SQL equivalent in Python

SQL Operation	Python Equivalent (pandas)	Example
SELECT	df['column_name']	**df['Age']** selects the 'Age' column.
SELECT Multiple Columns	df[['column1', 'column2']]	**df[['Age', 'Name']]** selects both 'Age' and 'Name' columns.
SELECT DISTINCT	df['column'].drop_duplicates()	**df['Country'].drop_duplicates()** selects unique country names.
WHERE	df[df['column'] condition]	**df[df['Age'] > 30]** selects rows where age is greater than 30.
ORDER BY	df.sort_values('column')	**df.sort_values('Age')** sorts the data by the 'Age' column.
GROUP BY	df.groupby('column')	**df.groupby('Country').mean()** calculates mean for each group of 'Country'.
JOIN	pd.merge(df1, df2, on='column', how='type')	**pd.merge(df1, df2, on='ID', how='inner')** performs an inner join on 'ID'.
INSERT INTO	df.append(new_row, ignore_index=True)	**df.append({'Name': 'John', 'Age': 30}, ignore_index=True)** inserts a new row.
UPDATE	df.loc[df['column'] condition, 'column'] = new_value	**df.loc[df['Age'] > 30, 'Category'] = 'Senior'** updates 'Category' based on condition.
DELETE FROM	df = df[~(df['column'] condition)]	**df = df[~(df['Age'] < 18)]** deletes rows where age is less than 18.
LIMIT	df.head(n)	**df.head(5)** selects the first 5 rows of the dataframe.

Explanation

1. **SELECT**: To select specific columns from a DataFrame, you use the column names as keys.
    - **SQL**: **SELECT Age FROM table;**
    - **Python**: **df['Age']**
2. **WHERE**: To filter rows based on a condition, you use boolean indexing.
    - **SQL**: **SELECT * FROM table WHERE Age > 30;**
    - **Python**: **df[df['Age'] > 30]**
3. **ORDER BY**: To sort the data, **sort_values** method is used, specifying the column name.
    - **SQL**: **SELECT * FROM table ORDER BY Age;**
    - **Python**: **df.sort_values('Age')**
4. **GROUP BY**: Grouping in pandas is done using the **groupby** method, often followed by an aggregation function like **mean()**, **sum()**, etc.
    - **SQL**: **SELECT Country, AVG(Age) FROM table GROUP BY Country;**

- **Python**: **df.groupby('Country')['Age'].mean()**

5. **JOIN**: To join two DataFrames, **merge** function is used, specifying the column to join on and the type of join.
    - **SQL**: **SELECT * FROM table1 INNER JOIN table2 ON table1.ID = table2.ID;**
    - **Python**: **pd.merge(df1, df2, on='ID', how='inner')**
6. **INSERT INTO**: To add a new row to a DataFrame, **append** method is used, often with **ignore_index=True** to reindex.
    - **SQL**: **INSERT INTO table (Name, Age) VALUES ('John', 30);**
    - **Python**: **df.append({'Name': 'John', 'Age': 30}, ignore_index=True)**
7. **UPDATE**: To update values in a DataFrame, **loc** method combined with a condition is used.
    - **SQL**: **UPDATE table SET Category = 'Senior' WHERE Age > 30;**
    - **Python**: **df.loc[df['Age'] > 30, 'Category'] = 'Senior'**
8. **DELETE FROM**: To delete rows based on a condition, you filter the DataFrame with the negated condition.
    - **SQL**: **DELETE FROM table WHERE Age < 18;**
    - **Python**: **df = df[~(df['Age'] < 18)]**
9. **LIMIT**: To limit the number of rows returned, **head** method is used with the number of rows as an argument.
    - **SQL**: **SELECT * FROM table LIMIT 5;**
    - **Python**: **df.head(5)**

More commands

SQL Operation	Python Equivalent (pandas)	Example
COUNT	df['column'].count()	**df['Age'].count()** counts the non-NA/null entries of the 'Age' column.
SUM	df['column'].sum()	**df['Sales'].sum()** sums up the 'Sales' column.
AVG (Average)	df['column'].mean()	**df['Price'].mean()** calculates the average of the 'Price' column.
MIN (Minimum)	df['column'].min()	**df['Age'].min()** finds the minimum value in the 'Age' column.
MAX (Maximum)	df['column'].max()	**df['Age'].max()** finds the maximum value in the 'Age' column.
HAVING	df.groupby('column').filter(lambda x: condition)	**df.groupby('Department').filter(lambda x: x['Sales'].sum() > 1000)** filters groups with total sales greater than 1000.

CONCATENATE	df['new_column'] = df['column1'] + df['column2']	**df['FullName'] = df['FirstName'] + ' ' + df['LastName']** concatenates two columns with a space.
LIKE (Pattern Match)	df[df['column'].str.contains('pattern')]	**df[df['Name'].str.contains('John')]** selects rows where the 'Name' column contains 'John'.
IN (List)	df[df['column'].isin(['value1', 'value2'])]	**df[df['Country'].isin(['USA', 'Canada'])]** selects rows where the 'Country' column is either 'USA' or 'Canada'.
BETWEEN	df[df['column'].between(value1, value2)]	**df[df['Age'].between(18, 30)]** selects rows where the 'Age' column is between 18 and 30.
CASE WHEN THEN ELSE	df['column'].apply(lambda x: 'value_if_true' if condition else 'value_if_false')	**df['Category'] = df['Age'].apply(lambda x: 'Adult' if x >= 18 else 'Minor')** assigns 'Adult' or 'Minor' based on the 'Age' column.
SUBSTRING	df['column'].str.slice(start, end)	**df['Initials'] = df['Name'].str.slice(0, 1)** extracts the first letter from the 'Name' column.
LENGTH	df['column'].str.len()	**df['NameLength'] = df['Name'].str.len()** calculates the length of each string in the 'Name' column.
REPLACE	df['column'].str.replace('old', 'new')	**df['Phone'] = df['Phone'].str.replace('-', '')** removes dashes from the 'Phone' column.
DISTINCT COUNT	df['column'].nunique()	**df['Country'].nunique()** counts the number of unique countries.

Explanation

1. **COUNT**: To count the number of non-null entries in a column.
    - **SQL**: **SELECT COUNT(Age) FROM table;**
    - **Python**: **df['Age'].count()**
2. **SUM**: To sum up the values in a column.
    - **SQL**: **SELECT SUM(Sales) FROM table;**
    - **Python**: **df['Sales'].sum()**
3. **AVG (Average)**: To calculate the average of a column.
    - **SQL**: **SELECT AVG(Price) FROM table;**
    - **Python**: **df['Price'].mean()**
4. **MIN (Minimum)** and **MAX (Maximum)**: To find the minimum or maximum value in a column.
    - **SQL**: **SELECT MIN(Age) FROM table;** / **SELECT MAX(Age) FROM table;**

- **Python**: **df['Age'].min() / df['Age'].max()**

5. **HAVING**: To filter aggregated data based on a condition.
    - **SQL**: **SELECT Department, SUM(Sales) FROM table GROUP BY Department HAVING SUM(Sales) > 1000;**
    - **Python**: **df.groupby('Department').filter(lambda x: x['Sales'].sum() > 1000)**
6. **CONCATENATE**: To combine two or more columns into a new column.
    - **SQL**: **SELECT FirstName \|\| ' ' \|\| LastName AS FullName FROM table;**
    - **Python**: **df['FullName'] = df['FirstName'] + ' ' + df['LastName']**
7. **LIKE (Pattern Match)**: To select rows based on a pattern in a column.
    - **SQL**: **SELECT * FROM table WHERE Name LIKE '%John%';**
    - **Python**: **df[df['Name'].str.contains('John')]**
8. **IN (List)**: To select rows where a column's value is in a specified list.
    - **SQL**: **SELECT * FROM table WHERE Country IN ('USA', 'Canada');**
    - **Python**: **df[df['Country'].isin(['USA', 'Canada'])]**
9. **BETWEEN**: To select rows where a column's value is between two values.
    - **SQL**: **SELECT * FROM table WHERE Age BETWEEN 18 AND 30;**
    - **Python**: **df[df['Age'].between(18, 30)]**
10. **CASE WHEN THEN ELSE**: To create a new column based on conditions applied to an existing column.
    - **SQL**: **SELECT CASE WHEN Age >= 18 THEN 'Adult' ELSE 'Minor' END AS Category FROM table;**
    - **Python**: **df['Category'] = df['Age'].apply(lambda x: 'Adult' if x >= 18 else 'Minor')**