*1.1. Tipos de datos e integridad de datos en SQL*

*Los tipos de datos en SQL son fundamentales para la gestión de bases de datos. Entre los tipos de datos más comunes en SQL se encuentran: enteros, decimales, texto, fecha y hora, binarios, entre otros. Es importante conocer los diferentes tipos de datos y cómo se almacenan en una base de datos para poder realizar consultas y operaciones de manera efectiva.*

*Los tipos de datos. además de su sintaxis, se describen a continuación:*

1. *Tipos de datos numéricos*

* *INTEGER: Este tipo de dato se utiliza para almacenar números enteros. Su tamaño suele variar entre 2 y 8 bytes, dependiendo del sistema de gestión de bases de datos (SGBD) utilizado. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INTEGER);* |
| --- |

* *FLOAT: Este tipo de dato se utiliza para almacenar números decimales. Su tamaño suele variar entre 4 y 8 bytes. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (precio FLOAT)* |
| --- |

* *DECIMAL: Este tipo de dato también se utiliza para almacenar números decimales, pero con una precisión fija. Su sintaxis incluye dos valores: la precisión (número total de dígitos) y la escala (número de dígitos después del punto decimal). Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (monto DECIMAL(10, 2));* |
| --- |

1. *Tipos de datos de cadena de caracteres*

* *CHAR: Este tipo de dato se utiliza para almacenar cadenas de caracteres de longitud fija. Si el tamaño de la cadena es menor que el tamaño definido, se rellena con espacios en blanco. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (nombre CHAR(20));* |
| --- |

* *VARCHAR: Este tipo de dato se utiliza para almacenar cadenas de caracteres de longitud variable. Solo se utiliza el espacio necesario para almacenar la cadena. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (descripcion VARCHAR(100));* |
| --- |

1. *Tipos de datos de fecha y hora*

* *DATE: Este tipo de dato se utiliza para almacenar fechas. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (fecha DATE);* |
| --- |

* *TIME: Este tipo de dato se utiliza para almacenar horas. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (hora TIME);* |
| --- |

* *DATETIME: Este tipo de dato se utiliza para almacenar fechas y horas. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (fecha\_hora DATETIME);* |
| --- |

*Estos son solo algunos ejemplos de los tipos de datos que se pueden encontrar en SQL. Es importante que los estudiantes comprendan las características de cada tipo de dato y su sintaxis, para poder utilizarlos adecuadamente en la creación de tablas y consultas.*

*Además de los tipos de datos, es importante asegurar la integridad de los datos en una base de datos. La integridad de los datos se refiere a la precisión, coherencia y consistencia de los datos almacenados en una base de datos. Cuando se garantiza la integridad de los datos, se asegura que los datos sean exactos y completos, lo que facilita la toma de decisiones basadas en los datos.*

*En SQL, la integridad de los datos se garantiza mediante el uso de restricciones, que son reglas que se aplican a las tablas para evitar la inserción de datos incorrectos o incompletos. A continuación, se describen los tipos de restricciones más comunes en SQL:*

* *NOT NULL: Esta restricción se aplica a una columna y especifica que no se pueden insertar valores nulos en esa columna. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INT NOT NULL, nombre VARCHAR(50));* |
| --- |

* *UNIQUE: Esta restricción se aplica a una columna y especifica que no se pueden insertar valores duplicados en esa columna. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INT UNIQUE, nombre VARCHAR(50));* |
| --- |

* *PRIMARY KEY: Esta restricción se aplica a una o más columnas y especifica que esa columna o conjunto de columnas son claves primarias de la tabla. La clave primaria es un valor único que identifica de manera única cada fila de la tabla. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INT PRIMARY KEY, nombre VARCHAR(50));* |
| --- |

* *FOREIGN KEY: Esta restricción se aplica a una o más columnas y especifica que esa columna o conjunto de columnas son claves foráneas de la tabla. La clave foránea es un valor que hace referencia a una clave primaria en otra tabla. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INT, id\_otra\_tabla INT, FOREIGN KEY (id\_otra\_tabla) REFERENCES otra\_tabla(id));* |
| --- |

* *CHECK: Esta restricción se aplica a una columna y especifica una condición que debe cumplirse para insertar valores en esa columna. Ejemplo de sintaxis:*

| *CREATE TABLE tabla\_ejemplo (id INT, edad INT CHECK (edad > 0));* |
| --- |

*Es importante que comprendas la importancia de la integridad de los datos y cómo se garantiza mediante restricciones en SQL. De esta forma serás capaz de utilizar estos tipos de restricciones correctamente al crear tablas y consultas.*

*Un ejemplo de una consulta utilizando las restricciones mencionadas es el siguiente:*

| *CREATE TABLE tabla\_ejemplo (  id INT NOT NULL,  nombre VARCHAR(50) NOT NULL,   email VARCHAR(50) UNIQUE,  fecha\_nacimiento DATE NOT NULL,   id\_otros INT,  CONSTRAINT pk\_id PRIMARY KEY (id),  CONSTRAINT fk\_id\_otros FOREIGN KEY (id\_otros) REFERENCES otra\_tabla(id),  CONSTRAINT ck\_fecha\_nacimiento CHECK (fecha\_nacimiento < CURRENT\_DATE),  CONSTRAINT uc\_nombre\_email UNIQUE (nombre, email) );* |
| --- |

*En este ejemplo, se crea una tabla llamada "tabla\_ejemplo" con las siguientes columnas y restricciones:*

* *“id”: es un campo requerido (NOT NULL) y es la clave primaria de la tabla (pk\_id).*
* *“nombre”: es un campo requerido (NOT NULL) y se asegura la unicidad del par “nombre”/”email” (“uc\_nombre\_email”).*
* *“email”: es un campo único, lo que significa que no puede haber dos filas con el mismo valor de email. Se asegura la unicidad del par “nombre”/”email” (“uc\_nombre\_email”).*
* *“fecha\_nacimiento”: es una fecha que se asegura que sea menor que la fecha actual (“ck\_fecha\_nacimiento”) y no puede ser nula.*
* *“id\_otros”: es una clave foránea que hace referencia a la tabla "otra\_tabla" y se asegura que los valores de esta columna existan en la tabla "otra\_tabla" (fk\_id\_otros).*

*Con este ejemplo se busca entender cómo se aplican las restricciones de integridad de datos en SQL.*

*1.2 Consultas avanzadas de SQL para la exploración de datos*

*Las consultas avanzadas de SQL permiten realizar análisis más complejos de los datos en una base de datos. Entre las técnicas avanzadas de SQL para la exploración de datos se encuentran:*

* *Funciones de ventana (window functions): Permiten realizar operaciones de agregación sobre un conjunto de filas. Por ejemplo, se pueden calcular promedios móviles o clasificar los datos por rangos.*

*La sintaxis básica para usar una función de ventana es la siguiente:*

| *SELECT columna1, columna2, ..., funcion\_de\_ventana(columna3) OVER (PARTITION BY columna4 ORDER BY columna5) FROM tabla;* |
| --- |

*Donde “columna1”, “columna2”, “columna3”, “columna4” y “columna5” son nombres de columnas de la tabla, “funcion\_de\_ventana” es la función de ventana que se quiere aplicar (por ejemplo, SUM, AVG, MAX, etc.), y “tabla” es el nombre de la tabla.*

*En la cláusula OVER, se define la ventana que se quiere usar para aplicar la función de ventana. PARTITION BY se utiliza para separar las filas en grupos según los valores de una columna específica, mientras que ORDER BY se utiliza para ordenar las filas dentro de cada grupo.*

*Ahora veamos un ejemplo de uso de una función de ventana. Supongamos que tenemos una tabla llamada ventas con las columnas producto, fecha y cantidad. Queremos calcular la cantidad total vendida de cada producto en cada fecha, así como la cantidad acumulada hasta esa fecha. La consulta SQL para realizar esta tarea utilizando la función de ventana SUM sería la siguiente:*

| *SELECT producto, fecha, cantidad, SUM(cantidad) OVER (PARTITION BY producto ORDER BY fecha) AS cantidad\_acumulada FROM ventas;* |
| --- |

*En esta consulta, se está calculando la suma acumulada de la cantidad vendida (“cantidad\_acumulada”) por “producto” (PARTITION BY producto) ordenado por fecha (ORDER BY fecha).*

* *Consultas recursivas: Permite realizar análisis de estructuras de datos complejas, como árboles o grafos. En una consulta recursiva, una subconsulta se ejecuta varias veces y el resultado se va acumulando en una tabla temporal.*

*La sintaxis básica de una consulta recursiva en SQL es la siguiente:*

| *WITH RECURSIVE nombre\_cte (columna1, columna2, ...) AS ( -- Consulta no recursiva SELECT ... UNION ALL -- Consulta recursiva SELECT ... FROM nombre\_cte WHERE ... ) SELECT \* FROM nombre\_cte;* |
| --- |

*Donde:*

* *"WITH RECURSIVE" es una cláusula que indica que se está utilizando una consulta recursiva.*
* *"nombre\_cte" es un nombre arbitrario que se le da a la tabla temporal que se va a crear.*
* *"columna1, columna2, ..." son las columnas que se van a seleccionar en la consulta.*
* *"SELECT ..." es la consulta no recursiva, es decir, la primera vez que se ejecuta la subconsulta.*
* *"UNION ALL" es un operador que combina los resultados de dos consultas.*
* *La segunda consulta es la consulta recursiva. Aquí se utiliza la tabla temporal creada en la primera consulta.*
* *"WHERE ..." es una condición que se utiliza para definir cuándo se debe detener la recursión.*
* *"SELECT \* FROM nombre\_cte" es la consulta final que devuelve el resultado acumulado.*

*Por ejemplo, supongamos que tenemos una tabla llamada "Categorías" con las siguientes columnas: "ID", "Nombre" y "ID\_padre". La columna "ID\_padre" indica el ID de la categoría “padre”, o NULL si no tiene un padre. Podemos utilizar una consulta recursiva para obtener una estructura jerárquica de las categorías. Un ejemplo de consulta recursiva para obtener la estructura jerárquica de las categorías es el siguiente:*

| *WITH RECURSIVE jerarquia AS ( -- Consulta no recursiva SELECT id, nombre, id\_padre, 0 AS nivel FROM categorias WHERE id\_padre IS NULL UNION ALL -- Consulta recursiva SELECT c.id, c.nombre, c.id\_padre, j.nivel + 1 AS nivel FROM categorias c JOIN jerarquia j ON c.id\_padre = j.id ) SELECT id, nombre, nivel  FROM jerarquia ORDER BY nivel, id;* |
| --- |

*En el ejemplo, "jerarquia" es el nombre de la tabla temporal que se va a crear.*

*La primera consulta selecciona todas las categorías que no tienen un padre, es decir, las categorías raíz. La segunda consulta utiliza la tabla "categorias" y la tabla "jerarquia" para unir las categorías con sus padres y calcular el nivel de profundidad de cada categoría.*

*"ORDER BY nivel, id" es una cláusula que ordena el resultado por “nivel” y por “ID” de categoría.*

* *Uniones (joins) complejas: Permiten combinar múltiples tablas para realizar análisis más complejos. Se utilizan cuando una consulta necesita obtener información de varias tablas relacionadas entre sí. Por ejemplo, se pueden unir varias tablas para realizar análisis de ventas por categoría de producto y por región. La sintaxis básica de una unión compleja es:*

| *SELECT columnas FROM tabla1 JOIN tabla2 ON condicion1 JOIN tabla3 ON condicion2 WHERE condicion3;* |
| --- |

*En esta sintaxis, JOIN se utiliza para unir dos tablas y ON se utiliza para especificar la condición de unión. La cláusula WHERE se utiliza para aplicar filtros adicionales a la consulta.*

*Veamos el siguiente ejemplo:*

| *SELECT productos.nombre, ventas.cantidad, regiones.nombre FROM productos JOIN ventas ON productos.id = ventas.id\_producto JOIN tiendas ON ventas.id\_tienda = tiendas.id JOIN regiones ON tiendas.id\_region = regiones.id WHERE regiones.nombre = 'Norte';* |
| --- |

*En este ejemplo, se quiere obtener el nombre de los productos vendidos, la cantidad vendida y el nombre de la región donde se realizaron las ventas, para las ventas realizadas en la región Norte. Se realiza una unión entre las tablas de productos, ventas, tiendas y regiones utilizando las columnas “id” y “id\_producto”, “id\_tienda” y “id\_region”, respectivamente. Luego se utiliza la cláusula WHERE para filtrar los resultados y obtener solo las ventas realizadas en la región Norte.*

* *Subconsultas: Son consultas dentro de una consulta principal. Se utilizan para obtener datos que no pueden ser obtenidos directamente de una sola tabla o vista. Las subconsultas pueden ser utilizadas en la cláusula WHERE, FROM o SELECT de la consulta principal. La sintaxis de una subconsulta es la siguiente:*

| *SELECT columna1, columna2, ... FROM tabla1 WHERE columna1 IN (SELECT columna1 FROM tabla2 WHERE condicion)* |
| --- |

*La subconsulta se encuentra entre paréntesis después de la cláusula IN. La subconsulta selecciona los valores de la “columna1” de la “tabla2” que cumplen con la condición especificada, y la consulta principal selecciona las columnas específicas de la “tabla1” donde la “columna1” está dentro de los resultados de la subconsulta.*

*Para entender mejor el uso de las subconsultas, supongamos que tenemos dos tablas: una tabla llamada "ventas" que contiene información sobre ventas diarias de productos, y otra tabla llamada "productos" que contiene información sobre los productos vendidos. Queremos obtener el nombre del producto y su precio máximo de venta en la tabla "ventas". Podríamos utilizar una subconsulta para lograr esto de la siguiente manera:*

| *SELECT nombre\_producto, precio\_maximo FROM productos JOIN (  SELECT id\_producto, MAX(precio\_venta) AS precio\_maximo  FROM ventas  GROUP BY id\_producto ) AS precios ON productos.id = precios.id\_producto;* |
| --- |

*En este ejemplo, la subconsulta se utiliza para obtener el precio máximo de venta para cada producto en la tabla "ventas", y se une con la tabla "productos" utilizando la cláusula JOIN para obtener el nombre del producto correspondiente.*

*Es importante notar que la subconsulta está entre paréntesis y se utiliza como una tabla temporal dentro de la consulta principal. También se utiliza la función de agregación MAX para obtener el precio máximo de venta de cada producto en la tabla "ventas".*

* *Consultas temporales: Las consultas temporales en SQL permiten acceder a datos históricos de una tabla. Esto es útil para realizar análisis y comparaciones de datos en diferentes momentos del tiempo.*

*La sintaxis básica de una consulta temporal implica la creación de una tabla derivada que contiene una instantánea de los datos en un momento específico. Para hacer esto, se utiliza la cláusula "AS OF TIMESTAMP" seguida de una marca de tiempo específica.*

*Por ejemplo, si queremos obtener una instantánea de los datos en una tabla "ventas" a las 12:00 PM del 1 de enero de 2022, la consulta temporal se vería así:*

| *SELECT \*  FROM ventas AS OF TIMESTAMP '2022-01-01 12:00:00';* |
| --- |

*En esta consulta, la tabla "ventas" se convierte en una tabla derivada que contiene los datos tal como se veían a las 12:00 PM del 1 de enero de 2022.*

*También es posible usar la cláusula "VERSIONS BETWEEN" para recuperar datos en un rango de tiempo específico. Por ejemplo, si queremos obtener todas las versiones de la tabla "ventas" entre el 1 de enero de 2022 y el 31 de enero de 2022, la consulta se vería así:*

| *SELECT \* FROM ventas VERSIONS BETWEEN TIMESTAMP '2022-01-01 00:00:00'  AND TIMESTAMP '2022-01-31 23:59:59';* |
| --- |

*En esta consulta, la tabla "ventas" se convierte en una tabla derivada que contiene todas las versiones de los datos entre el 1 de enero de 2022 y el 31 de enero de 2022.*

*Las consultas temporales son útiles para el análisis de series de tiempo, la comparación de datos a lo largo del tiempo y la recuperación de datos históricos.*

* *Tablas dinámicas (pivot tables): Son una herramienta útil para resumir grandes conjuntos de datos y analizarlos desde diferentes perspectivas.La sintaxis básica para crear una tabla dinámica en SQL es la siguiente:*

| *SELECT columna1, [agregado1], [agregado2], ... FROM tabla PIVOT (  [funcion\_agregada](columna2)  FOR columna3 IN ([valor1], [valor2], ..., [valorN]) ) AS alias\_tabla\_pivote;* |
| --- |

*Donde:*

* *“columna1” es la columna que se utilizará como categoría o eje X en la tabla dinámica.*
* *[agregado1], [agregado2], ... son las columnas que se quieren agregar o resumir en la tabla dinámica. Por ejemplo, la suma de ventas o el promedio de ganancias.*
* *“tabla” es la tabla original que contiene los datos.*
* *“columna2” es la columna que se quiere agregar o resumir en la tabla dinámica.*
* *“columna3” es la columna que se utilizará como categoría o eje Y en la tabla dinámica.*
* *[valor1], [valor2], ... [valorN] son los valores únicos que aparecen en la “columna3” y que se quieren mostrar como columnas en la tabla dinámica.*
* *“alias\_tabla\_pivote” es el nombre que se le da a la tabla dinámica resultante.*

*Ahora, veamos un ejemplo para dar mayor claridad al uso de tablas dinámicas. Supongamos que tenemos una tabla ventas que contiene información sobre las ventas realizadas en una tienda, con las siguientes columnas:*

* *“producto”: el nombre del producto vendido.*
* *“fecha”: la fecha en que se realizó la venta.*
* *“cantidad”: la cantidad vendida.*
* *“precio”: el precio unitario del producto.*

*Queremos crear una tabla dinámica que resuma las ventas por producto y por fecha, mostrando la cantidad total vendida y el precio promedio por producto y por fecha.*

*La consulta para crear la tabla dinámica sería la siguiente:*

| *SELECT producto, [2019-01-01], [2019-01-02], [2019-01-03] FROM (  SELECT producto, CONVERT(date, fecha) AS fecha, cantidad, precio  FROM ventas ) AS t PIVOT (  SUM(cantidad) AS cantidad\_total, AVG(precio) AS precio\_promedio  FOR fecha in ([2019-01-01], [2019-01-02], [2019-01-03]) ) AS tabla\_pivote;* |
| --- |

*En este ejemplo, la consulta utiliza una subconsulta para convertir la columna fecha en un tipo de datos “date”, para que pueda ser utilizada como eje Y en la tabla dinámica. Luego, se utiliza la función SUM para sumar la cantidad vendida y la función AVG para calcular el precio promedio por producto y por fecha.*

*1.3. Visualización de datos utilizando gráficos, tablas y otros elementos visuales*

*SQL es un lenguaje de consulta que se utiliza para acceder y manipular datos almacenados en una base de datos. Además de realizar consultas y análisis de datos, SQL también puede utilizarse para visualizar datos utilizando diferentes elementos visuales, como gráficos y tablas.*

*Para visualizar datos con SQL, es importante tener un buen conocimiento de las estructuras de datos y cómo se relacionan entre sí. Esto permitirá que los datos se seleccionen y se muestren de manera efectiva. A continuación, se presentan algunas técnicas comunes para visualizar datos con SQL:*

1. *Selección de datos: Para visualizar datos, primero se deben seleccionar los datos que se utilizarán. Esto se puede hacer utilizando consultas SELECT en SQL. Las consultas SELECT permiten seleccionar columnas específicas de una tabla y filtrar los resultados utilizando condiciones.*
2. *Agrupación de datos: Los datos a menudo deben agruparse para que puedan analizarse y visualizarse de manera efectiva. Esto se puede hacer utilizando la cláusula GROUP BY en SQL. La cláusula GROUP BY permite agrupar los resultados de una consulta por una o varias columnas.*
3. *Funciones de agregación: Las funciones de agregación se utilizan para calcular estadísticas y resúmenes de datos, como la suma, el promedio o el máximo. Estas funciones se pueden utilizar en combinación con la cláusula GROUP BY para obtener estadísticas por grupos de datos.*
4. *Gráficos y tablas: Una vez que se han seleccionado y agrupado los datos, se pueden utilizar diferentes tipos de gráficos y tablas para visualizar los resultados. En SQL, esto se puede hacer utilizando diferentes herramientas y lenguajes de programación, como SQL Server Reporting Services o Python.*

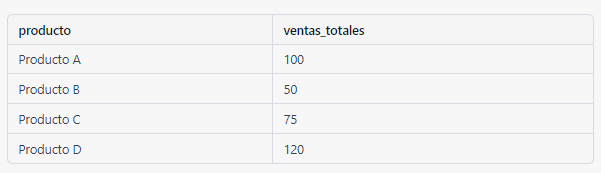
*Para explicar el proceso que conlleva la visualización de datos, veamos el siguiente ejemplo:*

*Supongamos que tienes una tabla de ventas con las columnas "producto", "fecha" y "cantidad", que registra la cantidad de cada producto que se vendió en una fecha determinada. Quieres visualizar los datos de ventas totales por producto para el mes de enero. Para hacerlo, primero seleccionaremos los datos que necesitamos utilizando la consulta SELECT, y filtraremos por fecha para seleccionar solo los datos de enero:*

| *SELECT producto, SUM(cantidad) AS ventas\_totales FROM ventas WHERE fecha >= '2022-01-01' AND fecha < '2022-02-01' GROUP BY producto;* |
| --- |

*Esta consulta nos devolverá una tabla con dos columnas: "producto" y "ventas\_totales". Los datos estarán agrupados por producto y se mostrará la suma de la cantidad vendida para cada producto durante el mes de enero.*

*Para visualizar estos datos, podríamos utilizar una tabla o un gráfico de barras. Por ejemplo, si queremos utilizar una tabla, podemos exportar los resultados de nuestra consulta a un archivo CSV y abrirlo en una herramienta de visualización como Excel:*

**

*Otra opción sería utilizar un gráfico de barras para mostrar los datos de manera más visual. Podríamos utilizar una herramienta de visualización como Python o R para crear el gráfico. Por ejemplo, una solución utilizando Python y la librería matplotlib:*

| *import matplotlib.pyplot as plt import pandas as pd # Lee los datos desde un archivo CSV df = pd.read\_csv('ventas\_por\_producto.csv')  # Crea un gráfico de barras plt.bar(df['producto'], df['ventas\_totales'])  # Añade etiquetas a los ejes y título plt.xlabel('Producto') plt.ylabel('Ventas totales') plt.title('Ventas por producto en enero')  # Mostramos gráfica plt.show()* |
| --- |

*Este código nos generará un gráfico de barras con las ventas totales por producto durante el mes de enero. Cada barra representa un producto y su altura representa la cantidad de ventas totales.*

*1.4. Trabajando con bibliotecas de visualización en Python:*

*Python es uno de los lenguajes de programación más populares para la visualización de datos, y tiene muchas bibliotecas y herramientas disponibles para ayudar a los programadores a crear gráficos y visualizaciones interactivas. SQL, por otro lado, es una herramienta poderosa para la gestión y análisis de datos. A menudo, los usuarios necesitan conectar las dos herramientas para combinar y visualizar datos.*

*Una forma común de conectar SQL y Python es utilizar una biblioteca de Python para conectarse a la base de datos y leer los datos de SQL en Python. Luego, se pueden utilizar diferentes bibliotecas de visualización de Python, como Matplotlib, Seaborn o Plotly, para crear gráficos y visualizaciones interactivas.*

*Para conectarse a una base de datos de SQL desde Python, se puede utilizar una biblioteca de Python llamada "pyodbc". Esta biblioteca permite a los usuarios conectarse a una base de datos utilizando el protocolo ODBC (Open Database Connectivity). Para usar pyodbc, primero es necesario instalar el controlador ODBC correspondiente para la base de datos a la que se desea conectarse.*

*Una vez que se ha establecido la conexión entre SQL y Python, se pueden usar diferentes bibliotecas de visualización de Python para crear gráficos y visualizaciones interactivas. Por ejemplo, Matplotlib es una biblioteca de visualización de Python muy popular que permite a los usuarios crear diferentes tipos de gráficos, como gráficos de barras, gráficos de líneas, gráficos de dispersión y muchos otros. Seaborn es otra biblioteca popular que se basa en Matplotlib y proporciona una sintaxis más fácil de usar para la creación de gráficos estadísticos. Plotly es otra biblioteca de visualización de Python que permite a los usuarios crear gráficos interactivos y dinámicos que se pueden incrustar en sitios web o aplicaciones.*

*En resumen, la conexión de SQL y bibliotecas de visualización de Python permite a los usuarios combinar y visualizar datos de manera efectiva y eficiente. Al conectar SQL y Python, los usuarios pueden aprovechar la potencia de ambas herramientas para analizar y visualizar datos de manera efectiva.*

*Veamos ahora un ejemplo de cómo se realiza esta interacción entre SQ y Python:*

1. *Primero, es necesario instalar la biblioteca pyodbc en Python, que proporciona una forma de conectarse a una base de datos SQL Server. Esto se puede hacer a través de la línea de comando de Python o utilizando un administrador de paquetes como pip:*

| *pip install pyodbc* |
| --- |

1. *A continuación, se debe establecer una conexión a la base de datos SQL Server utilizando la función connect() de pyodbc. Esto requiere información como el nombre del servidor, la base de datos, el nombre de usuario y la contraseña.*

| *import pyodbc  # Establecer la conexión a la base de datos SQL Server  conn = pyodbc.connect('Driver = {SQL Server};'  'Server = nombre\_servidor;'  'Database = nombre\_bd;'  'UID = nombre\_usuario;'  'PWD = contraseña;')* |
| --- |

1. *Después de establecer la conexión, se puede utilizar la función read\_sql() de pandas para ejecutar una consulta SQL y obtener los resultados como un DataFrame de pandas:*

| *import pandas as pd  # Ejecutar una consulta SQL y obtener los resultados como un DataFrame  df = pd.read\_sql('SELECT columna1, columna2 FROM tabla', conn)* |
| --- |

1. *Una vez que se tienen los datos en un DataFrame de pandas, se puede utilizar Matplotlib para visualizarlos. Por ejemplo, se podría crear un gráfico de barras que muestre la cantidad de registros en la tabla por valor en la “columna1”:*

| *import matplotlib.pyplot as plt  # Crear un gráfico de barras de la cantidad de registros por valor en la columna 1 df.groupby('columna1').size().plot(kind = 'bar') plt.show()* |
| --- |

*¡Eso es todo! Este es solo un ejemplo simple, pero con esta técnica se pueden visualizar una gran variedad de datos de SQL en Python utilizando Matplotlib o cualquier otra biblioteca de visualización de datos de Python.*

*1.5. Integrando SQL con Tableau:*

*Tableau es una herramienta de visualización de datos que permite crear informes interactivos y dashboards. También es compatible con la integración de datos de SQL. Para conectar SQL y Tableau, hay varios pasos que deben seguirse:*

1. *Descargar el controlador de base de datos: El primer paso es descargar el controlador de base de datos de SQL que se utilizará para la conexión con Tableau. Esto puede variar según el tipo de base de datos SQL que se esté utilizando.*
2. *Conectar Tableau a la base de datos SQL: Una vez que se ha descargado el controlador, se puede conectar Tableau a la base de datos SQL. Esto se hace mediante la opción de conexión de Tableau y proporcionando los detalles de la conexión, como la dirección IP, el nombre de la base de datos y las credenciales de inicio de sesión.*
3. *Importar los datos a Tableau: Después de establecer la conexión, se pueden importar los datos desde SQL a Tableau. Esto se hace mediante la opción "Conectar a los datos" en Tableau, donde se selecciona la base de datos SQL y se especifica la tabla o vista que se desea importar.*
4. *Crear visualizaciones: Una vez que los datos se han importado a Tableau, se pueden utilizar las herramientas de visualización de Tableau para crear gráficos, tablas y paneles interactivos. Tableau ofrece una amplia variedad de herramientas de visualización, como gráficos de barras, gráficos circulares, mapas, tablas y más.*

*En resumen, la conexión de SQL y Tableau permite a los usuarios acceder a datos almacenados en bases de datos SQL y visualizarlos de manera interactiva y dinámica utilizando las herramientas de visualización de Tableau. Esto puede ser muy útil para analizar grandes conjuntos de datos y obtener información valiosa para la toma de decisiones empresariales.*

*Un ejemplo sencillo de cómo conectar SQL y Tableau:*

1. *Primero, asegúrate de tener una base de datos SQL a la que puedas acceder. Puedes usar MySQL, PostgreSQL, Microsoft SQL Server, entre otras.*
2. *Abre Tableau y selecciona "Conectar" en la pantalla de inicio.*
3. *En la ventana "Conectar", selecciona "SQL Server" o "MySQL" (dependiendo de la base de datos que estés usando) y completa la información requerida, como el nombre del servidor, el nombre de usuario y la contraseña.*
4. *Después de completar la información, selecciona "Conectar".*
5. *Ahora, en Tableau, puedes seleccionar las tablas que deseas utilizar y arrastrarlas al lienzo de trabajo.*
6. *A partir de aquí, puedes comenzar a visualizar los datos. Tableau te permite crear diferentes tipos de visualizaciones, como gráficos, mapas, tablas, etc. Simplemente selecciona la visualización que deseas crear y arrastra los campos que deseas utilizar a las áreas correspondientes.*
7. *Una vez que hayas creado una visualización, puedes guardarla en Tableau y actualizarla en tiempo real con los datos de tu base de datos SQL. Esto significa que si los datos cambian en la base de datos, tu visualización también cambiará automáticamente.*