas.

Ejemplo. 4. Suponga que desea crear una tabla llamada `clientes`, con tres columnas: `id`, para identificar al cliente; `nombre`, para almacenar el nombre del cliente, y `correo`, para su dirección de correo electrónico. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE TABLE clientes (
    id INT,
    nombre VARCHAR(50),
    correo VARCHAR(100)
);
```

El resultado de este comando es:

id	nombre	correo
INT	VARCHAR(50)	VARCHAR(100)

Tabla 2. Tabla `clientes` con tres columnas definidas.

6.6.2. Restricciones

Las restricciones son reglas que se aplican a las columnas de una tabla para controlar los valores que pueden ser insertados en ellas. Estas restricciones garantizan que los datos almacenados en la base de datos sean válidos y cumplan con ciertas condiciones o requisitos. Al utilizar restricciones, se garantiza la integridad de los datos y se previenen errores o inconsistencias en la base de datos.

6.6.2.1. NOT NULL

La restricción `NOT NULL` se utiliza para garantizar que una columna no pueda contener valores vacíos o nulos. Esto significa que, para cualquier fila de la tabla, se debe insertar un valor válido en la columna con esta restricción. Si no se especifica un valor para la columna en una operación de inserción, la base de datos lanzará un error[24].

La estructura básica es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato NOT NULL,
    columna2 tipo_de_dato,
```

[24] (MySQL Documentation Team: *MySQL 8.0 Reference Manual: Table Constraints*. Recuperado el 26 de marzo de 2025. 2023. `https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html`)

```
    ...
);
```

En este caso, la columna `columna1` debe recibir un valor en cada inserción de datos, mientras que `columna2` puede contener valores nulos.

Ejemplo. 5. Suponga que desea crear una tabla llamada `empleados`, donde el nombre del empleado sea obligatorio, es decir, no puede quedar en blanco o nulo. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE TABLE empleados (
    id INT,
    nombre VARCHAR(100) NOT NULL,
    cargo VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

Este comando crea la tabla `empleados`, donde la columna `nombre` es obligatoria, mientras que las demás columnas pueden aceptar valores nulos.

6.6.2.2. PRIMARY KEY

La restricción `PRIMARY KEY` se utiliza para identificar de manera única cada registro en una tabla. Una columna o un conjunto de columnas designado como clave primaria debe cumplir dos condiciones: los valores deben ser únicos y no nulos. En otras palabras, no puede haber dos filas con el mismo valor en la columna `PRIMARY KEY`, y ninguna fila puede tener un valor nulo en esa columna.

La restricción `PRIMARY KEY` puede aplicarse a una columna o a un conjunto de columnas. La estructura básica es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato PRIMARY KEY,
    columna2 tipo_de_dato,
    ...
);
```

En este caso, la columna `columna1` es la clave primaria, lo que asegura que cada valor en esa columna sea único y no nulo.

Ejemplo. 6. Suponga que desea crear una tabla llamada `empleados`, donde el número de identificación (`id`) de cada empleado sea único y obligatorio. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(100),
    cargo VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

Este comando crea una tabla `empleados` donde la columna `id` es la clave primaria. Esto garantiza que cada valor de `id` sea único y que no pueda estar vacío (nulo).

6.6.2.3. UNIQUE

La restricción `UNIQUE` se utiliza para garantizar que todos los valores en una columna o un conjunto de columnas sean únicos. Esto significa que no puede haber dos filas con el mismo valor en una columna que tenga la restricción `UNIQUE`. Esta restricción es útil para asegurar que los datos no se dupliquen, por ejemplo, en campos como correos electrónicos o números de identificación.

La estructura básica para utilizar UNIQUE en una consulta es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato UNIQUE,
    columna2 tipo_de_dato,
    ...
);
```

Ejemplo. 7. Suponga que desea crear una tabla llamada empleados, donde el número de identificación del empleado debe ser único para cada registro. A continuación, se muestra el código necesario para ejecutar este ejemplo:

```
CREATE TABLE empleados (
    id INT UNIQUE,
    nombre VARCHAR(100),
    cargo VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

Este comando crea una tabla empleados en la cual la columna id debe contener valores únicos, lo que significa que no puede haber dos empleados con el mismo id.

6.6.2.4. CHECK

La restricción CHECK se utiliza para asegurar que los valores insertados en una columna cumplan con una condición específica. Por ejemplo, se puede utilizar para restringir que los valores de una columna numérica estén dentro de un cierto rango, o que los valores de una columna de texto pertenezcan a un conjunto específico de opciones.

La estructura básica para utilizar CHECK en una consulta es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato,
    columna2 tipo_de_dato,
    ...
    CONSTRAINT nombre_de_la_restriccion CHECK (condicion)
);
```

Ejemplo. 8. Suponga que se desea crear una tabla llamada `empleados` en la cual los valores de la columna `salario` deban estar en un rango entre 1000 y 5000.

La siguiente tabla muestra algunos datos de ejemplo que cumplen con esta condición:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500
2	María Gómez	Secretaria	1800
3	Carlos López	Contador	2500

Tabla 3. Ejemplo de datos para la tabla `empleados` con salario entre 1000 y 5000.

A continuación, se muestra el código necesario para crear la tabla con la restricción deseada:

```
CREATE TABLE empleados (
    id INT,
    nombre VARCHAR(100),
    cargo VARCHAR(50),
    salario DECIMAL(10, 2),
    CONSTRAINT salario_rango CHECK (salario >= 1000 AND salario <= 5000)
);
```

El resultado de este comando es la creación de una tabla donde los valores insertados

en la columna `salario` deben cumplir con la condición establecida. Si se intenta insertar un salario fuera de este rango, el sistema rechazará la operación.

6.6.2.5. DEFAULT

La restricción `DEFAULT` se utiliza para asignar un valor predeterminado a una columna en caso de que no se especifique un valor al insertar un nuevo registro. Esto asegura que, si no se proporciona un valor para una columna, se utilizará un valor por defecto previamente definido.

La estructura básica para utilizar `DEFAULT` en una consulta es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato DEFAULT valor_por_defecto,
    columna2 tipo_de_dato,
    ...
);
```

Ejemplo. 9. Suponga que se desea crear una tabla llamada `empleados`, en la cual la columna `cargo` tenga como valor predeterminado “Empleado” cuando no se especifique uno explícitamente.

La siguiente tabla muestra algunos empleados registrados:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 4. Tabla `empleados` con valor por defecto en la columna `cargo`.

A continuación, se muestra el código necesario para crear esta tabla:

```
CREATE TABLE empleados (
    id INT,
    nombre VARCHAR(100),
    cargo VARCHAR(50) DEFAULT 'Empleado',
    salario DECIMAL(10, 2)
);
```

El resultado de este comando es la creación de una tabla `empleados` donde la columna `cargo` tomará automáticamente el valor "Empleado" si no se proporciona uno al momento de insertar un registro.

Ejemplo. 10. Si se inserta un registro sin especificar un valor para la columna `cargo`, se aplicará automáticamente el valor predeterminado. Por ejemplo:

```
INSERT INTO empleados (id, nombre, salario)
VALUES (4, 'Ana Fernández', 2200.00);
```

En este caso, la columna `cargo` tomará el valor "Empleado" para el registro correspondiente a `Ana Fernández`.

6.6.2.6. AUTO_INCREMENT

La restricción `AUTO_INCREMENT` se utiliza para generar automáticamente un valor único para una columna cada vez que se inserta un nuevo registro en la tabla. Esta restricción es comúnmente utilizada en columnas de tipo `id` para identificar de manera única a cada registro sin necesidad de que el usuario proporcione un valor explícito.

La estructura básica para utilizar `AUTO_INCREMENT` en una consulta es la siguiente:

```
CREATE TABLE nombre_de_la_tabla (
    columna1 tipo_de_dato AUTO_INCREMENT,
    columna2 tipo_de_dato,
    ...
    PRIMARY KEY (columna1)
);
```

Ejemplo. 11. Suponga que se desea crear una tabla llamada `empleados`, en la cual la columna `id` se incremente automáticamente con cada nuevo registro insertado. Esta funcionalidad permite asignar un identificador único sin necesidad de especificarlo manualmente.

La siguiente tabla muestra algunos registros de ejemplo:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 5. Tabla `empleados` con identificador auto-incrementable.

A continuación, se muestra el código necesario para crear esta tabla:

```
CREATE TABLE empleados (
    id INT AUTO_INCREMENT,
    nombre VARCHAR(100),
    cargo VARCHAR(50),
    salario DECIMAL(10, 2),
    PRIMARY KEY (id)
);
```

El resultado de esta consulta es la creación de una tabla donde la columna `id` se incrementará automáticamente cada vez que se inserte un nuevo registro, garantizando que cada empleado tenga un identificador único.

6.6.3. Eliminación de tablas

El comando `DROP TABLE` se utiliza para eliminar una tabla existente, junto con todos los datos almacenados en ella. Una vez que una tabla es eliminada, los datos no se pueden recuperar, por lo que debe utilizarse con precaución.

La estructura básica para eliminar una tabla es la siguiente:

```
DROP TABLE nombre_de_la_tabla;
```

Ejemplo. 12. Suponga que se tiene una tabla llamada `empleados`:

Si desea eliminar la tabla `empleados`, puede utilizar el siguiente comando:

```
DROP TABLE empleados;
```

Este comando eliminará la tabla `empleados` junto con todos los datos que contiene.

6.6.4. Modificación de tablas

6.6.4.1. Añadir columnas

El comando `ALTER TABLE` se utiliza para modificar una tabla existente, permitiendo agregar, eliminar o modificar columnas. Cuando se utiliza para añadir una columna, se puede especificar el nombre de la nueva columna y su tipo de dato.

La estructura básica para añadir una columna es la siguiente:

```
ALTER TABLE nombre_de_la_tabla
ADD COLUMN nombre_columna tipo_de_dato;
```

Ejemplo. 13. Suponga que se tiene la siguiente tabla de `empleados`:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 6. Tabla de empleados.

Si se desea añadir una nueva columna llamada `fecha_ingreso` para almacenar la fecha en la que cada empleado fue contratado, se puede utilizar la siguiente consulta:

```
ALTER TABLE empleados
ADD COLUMN fecha_ingreso DATE;
```

El resultado de esta consulta es:

id	nombre	cargo	salario	fecha_ingreso
1	Juan Pérez	Gerente	3500.00	*NULL*
2	María Gómez	Secretaria	1800.00	*NULL*
3	Carlos López	Contador	2500.00	*NULL*

Tabla 7. Tabla de empleados con la nueva columna `fecha_ingreso`.

Ejemplo. 14. Si se desea añadir una nueva columna en una posición específica dentro de la tabla, se puede utilizar la opción `AFTER`. Por ejemplo, si se quiere que la nueva columna `fecha_ingreso` aparezca después de la columna `nombre`, se puede hacer de la siguiente manera:

```
ALTER TABLE empleados
ADD COLUMN fecha_ingreso DATE AFTER nombre;
```

Esto colocará la columna `fecha_ingreso` inmediatamente después de la columna `nombre` en la tabla.

6.6.4.2. Renombrar columnas

El comando `ALTER TABLE` en MySQL también se utiliza para renombrar una columna existente en una tabla. Este comando permite cambiar el nombre de una columna manteniendo su tipo de dato y los datos existentes.

La estructura básica para renombrar una columna es la siguiente:

```
ALTER TABLE nombre_de_la_tabla
RENAME COLUMN nombre_actual TO nuevo_nombre;
```

Ejemplo. 15. Suponga que se tiene la siguiente tabla de `empleados`:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 8. Tabla de empleados.

Si se desea renombrar la columna `cargo` a `puesto`, se puede utilizar la siguiente consulta:

```
ALTER TABLE empleados
RENAME COLUMN cargo TO puesto;
```

El resultado de esta consulta es:

id	nombre	puesto	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 9. Tabla de empleados con la columna `cargo` renombrada a `puesto`.

6.6.4.3. Modificar columnas

El comando `ALTER TABLE` con la opción `MODIFY COLUMN` en MySQL se utiliza para modificar el tipo de dato o las propiedades de una columna existente en una tabla, sin cambiar su nombre. Este comando es útil cuando se necesita aumentar el tamaño de una columna, cambiar su tipo de dato o agregar una restricción como `NOT NULL`.

La estructura básica para modificar una columna es la siguiente:

```
ALTER TABLE nombre_de_la_tabla
MODIFY COLUMN nombre_columna tipo_de_dato;
```

Ejemplo. 16. Suponga que se tiene la siguiente tabla de `empleados`:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 10. Tabla de empleados.

Si se desea modificar la columna `salario` para aumentar el tamaño y permitir más decimales, se puede utilizar la siguiente consulta:

```
ALTER TABLE empleados
MODIFY COLUMN salario DECIMAL(12, 4);
```

El resultado de esta consulta es:

id	nombre	cargo	salario (DECIMAL(12,4))
1	Juan Pérez	Gerente	3500.0000
2	María Gómez	Secretaria	1800.0000
3	Carlos López	Contador	2500.0000

Tabla 11. Tabla de empleados con la columna `salario` modificada.

Donde la columna `salario` permite un total de 12 dígitos, de los cuales 4 son decimales.

6.6.4.4. Eliminar columnas

El comando `ALTER TABLE` con la opción `DROP COLUMN` en MySQL se utiliza para eliminar una columna existente de una tabla. Esta acción es irreversible, por lo que al eliminar una columna, se perderán todos los datos que esta contenía.

La estructura básica para eliminar una columna es la siguiente:

```
ALTER TABLE nombre_de_la_tabla
DROP COLUMN nombre_columna;
```

Ejemplo. 17. Suponga que se tiene la siguiente tabla de `empleados`:

id	nombre	cargo	salario
1	Juan Pérez	Gerente	3500.00
2	María Gómez	Secretaria	1800.00
3	Carlos López	Contador	2500.00

Tabla 12. Tabla de empleados.

Si se desea eliminar la columna `cargo`, se puede utilizar la siguiente consulta:

```
ALTER TABLE empleados
DROP COLUMN cargo;
```

El resultado de esta consulta es:

id	nombre	salario
1	Juan Pérez	3500.00
2	María Gómez	1800.00
3	Carlos López	2500.00

Tabla 13. Tabla de empleados sin la columna `cargo`.

Donde la columna `cargo` ha sido eliminada.

6.7. Consulta de datos con el comando `SELECT`

En este capítulo se explica cómo consultar datos específicos de una base de datos mediante el uso del comando `SELECT`. Esta instrucción es uno de los elementos más fundamentales del lenguaje SQL, ya que permite recuperar la información almacenada en una o varias tablas, según las necesidades del usuario.

El comando `SELECT` se utiliza para indicar qué columnas o expresiones se desean obtener

como resultado de una consulta. Es la parte de la instrucción SQL que define los datos que se mostrarán, ya sea de una tabla completa o únicamente de algunas columnas específicas[25].

La estructura general del comando es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla;
```

En esta estructura, la cláusula `SELECT` indica las columnas que se desean recuperar, mientras que la cláusula `FROM` especifica la tabla desde la cual se obtendrán los datos.

También es posible utilizar el asterisco $*$ para seleccionar todas las columnas disponibles en la tabla:

```
SELECT *
FROM nombre_de_la_tabla;
```

Esta forma es útil cuando se desea recuperar todos los datos sin necesidad de escribir cada nombre de columna, aunque se recomienda usarla solo cuando sea necesario, ya que puede afectar el rendimiento en bases de datos grandes.

Asignación de alias con `AS`

La palabra clave `AS` se utiliza para asignar un alias o nombre alternativo a una columna o expresión en los resultados de una consulta. Esto permite mejorar la legibilidad del resultado, especialmente cuando se usan expresiones o funciones.

Por ejemplo, si se desea mostrar el resultado de una multiplicación con un encabezado

---

[25] (MySQL Documentation Team: *MySQL 8.0 Reference Manual: SELECT Statement*. Recuperado el 26 de marzo de 2025. 2023. `https://dev.mysql.com/doc/refman/8.0/en/select.html`)

personalizado, se puede hacer lo siguiente:

```
SELECT precio * cantidad AS total
FROM ventas;
```

En este caso, la columna calculada aparecerá en el resultado con el nombre `total`, en lugar de mostrar directamente la expresión `precio * cantidad`.

El comando `SELECT` es la base de cualquier operación de lectura en una base de datos. Su uso permite visualizar datos, generar reportes y preparar información para su análisis.

6.8. Comandos y funciones clave para consultas SQL

En este capítulo se explican los principales elementos que se utilizan al construir consultas SQL, tales como **modificadores**, **funciones**, **operadores** y **estructuras condicionales**. Estos elementos permiten realizar consultas más precisas, eficientes y adaptadas a distintos requerimientos de análisis de datos.

- **Modificadores:** son palabras clave que alteran el comportamiento de una consulta. Ejemplos: `DISTINCT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`.
- **Funciones de agregación:** permiten realizar cálculos sobre un conjunto de valores. Ejemplos: `SUM`, `AVG`, `MIN`, `MAX`, `COUNT`.
- **Funciones de manejo de datos:** procesan o transforman los datos dentro de una consulta. Ejemplos: `CONCAT`, `IFNULL`.
- **Operadores lógicos:** se utilizan para establecer condiciones en las consultas. Ejemplos: `AND`, `OR`, `NOT`, `IN`, `BETWEEN`, `LIKE`.
- **Estructuras condicionales:** permiten definir lógica dentro de una consulta para

retornar valores específicos según condiciones[26]. Ejemplo: `CASE`.

Además, se presentan mediante un listado ejemplos prácticos de cada modificador que ilustran su uso en distintos contextos.

6.8.1. DISTINCT

El modificador `DISTINCT` se utiliza para eliminar duplicados en los resultados de una consulta. Esto es útil cuando solo se desea recuperar valores únicos de una columna o un conjunto de columnas.

La estructura básica para utilizar `DISTINCT` en una consulta es la siguiente:

```
SELECT DISTINCT columna1, columna2, ...
FROM nombre_de_la_tabla;
```

Ejemplo. 18. Suponga que se tiene la siguiente tabla de `usuarios`, donde varios usuarios pueden residir en la misma ciudad:

nombre_usuario	ciudad
Juan Pérez	Madrid
María Gómez	Barcelona
Luis Rodríguez	Madrid
Ana Fernández	Sevilla
Carlos López	Barcelona

Tabla 14. Tabla de usuarios.

[26] MySQL Documentation Team. MySQL 8.0 Reference Manual: SELECT Syntax - Modifiers and Clauses. Recuperado el 26 de marzo de 2025. 2023. URL: https://dev.mysql.com/doc/refman/8.0/en/ select.html

Si se quiere obtener una lista de las diferentes ciudades donde residen los usuarios, sin incluir duplicados, se utilizaría la siguiente consulta con `DISTINCT`:

```
SELECT DISTINCT ciudad
FROM usuarios;
```

El resultado de esta consulta es:

ciudad
Madrid
Barcelona
Sevilla

Tabla 15. Ciudades distintas donde residen los usuarios.

Ejemplo. 19. Si se desea obtener los nombres de usuarios y sus respectivas ciudades, asegurándose de que no se repitan las combinaciones de nombre de usuario y ciudad, la consulta es:

```
SELECT DISTINCT nombre_usuario, ciudad
FROM usuarios;
```

El resultado de esta consulta es:

nombre_usuario	ciudad
Juan Pérez	Madrid
María Gómez	Barcelona
Luis Rodríguez	Madrid
Ana Fernández	Sevilla
Carlos López	Barcelona

Tabla 16. Nombres de usuarios y ciudades sin duplicar combinaciones.

6.8.2. WHERE

El modificador `WHERE` se utiliza para establecer una condición que los registros deben cumplir para ser incluidos en el resultado de una consulta. Permite filtrar los datos y trabajar únicamente con aquellos que satisfacen ciertos criterios, mejorando así la precisión y relevancia de la información obtenida.

La estructura básica para utilizar `WHERE` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE condicion;
```

Ejemplo. 20. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 17. Tabla de usuarios.

Si se quiere obtener una lista de los usuarios que viven en la ciudad de `Madrid`, se utiliza la siguiente consulta con `WHERE`:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE ciudad = 'Madrid';
```

El resultado de esta consulta es:

nombre_usuario	ciudad
Juan Pérez	Madrid
Luis Rodríguez	Madrid

Tabla 18. Usuarios que viven en Madrid.

Ejemplo. 21. Si se desea obtener una lista de los usuarios cuya edad es mayor a 30 años, se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, edad
FROM usuarios
WHERE edad > 30;
```

El resultado de esta consulta es:

nombre_usuario	edad
Carlos López	35

Tabla 19. Usuarios mayores de 30 años.

6.8.3. ORDER BY

El modificador `ORDER BY` se utiliza para ordenar los resultados de una consulta en un orden específico. Puedes ordenar los datos de forma ascendente o descendente, según una o más columnas.

La estructura básica para utilizar `ORDER BY` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
ORDER BY columna1 [ASC|DESC], columna2 [ASC|DESC], ...;
```

Ejemplo. 22. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 20. Tabla de usuarios.

Si se quiere obtener una lista de los usuarios ordenados por la columna `ciudad` en orden ascendente (`ASC`), se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad
FROM usuarios
ORDER BY ciudad ASC;
```

El resultado de esta consulta es:

nombre_usuario	ciudad
María Gómez	Barcelona
Carlos López	Barcelona
Juan Pérez	Madrid
Luis Rodríguez	Madrid
Ana Fernández	Sevilla

Tabla 21. Usuarios ordenados por ciudad en orden ascendente.

Ejemplo. 23. Si se quiere ordenar a los usuarios por la columna `edad` en orden descendente (`DESC`), la consulta es:

```sql
SELECT nombre_usuario, edad
FROM usuarios
ORDER BY edad DESC;
```

El resultado de esta consulta es:

nombre_usuario	edad
Carlos López	35
María Gómez	30
Ana Fernández	28
Juan Pérez	25
Luis Rodríguez	22

Tabla 22. Usuarios ordenados por edad en orden descendente.

6.8.4. LIKE

El operador lógico `LIKE` se utiliza para buscar patrones dentro de una columna de texto. Es especialmente útil cuando se necesita encontrar registros que coincidan con un criterio parcial en lugar de un valor exacto.

La estructura básica para utilizar `LIKE` en una consulta es la siguiente:

```sql
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE columna1 LIKE patron;
```

El `patrón` puede incluir los caracteres comodín `%` y `_`:

- `%`: Coincide con cero o más caracteres.
- `_`: Coincide con exactamente un carácter.

Ejemplo. 24. Supongamos que tenemos la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 23. Tabla de usuarios.

Si se quiere obtener una lista de los usuarios cuyos nombres comienzan con la letra “M”, se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE nombre_usuario LIKE 'M%';
```

El resultado de esta consulta es:

nombre_usuario	ciudad
María Gómez	Barcelona

Tabla 24. Usuarios cuyo nombre empieza con “M”.

Ejemplo. 25. Si se desea obtener una lista de los usuarios cuyos nombres contienen la letra “a” en cualquier posición, se utiliza la siguiente consulta:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE nombre_usuario LIKE '%a%';
```

El resultado de esta consulta es:

nombre_usuario	ciudad
Juan Pérez	Madrid
María Gómez	Barcelona
Ana Fernández	Sevilla
Carlos López	Barcelona

Tabla 25. Usuarios cuyos nombres contienen la letra “a”.

Ejemplo. 26. Si se quiere obtener una lista de los usuarios cuyos nombres tienen exactamente 4 letras, se puede utilizar el comodín _ que representa un solo carácter:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE nombre_usuario LIKE '____';
```

El resultado de esta consulta es una tabla vacía.

6.8.5. AND

El operador lógico `AND` se utiliza para combinar múltiples condiciones en una consulta. Todas las condiciones combinadas con `AND` deben ser verdaderas para que un registro sea incluido en el resultado.

La estructura básica para utilizar `AND` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE condicion1 AND condicion2 AND ...;
```

Ejemplo. 27. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 26. Tabla de usuarios.

Si se desea obtener una lista de los usuarios que residen en la ciudad de `Madrid` y tienen menos de 30 años, se puede utilizar la siguiente consulta con `AND`:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
WHERE ciudad = 'Madrid' AND edad < 30;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
Luis Rodríguez	Madrid	22

Tabla 27. Usuarios que viven en Madrid y tienen menos de 30 años.

Ejemplo. 28. Si se busca obtener una lista de los usuarios que residen en `Barcelona` y tienen más de 30 años, la consulta es la siguiente:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
WHERE ciudad = 'Barcelona' AND edad > 30;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Carlos López	Barcelona	35

Tabla 28. Usuarios que viven en Barcelona y tienen más de 30 años.

6.8.6. OR

El operador lógico `OR` se utiliza para combinar múltiples condiciones en una consulta, donde se desea que al menos una de las condiciones sea verdadera para que un registro sea incluido en el resultado. A diferencia de `AND`, que requiere que todas las condiciones sean verdaderas, `OR` solo requiere que una de las condiciones sea verdadera.

La estructura básica para utilizar `OR` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE condicion1 OR condicion2 OR ...;
```

Ejemplo. 29. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 29. Tabla de usuarios.

Si se desea obtener una lista de los usuarios que residen en la ciudad de `Madrid` o que tienen más de 30 años, se puede utilizar la siguiente consulta con `OR`:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
WHERE ciudad = 'Madrid' OR edad > 30;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
Luis Rodríguez	Madrid	22
Carlos López	Barcelona	35

Tabla 30. Usuarios que viven en Madrid o tienen más de 30 años.

Ejemplo. 30. Si se busca obtener una lista de los usuarios que residen en `Sevilla` o que tienen menos de 30 años, se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
WHERE ciudad = 'Sevilla' OR edad < 30;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28

Tabla 31. Usuarios que viven en Sevilla o tienen menos de 30 años.

6.8.7. NOT

El operador lógico `NOT` se utiliza para negar una condición en una consulta. Permite filtrar los resultados para excluir aquellos que cumplen con la condición especificada.

La estructura básica para utilizar `NOT` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE NOT condicion;
```

Ejemplo. 31. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 32. Tabla de usuarios.

Si se desea obtener una lista de los usuarios que **no** viven en la ciudad de `Madrid`, se puede utilizar la siguiente consulta con `NOT`:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE NOT ciudad = 'Madrid';
```

El resultado de esta consulta es:

nombre_usuario	ciudad
María Gómez	Barcelona
Ana Fernández	Sevilla
Carlos López	Barcelona

Tabla 33. Usuarios que no viven en Madrid.

Ejemplo. 32. Si se busca obtener una lista de los usuarios que **no** tienen más de 30 años, se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
WHERE NOT edad > 30;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28

Tabla 34. Usuarios que no tienen más de 30 años.

6.8.8. LIMIT

El modificador `LIMIT` se utiliza para restringir el número de filas que se devuelven en el resultado de una consulta. Es útil para obtener un subconjunto específico de datos, especialmente cuando se trabaja con grandes conjuntos de datos o se desea paginar los resultados.

La estructura básica para utilizar `LIMIT` en una consulta es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
LIMIT numero_de_filas;
```

Ejemplo. 33. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 35. Tabla de usuarios.

Si se desea obtener solo los primeros 3 usuarios de la tabla `usuarios`, se puede utilizar la siguiente consulta con `LIMIT`:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
LIMIT 3;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22

Tabla 36. Primeros 3 usuarios de la tabla.

Ejemplo. 34. Si se desea obtener el segundo y el tercer usuario de la tabla (omitiendo el primero), se puede utilizar la siguiente consulta, aplicando el modificador `LIMIT` con desplazamiento:

```
SELECT nombre_usuario, ciudad, edad
FROM usuarios
LIMIT 2 OFFSET 1;
```

El resultado de esta consulta es:

nombre_usuario	ciudad	edad
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22

Tabla 37. Segundo y tercer usuario de la tabla.

6.8.9. COMENTARIOS

En SQL, los comentarios se utilizan para incluir anotaciones o explicaciones dentro del código de consulta. Los comentarios no afectan la ejecución de la consulta y sirven para documentar el código, facilitar su comprensión y mantenerlo organizado.

Existen dos tipos principales de comentarios en SQL:

1. **Comentarios de una sola línea:** Se utilizan para agregar comentarios en una sola línea de código. Estos comentarios comienzan con dos guiones consecutivos `-`.
2. **Comentarios de varias líneas:** Se utilizan para agregar comentarios que abarcan múltiples líneas. Estos comentarios están delimitados por `/*` al inicio y `*/` al final.

Para agregar un comentario de una sola línea en SQL, se utilizan dos guiones seguidos. Todo el texto después de los guiones en la misma línea será tratado como un comentario. Por ejemplo:

```
-- Este es un comentario de una sola linea
SELECT nombre_usuario
FROM usuarios;
```

En este caso, el comentario `- Este es un comentario de una sola línea` no afectará la ejecución de la consulta y sirve únicamente para proporcionar una explicación o nota.

Para agregar comentarios que ocupen varias líneas, se utilizan los delimitadores `/*` y `*/`. Todo el texto entre estos delimitadores será tratado como un comentario. Por ejemplo:

```
/*
Este es un comentario de varias lineas.
Se puede extender por varias lineas y es util para proporcionar
descripciones detalladas o explicaciones extensas.
*/
SELECT nombre_usuario
FROM usuarios;
```

6.8.10. NULL

El valor especial `NULL` se utiliza para representar la ausencia de un valor en una columna. Un valor `NULL` indica que el dato es desconocido o no aplica, y no debe ser confundido con un valor vacío o cero.

Ejemplo. 35. Suponga que se tiene la siguiente tabla de `usuarios`, donde algunos usuarios no tienen registrada la ciudad en la que residen (es decir, el valor es `NULL`):

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	NULL	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	NULL	35

Tabla 38. Tabla de usuarios.

Si se desea obtener una lista de los usuarios cuya ciudad no está registrada (`NULL`), se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE ciudad IS NULL;
```

El resultado de esta consulta es:

nombre_usuario	ciudad
María Gómez	NULL
Carlos López	NULL

Tabla 39. Usuarios cuya ciudad es `NULL`.

Ejemplo. 36. Si se busca obtener una lista de los usuarios que tienen la ciudad registrada (es decir, la columna `ciudad` no es `NULL`), se puede utilizar la siguiente consulta:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE ciudad IS NOT NULL;
```

El resultado de esta consulta es:

nombre_usuario	ciudad
Juan Pérez	Madrid
Luis Rodríguez	Madrid
Ana Fernández	Sevilla

Tabla 40. Usuarios cuya ciudad no es `NULL`.

6.8.11. MIN

La función de agregación `MIN` se utiliza para obtener el valor mínimo de una columna en un conjunto de resultados. Es una función de agregación que devuelve el valor más bajo en el grupo de valores seleccionados.

La estructura básica para utilizar `MIN` en una consulta SQL es la siguiente:

```
SELECT MIN(columna)
FROM nombre_de_la_tabla;
```

Ejemplo. 37. Suponga que se tiene la siguiente tabla de `usuarios`, que contiene información sobre la edad de los usuarios:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 41. Tabla de usuarios.

Si se desea obtener la edad mínima entre todos los usuarios de la tabla `usuarios`, se puede utilizar la siguiente consulta con la función `MIN`:

```
SELECT MIN(edad) AS edad_minima
FROM usuarios;
```

El resultado de esta consulta es:

edad_minima
22

Tabla 42. Edad mínima de los usuarios.

Ejemplo. 38. Si se busca obtener el nombre del usuario con la edad mínima, se puede utilizar la función `MIN` en combinación con una subconsulta:

```
SELECT nombre_usuario, edad
FROM usuarios
WHERE edad = (SELECT MIN(edad) FROM usuarios);
```

El resultado de esta consulta es:

nombre_usuario	edad
Luis Rodríguez	22

Tabla 43. Usuario más joven.

6.8.12. MAX

La función de agregación `MAX` se utiliza para obtener el valor máximo de una columna en un conjunto de resultados. Es una función de agregación que devuelve el valor más alto dentro del grupo de valores seleccionados.

La estructura básica para utilizar `MAX` en una consulta SQL es la siguiente:

```
SELECT MAX(columna)
FROM nombre_de_la_tabla;
```

Ejemplo. 39. Suponga que se tiene la siguiente tabla de `usuarios`, que contiene información sobre la edad de los usuarios:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 44. Tabla de usuarios.

Si se desea obtener la edad máxima entre todos los usuarios de la tabla `usuarios`, se puede utilizar la siguiente consulta con la función `MAX`:

```
SELECT MAX(edad) AS edad_maxima
FROM usuarios;
```

El resultado de esta consulta es:

edad_maxima
35

Tabla 45. Edad máxima de los usuarios.

6.8.13. COUNT

La función de agregación `COUNT` se utiliza para contar el número de filas que cumplen con una condición en un conjunto de resultados. Es una función de agregación que devuelve el total de registros en una columna o en el conjunto de datos.

La estructura básica para utilizar `COUNT` en una consulta SQL es la siguiente:

```
SELECT COUNT(columna)
FROM nombre_de_la_tabla;
```

Ejemplo. 40. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 46. Tabla de usuarios.

Si se desea contar el número total de usuarios en la tabla `usuarios`, se puede utilizar la siguiente consulta con la función `COUNT`:

```
SELECT COUNT(*) AS total_usuarios
FROM usuarios;
```

El resultado de esta consulta es:

total_usuarios
5

Tabla 47. Número total de usuarios.

Ejemplo. 41. Si se desea contar el número de usuarios que viven en la ciudad de `Madrid`, se puede utilizar la función `COUNT` junto con una condición `WHERE`:

```
SELECT COUNT(*) AS usuarios_madrid
FROM usuarios
WHERE ciudad = 'Madrid';
```

El resultado de esta consulta es:

usuarios_madrid
2

Tabla 48. Número de usuarios que viven en Madrid.

6.8.14. SUM

La función de agregación `SUM` se utiliza para calcular la suma total de los valores en una columna específica de un conjunto de resultados. Es una función de agregación que devuelve el total acumulado de los valores numéricos de una columna.

La estructura básica para utilizar `SUM` en una consulta SQL es la siguiente:

```
SELECT SUM(columna)
FROM nombre_de_la_tabla;
```

Ejemplo. 42. Suponga que se tiene la siguiente tabla de `usuarios`, que contiene información sobre la edad de los usuarios:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 49. Tabla de usuarios.

Si se desea obtener la suma de las edades de todos los usuarios en la tabla `usuarios`, se puede utilizar la siguiente consulta con la función `SUM`:

```
SELECT SUM(edad) AS suma_edades
FROM usuarios;
```

El resultado de esta consulta es:

suma_edades
140

Tabla 50. Suma total de las edades de los usuarios.

Ejemplo. 43. Si se busca sumar la edad total de los usuarios que viven en `Madrid`, se puede utilizar la función `SUM` junto con una condición `WHERE`:

```
SELECT SUM(edad) AS suma_edades_madrid
FROM usuarios
WHERE ciudad = 'Madrid';
```

El resultado de esta consulta es:

suma_edades_madrid
47

Tabla 51. Suma total de las edades de los usuarios que viven en Madrid.

6.8.15. AVG

La función de agregación `AVG` se utiliza para calcular el valor promedio de una columna específica en un conjunto de resultados. Es una función de agregación que devuelve el promedio de los valores numéricos de una columna.

La estructura básica para utilizar `AVG` en una consulta SQL es la siguiente:

```
SELECT AVG(columna)
FROM nombre_de_la_tabla;
```

Ejemplo. 44. Suponga que se tiene la siguiente tabla de `usuarios`, que contiene información sobre la edad de los usuarios:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 52. Tabla de usuarios.

Si se desea calcular el promedio de las edades de todos los usuarios en la tabla `usuarios`, se puede utilizar la siguiente consulta con la función `AVG`:

```
SELECT AVG(edad) AS promedio_edad
FROM usuarios;
```

El resultado de esta consulta es:

promedio_edad
28

Tabla 53. Edad promedio de los usuarios.

Ejemplo. 45. Si se desea calcular el promedio de la edad de los usuarios que viven en `Madrid`, se puede utilizar la función `AVG` junto con una condición `WHERE`:

```
SELECT AVG(edad) AS promedio_edad_madrid
FROM usuarios
WHERE ciudad = 'Madrid';
```

El resultado de esta consulta es:

promedio_edad_madrid
23.5

Tabla 54. Edad promedio de los usuarios que viven en Madrid.

6.8.16. IN

El operador lógico `IN` se utiliza para filtrar los resultados de una consulta, verificando si el valor de una columna específica coincide con cualquiera de los valores de una lista. Este modificador simplifica la sintaxis cuando se desea comparar un valor con múltiples opciones.

La estructura básica para utilizar `IN` en una consulta SQL es la siguiente:

```
SELECT columnas
FROM nombre_de_la_tabla
WHERE columna IN (valor1, valor2, ...);
```

Ejemplo. 46. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 55. Tabla de usuarios.

Si se desea obtener una lista de los usuarios que viven en las ciudades de `Madrid` o `Barcelona`, se puede utilizar el operador `IN` de la siguiente manera:

```
SELECT nombre_usuario, ciudad
FROM usuarios
WHERE ciudad IN ('Madrid', 'Barcelona');
```

El resultado de esta consulta es:

nombre_usuario	ciudad
Juan Pérez	Madrid
Luis Rodríguez	Madrid
María Gómez	Barcelona
Carlos López	Barcelona

Tabla 56. Usuarios que viven en Madrid o Barcelona.

Ejemplo. 47. Si se desea obtener una lista de los usuarios cuya edad es exactamente 22, 28 o 35 años, se puede utilizar el operador `IN` de la siguiente manera:

```
SELECT nombre_usuario, edad
FROM usuarios
WHERE edad IN (22, 28, 35);
```

El resultado de esta consulta es:

nombre_usuario	edad
Luis Rodríguez	22
Ana Fernández	28
Carlos López	35

Tabla 57. Usuarios cuya edad es 22, 28 o 35 años.

6.8.17. BETWEEN

El operador lógico `BETWEEN` se utiliza para filtrar los resultados de una consulta, seleccionando los registros cuyo valor en una columna específica se encuentre dentro de un rango determinado.

La estructura básica para utilizar `BETWEEN` en una consulta SQL es la siguiente:

```
SELECT columnas
FROM nombre_de_la_tabla
WHERE columna BETWEEN valor_inicial AND valor_final;
```

Ejemplo. 48. Suponga que se tiene la siguiente tabla de `usuarios`:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 58. Tabla de usuarios.

Si se desea obtener una lista de los usuarios cuya edad esté comprendida entre 25 y 35 años, se puede utilizar el operador `BETWEEN` de la siguiente manera:

```
SELECT nombre_usuario, edad
FROM usuarios
WHERE edad BETWEEN 25 AND 35;
```

El resultado de esta consulta es:

nombre_usuario	edad
Juan Pérez	25
Ana Fernández	28
María Gómez	30
Carlos López	35

Tabla 59. Usuarios cuya edad está entre 25 y 35 años.

Ejemplo. 49. Si se desea obtener una lista de los usuarios cuya edad esté comprendida entre 20 y 30 años, se puede utilizar el operador `BETWEEN` de la siguiente manera:

```
SELECT nombre_usuario, edad
FROM usuarios
WHERE edad BETWEEN 20 AND 30;
```

El resultado de esta consulta es:

nombre_usuario	edad
Juan Pérez	25
María Gómez	30
Luis Rodríguez	22
Ana Fernández	28

Tabla 60. Usuarios cuya edad está entre 20 y 30 años.

6.8.18. CONCAT

La función de cadena `CONCAT` se utiliza para combinar dos o más cadenas de texto en una sola.

La estructura básica para utilizar `CONCAT` en una consulta SQL es la siguiente:

```
SELECT CONCAT(columna1, columna2, ...) AS alias_columna
FROM nombre_de_la_tabla;
```

Ejemplo. 50. Suponga que se tiene la siguiente tabla de `usuarios`, donde el nombre y el apellido están almacenados en columnas separadas:

nombre	apellido	ciudad
Juan	Pérez	Madrid
María	Gómez	Barcelona
Luis	Rodríguez	Madrid
Ana	Fernández	Sevilla
Carlos	López	Barcelona

Tabla 61. Tabla de usuarios.

Si se desea obtener una lista de usuarios con su nombre completo (nombre y apellido concatenados), se puede utilizar la función `CONCAT` de la siguiente manera:

```
SELECT CONCAT(nombre, ' ', apellido) AS nombre_completo
FROM usuarios;
```

El resultado de esta consulta es:

nombre_completo
Juan Pérez
María Gómez
Luis Rodríguez
Ana Fernández
Carlos López

Tabla 62. Usuarios con nombre y apellido concatenados.

Ejemplo. 51. Si se desea concatenar el nombre completo de los usuarios con la ciudad en la que residen, se puede utilizar la siguiente consulta con `CONCAT`:

```
SELECT CONCAT(nombre, ' ', apellido, ' - ', ciudad) AS info_usuario
FROM usuarios;
```

El resultado de esta consulta es:

info_usuario
Juan Pérez - Madrid
María Gómez - Barcelona
Luis Rodríguez - Madrid
Ana Fernández - Sevilla
Carlos López - Barcelona

Tabla 63. Usuarios con nombre completo y ciudad concatenados.

6.8.19. GROUP BY

El modificador `GROUP BY` se utiliza para agrupar filas que tienen valores idénticos en una o más columnas, permitiendo realizar operaciones de agregación en cada grupo. Este modificador es esencial cuando se necesita resumir datos y aplicar funciones como `COUNT`, `SUM`, `AVG`, `MIN`, o `MAX` sobre grupos específicos de registros.

La estructura básica para utilizar `GROUP BY` en una consulta SQL es la siguiente:

```
SELECT columna1, función_de_agregación(columna2)
FROM nombre_de_la_tabla
GROUP BY columna1;
```

Ejemplo. 52. Suponga que se tiene la siguiente tabla de `usuarios`, donde varios usuarios pueden residir en la misma ciudad:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 64. Tabla de usuarios.

Si se desea obtener una lista de cuántos usuarios viven en cada ciudad, se puede utilizar `GROUP BY` junto con la función `COUNT`:

```
SELECT ciudad, COUNT(*) AS total_usuarios
FROM usuarios
GROUP BY ciudad;
```

El resultado de esta consulta es:

ciudad	total_usuarios
Barcelona	2
Madrid	2
Sevilla	1

Tabla 65. Cantidad de usuarios por ciudad.

Ejemplo. 53. Si se desea calcular la edad promedio de los usuarios agrupados por ciudad, se puede utilizar `GROUP BY` junto con la función `AVG`:

```
SELECT ciudad, AVG(edad) AS promedio_edad
FROM usuarios
GROUP BY ciudad;
```

El resultado de esta consulta es:

ciudad	promedio_edad
Barcelona	32.5
Madrid	23.5
Sevilla	28

Tabla 66. Edad promedio por ciudad.

6.8.20. HAVING

El modificador `HAVING` se utiliza para filtrar los resultados de un conjunto de datos agrupados. A diferencia de `WHERE`, que se utiliza para filtrar filas antes de la agregación, `HAVING` se aplica después de que se ha realizado la agregación y permite filtrar los grupos de resultados basados en condiciones específicas.

La estructura básica para utilizar `HAVING` en una consulta SQL es la siguiente:

```
SELECT columna1, funcion_de_agregacion(columna2)
FROM nombre_de_la_tabla
GROUP BY columna1
HAVING condicion;
```

Ejemplo. 54. Suponga que se tiene la siguiente tabla de `usuarios`, donde varios usuarios pueden residir en la misma ciudad:

nombre_usuario	ciudad	edad
Juan Pérez	Madrid	25
María Gómez	Barcelona	30
Luis Rodríguez	Madrid	22
Ana Fernández	Sevilla	28
Carlos López	Barcelona	35

Tabla 67. Tabla de usuarios.

Si se desea obtener una lista de las ciudades que tienen más de un usuario registrado, se puede utilizar `GROUP BY` y luego aplicar `HAVING` para filtrar los grupos que tienen más de un usuario:

```
SELECT ciudad, COUNT(*) AS total_usuarios
FROM usuarios
GROUP BY ciudad
HAVING COUNT(*) > 1;
```

El resultado de esta consula es:

ciudad	total_usuarios
Barcelona	2
Madrid	2

Tabla 68. Ciudades con más de un usuario.

Ejemplo. 55. Si se desea obtener una lista de ciudades donde la edad promedio de los usuarios sea mayor a 25 años, se puede utilizar `HAVING` junto con `AVG`:

```
SELECT ciudad, AVG(edad) AS promedio_edad
FROM usuarios
GROUP BY ciudad
HAVING AVG(edad) > 25;
```

El resultado de esta consulta es:

ciudad	promedio_edad
Barcelona	32.5
Sevilla	28

Tabla 69. Ciudades con edad promedio mayor a 25 años.

6.8.21. CASE

La expresión condicional `CASE` se utiliza para ejecutar lógica condicional dentro de una consulta. Permite devolver valores específicos basados en condiciones definidas, similar a una estructura de control de flujo `IF-THEN-ELSE` en lenguajes de programación.

La estructura básica para utilizar `CASE` en una consulta SQL es la siguiente:

```
SELECT
    CASE
```

```
        WHEN condicion1 THEN resultado1
        WHEN condicion2 THEN resultado2
        ...
        ELSE resultado_defecto
    END AS alias_columna
FROM nombre_de_la_tabla;
```

Ejemplo. 56. Suponga que se tiene la siguiente tabla de `usuarios`, donde se registran el nombre del usuario y su edad:

nombre_usuario	edad
Juan Pérez	25
María Gómez	30
Luis Rodríguez	22
Ana Fernández	28
Carlos López	35

Tabla 70. Tabla de usuarios.

Si se desea clasificar a los usuarios en dos grupos, “Joven” para aquellos menores de 30 años y “Adulto” para aquellos de 30 años o más, se puede utilizar el operador `CASE` de la siguiente manera:

```
SELECT nombre_usuario,
       CASE
           WHEN edad < 30 THEN 'Joven'
           ELSE 'Adulto'
       END AS categoria
FROM usuarios;
```

El resultado de esta consulta es:

nombre_usuario	categoria
Juan Pérez	Joven
María Gómez	Adulto
Luis Rodríguez	Joven
Ana Fernández	Joven
Carlos López	Adulto

Tabla 71. Clasificación de usuarios por categoría de edad.

Ejemplo. 57. Si se desea ofrecer un descuento del 10 % para los usuarios menores de 30 años y un descuento del 5 % para los mayores o iguales a 30 años, se puede utilizar la siguiente consulta con `CASE`:

```
SELECT nombre_usuario,
       CASE
           WHEN edad < 30 THEN '10%'
           ELSE '5%'
       END AS descuento
FROM usuarios;
```

El resultado de esta consulta es:

nombre_usuario	descuento
Juan Pérez	10 %
María Gómez	5 %
Luis Rodríguez	10 %
Ana Fernández	10 %
Carlos López	5 %

Tabla 72. Descuento ofrecido según la edad del usuario.

6.8.22. IFNULL

La función de agregación IFNULL se utiliza para gestionar valores NULL en los datos. Esta función permite reemplazar un valor NULL con un valor predeterminado especificado, asegurando que los resultados de una consulta no contengan valores NULL donde no sean deseados. IFNULL es particularmente útil en cálculos y agregaciones donde un valor NULL podría causar errores o resultados inesperados.

La estructura básica para utilizar IFNULL en una consulta SQL es la siguiente:

```
SELECT IFNULL(columna, valor_predeterminado) AS alias_columna
FROM nombre_de_la_tabla;
```

Ejemplo. 58. Suponga que se tiene la siguiente tabla de usuarios, donde algunos usuarios no tienen registrada la ciudad en la que residen (es decir, el valor es NULL):

nombre_usuario	ciudad
Juan Pérez	Madrid
María Gómez	NULL
Luis Rodríguez	Madrid
Ana Fernández	Sevilla
Carlos López	NULL

Tabla 73. Tabla de usuarios con valores NULL en la columna ciudad.

Si se desea reemplazar los valores NULL en la columna ciudad por un valor predeterminado, como “Desconocido”, se puede utilizar la función IFNULL de la siguiente manera:

```
SELECT nombre_usuario,
       IFNULL(ciudad, 'Desconocido') AS ciudad_modificada
FROM usuarios;
```

El resultado de esta consulta es:

nombre_usuario	ciudad_modificada
Juan Pérez	Madrid
María Gómez	Desconocido
Luis Rodríguez	Madrid
Ana Fernández	Sevilla
Carlos López	Desconocido

Tabla 74. Usuarios con valores `NULL` reemplazados por “Desconocido”.

Ejemplo. 59. Si se desea calcular cuántos usuarios no tienen la ciudad registrada (`NULL`), pero primero se desea mostrar un valor predeterminado para esos valores nulos, se puede hacer lo siguiente:

```
SELECT nombre_usuario,
       IFNULL(ciudad, 'Desconocido') AS ciudad_modificada
FROM usuarios
WHERE ciudad IS NULL;
```

El resultado de esta consulta es:

nombre_usuario	ciudad_modificada
María Gómez	Desconocido
Carlos López	Desconocido

Tabla 75. Usuarios sin ciudad registrada (valores `NULL`).

6.9. Escritura de datos

En este capítulo se explican los comandos usados en SQL para la escritura de datos. Esta escritura de datos consiste en añadir, modificar y eliminar registros de datos en

columnas y filas de una tabla ya existente. Para cada tipo de escritura existe una sentencia correspondiente[27].
A continuación, se muestra mediante ejemplos el uso y significado de las sentencias relacionadas a la escritura de datos.

6.9.1. INSERT INTO

El comando `INSERT INTO` en SQL se utiliza para agregar nuevas filas a una tabla en una base de datos y permite especificar los valores que se deben insertar en cada columna de la tabla. A continuación, un ejemplo del uso de este comando.

La estructura básica para utilizar el comando `INSERT INTO` en una consulta SQL es la siguiente:

```
INSERT INTO nombre_de_la_tabla (columna1, columna2, ...)
VALUES (valor1, valor2, ...);
```

Ejemplo. 60. Suponga que se tiene una tabla llamada `clientes` con las siguientes columnas:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	NULL	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 76. Tabla de clientes después de las inserciones.

[27] W3schools. SQL Tutorial. Recuperado el 6 de abril del 2025. URL: https://www.w3schools.com/sql/default.asp

Si se desea insertar un nuevo registro, la consulta es:

```
INSERT INTO clientes (nombre, telefono, email)
VALUES ('Juan Pérez', '1234567890', 'juan.perez@example.com');
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	NULL	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com
Juan Pérez	1234567890	juan.perez@example.com

Tabla 77. Tabla de clientes después de la inserción de Juan Pérez.

Ejemplo. 61. Si no se desea insertar valores para todas las columnas, se puede omitir la columna `telefono`:

```
INSERT INTO clientes (nombre, email)
VALUES ('Carlos Ruiz', 'carlos.ruiz@example.com');
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	NULL	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com
Carlos Ruiz	NULL	carlos.ruiz@example.com

Tabla 78. Tabla de clientes después de la inserción de Carlos Ruiz.

Ejemplo. 62. El comando `INSERT` también permite insertar múltiples filas en una sola consulta:

```
INSERT INTO clientes (nombre, telefono, email)
VALUES
('Ana López', '1112223333', 'ana.lopez@example.com'),
('Luis Torres', '4445556666', 'luistorres@example.com');
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	NULL	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 79. Tabla de clientes después de la inserción de Ana López y Luis Torres.

6.9.2. UPDATE

El comando `UPDATE` en SQL se utiliza para modificar los datos existentes en una tabla. Este comando permite actualizar los valores de una o más columnas en filas específicas, según se definan las condiciones.

La estructura básica para utilizar el comando `UPDATE` en una consulta SQL es la siguiente:

```
UPDATE nombre_de_la_tabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condicion;
```

Ejemplo. 63. Suponga que se tiene una tabla llamada `clientes` con las siguientes columnas:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 80. Tabla de clientes antes de las actualizaciones.

Si se desea actualizar el número de teléfono de un cliente específico, la consulta es:

```
UPDATE clientes
SET telefono = '9876543210'
WHERE nombre = 'Juan Pérez';
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	9876543210	juan.perez@example.com
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 81. Tabla de clientes después de la actualización del teléfono de Juan Pérez.

Ejemplo. 64. El comando `UPDATE` también permite modificar múltiples columnas en una misma operación. Si se desea actualizar tanto el número de teléfono como el correo electrónico de un cliente, la consulta es:

```
UPDATE clientes
```

```
SET telefono = '9876543210', email = 'juan.perez@nuevoemail.com'
WHERE nombre = 'Juan Pérez';
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	9876543210	juan.perez@nuevoemail.com
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 82. Tabla de clientes después de la actualización de Juan Pérez.

Ejemplo. 65. Si se desea asignar un nuevo valor a todos los clientes cuyo nombre comience con `Ana`, la consulta es:

```
UPDATE clientes
SET telefono = '1234567890'
WHERE nombre LIKE 'Ana%';
```

El resultado de esta consulta es:

nombre	telefono	email
Juan Pérez	9876543210	juan.perez@nuevoemail.com
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1234567890	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 83. Tabla de clientes después de actualizar los registros cuyo nombre comienza con “Ana”.

Ejemplo. 66. El comando `UPDATE` también puede combinarse con otros modificadores

como CASE para realizar actualizaciones condicionales. Si se desea ajustar los descuentos de los clientes en función de sus niveles de compra, la consulta es:

```
UPDATE clientes
SET descuento =
    CASE
        WHEN compras_totales > 1000 THEN 0.15
        WHEN compras_totales BETWEEN 500 AND 1000 THEN 0.10
        ELSE 0.05
    END;
```

El resultado de esta consulta, si se asume que las columnas compras_totales y descuento existen, podría ser algo así:

nombre	telefono	compras_totales	descuento
Juan Pérez	9876543210	1200	0.15
Carlos Ruiz	9876543210	800	0.10
Ana López	1234567890	300	0.05
Luis Torres	4445556666	1500	0.15

Tabla 84. Tabla de clientes después de ajustar el descuento en función de las compras.

6.9.3. DELETE

El comando DELETE en SQL se utiliza para eliminar una o más filas de una tabla en una base de datos. La eliminación de datos es una operación crítica que debe realizarse con cuidado para evitar la pérdida de información importante.

La estructura básica para utilizar el comando DELETE en una consulta SQL es la siguiente:

```
DELETE FROM nombre_de_la_tabla
```

```
WHERE condicion;
```

Ejemplo. 67. Suponga que se tiene una tabla llamada `clientes` con las siguientes columnas:

nombre	telefono	email
Juan Pérez	1234567890	juan.perez@example.com
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 85. Tabla de clientes antes de eliminar registros.

Si se desea eliminar el registro de un cliente específico, la consulta es:

```
DELETE FROM clientes
WHERE nombre = 'Juan Pérez';
```

El resultado de esta consulta es:

nombre	telefono	email
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 86. Tabla de clientes después de eliminar a Juan Pérez.

Ejemplo. 68. Para eliminar múltiples filas al mismo tiempo, se puede definir una condición que abarque varios registros. Si se desea eliminar todos los clientes que no tienen un número de teléfono registrado, la consulta es:

```
DELETE FROM clientes
WHERE telefono IS NULL;
```

El resultado de esta consulta, si hubiera registros sin número de teléfono, sería una tabla sin esas filas. En este caso, no se muestra ningún cambio porque todos los clientes tienen un número de teléfono registrado:

nombre	telefono	email
Carlos Ruiz	9876543210	carlos.ruiz@example.com
Ana López	1112223333	ana.lopez@example.com
Luis Torres	4445556666	luis.torres@example.com

Tabla 87. Tabla de clientes sin cambios, ya que no hay registros con teléfono `NULL`.

Ejemplo. 69. El comando `DELETE` también puede usarse para eliminar todas las filas de una tabla. Sin embargo, es crucial utilizar esta operación con precaución. La consulta para eliminar todos los datos de una tabla es:

```
DELETE FROM clientes;
```

El resultado de esta consulta es:

nombre	telefono	email
No hay registros		

Tabla 88. Tabla de clientes vacía después de eliminar todos los registros.

6.10. Tablas relacionadas y almacenamiento de datos

En este capítulo se explican los tres tipos de relaciones que pueden establecerse entre tablas en MySQL (uno a uno, uno a muchos, muchos a muchos), cómo crear dichas relaciones utilizando claves primarias y foráneas, y finalmente, cómo almacenar datos correctamente en tablas relacionadas. Una relación entre dos tablas es un vínculo que

organiza y conecta los datos de una forma estructurada [28].

Una `clave primaria` es un identificador único para cada registro en una tabla, y no puede contener valores repetidos ni nulos. Una `clave foránea` es un campo que establece una relación entre dos tablas, apuntando a una clave primaria en otra tabla.

6.10.1. Relación 1:1

En una base de datos relacional, la relación uno a uno (1:1) entre dos tablas se establece cuando un registro en la primera tabla está relacionado con exactamente un registro en la segunda tabla, y viceversa. Esta relación es útil cuando se desea dividir información en diferentes tablas, pero mantener una correspondencia directa entre los registros de ambas tablas.

La estructura básica para crear una relación 1:1 es la siguiente:

```
CREATE TABLE tabla_principal(
    pk_tabla_principal tipo_dato PRIMARY KEY,
    columna1 tipo_dato,
    ...
);

CREATE TABLE tabla_relacionada(
    pk_tabla_relacionada tipo_dato PRIMARY KEY,
    pk_tabla_principal tipo_dato UNIQUE,
    columna2 tipo_dato,
    FOREIGN KEY (pk_tabla_principal) REFERENCES
        tabla_principal(pk_tabla_principal)
```

[28] W3schools. SQL Tutorial. Recuperado el 6 de abril del 2025. URL: https://www.w3schools.com/sql/default.asp

```
);
```

Para almacenar los datos, primero se insertan en la tabla principal y luego en la tabla relacionada, asegurando que el valor de la clave foránea coincida con uno existente en la tabla principal.

Ejemplo. 70. Suponga que se tiene la siguiente tabla principal:

```
CREATE TABLE empleados_empresa(
    nombres varchar(100),
    apellidos varchar(100),
    n_id bigint PRIMARY KEY
);
```

Y se desea registrar un identificador interno para cada empleado:

```
CREATE TABLE ID(
    id_empresa int AUTO_INCREMENT,
    n_id bigint UNIQUE,
    FOREIGN KEY (n_id) REFERENCES empleados_empresa(n_id)
);
```

Para almacenar los datos:

```
INSERT INTO empleados_empresa (nombres, apellidos, n_id) VALUES ('nombre1',
   'apellido1', 1);
INSERT INTO ID (id_empresa, n_id) VALUES (11111111, 1);
```

6.10.2. Relación 1:N

En una base de datos relacional, la relación uno a muchos (1:N) se establece cuando un registro en la primera tabla puede estar relacionado con varios registros en la segun-

da, pero cada registro en la segunda tabla solo puede estar relacionado con uno en la primera.

La estructura básica para crear una relación 1:N es la siguiente:

```
CREATE TABLE area_trabajo(
    area_id int PRIMARY KEY,
    nombre varchar(100)
);

CREATE TABLE empleados_empresa(
    nombres varchar(100),
    apellidos varchar(100),
    n_id bigint PRIMARY KEY,
    area_id int,
    FOREIGN KEY (area_id) REFERENCES area_trabajo(area_id)
);
```

Ejemplo. 71. Primero se insertan los datos en la tabla secundaria:

```
INSERT INTO area_trabajo (area_id, nombre) VALUES (11, 'Recursos Humanos');
INSERT INTO area_trabajo (area_id, nombre) VALUES (22, 'Contabilidad');
```

Luego, se almacenan los datos en la tabla principal relacionados por la clave foránea:

```
INSERT INTO empleados_empresa (nombres, apellidos, n_id, area_id)
VALUES ('nombre1', 'apellido1', 1, 11);
INSERT INTO empleados_empresa (nombres, apellidos, n_id, area_id)
VALUES ('nombre2', 'apellido2', 2, 22);
INSERT INTO empleados_empresa (nombres, apellidos, n_id, area_id)
VALUES ('nombre3', 'apellido3', 3, 22);
```

6.10.3. Relación N:M

La relación muchos a muchos (N:M) se representa con una tabla intermedia que contiene claves foráneas de ambas tablas relacionadas. Esta tabla permite que múltiples registros de una tabla se relacionen con múltiples registros de otra.

La estructura básica para crear una relación N:M es la siguiente:

```
CREATE TABLE idiomas(
    idioma_id int PRIMARY KEY,
    nom_idioma varchar(100)
);

CREATE TABLE empleados_empresa(
    nombres varchar(100),
    apellidos varchar(100),
    n_id bigint PRIMARY KEY
);

CREATE TABLE empleado_idioma(
    n_id bigint,
    idioma_id int,
    FOREIGN KEY (n_id) REFERENCES empleados_empresa(n_id),
    FOREIGN KEY (idioma_id) REFERENCES idiomas(idioma_id),
    UNIQUE (n_id, idioma_id)
);
```

Ejemplo. 72. Primero se insertan los datos en las tablas principales:

```
INSERT INTO empleados_empresa (nombres, apellidos, n_id) VALUES ('nombre1',
    'apellido1', 1);
INSERT INTO empleados_empresa (nombres, apellidos, n_id) VALUES ('nombre2',
    'apellido2', 2);
INSERT INTO empleados_empresa (nombres, apellidos, n_id) VALUES ('nombre3',
    'apellido3', 3);

INSERT INTO idiomas (idioma_id, nom_idioma) VALUES (1, 'Español');
INSERT INTO idiomas (idioma_id, nom_idioma) VALUES (2, 'Inglés');
INSERT INTO idiomas (idioma_id, nom_idioma) VALUES (3, 'Alemán');
INSERT INTO idiomas (idioma_id, nom_idioma) VALUES (4, 'Portugués');
INSERT INTO idiomas (idioma_id, nom_idioma) VALUES (5, 'Francés');
```

Luego se insertan los datos en la tabla intermedia:

```
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (1, 1);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (1, 2);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (1, 5);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (2, 1);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (2, 3);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (2, 4);
INSERT INTO empleado_idioma (n_id, idioma_id) VALUES (3, 1);
```

Con estos ejemplos se pudo visualizar cómo se implementan y se almacenan datos en relaciones uno a uno, uno a muchos y muchos a muchos dentro de una base de datos MySQL. Además, se destacó el papel fundamental que cumplen las claves primarias y foráneas para garantizar la integridad referencial entre las tablas.

6.11. Consulta de datos relacionados

En este capítulo se explica cómo hacer una consulta de datos relacionados entre tablas en MySQL usando las sentencias correspondientes.
Las consultas de datos relacionales en bases de datos se refieren a las operaciones que se realizan para recuperar información de múltiples tablas relacionadas entre sí, basándose en un conjunto de condiciones o criterios. En un sistema de gestión de bases de datos relacionales como MySQL, los datos están organizados en tablas que pueden estar relacionadas mediante llaves primarias y foráneas, lo que permite la conexión lógica entre ellas [29].

6.11.1. INNER JOIN

El comando `INNER JOIN` en MySQL se utiliza para combinar filas de dos o más tablas basándose en una condición que relaciona las columnas de dichas tablas. Solo se devuelven las filas donde existe coincidencia entre las tablas involucradas.

La estructura básica para utilizar `INNER JOIN` es la siguiente:

```
SELECT columnas
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

Ejemplo. 73. Suponga que se tienen las siguientes tablas de `empleados` y `departamentos`:

---

[29] (W3schools. s.f.[b])

id	nombre	id_departamento
1	Juan Pérez	1
2	María Gómez	2
3	Carlos López	1

Tabla 89. Tabla de empleados.

id_departamento	nombre_departamento
1	Recursos Humanos
2	Finanzas

Tabla 90. Tabla de departamentos.

Si se desea obtener una lista de empleados junto con el nombre de su departamento, se puede utilizar la siguiente consulta con `INNER JOIN`:

```
SELECT empleados.nombre, departamentos.nombre_departamento
FROM empleados
INNER JOIN departamentos
ON empleados.id_departamento = departamentos.id_departamento;
```

El resultado de esta consulta es:

nombre	nombre_departamento
Juan Pérez	Recursos Humanos
María Gómez	Finanzas
Carlos López	Recursos Humanos

Tabla 91. Resultado del INNER JOIN entre empleados y departamentos.

6.11.2. LEFT JOIN

El comando `LEFT JOIN` en MySQL se utiliza para combinar filas de dos o más tablas. Devuelve todas las filas de la tabla de la izquierda (primer tabla), incluso si no hay coincidencias en la tabla de la derecha. Si no hay coincidencias, las columnas de la tabla de la derecha se rellenan con `NULL`.

La estructura básica para utilizar `LEFT JOIN` es la siguiente:

```
SELECT columnas
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

Ejemplo. 74. Suponga que se tienen las siguientes tablas de `empleados` y `departamentos`:

id	nombre	id_departamento
1	Juan Pérez	1
2	María Gómez	2
3	Carlos López	*NULL*

Tabla 92. Tabla de empleados.

id_departamento	nombre_departamento
1	Recursos Humanos
2	Finanzas

Tabla 93. Tabla de departamentos.

Si se desea obtener una lista de todos los empleados junto con el nombre de su departamento, incluyendo aquellos que no tienen asignado un departamento, se puede utilizar la

siguiente consulta con `LEFT JOIN`:

```
SELECT empleados.nombre, departamentos.nombre_departamento
FROM empleados
LEFT JOIN departamentos
ON empleados.id_departamento = departamentos.id_departamento;
```

El resultado de esta consulta es:

nombre	nombre_departamento
Juan Pérez	Recursos Humanos
María Gómez	Finanzas
Carlos López	*NULL*

Tabla 94. Resultado del LEFT JOIN entre empleados y departamentos.

6.11.3. RIGHT JOIN

El comando `RIGHT JOIN` en MySQL se utiliza para combinar filas de dos o más tablas. Devuelve todas las filas de la tabla de la derecha (segunda tabla), incluso si no hay coincidencias en la tabla de la izquierda. Si no hay coincidencias, las columnas de la tabla de la izquierda se rellenan con `NULL`.

La estructura básica para utilizar `RIGHT JOIN` es la siguiente:

```
SELECT columnas
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

Ejemplo. 75. Suponga que se tienen las siguientes tablas de `empleados` y `departamentos`:

id	nombre	id_departamento
1	Juan Pérez	1
2	María Gómez	2

Tabla 95. Tabla de empleados.

id_departamento	nombre_departamento
1	Recursos Humanos
2	Finanzas
3	Marketing

Tabla 96. Tabla de departamentos.

Si se desea obtener una lista de todos los departamentos junto con los nombres de los empleados asignados a ellos, incluyendo aquellos departamentos que no tienen empleados asignados, se puede utilizar la siguiente consulta con `RIGHT JOIN`:

```
SELECT empleados.nombre, departamentos.nombre_departamento
FROM empleados
RIGHT JOIN departamentos
ON empleados.id_departamento = departamentos.id_departamento;
```

El resultado de esta consulta es:

nombre	nombre_departamento
Juan Pérez	Recursos Humanos
María Gómez	Finanzas
NULL	Marketing

Tabla 97. Resultado del RIGHT JOIN entre empleados y departamentos.

6.11.4. UNION

El comando `UNION` en MySQL se utiliza para combinar los resultados de dos o más consultas `SELECT`. Devuelve todas las filas de ambas consultas, eliminando los duplicados de manera predeterminada. Las consultas deben tener el mismo número de columnas y los tipos de datos deben ser compatibles.

La estructura básica para utilizar `UNION` es la siguiente:

```
SELECT columnas
FROM tabla1
UNION
SELECT columnas
FROM tabla2;
```

Ejemplo. 76. Suponga que se tienen las siguientes tablas de `empleados_actuales` y `ex_empleados`:

id	nombre	cargo
1	Juan Pérez	Gerente
2	María Gómez	Secretaria

Tabla 98. Tabla de empleados actuales.

id	nombre	cargo
3	Carlos López	Contador
4	Ana Fernández	Asistente

Tabla 99. Tabla de ex-empleados.

Si se desea obtener una lista combinada de todos los empleados, tanto actuales como ex-empleados, se puede utilizar la siguiente consulta con `UNION`:

```
SELECT nombre, cargo
FROM empleados_actuales
UNION
SELECT nombre, cargo
FROM ex_empleados;
```

El resultado de esta consulta es:

nombre	cargo
Juan Pérez	Gerente
María Gómez	Secretaria
Carlos López	Contador
Ana Fernández	Asistente

Tabla 100. Resultado de la unión de empleados actuales y ex-empleados.

6.12. Instalación de Python

En este capítulo se proporciona una guía detallada para la descarga e instalación de Python en sistemas operativos con Windows. Se explicará cómo acceder al sitio oficial de Python para descargar la versión más reciente y adecuada para el sistema [30]. A continuación, el paso a paso de la instalación:

- En primer lugar, es necesario ingresar al sitio web oficial de Python mediante el siguiente enlace: Python.org.

[30] Python Software Foundation. Download Python. Recuperado el 26 de marzo de 2025. 2023. URL: https://www.python.org/downloads/

- Una vez en la página principal, se debe ubicar la sección de descargas y seleccionar el botón que aparece debajo del texto: `Download for Windows`.

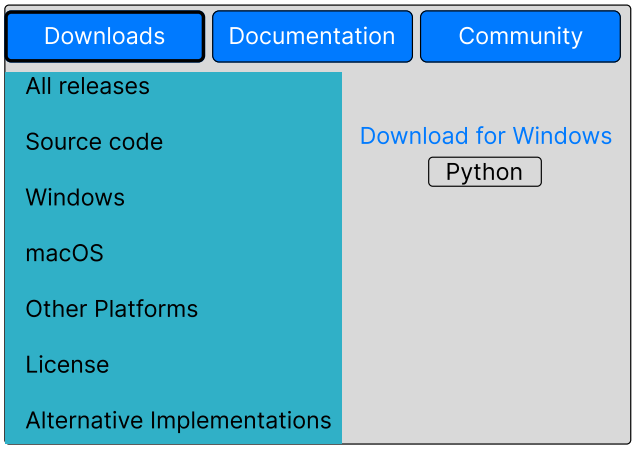

Figura 16. Elección de sistema operativo.

- Ya descargado el archivo `.exe`, se debe hacer clic para abrirlo y, a continuación, seleccionar la opción `Install Now`.

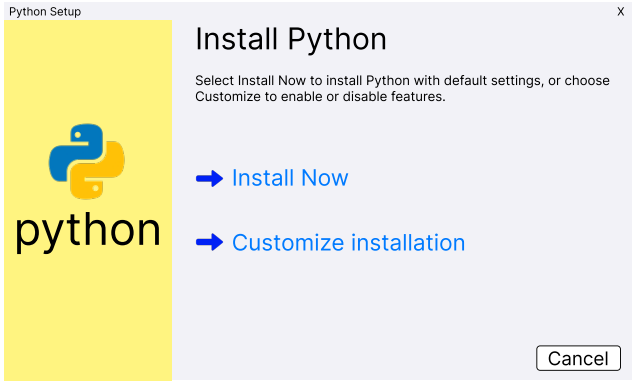

Figura 17. Interfaz de instalación de Python.

- La instalación se completará exitosamente. A continuación, se debe cerrar la ventana seleccionando la opción `Close`.

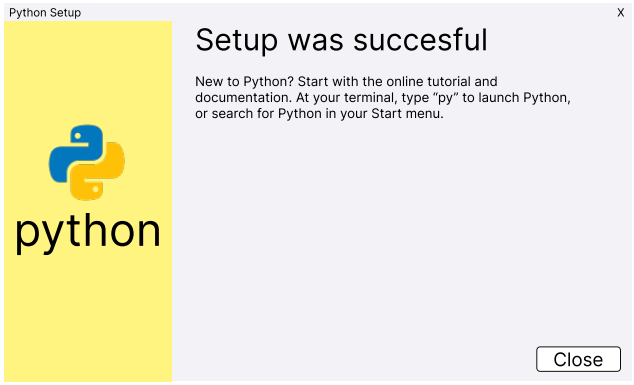

Figura 18. Instalación de Python completada

Los pasos mencionados anteriormente serán suficientes para tener Python instalado en el ordenador. En el siguiente apartado, se explicará cómo verificar que la instalación se haya realizado correctamente.

- En el panel de búsqueda principal de Windows, se debe escribir la palabra `Python`, lo cual mostrará una aplicación que coincide con dicho nombre; se debe seleccionar para abrirla.

Figura 19. Icono del programa.

- A continuación, se desplegará una consola similar a la de CMD, dedicada exclusivamente a Python, en la cual es posible escribir y ejecutar comandos propios de este lenguaje.

```
print("Instalación de Python exitosa")
```

Si la instalación ha sido correcta, la consola mostrará el siguiente mensaje: `Instalación de Python exitosa`.

- Para salir de esta consola, basta escribir el siguiente comando:

```
Exit()
```

6.13. Instalación de Jupyter Notebook y Miniconda

En este capítulo se explica la instalación de dos herramientas que complementan el uso de MySQL para el análisis de datos.
Jupyter Notebook es una herramienta creada para el desarrollo de software de código abierto que es compatible con varios lenguajes de programación, será usada para la visualización de datos mediante Python. Por otro lado, Miniconda es una versión reducida de Anaconda, esta tiene como recursos solamente a Conda y Python. Para la instalación de estas dos herramientas, es necesario tener instalado Python, como se muestra en la Sección 6.12, y seguir las siguientes instrucciones:

- https://docs.jupyter.org/en/latest/install/notebook-classic.html[31].
- Se ubica la sección llamada `Installing Jupyter using Anaconda and conda`, se presiona en la palabra `Anaconda` que está resaltada en azul y esto redigirá al usuario a una nueva pestaña.

---

[31] (Project Jupyter: *Installation — Jupyter Notebook*. Recuperado el 26 de marzo de 2025. 2023. `https://jupyter.org/install`)

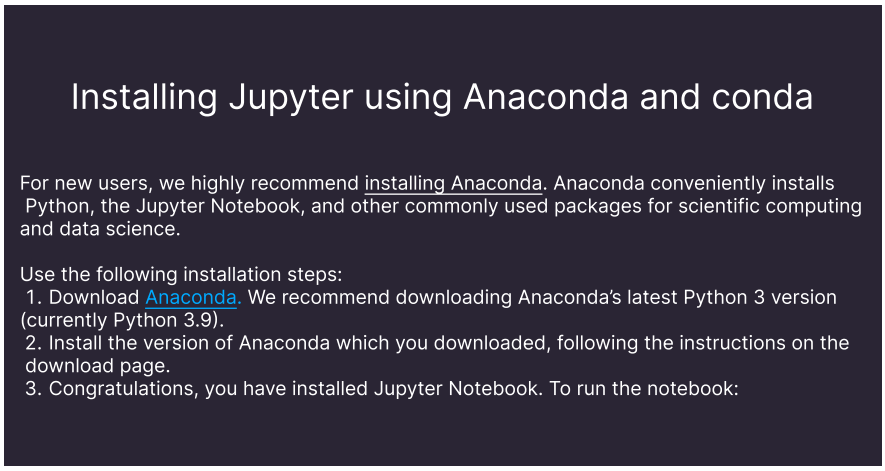

Figura 20. Pasos de instalación de Jupyter usando anaconda y conda.

- Una vez ubicados en la página de Anaconda, se selecciona la opción `Skip registration` lo cual permitirá descargar sin necesidad de un registro.

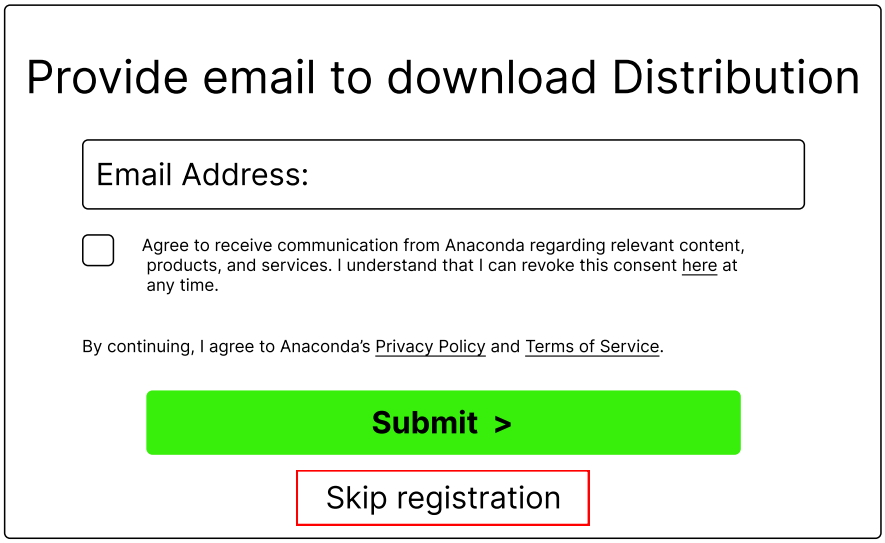

Figura 21. Acceso a la descarga del programa.

- Se selecciona la versión de Miniconda compatible con el ordenador y se descarga.

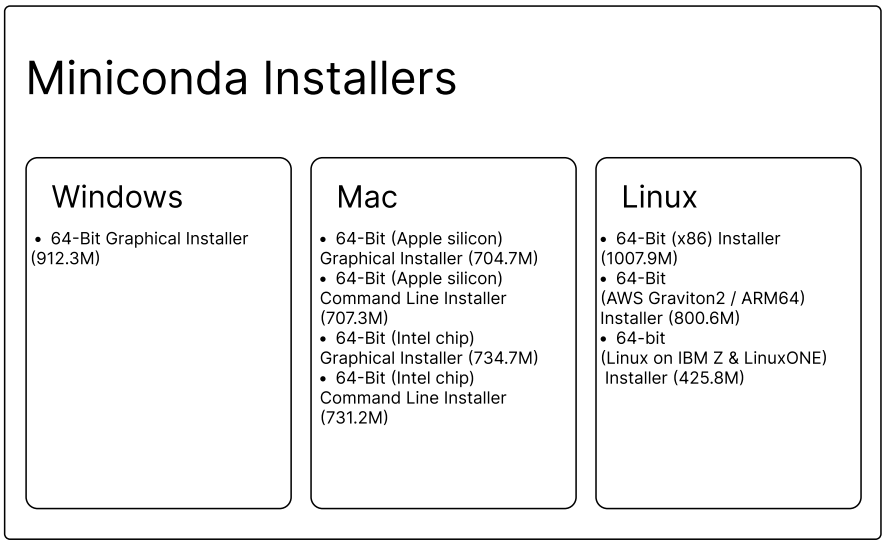

Figura 22. Instaladores de Miniconda.

- Tras haber finalizado la descarga, se ejecuta el archivo y se presiona en la opción `Next`.

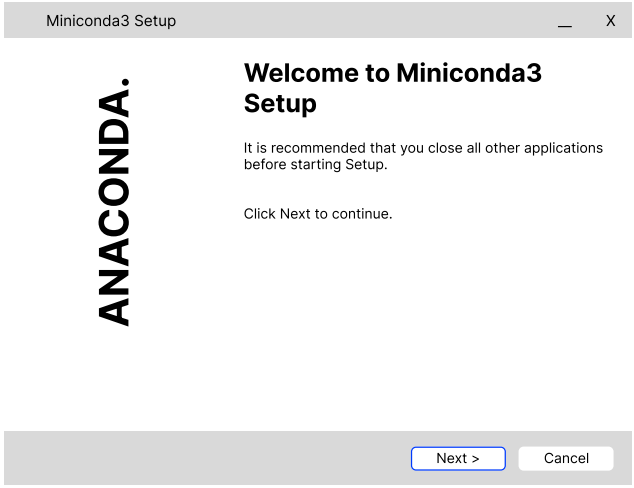

Figura 23. Bienvenida al programa.

- Para aceptar los términos y condiciones de la instalación, se selecciona la opción `I agree`.

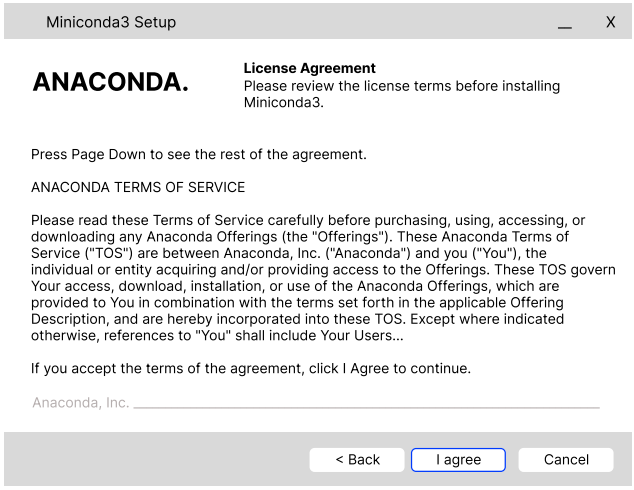

Figura 24. Términos y condiciones del programa.

- Se marca la casilla `Just Me` para continuar y se presiona la opción `Next`.

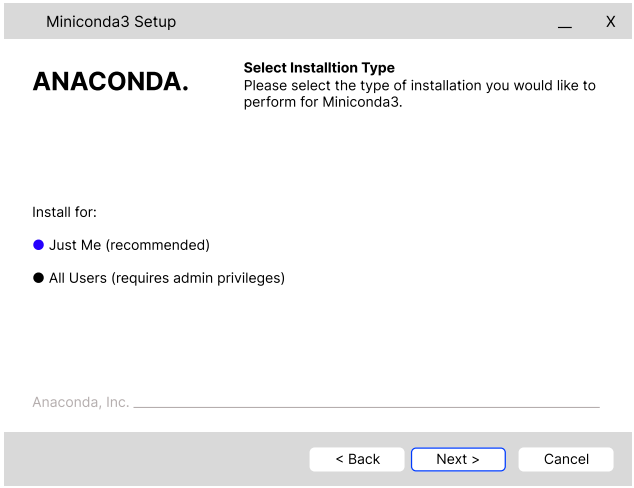

Figura 25. Tipo de instalación.

- Se deja la ubicación por defecto y se hace clic en la opción `Next`.

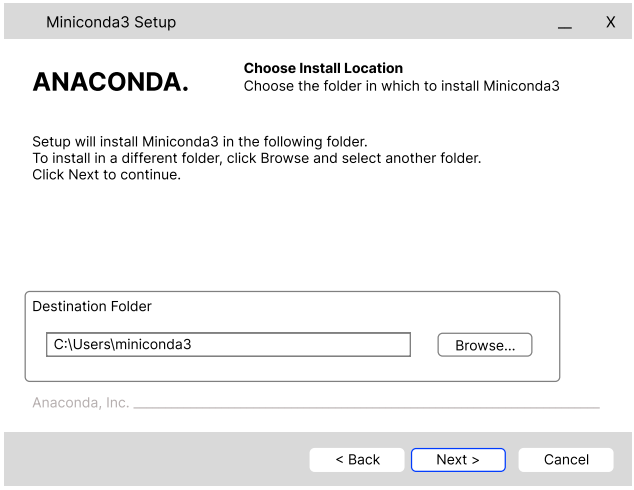

Figura 26. Destino de la descarga.

- Para una completa instalación, se seleccionan todas las casillas y se hace clic en la la opción `Install`.

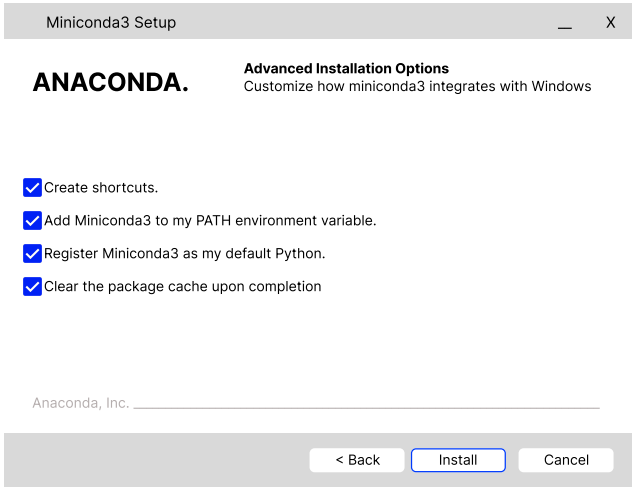

Figura 27. Opciones de instalación avanzada.

- Una vez completa la instalación, se continúa el proceso presionando sobre la opción `Next`.

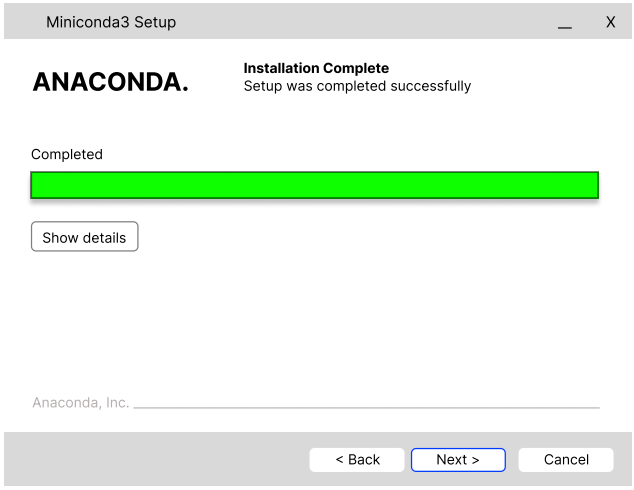

Figura 28. Instalación de anacaonda completada.

- Se seleccionan las casillas que se muestran en la ventana y se finaliza la instalación usando la opción `Finish`.

Figura 29. Instalación de Miniconda3.

- Para completar la configuración, se necesita abrir la consola de Windows. Para esto, se presionan los botones Windows + R en el teclado y muestran la siguiente ventana en la que se escribe el comando `cmd`.

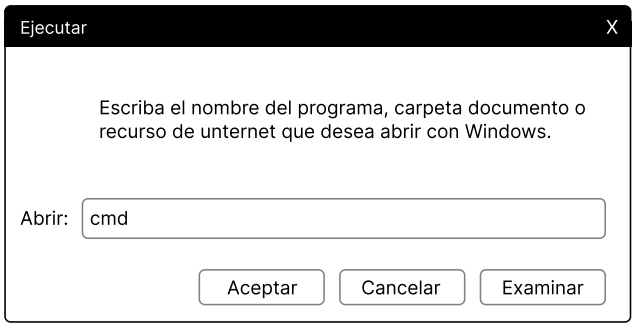

Figura 30. Interfaz de comandos.

- Al completar el paso anterior, Windows mostrará la siguiente ventana.

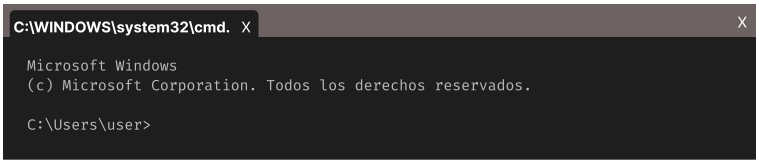

Figura 31. Interfaz de CMD.

- En esta ventana, se escribirá el siguiente comando y se ejecuta presionando enter:

```
conda install -c conda-forge notebook
```

- Al correr el comando, se hará una pregunta en la ventana, a la cual se le responderá sencillamente con la siguiente letra:

```
y
```

- Ya finalizada la configuración, para ejecutar Jupyter Notebook desde la consola, se debe escribir el siguiente comando:

```
jupyter notebook
```

- Se selecciona el navegador de preferencia y se muestra una interfaz como la siguiente:

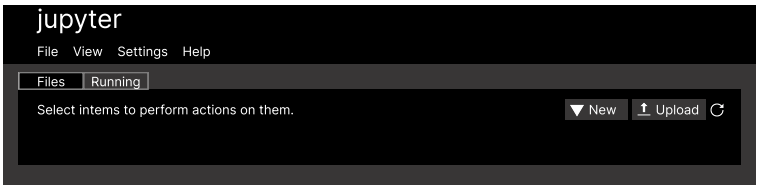

Figura 32. Interfaz de Jupyter.

- Para crear un archivo de código seleccionamos la opción `New` que se ubicada en la esquina superior derecha y se selecciona la opción `Python 3`. A continuación, se muestra la interfaz donde se escriben los comandos en lenguaje Python.

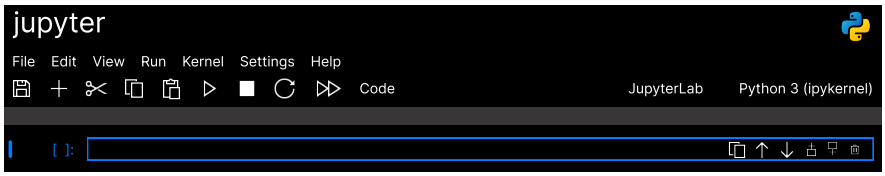

Figura 33. Creación de archivos en Jupyter.

  Para crear nuevas celdas, se debe hacer clic en el botón $+$, y para ejecutarlas, en el botón ▶.

6.14. Instalación de paquetes mediante el gestor de paquetes `pip`

En este capítulo se explica la instalación de recursos adicionales de Python para así poder trabajar con archivos CSV. Es necesario instalar las librerías requeridas para la manipulación, visualización y análisis de datos, así como para conectarse a bases de datos y manejar archivos geoespaciales.
Las principales librerías que se deben instalar son:

- **pandas**: permite leer, escribir y manipular archivos CSV y otros tipos de datos estructurados.

- **numpy**: se utiliza para operaciones numéricas y se complementa muy bien con pandas.
- **mysql-connector-python**: permite conectar Python con bases de datos MySQL para insertar, consultar y modificar datos.
- **matplotlib**: se utiliza para crear gráficos y visualizaciones.
- **geopandas**: permite trabajar con datos geoespaciales (por ejemplo, archivos GeoJSON) y combinarlos con datos tabulares.
- **minisom**: es una implementación simple de mapas autoorganizados (*Self-Organizing Maps*), útil en análisis de datos no supervisados.
- **seaborn**: es una librería de visualización de datos basada en `matplotlib`, que permite crear gráficos estadísticos de forma más sencilla y con una estética mejorada. Es ideal para generar mapas de calor, diagramas de dispersión, diagramas de caja, entre otros.

Para instalar estos paquetes es necesario abrir la consola de comandos del sistema operativo (por ejemplo, `CMD` en Windows, Terminal en macOS o Linux) y escribir los comandos correspondientes utilizando el gestor de paquetes `pip`. A continuación se muestra cómo hacerlo para cada una de las librerías mencionadas:

```
pip install pandas
pip install numpy
pip install mysql-connector-python
pip install matplotlib
pip install geopandas
pip install minisom
pip install seaborn
```

Es importante asegurarse de que Python esté correctamente configurado en el sistema. Esta comprobación se puede realizar tal como se hizo al final de la Sección 6.12.

6.15. Descargar el archivo CSV

En este capítulo se explica cómo descargar el archivo CSV necesario para el análisis de datos correspondiente. Se trabajará con la base de datos del Censo Nacional de Población y Vivienda 2018, cuyos datos son públicos y pueden descargarse desde el siguiente enlace:

`https://microdatos.dane.gov.co/index.php/catalog/643/get-microdata`[32].

Se utilizarán los registros correspondientes al departamento de Antioquia para la explicación de la descarga de archivos CSV.

1. Acceder al enlace previamente mencionado, Base de datos del Censo Nacional de Población y Vivienda 2018, donde se encuentra un listado de departamentos junto con un botón de descarga para una carpeta en formato ZIP.

2. Hacer clic en el botón `descargar`, lo que iniciará la descarga de una carpeta que contiene los datos del departamento de Antioquia.

3. Una vez finalizada la descarga, abrir la carpeta ZIP, dentro de la cual se encuentran tres subcarpetas. Ingresar a la carpeta denominada **“05_Antioquia_CSV”**.

4. En esta carpeta se encuentran cinco archivos en formato CSV. Se debe descargar el archivo denominado **“CNPV2018_3FALL_A2_05”**.

---

[32] (Departamento Administrativo Nacional de Estadística (DANE): *Descarga de microdatos - Encuesta del DANE*. Recuperado el 26 de marzo de 2025. 2023. `https://microdatos.dane.gov.co/index.php/catalog/643/get-microdata`)

5. Guarde el archivo en una carpeta o ruta de fácil acceso para facilitar su carga en la tabla de MySQL. Es importante no modificar la configuración del archivo.

6.16. Conexión a una base de datos en Python desde Jupyter Notebook

En este capítulo se explica el procedimiento para hacer una conexión a una base de datos desde Jupyter Notebook. Una vez completado la Sección 6.4, Sección 6.12, Sección 6.13, Sección 6.14 y Sección 6.15 es posible hacer la conexión mencionada anteriormente, para ello, se crea un notebook en Jupyter para conectarse a MySQL mediante un código de Python para posteriormente crear una base de datos y conectarse con ella.

6.16.1. Crear un Notebook

Para crear un Notebook, se deben seguir cuidadosamente los siguientes pasos:

1. En el buscador del sistema operativo, escribir `Jupyter Notebook` y ejecutar la aplicación.
2. Una vez abierto el entorno, localizar en la esquina superior derecha el botón `New` y hacer clic sobre él.
3. En el menú desplegable, seleccionar la opción `New Folder` para crear una nueva carpeta. Esta carpeta contendrá los archivos relacionados con el proyecto.
4. Asignar a la carpeta el nombre `Departamentos`, haciendo clic en el nombre predeterminado (que suele ser `Untitled Folder`) y escribiendo el nuevo nombre. Luego, presionar `Enter` para confirmar el cambio.
5. Ingresar a la carpeta `Departamentos` haciendo clic sobre ella.

6. Una vez dentro de la carpeta, volver a hacer clic en el botón `New` ubicado en la esquina superior derecha y seleccionar la opción `Notebook` (Python 3) para crear un nuevo cuaderno de trabajo.

7. Cambiar el nombre del cuaderno (notebook) haciendo clic sobre el título predeterminado (por ejemplo, `Untitled`) en la parte superior del cuaderno, y renombrarlo como `Fallecidos_Antioquia`. Luego, hacer clic en `Rename` para guardar el nuevo nombre.

8. El cuaderno `Fallecidos_Antioquia` ya está listo para comenzar a escribir el código correspondiente a la conexión con la base de datos y la manipulación de archivos CSV.

6.16.2. Conectar a MySQL

Para establecer una conexión con un servidor MySQL desde Python, es necesario especificar el `host` y el `user`. Estos valores dependen del entorno en el que se esté trabajando.

- `host`: representa la dirección donde se encuentra el servidor MySQL.
- `user`: es el nombre del usuario autorizado a acceder a la base de datos.

A continuación, se muestra una tabla con los valores recomendados según el caso:

Caso	`host`	`user`
Servidor local (en el mismo computador)	`localhost`	`root` (o usuario creado durante la instalación)
Servidor en red local	Dirección IP (ej: `192.168.1.50`)	Usuario con permisos definidos en el servidor
Servidor en la nube	Nombre de dominio (ej: `midb.mysql.database.azure.com`)	Usuario asignado por el proveedor del servicio

Tabla 101. Tabla de valores recomendados

Nota: Si MySQL fue instalado localmente, es común que el usuario por defecto sea `root`, y que el host sea `localhost`. La contraseña será la que se definió durante la instalación.

Por lo tanto, para definir correctamente el host y el user, en una nueva celda de Jupyter, ejecutamos el siguiente código:

```
import mysql.connector
from mysql.connector import Error


try:
    connection = mysql.connector.connect(
        host='localhost',     # Dirección del servidor MySQL
        user='root',          # Nombre de usuario
        password='tu_clave' # clave del usuario MySQL
    )
```

```
    if connection.is_connected():
        print("Conexión exitosa al servidor MySQL")
        cursor = connection.cursor()

except Error as e:
    print(f"Error al conectarse a MySQL: {e}")
```

6.16.3. Crear una Base de Datos

Una vez establecida la conexión con MySQL, el siguiente paso es crear una base de datos si aún no existe. Para esto, se utiliza el comando SQL `CREATE DATABASE`.

```
try:
    cursor.execute("CREATE DATABASE IF NOT EXISTS base_de_datos")
    print("Base de datos 'base_de_datos' creada exitosamente")

except Error as e:
    print(f"Error al crear la base de datos: {e}")
```

El método `cursor.execute` se usa para enviar una instrucción SQL desde el lenguaje de programación Python hacia la base de datos, `cursor.execute` actúa como un puente entre el programa y la base de datos, permitiendo realizar consultas y manipular los datos almacenados.

El lector puede asignarle el nombre que desee, reemplazando el texto `base_de_datos` por el nombre elegido. En este caso, a modo de ejemplo, la base de datos se denominará `Departamentos`.

```
try:
    cursor.execute("CREATE DATABASE IF NOT EXISTS Departamentos")
```

```
    print("Base de datos 'Departamentos' creada exitosamente")

except Error as e:
    print(f"Error al crear la base de datos: {e}")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter.

6.16.4. Conectar a la Base de Datos

Después de crear la base de datos, es necesario conectarse a ella para poder trabajar (por ejemplo, para crear tablas o realizar consultas). Esto se hace especificando el nombre de la base de datos en la conexión.

```
try:
    connection = mysql.connector.connect(
        host='localhost',
        user='root',
        password='tu_clave',
        database='Departamentos'
    )

    if connection.is_connected():
        print("Conexión exitosa a la base de datos")
        cursor = connection.cursor()

except Error as e:
    print(f"Error al conectarse a la base de datos: {e}")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter.

6.17. Creación de Tabla y Carga de Datos desde CSV a MySQL

En este capítulo se explica cómo crear una tabla en MySQL usando el lenguaje Python desde Jupyter para después llenarla con datos obtenidos de un archivo CSV. Se explica también cómo el archivo CSV es leído y cargado para poder usar sus datos almacenados.

6.17.1. Crear una Tabla

Con la conexión a la base de datos establecida, el siguiente paso es crear una tabla.

```
try:
    cursor = connection.cursor()

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS nombre_tabla (
    id INT AUTO_INCREMENT PRIMARY KEY
    )
    """)
    print("Tabla 'nombre_tabla' creada exitosamente")

except Error as e:
    print(f"Error al crear la tabla: {e}")
```

El lector puede asignarle el nombre que desee, reemplazando el texto `nombre_tabla` por el nombre elegido. En este caso, a modo de ejemplo, la tabla se denominará `Fallecidos_Antioquia` .

```
try:
    cursor = connection.cursor()

    cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS Fallecidos_Antioquia (
    id INT AUTO_INCREMENT PRIMARY KEY
    )
    """)
    print("Tabla 'Fallecidos_Antioquia' creada exitosamente")

except Error as e:
    print(f"Error al crear la tabla: {e}")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter para crear una tabla.

6.17.2. Función para Cargar un Archivo CSV

Se define una función encargada de abrir una ventana de selección de archivos que permite al usuario seleccionar un archivo CSV desde su sistema de archivos. Luego, el archivo es leído y cargado en un `DataFrame`. Si el archivo está vacío, se informa al usuario.

```
import pandas as pd
from tkinter import filedialog
from tkinter import Tk
import tkinter as tk

def cargar_csv():
    archivo = filedialog.askopenfilename(
        title="Selecciona un archivo CSV",
        filetypes=[("Archivos CSV", "*.csv"), ("Todos los archivos", "*.*")]
    )
    if archivo:
        try:
            df = pd.read_csv(archivo)
```

```
        if df.empty:
            print("El archivo CSV esta vacio.")
            return

        print(f"Archivo cargado: {archivo}")
        print(f"Nombre del archivo: {archivo.split('/')[-1]}")

        global df_global
        df_global = df
        print("Datos cargados y listos para ser usados en MySQL.")

    except Exception as e:
        print(f"Error al leer el archivo: {e}")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter.

6.17.3. Función para Subir el Archivo CSV a MySQL

Una vez cargado el archivo CSV, se define una función que toma el `DataFrame` global y lo sube a una tabla en MySQL. La función verifica si el `DataFrame` está vacío antes de proceder. Si la tabla ya existe, es eliminada y recreada con los nombres de columnas extraídos del archivo CSV. Finalmente, los datos se insertan en la tabla.

```
def subir_a_mysql():
    try:

        df = df_global
        nombre_tabla = "Fallecidos_Antioquia"

        if df.empty:
            print("El DataFrame está vacío, no hay datos para subir.")
            return

        columnas = ', '.join(df.columns)

        cursor.execute(f"DROP TABLE IF EXISTS {nombre_tabla}")
        tipos_datos = ', '.join([f'{col} VARCHAR(255)' for col in df.columns])
        cursor.execute(f"CREATE TABLE {nombre_tabla} ({tipos_datos})")

        filas_iniciales = df.shape[0]
        for i, row in df.iterrows():
            valores = ', '.join([f"'{str(x)}'" for x in row])
            cursor.execute(f"INSERT INTO {nombre_tabla} ({columnas}) VALUES
                ({valores})")

        connection.commit()
        print(f"Datos subidos a la tabla '{nombre_tabla}' en MySQL. Filas subidas:
            {filas_iniciales}.")

    except Exception as e:
        print(f"Error al subir los datos: {e}")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter.

6.17.4. Interfaz Gráfica

La interfaz gráfica se implementa utilizando `tkinter`. Se crea una ventana con dos botones: uno para cargar el archivo CSV y otro para subir los datos a MySQL. Cada botón

está asociado a la función correspondiente.

```
ventana = tk.Tk()
ventana.title("Cargador de CSV a MySQL")
ventana.geometry("300x200")

boton_cargar = tk.Button(ventana, text="Cargar archivo CSV", command=cargar_csv)
boton_cargar.pack(pady=10)

boton_subir = tk.Button(ventana, text="Subir a MySQL", command=subir_a_mysql)
boton_subir.pack(pady=10)

ventana.mainloop()
```

El código anterior debe ejecutarse en una nueva celda de Jupyter.

Esta configuración permite al usuario seleccionar un archivo CSV de su sistema, cargarlo en Python y luego subirlo a una base de datos MySQL. Esta celda debe ejecutarse después de haber descargado el archivo CSV y guardado en una carpeta de fácil acceso, tal y como se explica en la Sección 6.15.

Al ejecutar esta celda, aparecerá una ventana emergente como la que se muestra a continuación:

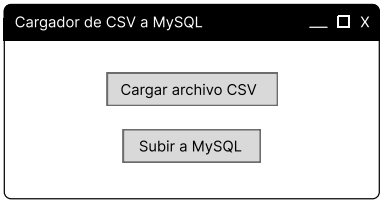

Figura 34. Ventana de carga de CSV a SQL

1. Hacer clic en el botón `Cargar archivo CSV`, lo que abrirá el explorador de archivos.

Seleccionar el archivo CSV previamente guardado.

2. Una vez cargado el archivo, hacer clic en el botón `Subir a MySQL`. El tiempo de procesamiento dependerá del tamaño del archivo y la cantidad de datos que contiene.

3. La carga se considerará exitosa cuando en Jupyter se muestre el siguiente mensaje:

```
Datos subidos a la tabla Fallecidos_Antioquia en MySQL. Filas subidas:
31617.
```

6.18. Ejemplos de sistemas de consultas

En este capítulo se explica el procedimiento necesario para la creación del ejemplo del sistema de consultas, recopilando y usando todo el contenido visto anteriormente. Cada base de datos para analizar proviene de un archivo CSV diferente y los datos son los siguientes:

- Tabla “Fallecidos” del departamento de Antioquia.
- Tabla “Hogares” del departamento de Magdalena.
- Tabla “Personas” del departamento de La Guajira.
- Tabla “Viviendas” del departamento de Santander.

Donde cada base de datos es analizada de manera distinta.
El código correspondiente a cada departamento y municipio está disponible en la siguiente tabla:

`https://www.fopep.gov.co/sheempoo/2019/02/Tabla-CÃşdigos-Dane.pdf`[33].

[33] (Fondo de Pensiones Públicas del Nivel Nacional (FOPEP): *Tabla de Códigos DANE*. Recuperado el

6.18.1. Tabla “Fallecidos” del departamento de Antioquia

Para una mejor comprensión de la tabla, previamente descargada y explicada en la Sección 6.15, se detallará el significado de cada una de sus columnas.

- **TIPO_REG**: Tipo de registro. Indica el tipo de reporte o la clasificación del fallecimiento.
- **U_DPTO**: Código del departamento. Representa el departamento geográfico donde ocurrió el evento.
- **U_MPIO**: Código del municipio. Identifica el municipio específico.
- **UA_CLASE**: Clase de área. Distingue entre áreas urbanas y rurales.
- **COD_ENCUESTAS**: Código de encuesta. Número único para identificar encuestas o registros específicos.
- **U_VIVIENDA**: Número de orden de la Vivienda.
- **F_NROHOG**: Número de hogar. Indica un hogar específico dentro de la vivienda.
- **FA1_NRO_FALL**: Número persona fallecida.
- **FA2_SEXO_FALL**: Sexo del fallecido. Se codifica como 1 para masculino y 2 para femenino.(9-no informa).
- **FA3_EDAD_FALL**: Edad del fallecido. Indica la edad de la persona fallecida.(999 no informa).

---

26 de marzo de 2025. 2019. https://www.fopep.gov.co/sheempoo/2019/02/Tabla-C%C3%B3digos-Dane.pdf)

- **FA4_CERT_DEFUN**: Certificación de defunción. (1-Si 2-No 3-No sabe 9-No informa).

Con la tabla `Fallecidos_Antioquia` cargada, se pueden realizar consultas empleando comandos SQL en Python y aplicar los modificadores explicados en la Sección 6.8 A continuación, se presentan algunos ejemplos prácticos de consultas que se pueden realizar sobre la tabla `Fallecidos_Antioquia`. Cada uno de estos ejemplos deben ejecutarse en una nueva celda de Jupyter.

Ejemplo. 77. De esta consulta se obtiene el promedio de edad de los fallecidos (excluyendo registros sin información).

```
cursor.execute("""
SELECT AVG(FA3_EDAD_FALL) AS edad_promedio
FROM Fallecidos_Antioquia
WHERE FA3_EDAD_FALL != 999;
""")
resultado = cursor.fetchone()
print(f"Edad promedio de los fallecidos: {resultado[0]:.2f} años")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `AVG` (Subsección 6.8.15) y `WHERE` (Subsección 6.8.2).

Ejemplo. 78. De esta consulta se obtiene la cantidad total de fallecidos por sexo.

```
cursor.execute("""
    SELECT
        CASE
            WHEN FA2_SEXO_FALL = 1 THEN 'Masculino'
            WHEN FA2_SEXO_FALL = 2 THEN 'Femenino'
            ELSE 'No informa'
```

```
        END AS sexo,
        SUM(FA1_NRO_FALL) AS total_fallecidos
    FROM Fallecidos_Antioquia
    GROUP BY FA2_SEXO_FALL;
""")

for fila in cursor.fetchall():
    print(f"{fila[0]}: {fila[1]} fallecidos")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13), `CASE` (Subsección 6.8.21) y `GROUP BY` (Subsección 6.8.19).

Ejemplo. 79. De esta consulta se obtiene el número total de personas fallecidas sin certificado.

```
cursor.execute("""
    SELECT SUM(FA1_NRO_FALL)
    FROM Fallecidos_Antioquia
    WHERE FA4_CERT_DEFUN = 2;
""")
resultado = cursor.fetchone()
print(f"Número total de personas fallecidas sin certificado: {resultado[0]}")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 80. De esta consulta se obtienen los municipios junto con el número total de fallecidos.

```
import pandas as pd
cursor.execute("""
    SELECT U_MPIO, SUM(FA1_NRO_FALL) AS total_fallecidos
```

```
    FROM Fallecidos_Antioquia
    GROUP BY U_MPIO
""")

resultados = cursor.fetchall()
df = pd.DataFrame(resultados, columns=['Codigo del Municipio', 'Fallecidos
    Totales'])
df = df.sort_values(by='Fallecidos Totales', ascending=False)
pd.set_option('display.max_rows', len(df))

df
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `SUM` (Subsección 6.8.14) y `GROUP BY` (Subsección 6.8.19).

Ejemplo. 81. Mediante un mapa de calor, se representa gráficamente la distribución de fallecimientos por municipio en el departamento de Antioquia.

Es importante destacar que, para ejecutar este ejemplo, es necesario descargar previamente un archivo en formato GeoJSON. Un archivo GeoJSON contiene datos geoespaciales que permiten representar información geográfica de manera detallada y estructurada, y es fundamental para poder generar la visualización geográfica presentada en este ejemplo.

Para ejecutar el ejemplo, se deben seguir los siguientes pasos:

1. Acceder al enlace: `https://data.opendatasoft.com/explore/dataset/shapes%40bogota-laburbano/export/?refine.nombre_dpt=ANTIOQUIA&location=7,7`.

17528,-75.5226&basemap=jawg.streets[34].

2. Para acceder a la información completa en formato **GeoJSON**, es necesario hacer clic en la opción `Whole dataset`, ubicada junto a la etiqueta **GeoJSON**. Esto permitirá descargar el conjunto de datos en su totalidad, en lugar de una selección limitada de registros.

3. Guarde el archivo **"shapes@bogota-laburbano.geojson"** en una carpeta o ruta de fácil acceso para facilitar su carga.

4. En una nueva celda de Jupyter ejecute el siguiente código:

```
import tkinter as tk
from tkinter.filedialog import askopenfilename

import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt

def seleccionar_archivo_geojson():
    root = tk.Tk()
    root.withdraw()
    return askopenfilename(
        filetypes=[("GeoJSON files", "*.geojson")],
        title="Seleccione el archivo GeoJSON para visualizar"
    )
```

[34] (Opendatasoft: *Shapes — Bogotá Lab Urbano*. Recuperado el 26 de marzo de 2025. 2023. https://data.opendatasoft.com/explore/dataset/shapes%40bogota-laburbano/export/?refine.nombre_dpt=ANTIOQUIA&location=7,7.17528,-75.5226&basemap=jawg.streets)

```
try:
    archivo_geojson = seleccionar_archivo_geojson()
    if not archivo_geojson:
        raise FileNotFoundError(
            "No se seleccionó ningún archivo GeoJSON."
        )

    mapa = gpd.read_file(archivo_geojson)

    columna_dpt = next(
        (col for col in ['nombre_dpt', 'dpto']
         if col in mapa.columns),
        None
    )
    if not columna_dpt:
        raise KeyError(
            "No se encontró una columna adecuada para "
            "identificar el departamento de Antioquia."
        )

    mapa_antioquia = mapa[
        mapa[columna_dpt].str.contains(
            "Antioquia", case=False, na=False
        )
    ].copy()

    query = (
        "SELECT U_MPIO, FA1_NRO_FALL "
```

```
    "FROM Fallecidos_Antioquia"
)
cursor.execute(query)
df = pd.DataFrame(
    cursor.fetchall(),
    columns=['U_MPIO', 'FA1_NRO_FALL']
)

df['FA1_NRO_FALL'] = pd.to_numeric(
    df['FA1_NRO_FALL'],
    errors='coerce'
).fillna(0)

df['U_MPIO'] = df['U_MPIO'].astype(str).str.zfill(3)

fallecidos_por_municipio = df.groupby(
    'U_MPIO', as_index=False
)['FA1_NRO_FALL'].sum()

mapa_antioquia['mpio'] = (
    mapa_antioquia['mpio']
    .astype(str).str.zfill(3)
)

mapa_antioquia = mapa_antioquia.merge(
    fallecidos_por_municipio,
    left_on='mpio',
    right_on='U_MPIO',
    how='left'
```

```
).fillna({'FA1_NRO_FALL': 0})

mapa_antioquia['log_fallecidos'] = np.log1p(
    mapa_antioquia['FA1_NRO_FALL']
)

fig, ax = plt.subplots(figsize=(12, 10))

mapa_antioquia.plot(
    column='log_fallecidos',
    cmap='Reds',
    linewidth=0.8,
    edgecolor='black',
    legend=True,
    ax=ax,
    legend_kwds={
        'label': "Log del Número de Fallecimientos",
        'orientation': "vertical",
        'shrink': 0.5
    }
)

for _, row in mapa_antioquia.iterrows():
    centroid = row['geometry'].centroid
    plt.text(
        centroid.x,
        centroid.y,
        str(row['mpio']),
        ha='center',
```

```
                fontsize=6,
                color='black',
                weight='bold'
            )

        plt.title(
            'Mapa de Calor de Fallecimientos '
            'por Municipio en Antioquia'
        )
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    except Exception as e:
        print(
            f"Error al generar el mapa de calor: {e}"
        )
```

5. Se abrirá la ventana del explorador de archivos, donde el usuario deberá seleccionar el archivo previamente guardado y hacer clic en la opción `Abrir`.

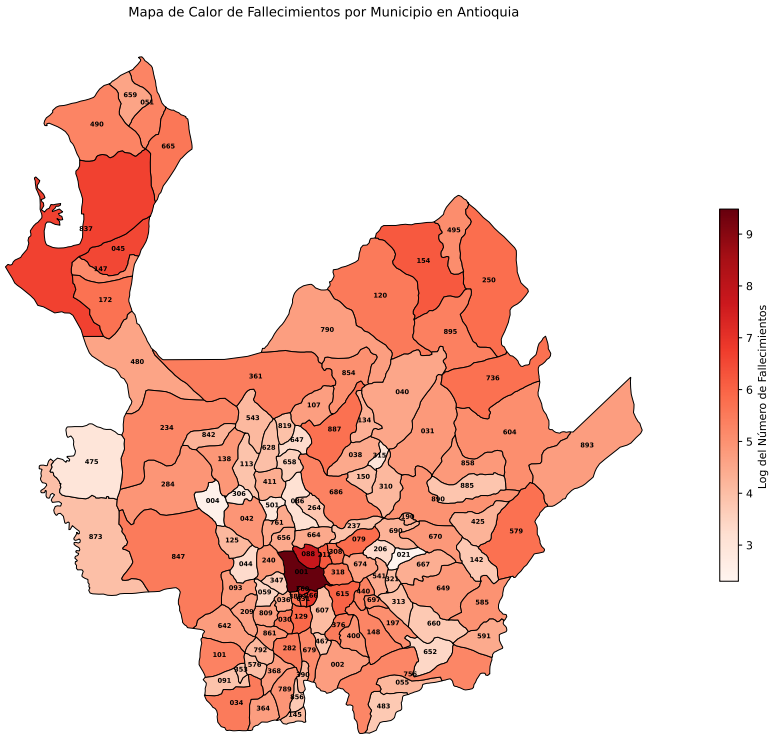

Figura 35. Mapa de calor de fallecimientos por municipio en Antioquia.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7).

El análisis del gráfico de mapa de calor evidencia que el municipio 001, que corresponde a Medellín, presenta la mayor concentración de fallecimientos en Antioquia, representado por la tonalidad más oscura en el mapa. Esto sugiere que Medellín, al ser la capital del departamento y la ciudad con mayor densidad poblacional, registra un número significativamente más alto de defunciones en comparación con otros municipios. En contraste, los municipios con tonalidades más claras muestran una menor incidencia de fallecimientos, lo que puede estar relacionado con una menor densidad poblacional o mejores condiciones sanitarias y de salud pública. Este análisis permite identificar patrones de mortalidad en el departamento y puede ser útil para la planificación de políticas de salud y asignación de recursos, priorizando intervenciones en las zonas con mayor impacto en la mortalidad.

Ejemplo. 82. Mediante un Mapa Autoorganizado (SOM), se representa gráficamente la distribución de fallecimientos en función de variables como el número de persona fallecida, el sexo, la edad y la clasificación del área (urbana o rural).

En una nueva celda de Jupyter ejecute el siguiente código:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from minisom import MiniSom


query = (
    "SELECT FA1_NRO_FALL, FA2_SEXO_FALL, FA3_EDAD_FALL, "
    "UA_CLASE, U_MPIO FROM Fallecidos_Antioquia"
)
cursor.execute(query)
resultados = cursor.fetchall()


df = pd.DataFrame(
    resultados,
    columns=[
        'FA1_NRO_FALL',
        'FA2_SEXO_FALL',
        'FA3_EDAD_FALL',
        'UA_CLASE',
        'U_MPIO'
    ]
)
```

```
df['FA1_NRO_FALL'] = pd.to_numeric(
    df['FA1_NRO_FALL'], errors='coerce'
).fillna(0)

df['FA2_SEXO_FALL'] = pd.to_numeric(
    df['FA2_SEXO_FALL'], errors='coerce'
).replace(9, np.nan).fillna(0)

df['FA3_EDAD_FALL'] = pd.to_numeric(
    df['FA3_EDAD_FALL'], errors='coerce'
).replace(999, np.nan)

df = df.dropna(subset=['FA3_EDAD_FALL'])
df.loc[
    df['FA3_EDAD_FALL'] > 120,
    'FA3_EDAD_FALL'
] = np.nan

df['FA3_EDAD_FALL'] = df['FA3_EDAD_FALL'].fillna(
    df['FA3_EDAD_FALL'].mean(skipna=True)
)

df['UA_CLASE'] = pd.to_numeric(
    df['UA_CLASE'], errors='coerce'
).fillna(0)

scaler = MinMaxScaler()
data = scaler.fit_transform(df)
```

```
som_shape = (10, 10)
som = MiniSom(
    som_shape[0],
    som_shape[1],
    data.shape[1],
    sigma=1.0,
    learning_rate=0.5
)

som.random_weights_init(data)
som.train_random(data, 1000)

plt.figure(figsize=(10, 10))
plt.pcolor(
    som.distance_map().T,
    cmap='Blues'
)
plt.colorbar(label='Distancia')

for i, x in enumerate(data):
    w = som.winner(x)
    plt.plot(
        w[0] + 0.5,
        w[1] + 0.5,
        'o',
        markerfacecolor='None',
        markeredgecolor='red',
        markersize=10,
        markeredgewidth=2
```

```
    )

plt.title(
    "Mapa Auto Organizado (SOM) "
    "de Fallecimientos por Municipio"
)
plt.axis('off')
plt.show()
```

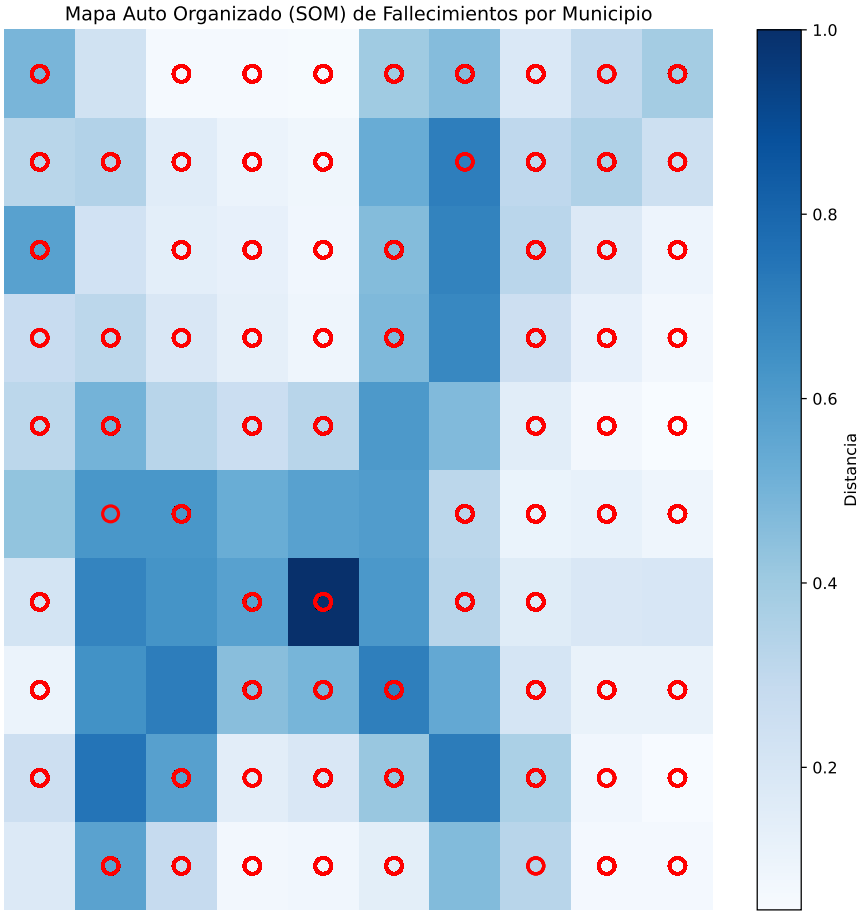

Figura 36. Mapa Autoorganizado (SOM) de municipios basado en características de fallecimientos.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7).

El objetivo del análisis es encontrar patrones en los datos que permitan entender mejor las diferencias territoriales y demográficas asociadas a los fallecimientos en la región. Por ejemplo, podrían existir municipios donde predominan los fallecimientos en zonas rurales con edades avanzadas, mientras que en otros los casos ocurren en zonas urbanas o con características diferentes. Este tipo de hallazgos puede ser útil para diseñar políticas públicas, asignar recursos sanitarios o detectar comportamientos atípicos.

Un **Mapa Autoorganizado (Self-Organizing Map, SOM)** es una técnica de aprendizaje no supervisado diseñada para proyectar datos de alta dimensión en un espacio bidimensional, conservando la estructura topológica de los datos. Es decir, registros similares en el conjunto de datos original tienden a ubicarse en celdas cercanas dentro del mapa. Cada celda (o nodo) representa un prototipo que resume características comunes de los registros que se agrupan en ese punto.

Mediante un Mapa Autoorganizado (SOM), se representa gráficamente esta proyección, donde cada celda de una cuadrícula bidimensional representa un nodo del mapa, y los registros similares se agrupan en regiones cercanas.

- Los colores representan la distancia entre los nodos vecinos del SOM.
- Los círculos rojos indican la ubicación de los registros en el SOM, es decir, cada círculo representa un municipio o conjunto de municipios con características similares.

El Mapa de Distancia U (U-Matrix) utiliza una codificación de colores, donde la intensidad del color indica la distancia entre los nodos:

- **Colores claros (blanco, azul claro)**: Representan **zonas homogéneas**, donde los

registros asignados tienen **características similares**. Municipios con **patrones de fallecimientos parecidos** aparecen agrupados en estas áreas.

- **Colores oscuros (azul oscuro, negro)**: Indican **zonas de alta distancia entre nodos**, es decir, **fronteras entre grupos distintos de municipios**. Representan municipios con características **muy diferentes**, como diferencias significativas en edad promedio, proporción de fallecimientos urbanos/rurales, o tasas de certificación de defunción.

Cada **círculo rojo** representa un municipio en el SOM, ubicado en el nodo que mejor lo representa.

- Si varios municipios están en el mismo nodo (círculos rojos superpuestos), significa que **tienen características muy parecidas**, como edad de fallecimiento, sexo de los fallecidos, tipo de área urbana/rural o certificación de defunción.
- Si un nodo contiene **pocos o un solo círculo rojo**, indica que esos municipios tienen **patrones únicos o poco comunes**.

Además de la visualización del Mapa Autoorganizado (SOM), es útil realizar un análisis cuantitativo que permita entender cómo se distribuyen los registros —en este caso, los municipios— dentro de la cuadrícula del SOM. Para ello, se implementa un bloque de código en Python que contabiliza cuántos municipios únicos han sido asignados a cada nodo del mapa.

Cada vector de datos (correspondiente a un municipio) es proyectado sobre el SOM, y se registra el nodo (coordenada fila, columna) que le corresponde. Luego, se agrupan los municipios según su nodo ganador y se construye una tabla que muestra:

- El total de nodos del SOM.

- Cuántos nodos están ocupados por al menos un municipio.
- Cuántos nodos están vacíos, es decir, no representan a ningún municipio.
- La cantidad de municipios únicos asignados a cada nodo ocupado.

Este análisis tiene varios propósitos:

- Detectar **nodos sobrecargados**, donde se agrupan muchos municipios con características similares. Esto puede indicar patrones demográficos o territoriales comunes.
- Identificar **nodos vacíos**, lo cual puede interpretarse como regiones del espacio de características que no corresponden a ningún municipio real, o que representan combinaciones de variables poco frecuentes.
- Localizar **nodos con pocos municipios (incluso uno solo)**, lo cual puede señalar comportamientos atípicos o únicos en ciertos municipios.

Este tipo de análisis complementa la visualización de la U-Matrix, brindando una perspectiva más estructurada y cuantificable sobre cómo los municipios se agrupan en el SOM, facilitando la interpretación de los patrones detectados en los datos.

```
import pandas as pd


nodos_municipios = {}


for i, x in enumerate(data):
    w = som.winner(x)
    municipio = df.iloc[i]['U_MPIO'] if 'U_MPIO' in df.columns else "Desconocido"
```

```
        if w in nodos_municipios:
            nodos_municipios[w].add(municipio)
        else:
            nodos_municipios[w] = {municipio}

nodos_municipios_df = pd.DataFrame(
    [(nodo, len(municipios)) for nodo, municipios in nodos_municipios.items()],
    columns=["Nodo SOM (Fila, Columna)", "Cantidad de Municipios Únicos"]
)

total_nodos = som_shape[0] * som_shape[1]
nodos_ocupados = len(set(nodos_municipios.keys()))
nodos_vacios = total_nodos - nodos_ocupados

print(f"Total de nodos en el SOM: {total_nodos}")
print(f"Nodos ocupados con registros: {nodos_ocupados}")
print(f"Nodos vacíos sin registros: {nodos_vacios}")

pd.set_option('display.max_rows', len(nodos_municipios_df))
nodos_municipios_df
```

Al finalizar el proceso de conexión a la base de datos, es esencial cerrar tanto el cursor como la conexión. Esto es importante porque permite liberar los recursos utilizados y evitar problemas de rendimiento o posibles bloqueos en futuras conexiones.

```
if connection.is_connected():
    cursor.close()
    connection.close()
    print("Conexión cerrada.")
```

El código anterior debe ejecutarse en una nueva celda de Jupyter al terminar de trabajar con la tabla.

Después de cerrar la conexión con MySQL, se recomienda presionar `Ctrl + S` para guardar los cambios y luego cerrar el archivo. Si posteriormente se desea volver a trabajar en el notebook `Fallecidos_Antioquia`, no es necesario compilar nuevamente todo el proyecto. Basta con seguir los siguientes pasos:

1. Compilar el bloque de código encargado de establecer la conexión con MySQL, descrito en la Subsección 6.16.2.
2. Compilar el bloque de código encargado de conectarse a la base de datos, explicado en la Subsección 6.16.4.

Una vez realizados estos pasos, se podrán ejecutar nuevamente las consultas previamente realizadas, así como agregar nuevas consultas a la base de datos.

6.18.2. Tabla “Hogares” del departamento de Magdalena

Para este ejemplo, se utilizarán los registros correspondientes al Departamento del Magdalena. Para acceder a ellos, se deben seguir los siguientes pasos.

1. Acceder a la carpeta `departamentos`, la cual fue creada en el paso 4 de la Subsección 6.16.1.
2. Una vez dentro de la carpeta, volver a hacer clic en el botón **New** ubicado en la esquina superior derecha y seleccionar la opción **Notebook** (Python 3) para crear un nuevo cuaderno de trabajo.

3. Cambiar el nombre del cuaderno (notebook) haciendo clic sobre el título predeterminado (por ejemplo, `Untitled`) en la parte superior del cuaderno, y renombrarlo como `Hogares_Magdalena`. Luego, hacer clic en **Rename** para guardar el nuevo nombre.

4. Repetir los pasos necesarios para establecer la conexión con MySQL, según lo explicado en la Subsección 6.16.2.

5. Repetir los pasos necesarios para realizar la conexión a la Base de Datos `Departamentos` , según lo explicado en la Subsección 6.16.4.

6. Cree una nueva tabla para almacenar los datos correspondientes al Departamento del Magdalena. En este caso, la tabla se denominará `Hogares_Magdalena`.

```
try:
    cursor = connection.cursor()


    cursor.execute("""
    CREATE TABLE IF NOT EXISTS Hogares_Magdalena (
    id INT AUTO_INCREMENT PRIMARY KEY
    )
    """)
    print("Tabla 'Hogares_Magdalena' creada exitosamente")


except Error as e:
    print(f"Error al crear la tabla: {e}")
```

7. Repetir los pasos explicados en la Subsección 6.17.2.

8. Repetir los pasos explicados en la Subsección 6.17.3, cambiando el nombre de la tabla por `Hogares_Magdalena`:

```
nombre_tabla = "Hogares_Magdalena"
```

9. Acceder al enlace previamente mencionado, Base de datos del Censo Nacional de Población y Vivienda 2018, donde se encuentra un listado de departamentos junto con un botón de descarga para una carpeta en formato ZIP.
10. Hacer clic en el botón `Descargar`, lo que iniciará la descarga de una carpeta que contiene los datos del departamento de Magdalena.
11. Una vez finalizada la descarga, abrir la carpeta ZIP, dentro de la cual se encuentran tres subcarpetas. Ingresar a la carpeta denominada **"47_Magdalena_CSV"**.
12. En esta carpeta se encuentran cinco archivos en formato CSV. Se debe descargar el archivo denominado **"CNPV2018_2HOG_A2_47"**.
13. Guarde el archivo en una carpeta o ruta de fácil acceso para facilitar su carga en la tabla de MySQL. Es importante no modificar la configuración del archivo.
14. Repetir los pasos explicados en la Subsección 6.17.4.

Para una mejor comprensión de la tabla, se detallará el significado de cada una de sus columnas.

- **TIPO_REG**: Muestra el tipo de registro del hogar.
- **U_DPTO**: Código del departamento. En este caso, se asocia el número 47 al departamento del Magdalena.
- **U_MPIO**: Código del municipio. Identifica un municipio en específico del Magdalena.
- **UA_CLASE**: Clase de área. Clasifica los hogares con 1 y 4 para Cabecera municipal y resto respectivamente.
- **COD_ENCUESTAS**: Código de encuesta. Número único para identificar y diferenciar las encuestas hechas en los hogares.

- **U_VIVIENDA**: Número de orden de la Vivienda.
- **H_NROHOG**: Indica el número de hogar en la vivienda.
- **H_NRO_CUARTOS**: Numera la cantidad total de cuartos en el hogar. Se cuenta desde el número 1 hasta el 20 para mostrar la cantidad de cuartos, se usa 99 para los hogares que no informan.
- **H_NRO_DORMIT**: Indica la cantidas de cuartos para dormir. Se numera igual que el anterior ítem.
- **H_DONDE_PREPALIM**: Informa el lugar donde se preparan los alimentos de la siguiente manera.
    - 1 - En un cuarto usado solo para cocinar.
    - 2 - En un cuarto usado también para dormir.
    - 3 - En una sala-comedor con lavaplatos.
    - 4 - En una sala-comedor sin lavaplatos.
    - 5 - En un patio, corredor, enramada o al aire libre.
    - 6 - No preparan alimentos en la vivienda.
    - 9 - No informa.
- **H_AGUA_COCIN**: Muestra la fuente principal para obtener el agua para cocinar con la siguiente lista.
    - 1 - Acueducto público.
    - 2 - Acueducto veredal.
    - 3 - Red de distribución comunitaria.
    - 4 - Pozo con bomba.

- 5 - Pozo sin bomba, aljibe, jaguey o barreno.
- 6 - Agua lluvia.
- 7 - Río, quebrada, manantial, nacimiento.
- 8 - Pila pública.
- 9 - Carrotanque.
- 10 - Aguatero.
- 11 - Agua embotellada o en bolsa.
- 12 - no preparan alimentos.
- 99 - No informa.

- **HA_NRO_FALL**: Informa la cantidad de fallecidos en el hogar hasta 2017. Numera desde 0 hasta 20 personas.
- **HA_TOT_PER**: Indica el número total de personas en el hogar. Numera desde 0 hasta 40 personas.

Con la tabla `Hogares_Magdalena` cargada, se pueden realizar consultas empleando comandos SQL en Python y aplicar los modificadores explicados en la Sección 6.8. A continuación, se presentan algunos ejemplos prácticos de consultas que se pueden realizar sobre la tabla `Hogares_Magdalena`. Cada uno de estos ejemplos debe ejecutarse en una nueva celda de Jupyter.

Ejemplo. 83. De esta consulta se obtiene la media de personas en cada hogar del departamento del Magdalena.

```
cursor.execute(
    "SELECT AVG(HA_TOT_PER) AS media_personas "
    "FROM Hogares_Magdalena "
    "WHERE U_DPTO = 47;"
```

```
)
resultado = cursor.fetchone()
print(
    f"La media de personas en cada hogar del "
    f"departamento del Magdalena es: {resultado[0]:.2f}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `AVG` (Subsección 6.8.15) y `WHERE` (Subsección 6.8.2).

Ejemplo. 84. De esta consulta se obtiene el municipio del Magdalena más consultado, es decir, el municipio con mayor número de encuestas respondidas.

```
cursor.execute(
    "SELECT U_MPIO, COUNT(*) AS total_encuestas "
    "FROM Hogares_Magdalena "
    "WHERE U_DPTO = 47 "
    "GROUP BY U_MPIO "
    "ORDER BY total_encuestas DESC "
    "LIMIT 1;"
)

resultado = cursor.fetchone()

print(
    f"El municipio del Magdalena con mayor número de "
    f"encuestas respondidas es: {resultado[0]}, con "
    f"{resultado[1]} encuestas."
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13), `WHERE` (Subsección 6.8.2), `GROUP BY` (Subsección 6.8.19), `ORDER BY` (Subsección 6.8.3) y `LIMIT` (Subsección 6.8.8).

Ejemplo. 85. De esta consulta se obtiene el municipio con la mayor cantidad de hogares que no preparan alimentos en la vivienda, permitiendo identificar zonas donde esta condición es más frecuente.

```
cursor.execute(
    "SELECT U_MPIO, COUNT(*) AS hogares_sin_cocina "
    "FROM Hogares_Magdalena "
    "WHERE H_DONDE_PREPALIM = 6 "
    "GROUP BY U_MPIO "
    "ORDER BY hogares_sin_cocina DESC "
    "LIMIT 1;"
)

resultado = cursor.fetchone()

print(
    f"El municipio con más hogares que no preparan "
    f"alimentos en la vivienda es: {resultado[0]}, con "
    f"un total de {resultado[1]} hogares en esta condición."
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13), `WHERE` (Subsección 6.8.2), `GROUP BY` (Subsección 6.8.19), `ORDER BY` (Subsección 6.8.3) y `LIMIT` (Subsección 6.8.8).

Ejemplo. 86. De esta consulta se obtiene la media de cuartos de un hogar cualquiera ubicado en la cabecera municipal.

```
cursor.execute(
    "SELECT AVG(H_NRO_CUARTOS) AS media_cuartos "
    "FROM Hogares_Magdalena "
    "WHERE UA_CLASE = 1;"
)

resultado = cursor.fetchone()

print(
    f"Media de cuartos en hogares ubicados en la "
    f"cabecera municipal: {resultado[0]:.2f}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `AVG` (Subsección 6.8.15) y `WHERE` (Subsección 6.8.2).

Ejemplo. 87. De esta consulta se obtiene información sobre cuántos registros hay de personas fallecidas en los hogares.

```
cursor.execute(
    "SELECT COUNT(*) AS total_fallecidos "
    "FROM Hogares_Magdalena "
    "WHERE H_NRO_FALL > 0;")

resultado = cursor.fetchone()

print(
    f"Cantidad de registros de personas fallecidas "
    f"en los hogares: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 88. Mediante un gráfico de dispersión, se representa gráficamente la relación entre el número de cuartos en el hogar y el número total de personas que lo habitan.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

cursor.execute("""
    SELECT H_NRO_CUARTOS, HA_TOT_PER
    FROM Hogares_Magdalena
    WHERE H_NRO_CUARTOS BETWEEN 1 AND 20
      AND HA_TOT_PER BETWEEN 1 AND 40;
""")
datos = cursor.fetchall()

df = pd.DataFrame(
    datos,
    columns=['H_NRO_CUARTOS', 'HA_TOT_PER']
)

if df.empty:
    print("No hay datos válidos para graficar.")
else:
    df['H_NRO_CUARTOS'] = pd.to_numeric(
        df['H_NRO_CUARTOS']
    )
    df['HA_TOT_PER'] = pd.to_numeric(
        df['HA_TOT_PER']
```

```
)

sns.set_theme(
    style='whitegrid',
    palette='muted'
)

plt.figure(figsize=(10, 6))

sns.scatterplot(
    data=df,
    x='H_NRO_CUARTOS',
    y='HA_TOT_PER',
    color='#4C72B0',
    s=50,
    alpha=0.7,
    edgecolor='white'
)

plt.xlabel(
    'Número de Cuartos en el Hogar',
    fontsize=14
)
plt.ylabel(
    'Número Total de Personas',
    fontsize=14
)
plt.title(
    'Relación entre Cuartos y Personas por Hogar',
```

```
    fontsize=16,
    fontweight='bold'
)

plt.show()
```

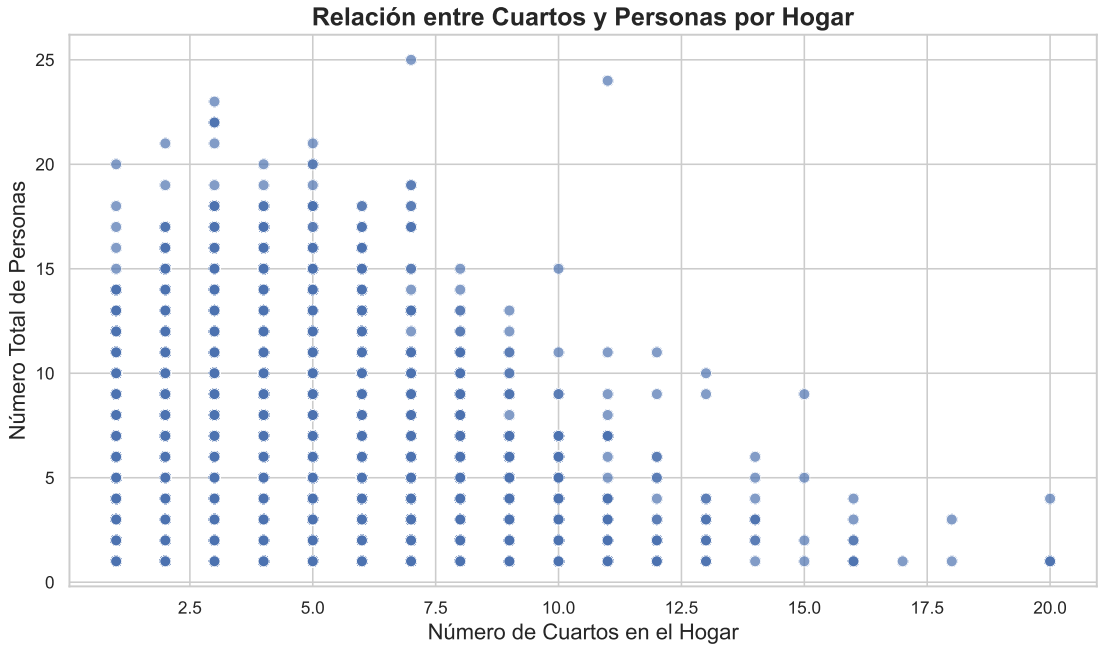

Figura 37. Relación entre el número de cuartos en el hogar y el número total de personas que lo habitan.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `WHERE` (Subsección 6.8.2).

El análisis del gráfico de dispersión evidencia una relación inversa entre el número de cuartos en el hogar y la cantidad de personas que lo habitan. Se observa que los hogares con menor cantidad de cuartos, particularmente entre 1 y 5, concentran un mayor número de personas, lo que sugiere posibles condiciones de hacinamiento. En contraste, a

medida que aumenta el número de cuartos, la cantidad de ocupantes tiende a disminuir, identificándose viviendas con más de 10 cuartos que albergan a un número reducido de personas. Asimismo, se destacan casos de hogares con un alto número de cuartos y una baja densidad de ocupantes, lo que indica la existencia de espacios subutilizados. Estos resultados permiten inferir desigualdades en la distribución del espacio habitacional y pueden servir como insumo para el diseño de políticas de vivienda orientadas a mejorar las condiciones de habitabilidad y mitigar el hacinamiento.

Ejemplo. 89. Mediante un gráfico de líneas, se representa gráficamente la evolución del número promedio de personas en los hogares por municipio.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


cursor.execute("""
    SELECT U_MPIO, HA_TOT_PER
    FROM Hogares_Magdalena
    WHERE HA_TOT_PER IS NOT NULL
      AND HA_TOT_PER <> ''
      AND CAST(HA_TOT_PER AS DECIMAL(10,2))
          BETWEEN 1 AND 40;
""")
datos = cursor.fetchall()


df_evolucion = pd.DataFrame(
    datos,
    columns=['Municipio', 'Numero_Personas']
)
```

```
df_evolucion['Numero_Personas'] = pd.to_numeric(
    df_evolucion['Numero_Personas'],
    errors='coerce'
)

df_evolucion = df_evolucion.groupby(
    'Municipio',
    as_index=False
)['Numero_Personas'].mean()

df_evolucion = df_evolucion.sort_values(
    by='Numero_Personas'
)

if not df_evolucion.empty:
    sns.set_theme(
        style="whitegrid",
        palette="muted"
    )
    plt.figure(figsize=(12, 6))

    sns.lineplot(
        data=df_evolucion,
        x='Municipio',
        y='Numero_Personas',
        marker='o',
        color='#4C72B0'
    )
```

```
    plt.xlabel(
        'Municipio',
        fontsize=14
    )
    plt.ylabel(
        'Número Promedio de Personas por Hogar',
        fontsize=14
    )
    plt.title(
        'Evolución del Número de Personas en '
        'los Hogares por Municipio',
        fontsize=16,
        fontweight='bold'
    )

    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
    plt.close()

else:
    print("No hay datos disponibles para graficar.")
```

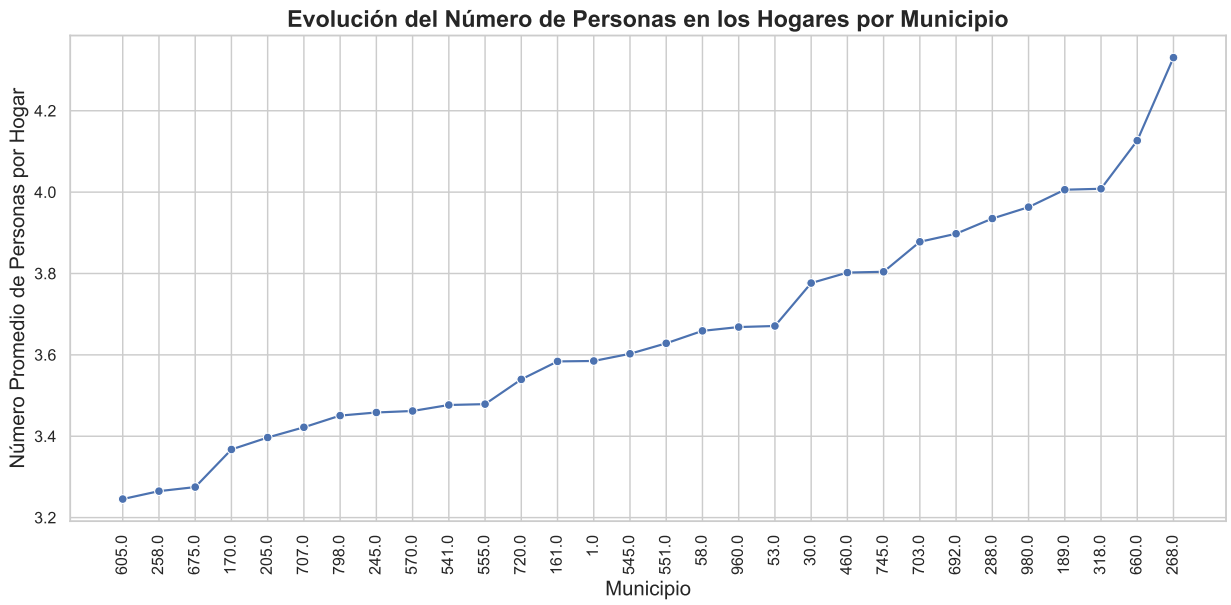

Figura 38. Evolución del número promedio de personas en los hogares por municipio en el Magdalena.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `WHERE` (Subsección 6.8.2).

El análisis del gráfico de líneas evidencia una variabilidad en la densidad de ocupación de los hogares en los municipios del departamento del Magdalena. Se observa una tendencia creciente en el número promedio de personas por hogar, lo que indica que ciertos municipios presentan una mayor cantidad de habitantes por vivienda, posiblemente debido a factores socioeconómicos o demográficos. Los municipios con promedios superiores a cuatro personas por hogar podrían reflejar la presencia de familias numerosas o viviendas compartidas, mientras que aquellos con menor densidad de ocupación, con promedios cercanos a 3.2 personas por hogar, podrían estar asociados a estructuras familiares más pequeñas o una menor ocupación de las viviendas. Este análisis permite identificar patrones de ocupación habitacional y es fundamental para la planificación de políticas públicas en materia de vivienda y distribución de recursos.

6.18.3. Tabla “Personas” del departamento de La Guajira

Para este ejemplo, se utilizarán los registros correspondientes al Departamento de La Guajira. Para acceder a ellos, se deben seguir los siguientes pasos.

1. Acceder a la carpeta `departamentos`, la cual fue creada en el paso 4 de la Subsección 6.16.1.

2. Una vez dentro de la carpeta, volver a hacer clic en el botón `New` ubicado en la esquina superior derecha y seleccionar la opción `Notebook` (Python 3) para crear un nuevo cuaderno de trabajo.

3. Cambiar el nombre del cuaderno (notebook) haciendo clic sobre el título predeterminado (por ejemplo, `Untitled`) en la parte superior del cuaderno, y renombrarlo como `Personas_Guajira`. Luego, hacer clic en `Rename` para guardar el nuevo nombre.

4. Repetir los pasos necesarios para establecer la conexión con MySQL, según lo explicado en la Subsección 6.16.2.

5. Repetir los pasos necesarios para realizar la conexión a la Base de Datos `Departamentos` , según lo explicado en la Subsección 6.16.4.

6. Cree una nueva tabla para almacenar los datos correspondientes al Departamento de La Guajira. En este caso, la tabla se denominará `Personas_Guajira`.

```
try:
    cursor = connection.cursor()


    cursor.execute("""
    CREATE TABLE IF NOT EXISTS Personas_Guajira (
    id INT AUTO_INCREMENT PRIMARY KEY
```

```
        )
        """)
        print("Tabla 'Personas_Guajira' creada exitosamente")

    except Error as e:
        print(f"Error al crear la tabla: {e}")
```

7. Repetir los pasos explicados en la Subsección 6.17.2.

8. Repetir los pasos explicados en la Subsección 6.17.3, cambiando el nombre de la tabla por `Personas_Guajira`:

```
nombre_tabla = "Personas_Guajira"
```

9. Acceder al enlace previamente mencionado, Base de datos del Censo Nacional de Población y Vivienda 2018, donde se encuentra un listado de departamentos junto con un botón de descarga para una carpeta en formato ZIP.

10. Hacer clic en el botón `Descargar`, lo que iniciará la descarga de una carpeta que contiene los datos del Departamento de La Guajira.

11. Una vez finalizada la descarga, abrir la carpeta ZIP, dentro de la cual se encuentran tres subcarpetas. Ingresar a la carpeta denominada **"44_LaGuajira_CSV"**.

12. En esta carpeta se encuentran cinco archivos en formato CSV. Se debe descargar el archivo denominado **"CNPV2018_5PER_A2_44"**.

13. Guarde el archivo en una carpeta o ruta de fácil acceso para facilitar su carga en la tabla de MySQL. Es importante no modificar la configuración del archivo.

14. Repetir los pasos explicados en la Subsección 6.17.3.

Para facilitar la comprensión de la tabla, se explicará el significado de cada una de sus columnas.

- **TIPO_REG**: Tipo de registro.
- **U_DPTO**: Departamento.
- **U_MPIO**: Municipio.
- **UA_CLASE**: Clase.
- **U_EDIFICA**: Número de orden de la Edificación.
- **COD_ENCUESTAS**: Código Encuesta.
- **U_VIVIENDA**: Número de orden de la Vivienda.
- **P_NROHOG**:Número de orden del Hogar dentro de la vivienda.
- **P_NRO_PER**: Número de personas en el hogar.
- **P_SEXO**: Sexo.
- **P_EDADR**: Edad en Grupos Quinquenales.
    - 1 - 00 A 04 Años.
    - 2 - 05 A 09 Años.
    - 3 - 10 A 14 Años.
    - 4 - 15 A 19 Años.
    - 5 - 20 A 24 Años.
    - 6 - 25 A 29 Años.
    - 7 - 30 A 34 Años.

- 8 - 35 A 39 Años.
- 9 - 40 A 44 Años.
- 10 - 45 A 49 Años.
- 11 - 50 A 54 Años.
- 12 - 55 A 59 Años.
- 13 - 60 A 64 Años.
- 14 - 65 A 69 Años.
- 15 - 70 A 74 Años.
- 16 - 75 A 79 Años.
- 17 - 80 A 84 Años.
- 18 - 85 A 89 Años.
- 19 - 90 A 94 Años.
- 20 - 95 A 99 Años.
- 21 - 100 y más Años.

- **P_PARENTESCOR**: Relación de parentesco con el jefe(a) del hogar (recodificada).
- **PA1_GRP_ETNIC**: Reconocimiento étnico.
- **PA11_COD_ETNIA**: Pueblo indígena de pertenencia.
- **PA12_CLAN**: Clan de pertenencia.
- **PA21_COD_VITSA**: Vitsa de pertenencia.
- **PA22_COD_KUMPA**: Kumpania de pertenencia.
- **PA_HABLA_LENG**: Habla la lengua nativa de su pueblo.

- **PA1_ENTIENDE**: Entiende la lengua nativa de su pueblo.
- **PB_OTRAS_LENG**: Habla otra(s) lengua(s) nativa(s).
- **PB1_QOTRAS_LENG**: Número de otra(s) lengua(s) nativa(s).
- **PA_LUG_NAC**: Lugar de nacimiento.
- **PA_VIVIA_5ANOS**: Lugar de residencia hace 5 años.
- **PA_VIVIA_1ANO**: Lugar de residencia hace 12 meses.
- **P_ENFERMO**: Algún problema de salud en los últimos 30 días, sin hospitalización.
- **P_QUEHIZO_PPAL**: Tratamiento principal del problema de salud.
- **PA_LO_ATENDIERON**: Atención del problema de salud.
- **PA1_CALIDAD_SERV**: Calidad de la prestación del servicio de salud.
- **CONDICION_FISICA**: Alguna dificultad en su vida diaria.
- **P_ALFABETA**: Sabe leer y escribir.
    - 1 - Si.
    - 2 - No.
    - 3 - No Aplica.
    - 4 - No Informa.
- **PA_ASISTENCIA**: Asistencia escolar (de forma presencial o virtual).
- **P_NIVEL_ANOSR**: Nivel educativo más alto alcanzado y último año o grado aprobado en ese nivel (recodificado).
    - 1 - Preescolar.

    - 2 - Básica primaria.
    - 3 - Básica secundaria.
    - 4 - Media academica o clasica.
    - 5 - Media tecnica.
    - 6 - Normalista.
    - 7 - Técnica profesional o Tecnológica.
    - 8 - Universitario.
    - 9 - Especialización, maestría, doctorado.
    - 10 - Ninguno.
    - 99 - No Informa.
- **P_TRABAJO**: Que hizo durante la semana pasada.
- **P_EST_CIVIL**: Estado civil.
- **PA_HNV**: Ha tenido algún hijo(a) nacido vivo(a).
- **PA1_THNV**: Hijos(as) nacidos vivos: Total.
- **PA2_HNVH**: Hijos(as) nacidos vivos: Hombres.
- **PA3_HNVM**: Hijos(as) nacidos vivos: Mujeres.
- **PA_HNVS**: Hijos(as) sobrevivientes.
- **PA1_THSV**: Hijos(as) sobrevivientes: Total.
- **PA2_HSVH**: Hijos(as) sobrevivientes: Hombres.
- **PA3_HSVM**: Hijos(as) sobrevivientes: Mujeres.
- **PA_HFC**: Hijos(as) viven actualmente fuera de Colombia.

- **PA2_HFCH**: Hijos(as) viven actualmente fuera de Colombia: Hombres.
- **PA3_HFCM**: Hijos(as) viven actualmente fuera de Colombia: Mujeres.
- **PA_UHNV**: Nacimiento último hijo(a) nacido(a) vivo(a).
- **PA1_MES_UHNV**: Nacimiento último hijo(a) nacido(a) vivo(a): Mes.
- **PA2_ANO_UHNV**: Nacimiento último hijo(a) nacido(a) vivo(a): Año.

Con la tabla `Personas_Guajira` cargada, se pueden realizar consultas empleando comandos SQL en Python y aplicar los modificadores explicados en la Sección 6.8. A continuación, se presentan algunos ejemplos prácticos de consultas que se pueden realizar sobre la tabla `Personas_Guajira`. Cada uno de estos ejemplos debe ejecutarse en una nueva celda de Jupyter.

Ejemplo. 90. De esta consulta obtiene la edad promedio de las personas cuyo nivel educativo es “ninguno” de la columna `P_NIVEL_ANOSR`.

```
cursor.execute("SELECT AVG(P_EDADR) FROM Personas_Guajira WHERE P_NIVEL_ANOSR =
    10")
resultado = cursor.fetchone()
print(f"Edad promedio de personas sin educación: {resultado[0]:.2f}")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `AVG` (Subsección 6.8.15) y `WHERE` (Subsección 6.8.2).

Ejemplo. 91. De esta consulta se obtiene la cantidad de personas que tienen entre 20 a 39 años de la columna `P_EDADR`.

```
cursor.execute("SELECT COUNT(*) FROM Personas_Guajira WHERE P_EDADR BETWEEN 5
    AND 8")
resultado = cursor.fetchone()
print(f"Cantidad de personas entre 20 y 39 años: {resultado[0]}")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 92. De esta consulta se obtiene la cantidad de personas que no tienen ningún estudio de la columna `P_NIVEL_ANOSR`.

```
cursor.execute("SELECT COUNT(*) FROM Personas_Guajira WHERE P_NIVEL_ANOSR = 10")
resultado = cursor.fetchone()
print(f"Cantidad de personas sin ningún estudio: {resultado[0]}")
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 93. De esta consulta se obtiene la cantidad de personas que tienen más de 20 años y no tienen ningún estudio, de acuerdo con las columnas `P_NIVEL_ANOSR` y `P_EDADR`.

```
cursor.execute(
    "SELECT COUNT(*) "
    "FROM Personas_Guajira "
    "WHERE P_NIVEL_ANOSR = 10 "
    "AND P_EDADR >= 5"
)

resultado = cursor.fetchone()

print(
    f"Cantidad de personas mayores de 20 años "
    f"sin estudios: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 94. De esta consulta se obtiene la cantidad de personas cuyo nivel educativo más alto alcanzado es básica secundaria o media técnica, de acuerdo con la columna P_NIVEL_ANOSR.

```
cursor.execute(
    "SELECT COUNT(*) "
    "FROM Personas_Guajira "
    "WHERE P_NIVEL_ANOSR IN (3, 5)"
)

resultado = cursor.fetchone()

print(
    f"Cantidad de personas con nivel educativo de "
    f"básica secundaria o media técnica: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción SELECT (Sección 6.7) junto con COUNT(*) (Subsección 6.8.13) y WHERE (Subsección 6.8.2).

Ejemplo. 95. Mediante un gráfico de torta, se representa gráficamente la proporción de personas mayores de 15 años que saben o no saben leer y escribir del departamento de La Guajira.

```
import matplotlib.pyplot as plt

try:
    plt.style.use('seaborn-whitegrid')
except OSError:
    plt.style.use('ggplot')
```

```
cursor.execute("""
    SELECT
        SUM(CASE WHEN P_ALFABETA = 1 THEN 1 ELSE 0 END) AS sabe_leer_escribir,
        SUM(CASE WHEN P_ALFABETA = 2 THEN 1 ELSE 0 END) AS no_sabe_leer_escribir
    FROM Personas_Guajira
    WHERE P_EDADR > 15
      AND P_ALFABETA IN (1,2);
""")
resultado = cursor.fetchone()

labels = ['Sabe leer y escribir', 'No sabe leer y escribir']
sizes = [resultado[0], resultado[1]]
colors = ['#8A54E7', '#4BC2C7']
explode = (0.05, 0.05)

fig, ax = plt.subplots(figsize=(6,6))

wedges, texts, autotexts = ax.pie(
    sizes,
    labels=labels,
    colors=colors,
    autopct='%1.1f%%',
    startangle=140,
    explode=explode,
    shadow=True,
    wedgeprops={'edgecolor':'white'}
)

for text in texts:
```

```
    text.set_fontsize(12)
for autotext in autotexts:
    autotext.set_fontsize(12)
    autotext.set_color('white')
    autotext.set_weight('bold')

ax.set_title('Proporción de personas mayores de 15 años que saben o no leer y
    escribir',
    fontsize=14,
    fontweight='bold'
)

plt.tight_layout()
plt.show()
```

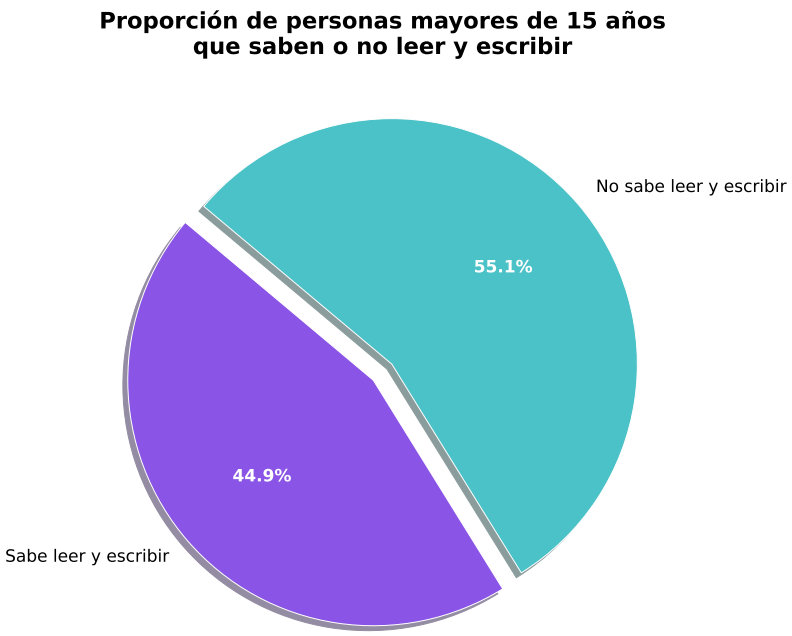

Figura 39. Proporción de personas mayores de 15 años que saben o no leer y escribir.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `SUM` (Subsección 6.8.14), `CASE` (Subsección 6.8.21) y `WHERE` (Subsección 6.8.2).

El análisis del gráfico de torta evidencia que una mayoría de la población mayor de 15 años no sabe leer ni escribir, representando el 55.1 % del total, mientras que el 44.9 % sí posee habilidades de lectoescritura. Esta distribución refleja una brecha significativa en el acceso a la educación básica, lo que puede estar relacionado con factores socioeconómicos, falta de infraestructura educativa, o barreras culturales y geográficas. La elevada proporción de personas analfabetas sugiere la necesidad de fortalecer políticas de alfabetización, promoviendo programas de educación para adultos y mejorando la cobertura educativa en las comunidades más vulnerables.

Ejemplo. 96. Mediante un gráfico de barras, se representa gráficamente la distribución del nivel educativo de la población del departamento de La Guajira.

```
import matplotlib.pyplot as plt
import math

try:
    plt.style.use('seaborn-whitegrid')
except OSError:
    plt.style.use('ggplot')

nivel_mapping = {
    1: "Preescolar",
    2: "Básica prim",
    3: "Básica sec",
    4: "Media académica",
    5: "Media técnica",
    6: "Normalista",
    7: "Técnica profesional",
    8: "Universitario",
    9: "Espec/Maest/Doc",
    10: "Ninguno",
    99: "No Informa"
}

cursor.execute("""
    SELECT P_NIVEL_ANOSR, COUNT(*) AS cantidad
    FROM Personas_Guajira
    GROUP BY P_NIVEL_ANOSR;
```

```
""")
resultados = cursor.fetchall()

if not resultados:
    print("No se encontraron datos para la consulta.")
else:
    codigos = [fila[0] for fila in resultados]
    cantidades = [fila[1] for fila in resultados]

    etiquetas = []
    for codigo in codigos:
        if codigo is None or (isinstance(codigo, float) and math.isnan(codigo)):
            etiquetas.append("Desconocido")
        else:
            try:
                codigo_float = float(codigo)
                codigo_int = int(codigo_float)
                etiqueta = nivel_mapping.get(codigo_int, str(codigo_int))
                etiquetas.append(etiqueta)
            except ValueError:
                etiquetas.append("Desconocido")


    fig, ax = plt.subplots(figsize=(10,6))

    bars = ax.bar(etiquetas, cantidades, color='#66b3ff', edgecolor='black')

    ax.set_xlabel('Nivel Educativo', fontsize=14)
    ax.set_ylabel('Cantidad de Personas', fontsize=14)
```

```
ax.set_title('Distribución de Nivel Educativo', fontsize=16,
    fontweight='bold')

plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)

for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height}',
                xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3),
                textcoords="offset points",
                ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.show()
```

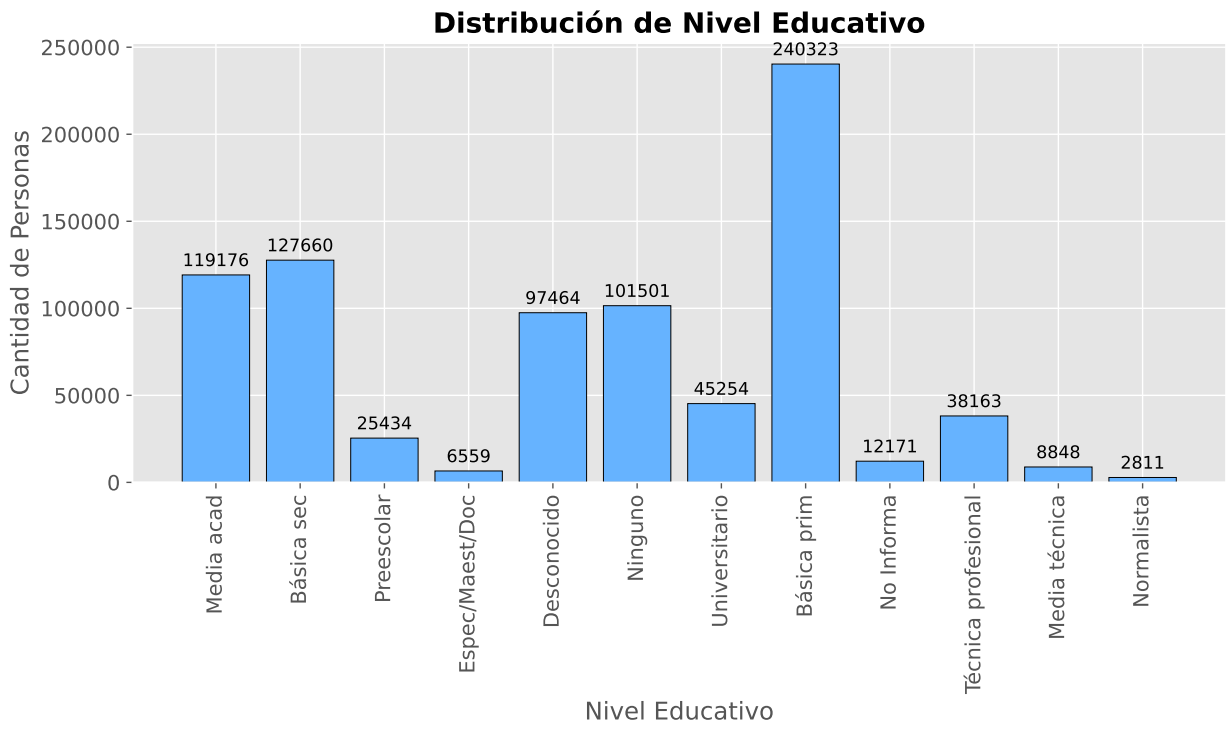

Figura 40. Distribución del nivel educativo de la población.

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con los `COUNT(*)` (Subsección 6.8.13) y `GROUP BY` (Subsección 6.8.19).

El análisis del gráfico de barras evidencia la distribución del nivel educativo de la población, resaltando que la mayoría de las personas han alcanzado el nivel de básica primaria, seguido por básica secundaria y media académica. Se observa que un número significativo de individuos no ha continuado con estudios superiores, ya que la cantidad de personas con nivel universitario o con estudios de especialización, maestría o doctorado es considerablemente menor en comparación con los niveles educativos básicos. Además, hay un grupo relevante de personas que no informa su nivel educativo o se encuentra en la categoría de “desconocido”, lo que puede indicar falta de datos o dificultades en la recopilación de información. Estos resultados pueden ser clave para la implementación de

políticas educativas, ya que reflejan la necesidad de fortalecer el acceso a la educación superior y técnica, así como reducir la brecha educativa en la población.

6.18.4. Tabla “Viviendas” del departamento de Santander

Para este ejemplo, se utilizarán los registros correspondientes al Departamento de Santander. Para acceder a ellos, se deben seguir los siguientes pasos.

1. Acceder a la carpeta `departamentos`, la cual fue creada en el paso 4 de la Subsección 6.16.1.
2. Una vez dentro de la carpeta, volver a hacer clic en el botón `New` ubicado en la esquina superior derecha y seleccionar la opción `Notebook` (Python 3) para crear un nuevo cuaderno de trabajo.
3. Cambiar el nombre del cuaderno (notebook) haciendo clic sobre el título predeterminado (por ejemplo, `Untitled`) en la parte superior del cuaderno, y renombrarlo como `Viviendas_Santander`. Luego, hacer clic en `Rename` para guardar el nuevo nombre.
4. Repetir los pasos necesarios para establecer la conexión con MySQL, según lo explicado en la Subsección 6.16.2.
5. Repetir los pasos necesarios para realizar la conexión a la Base de Datos `Departamentos` , según lo explicado en la Subsección 6.16.4.
6. Cree una nueva tabla para almacenar los datos correspondientes al Departamento del Magdalena. En este caso, la tabla se denominará `Viviendas_Santander`.

```
try:
    cursor = connection.cursor()
```

```
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS Viviendas_Santander (
        id INT AUTO_INCREMENT PRIMARY KEY
        )
        """)
        print("Tabla 'Viviendas_Santander' creada exitosamente")

    except Error as e:
        print(f"Error al crear la tabla: {e}")
```

7. Repetir los pasos explicados en la Subsección 6.17.2.

8. Repetir los pasos explicados en la Subsección 6.17.3, cambiando el nombre de la tabla por `Viviendas_Santander`:

```
nombre_tabla = "Viviendas_Santander"
```

9. Acceder al enlace previamente mencionado, Base de datos del Censo Nacional de Población y Vivienda 2018, donde se encuentra un listado de departamentos junto con un botón de descarga para una carpeta en formato ZIP.

10. Hacer clic en el botón `Descargar`, lo que iniciará la descarga de una carpeta que contiene los datos del departamento de Santander.

11. Una vez finalizada la descarga, abrir la carpeta ZIP, dentro de la cual se encuentran tres subcarpetas. Ingresar a la carpeta denominada **"68_Santander_CSV"**.

12. En esta carpeta se encuentran cinco archivos en formato CSV. Se debe descargar el archivo denominado **"CNPV2018_1VIV_A2_68"**.

13. Guarde el archivo en una carpeta o ruta de fácil acceso para facilitar su carga en la tabla de MySQL. Es importante no modificar la configuración del archivo.

14. Repetir los pasos explicados en la Subsección 6.17.4.

Para facilitar la comprensión de la tabla, se explicará el significado de cada una de sus columnas.

- **TIPO_REG**: Tipo de registro.
- **U_DPTO**: Código correspondiente al departamento.
- **U_MPIO**: Código correspondiente al municipio.
- **UA_CLASE**: Clase.
- **U_EDIFICA**: Número de orden de la edificación.
- **COD_ENCUESTAS**: Código de encuesta.
- **U_VIVIENDA**: Número de orden de la vivienda.
- **UVA_ESTATER**: Vivienda en una territorialidad étnica.
    - 1-Si.
    - 2-No.
- **UVA1_TIPOTER**: Tipo de territorialidad étnica.
- **UVA2_CODTER**: Código de territorialidad étnica.
- **UVA_ESTA_AREAPROT**: Vivienda en un área protegida.
    - 1-Si.
    - 2-No.
    - 4-No Aplica.

- 9-No Informa.

- **UVA1_COD_AREAPROT**: Código del área protegida.
- **UVA_USO_UNIDAD**: Uso de la Unidad.
  - 1-Vivienda.
  - 2-Mixto (Espacio independiente y separado que combina vivienda con otro uso no residencial).
  - 3-Unidad NO Residencial (Espacio independiente y separado con uso $<>$ vivienda).
  - 4-Lugar Especial de Alojamiento - LEA.
- **V_TIPO_VIV**: Tipo de vivienda.
  - 1-Casa.
  - 2-Apartamento.
  - 3-Tipo cuarto
  - 4-Vivienda tradicional Indígena.
  - 5-Vivienda tradicional Étnica (Afrocolombiana, Isleña, Rrom).
  - 6-Otro (contenedor, carpa, embarcación, vagón, cueva, refugio natural, etc).
- **V_CON_OCUP**: Condición de ocupación.
  - 1-Ocupada con personas presentes.
  - 2-Ocupada con todas las personas ausentes.
  - 3-Vivienda temporal (para vacaciones, trabajo, etc).
  - 4-Desocupada.
- **V_TOT_HOG**: Total de hogares en la vivienda.

- **V_MAT_PARED**: Material predominante en paredes exteriores.
    - 1-Bloque, ladrillo, piedra, madera pulida.
    - 2-Concreto vaciado.
    - 3-Material prefabricado.
    - 4-Guadua.
    - 5-Tapia pisada, bahareque, adobe.
    - 6-Madera burda, tabla, tablón.
    - 7-Caña, esterilla, otros vegetales.
    - 8-Materiales de deshecho (Zinc, tela, cartón, latas, plásticos, otros).
    - 9-No tiene paredes.
- **V_MAT_PISO**: Material predominante en los pisos.
    - 1-Mármol, parque, madera pulida y lacada.
    - 2-Baldosa, vinilo, tableta, ladrillo, laminado.
    - 3-Alfombra.
    - 4-Cemento, gravilla.
    - 5-Madera burda, tabla, tablón, otro vegetal.
    - 6-Tierra, arena, barro.
- **VA_EE**: Cuenta con servicio de energía eléctrica.
    - 1-Si.
    - 2-No.
- **VA1_ESTRATO**: Estrato de la vivienda, según servicio de energía.
    - 0-Sin Estrato.

- 1-Estrato 1.
- 2-Estrato 2.
- 3-Estrato 3.
- 4- Estrato 4.
- 5-Estrato 5.
- 6-Estrato 6.
- 9-No sabe el estrato.

- **VB_ACU**: Cuenta con servicio de acueducto.
  - 1-Si.
  - 2-No.
- **VC_ALC**: Cuenta con servicio de alcantarillado.
  - 1-Si.
  - 2-No.
- **VD_GAS**: Cuenta con servicio de gas natural conectado a red pública.
  - 1-Si.
  - 2-No.
  - 9-No Informa.
- **VE_RECBAS**: Cuenta con servicio de recolección de basura.
  - 1-Si.
  - 2-No.
  - 9-No Informa.

- **VE1_QSEM**: Veces por semana de recolección basura.
    - 1-1 Vez.
    - 2-2 Veces.
    - 3-3 Veces.
    - 4-4 Veces.
    - 5-5 Veces.
    - 6-6 Veces.
    - 7-7 Veces.
    - 8-Mayor periodicidad.
    - 9-No Sabe.
- **VF_INTERNET**: Cuenta con servicio de internet fijo o móvil.
    - 1-Si.
    - 2-No.
- **V_TIPO_SERSA**: Tipo de servicio sanitario.
    - 1-Inodoro conectado al alcantarillado.
    - 2-Inodoro conectado a pozo séptico.
    - 3-Inodoro sin conexión.
    - 4-Letrina.
    - 5-Inodoro con descarga directa a fuentes de agua.
    - 6-esta vivienda No tiene servicio sanitario.
- **L_TIPO_INST**: LEA_Tipo de institución o establecimiento.
    - 1-Centro penitenciario.

- 2- Institución de protección e internado preventivo para niños, niñas y adolescentes.
- 3-Centro de protección y atención al adulto mayor.
- 4-Convento, seminario, monasterio u otras instituciones similares.
- 5-Sede educativa con Población interna.
- 6-Cuartel, guarnición militar (Ejercito, Armada y Fuerza Aérea).
- 7-Comando de policía, estación de policía.
- 8-Campamento de trabajo.
- 9-Casa de lenocinio o prostíbulo.
- 10-Albergue de desplazados.
- 11-Hogar de paz.
- 12-Centro de rehabilitación funcional.
- 13-Casa de paso indígena.
- 14-No Aplica.

- **L_EXISTEHOG**: LEA_Existencia de hogar.
  - 1-Si.
  - 2-No.
  - 4-No Aplica.
  - 9-No Informa.
- **L_TOT_PERL**: LEA_Total de residentes (no pertenecen a hogares).

Con la tabla `Viviendas_Santander` cargada, se pueden realizar consultas empleando comandos SQL en Python y aplicar los modificadores explicados en la Sección 6.8. A continuación, se presentan algunos ejemplos prácticos de consultas que se pueden realizar

sobre la tabla `Viviendas_Santander`. Cada uno de estos ejemplos deben ejecutarse en una nueva celda de Jupyter.

Ejemplo. 97. De esta consulta se obtiene la cantidad de viviendas que tienen estrato 4 según el servicio de energía.

```
cursor.execute(
    "SELECT COUNT(*) AS total_viviendas "
    "FROM Viviendas_Santander "
    "WHERE VA_EE = 1 AND VA1_ESTRATO = 4;"
)

resultado = cursor.fetchone()

print(
    f"Cantidad de viviendas con estrato 4 según el "
    f"servicio de energía: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13), y `WHERE` (Subsección 6.8.2).

Ejemplo. 98. De esta consulta se obtiene el material predominante en las paredes de las viviendas que cuentan con servicio de acueducto y alcantarillado.

```
cursor.execute(
    "SELECT V_MAT_PARED, COUNT(*) AS total "
    "FROM Viviendas_Santander "
    "WHERE VB_ACU = 1 AND VC_ALC = 1 "
    "GROUP BY V_MAT_PARED "
    "ORDER BY total DESC "
    "LIMIT 1;"
```

```
)

resultado = cursor.fetchone()

print(
    f"El material predominante en las paredes de las "
    f"viviendas con acueducto y alcantarillado es: "
    f"{resultado[0]} (con {resultado[1]} registros)."
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13), `WHERE` (Subsección 6.8.2), `GROUP BY` (Subsección 6.8.19), `ORDER BY` (Subsección 6.8.3) y `LIMIT` (Subsección 6.8.8).

Ejemplo. 99. De esta consulta se obtiene el número de viviendas que no cuentan con servicio de gas.

```
cursor.execute(
    "SELECT COUNT(*) AS total_viviendas "
    "FROM Viviendas_Santander "
    "WHERE VD_GAS = 2;"
)

resultado = cursor.fetchone()

print(
    f"El número de viviendas que no cuentan con "
    f"servicio de gas es: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

Ejemplo. 100. De esta consulta se obtiene el promedio de hogares en las viviendas tradicionales étnicas que cuentan con servicio de internet y un inodoro conectado al alcantarillado.

```
cursor.execute(
    "SELECT AVG(V_TOT_HOG) AS prom_hogares "
    "FROM Viviendas_Santander "
    "WHERE V_TIPO_VIV = 5 "
    "AND VF_INTERNET = 1 "
    "AND V_TIPO_SERSA = 1;"
)

resultado = cursor.fetchone()

print(
    f"El promedio de hogares en viviendas tradicionales "
    f"étnicas con internet y servicio sanitario conectado "
    f"es: {resultado[0]:.2f}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `AVG` (Subsección 6.8.15) y `WHERE` (Subsección 6.8.2).

Ejemplo. 101. De esta consulta se obtiene la cantidad de viviendas que no cuentan con servicio de energía eléctrica.

```
cursor.execute(
    "SELECT COUNT(*) AS total_viviendas "
    "FROM Viviendas_Santander "
```

```
    "WHERE VA_EE = 2;"
)

resultado = cursor.fetchone()

print(
    f"El número de viviendas que no cuentan con "
    f"servicio de energía eléctrica es: {resultado[0]}"
)
```

Para este ejemplo se usó la instrucción `SELECT` (Sección 6.7) junto con `COUNT(*)` (Subsección 6.8.13) y `WHERE` (Subsección 6.8.2).

6.19. Extensiones y trabajos futuros

El presente manual se centra en las bases de datos relacionales usando lenguaje SQL mediante herramientas como MySQL y Python, también usa recursos gráficos fijos con los que no se puede interactuar de una manera dinámica y todo se usa desde el ámbito local. A partir de esto, surgen otras líneas de trabajo u otros temas que pueden complementar este proyecto. Entre ellos se encuentra el estudio de modelos de bases de datos no relacionales, la incorporación de herramientas de visualización dinámica de datos mediante tableros interactivos y el aprovechamiento de servicios en la nube, como Microsoft Azure, Amazon Web Services, entre otros. Se mencionan estos futuros trabajos en específico debido a que cada uno es una parte complementaria de este proyecto.
A continuación, se presentan algunas proyecciones que podrían guiar futuras proyectos u otros manuales prácticos derivados de este trabajo.

6.19.1. Manual para bases de datos no relacionales

La creación de un manual para el uso y manipulación de bases de datos no relacionales sería de gran utilidad ya que estas bases de datos no trabajan con tablas que guardan alguna relación, ni con estructuras definidas. Incluso, los datos podrían no estar guardados en tablas. El desarrollo del manual mencionado sería el complemento ideal del presente manual.

6.19.2. Manejo mediante gráficos dinámicos y sus herramientas

El uso de gráficos dinámicos y tableros interactivos son una herramienta muy importante en el análisis de datos, ya que ofrecen recursos gráficos para una visualización clara y estructurada de la información. Esto facilita la interpretación de grandes volúmenes de datos y facilita la toma de decisiones. Las herramientas más conocidas para la creación y manipulación de gráficos dinámicos o tableros interactivos son: Power BI, Tableau y Looker Studio Overview. Desarrollar algún proyecto que involucre las herramientas mencionadas anteriormente sería de gran utilidad debido a que se lograrían manejar grandes volúmenes de datos como los de una universidad o empresa y se mejoraría la comprensión de estos. También se crearía una interacción personalizada para cada usuario que le dé uso con actualización de datos en tiempo real.

6.19.3. Manejo de bases de datos desde la nube

El uso de bases de datos en la nube ofrece grandes ventajas frente a las bases de datos locales. La alta disponibilidad, el acceso remoto y la reducción de costos operativos permiten que las bases de datos en la nube sean una opción viable. Estas plataformas permiten administrar grandes volúmenes de datos de forma segura, rápida, eficiente y con menor intervención humana, gracias a la automatización de tareas como respaldos y actualizaciones.

6.19.3.1. Microsoft Azure

Esta herramienta de Microsoft ofrece un amplio catálogo de servicios para poder trabajar con bases de datos relacionales y no relacionales.
Azure ofrece una selección de bases de datos relacionales y no relacionales para todas las necesidades de la aplicación. La inteligencia integrada ayuda a automatizar tareas de administración como la alta disponibilidad, el escalado y la optimización del rendimiento de las consultas para que las aplicaciones estén siempre activas y listas[35].

6.19.3.2. Amazon Web Services

AWS o Amazon Web Services recoge una serie de servicios integrales que ofrece la empresa Amazon para la computación y almacenamiento en la nube, así como para la gestión de bases de datos y aplicaciones móviles, entre otros servicios de cloud computing[36].

El uso de Microsoft Azure o Amazon Services en un futuro proyecto de análisis de datos proporcionaría un entorno seguro y eficiente en el manejo de grandes volúmenes de datos. Además, estas plataformas ofrecen servicios especializados en almacenamiento, procesamiento y visualización de datos, lo que permite trabajar de manera rápida y remota, aumentando la productividad.

---

[35] (Microsoft: *Bases de datos en Azure*. Recuperado el 7 de abril de 2025. `https://azure.microsoft.com/es-es/products/category/databases#:~:text=Iniciar%20sesi%C3%B3n-,Bases%20de%20datos%20en%20Azure,respaldar%20el%20crecimiento%20del%20negocio.`)

[36] (Redacción The Information Lab: *Qué es Amazon Web Services y para qué sirve*. Recuperado el 7 de abril de 2025. 2021. `https://www.theinformationlab.es/blog/que-es-amazon-web-services-y-para-que-sirve/`)

7. Conclusiones y perspectivas

7.1. Conclusiones

Durante la elaboración de este seminario como trabajo de grado, se logró la unificación de herramientas fundamentales para la administración de bases de datos, mediante la combinación de MySQL y Python.

- En primer lugar, se crearon diversas bases de datos simuladas que facilitaron la demostración del uso de las distintas consultas mencionadas en este documento.
- En segundo lugar, se implementaron consultas sobre la base de datos del Censo Nacional de Población y Vivienda 2018, del DANE, lo que permitió aplicar lo aprendido en un contexto real.
- Por otra parte, se integró Python con MySQL mediante Jupyter Notebooks, lo que amplió las posibilidades de consulta y análisis de datos, demostrando la versatilidad de estas herramientas.

En conclusión, este manual constituye un recurso didáctico accesible para quienes desean iniciarse en la gestión de bases de datos, proporcionando explicaciones paso a paso y ejemplos prácticos. Además, su enfoque en datos reales y simulados permite a los usuarios desarrollar habilidades aplicables en diversos contextos académicos y profesionales.

7.2. Perspectivas

Este trabajo de grado presenta múltiples líneas de mejora para incrementar su utilidad en el futuro.

- En primer lugar, su uso como recurso adicional en programas universitarios de bases de datos o análisis de datos podría impulsar la instrucción de estos temas en la Universidad Industrial de Santander y otros lugares de educación superior.
- En segundo lugar, la inclusión de otras bases de datos, ya sean estructuradas o no, expandiría sus usos, facilitando la solución de nuevas problemáticas y contextos en el análisis de datos.
- Por último, el manual podría evolucionar con el tiempo, junto con nuevas formas del análisis de datos.

BIBLIOGRAFÍA

Armbrust, Michael et al.: *Spark SQL: Relational data processing in spark*. En: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), págs. 1383-1394.

Bayer, Mike: *SQLAlchemy Documentation*. `https://docs.sqlalchemy.org/`. 2021.

Beaulieu, Alan: *Learning SQL: Master SQL Fundamentals*. 2nd. O'Reilly, 2009. ISBN: 978-0596520830.

Canal MoureDev by Brais Moure: *Curso de SQL y BASES DE DATOS Desde Cero para PRINCIPIANTES*. Enlace web: `https://www.youtube.com/watch?v=OuJerKzV5T0`. 2024. Visitado 2018.

Codd, Edgar F.: *A Relational Model of Data for Large Shared Data Banks*. En: *Communications of the ACM* 13.6 (1970), págs. 377-387. DOI: `10.1145/362384.362685`.

Community, PyMySQL: *PyMySQL Documentation*. `https://pymysql.readthedocs.io/`. 2021.

Connolly, Thomas y Carolyn Begg: *Database Systems: A Practical Approach to Design, Implementation, and Management*. 4th. Boston, USA: Addison-Wesley, 2005. ISBN: 978-0321210258.

Coronel, Carlos y Steven Morris: *Database Systems: Design, Implementation, and Management*. 12th. Boston: Cengage Learning, 2016.

DANE, Dirección de Censos y Demografía: *Censo Nacional de Poblacion y Vivienda 2018*. Enlace web: `https://www.dane.gov.co/index.php/estadisticas-por-`

tema/demografia-y-poblacion/censo-nacional-de-poblacion-y-vivenda-2018. 2018. Visitado 2018.

Date, C. J.: *An Introduction to Database Systems*. 8th. Boston: Addison-Wesley, 2004.

DeBarros, Anthony: *Practical SQL, 2nd Edition: A Beginner's Guide to Storytelling with Data*. 2nd. 2022. ISBN: 978-1718501072.

Departamento Administrativo Nacional de Estadística (DANE): *Descarga de microdatos - Encuesta del DANE*. Recuperado el 26 de marzo de 2025. 2023. https://microdatos.dane.gov.co/index.php/catalog/643/get-microdata.

Din, Akeel: *STRUCTURED QUERY LANGUAGE (SQL): A Practical Introduction*. NCC BLACKWELL.

Elizabeth, Evangelista Nava: *MANUAL DE PRÁCTICA BÁSICA CON SQL*. En: *Centro Universitario UAEM Atlacomulco* (2015).

Elmasri, Ramez y Shamkant B. Navathe: *Fundamentals of Database Systems*. 7th. Boston: Pearson, 2015.

Fondo de Pensiones Públicas del Nivel Nacional (FOPEP): *Tabla de Códigos DANE*. Recuperado el 26 de marzo de 2025. 2019. https://www.fopep.gov.co/sheempoo/2019/02/Tabla-C%C3%B3digos-Dane.pdf.

Gregorio, Federico Di: *psycopg2 Documentation*. https://www.psycopg.org/docs/. 2021.

Héctor García-Molina Jeffrey D. Ullman, Jennifer Widom: *Sistemas de bases de datos*. 2ª. Madrid, España: Pearson Educación, 2014. ISBN: 978-6073222452.

IBM Corporation: *IBM DB2 Overview*. Recuperado el 26 de marzo de 2025. 2022. https://www.ibm.com/products/db2.

Joyce, Philip: *C and Python Applications: Embedding Python Code in C Programs, SQL Methods, and Python Sockets*. Apress, 2022. ISBN: 978-1-4842-7773-7.

Kohonen, Teuvo: *Self-Organizing Maps*. 3rd. Berlin: Springer, 2001.

Lutz, Mark: *Learning Python*. 5th. Sebastopol, CA, USA: O'Reilly Media, 2013. ISBN: 978-1449355739.

MariaDB Foundation: *What is MariaDB?* Recuperado el 26 de marzo de 2025. 2022. https://mariadb.org/about/.

McKinney, Wes: *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 2nd. Sebastopol, CA, USA: O'Reilly Media, 2017. ISBN: 978-1491957660.

Medina González, Raúl: *Juez de consultas SQL basado en Postgresql*. En: (2020).

Microsoft: *Bases de datos en Azure*. Recuperado el 7 de abril de 2025. https://azure.microsoft.com/es-es/products/category/databases#:~:text=Iniciar%20sesi%C3%B3n-,Bases%20de%20datos%20en%20Azure,respaldar%20el%20crecimiento%20del%20negocio..

Microsoft: *Visual Studio Code*. https://code.visualstudio.com/.

Microsoft Corporation: *Microsoft SQL Server*. Recuperado el 26 de marzo de 2025. 2021. https://www.microsoft.com/en-us/sql-server.

MongoDB, Inc.: *PyMongo Documentation*. https://pymongo.readthedocs.io/. 2021.

MongoDB, Inc.: *MongoDB Manual*. https://docs.mongodb.com/manual/. 2020.

MySQL Documentation Team: *Connecting to MySQL Using Connector/Python*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: CREATE DATABASE and DROP DATABASE*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/create-database.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: CREATE TABLE Statement*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/create-table.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: Data Types*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/data-types.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: SELECT Statement*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/select.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: SELECT Syntax - Modifiers and Clauses*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/select.html.

MySQL Documentation Team: *MySQL 8.0 Reference Manual: Table Constraints*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html.

MySQL Documentation Team: *MySQL Downloads*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/downloads/mysql/.

MySQL Documentation Team: *MySQL Workbench: Downloads*. Recuperado el 26 de marzo de 2025. 2023. https://dev.mysql.com/downloads/workbench/.

Opendatasoft: *Shapes — Bogotá Lab Urbano*. Recuperado el 26 de marzo de 2025. 2023. https://data.opendatasoft.com/explore/dataset/shapes%40bogota-laburbano/export/?refine.nombre_dpt=ANTIOQUIA&location=7,7.17528,-75.5226&basemap=jawg.streets.

Oracle Corporation: *MySQL 8.0 Reference Manual*. https://dev.mysql.com/doc/refman/8.0/en/. 2020.

Oracle Corporation: *MySQL Overview*. Recuperado el 26 de marzo de 2025. 2023. https://www.mysql.com/about/.

Oracle Corporation: *Oracle Database Documentation*. Recuperado el 26 de marzo de 2025. 2021. https://docs.oracle.com/en/database/.

PostgreSQL Global Development Group: *PostgreSQL Features*. Recuperado el 26 de marzo de 2025. 2023. https://www.postgresql.org/about/.

Project Jupyter: *Installation — Jupyter Notebook*. Recuperado el 26 de marzo de 2025. 2023. https://jupyter.org/install.

Python Software Foundation: *Download Python*. Recuperado el 26 de marzo de 2025. 2023. https://www.python.org/downloads/.

Redacción The Information Lab: *Qué es Amazon Web Services y para qué sirve*. Recuperado el 7 de abril de 2025. 2021. https://www.theinformationlab.es/blog/que-es-amazon-web-services-y-para-que-sirve/.

Rockoff, Larry: *The Language of SQL*. 3rd. Addison-Wesley, 2022. ISBN: 978-0-13-763269-5.

Rossum, Guido van: *Python Programming Language*. https://www.python.org/doc/essays/blurb/. 1991.

Spark, Apache: *Spark SQL Programming Guide*. https://spark.apache.org/docs/latest/sql-programming-guide.html.

SQLite Consortium: *About SQLite*. Recuperado el 26 de marzo de 2025. 2023. https://www.sqlite.org/about.html.

Starbuck, Craig: *The Fundamentals of People Analytics With Applications in R.* St. Louis, MO, USA: Springer, 2023. ISBN: 978-3-031-28674-2.

W3schools.: *SQL Tutorial*. https://www.w3schools.com/sql/.

W3schools.: *SQL Tutorial*. Recuperado el 6 de abril del 2025. https://www.w3schools.com/sql/default.asp.

Zaharia, Matei y Bill Wenig: *Spark: The Definitive Guide: Big Data Processing Made Simple*. O'Reilly Media, 2016.

Zaharia, Matei et al.: *Apache Spark: A Unified Engine for Big Data Processing*. En: *Communications of the ACM* 59.11 (2016), págs. 56-65. DOI: 10.1145/2934664.

Zhao, Alice: *SQL Pocket Guide: A Guide to SQL Usage*. 4th. O'Reilly, 2021. ISBN: 978-1-492-09040-3.