Clase 2

Alexis Ovalle

12 de marzo de 2025

Indice

GROUP BY y HAVING

- **GROUP BY**: Agrupa resultados según una o más columnas.
- **HAVING**: Filtra los grupos resultantes de un GROUP BY.

Ejemplo:

```
SELECT departamento, COUNT(*) AS empleados FROM empleados GROU
BY departamento HAVING COUNT(*) > 10;
```

BETWEEN

- Sirve para establecer un rango en la condición de búsqueda.

Ejemplo:

```
SELECT * FROM productos WHERE precio BETWEEN 100 AND 500;
```

ORDER BY

- Ordena los resultados en forma ascendente (ASC) o descendente (DESC).

Ejemplo:

```
SELECT nombre, salario FROM empleados ORDER BY salario DESC;
```

JOINS (INNER, LEFT, RIGHT, FULL)

- **INNER JOIN**: Devuelve solo los registros que coinciden en ambas tablas.
- **LEFT JOIN**: Todos los registros de la tabla izquierda y las coincidencias de la derecha.
- **RIGHT JOIN**: Al revés del LEFT JOIN.
- **FULL JOIN**: Combina LEFT y RIGHT JOIN.

Ejemplo INNER JOIN:

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

Subconsultas (Subqueries)

- Consultas anidadas dentro de otras consultas.

Ejemplo:

```
SELECT nombre, salario FROM empleados WHERE salario > ( SELECT
AVG(salario) FROM empleados );
```

EXISTS y NOT EXISTS

- **EXISTS**: Verifica si una subconsulta devuelve algún registro.
- **NOT EXISTS**: Lo contrario.

Ejemplo:

```
SELECT nombre FROM clientes c WHERE EXISTS ( SELECT 1 FROM pedidos p WHERE p.id_cliente = c.id);
```

CASE

- Permite realizar condiciones en la cláusula SELECT o en filtros.

Ejemplo:

```
SELECT nombre, CASE WHEN salario > 50000 THEN 'Alto' WHEN
salario BETWEEN 30000 AND 50000 THEN 'Medio' ELSE 'Bajo'
END AS categoria_salarioFROMempleados;
```

UNION y UNION ALL

- **UNION**: Combina los resultados de dos consultas eliminando duplicados.
- **UNION ALL**: Incluye duplicados.

Ejemplo:

```
SELECT nombre FROM clientes UNION SELECT nombre FROM proveedor
```

¿Qué es Pandas?

- **Pandas** es una librería de Python esencial para el análisis y manipulación de datos estructurados.
- Permite trabajar con datos de una manera muy flexible, ofreciendo herramientas tanto para datos unidimensionales como bidimensionales.
- Las dos estructuras principales de Pandas son:
  - **Series**: Se trata de un objeto unidimensional similar a un vector de Numpy, útil para almacenar datos de una sola columna.
  - **DataFrames**: Estructura bidimensional que organiza datos en filas y columnas, similar a una tabla en SQL o una hoja de cálculo en Excel.
- Es una librería extremadamente popular en el campo de la *Data Science*, *Machine Learning*, y *ETL (Extract, Transform, Load)*.

Carga de Datos

- Una de las primeras tareas al trabajar con datos es cargarlos desde diversas fuentes.
- Pandas ofrece funciones que permiten leer y escribir datos en varios formatos:
  - **CSV**: Un formato muy común para almacenar datos tabulares, ideal para exportar e importar datos.
  - **Excel**: Pandas puede leer archivos Excel con varias hojas, lo que facilita el trabajo con grandes volúmenes de datos.
  - **JSON**: Utilizado cuando los datos están organizados en formato jerárquico o en objetos.
  - **SQL**: Permite leer datos directamente desde bases de datos SQL, facilitando la integración con otras plataformas de datos.
- Algunos ejemplos básicos de carga de datos con Pandas:
  - **CSV:**
    - `df = pd.read_csv('archivo.csv')`
  - **Excel:**
    - `df = pd.read_excel('archivo.xlsx')`
  - **JSON:**
    - `df = pd.read_json('archivo.json')`

Inspección de Datos

- Una vez que los datos están cargados, es crucial inspeccionarlos para entender su estructura.
- Algunas funciones útiles de Pandas para inspeccionar los datos:
  - **df.head()**: Muestra las primeras 5 filas del DataFrame.
  - **df.tail()**: Muestra las últimas 5 filas.
  - **df.info()**: Proporciona información detallada sobre el DataFrame, como tipos de datos y valores nulos.
  - **df.describe()**: Calcula estadísticas descriptivas como la media, mediana, desviación estándar, etc.
  - **df.shape**: Devuelve las dimensiones del DataFrame (número de filas y columnas).
- Estas funciones nos permiten obtener rápidamente una visión general de la calidad y estructura de los datos.

Limpieza de Datos

- La limpieza de datos es una parte fundamental de cualquier proyecto de *Data Science*, ya que la calidad de los datos influye directamente en los resultados obtenidos.
- Algunos de los pasos más comunes en la limpieza de datos incluyen:
  - **Manejo de valores nulos**: Detectar y manejar valores faltantes es esencial para evitar errores en el análisis. Se pueden eliminar o rellenar con un valor.
  - **Eliminación de duplicados**: Detectar registros duplicados para mantener la calidad y exactitud de los datos.
  - **Conversión de tipos de datos**: Asegurarse de que las columnas tengan el tipo de dato adecuado (por ejemplo, convertir una columna de texto que contiene fechas en un tipo `datetime`).
- Algunos ejemplos de funciones de limpieza en Pandas:
  - **Valores nulos:**
    - `df.isnull().sum()` - Verifica la cantidad de valores nulos en cada columna.
    - `df.dropna(inplace=True)` - Elimina filas con valores nulos.
    - `df.fillna(0, inplace=True)` - Rellena valores nulos con un valor

Selección y Filtrado de Datos

- Una de las tareas más frecuentes es seleccionar y filtrar los datos según ciertas condiciones.
- Con Pandas, se pueden seleccionar columnas, filas y filtrar datos de manera eficiente.
- Algunas técnicas comunes de selección y filtrado:
  - **Selección de columnas**: Para obtener una o varias columnas de datos.
  - **Filtrado de filas**: Para seleccionar filas que cumplan con ciertas condiciones (por ejemplo, valores mayores o menores a un umbral).
  - **Selección basada en etiquetas y posiciones**: Usando `loc` para etiquetas y `iloc` para posiciones numéricas.
- Ejemplos de filtrado:
  - Filtrado por condiciones simples:
    - `df[df['edad'] > 30]`
  - Filtrado por condiciones compuestas:
    - `df[(df['edad'] > 30) & (df['pais'] == 'México')]`