SQL Operation	Pandas Equivalent	PySpark Equivalent	Example
SELECT	df['column_name']	df.select('column_name')	Select the 'Age' column:**SQL:** SELECT Age FROM table;**Pandas:** df['Age']**PySpark:** df.select('Age')
SELECT Multiple Columns	df[['column1', 'column2']]	df.select('column1', 'column2')	Select 'Age' and 'Name' columns:**SQL:** SELECT Age, Name FROM table;**Pandas:** df[['Age', 'Name']]**PySpark:** df.select('Age', 'Name')
SELECT DISTINCT	df['column'].drop_duplicates()	df.select('column').distinct()	Select unique country names:**SQL:** SELECT DISTINCT Country FROM table;**Pandas:** df['Country'].drop_duplicates()**PySpark:** df.select('Country').distinct()
WHERE	df[df['column'] condition]	df.filter(df['column'] condition)	Select rows where age is greater than 30:**SQL:** SELECT * FROM table WHERE Age > 30;**Pandas:** df[df['Age'] > 30]**PySpark:** df.filter(df['Age'] > 30)
ORDER BY	df.sort_values('column')	df.orderBy('column')	Sort data by 'Age' column:**SQL:** SELECT * FROM table ORDER BY Age;**Pandas:** df.sort_values('Age')**PySpark:** df.orderBy('Age')
GROUP BY	df.groupby('column').mean()	df.groupBy('column').agg({'column': 'mean'})	Calculate mean for each 'Country':**SQL:** SELECT Country, AVG(Age) FROM table GROUP BY Country;**Pandas:** df.groupby('Country')['Age'].mean()**PySpark:** df.groupBy('Country').agg({'Age': 'avg'})

JOIN	pd.merge(df1, df2, on='column', how='type')	df1.join(df2, on='column', how='type')	Inner join on 'ID':**SQL:** SELECT * FROM table1 INNER JOIN table2 ON table1.ID = table2.ID;**Pandas:** pd.merge(df1, df2, on='ID', how='inner')**PySpark:** df1.join(df2, on='ID', how='inner')
INSERT INTO	df.append(new_row, ignore_index=True)	df.union(spark.createDataFrame([new_row]))	Insert a new row:**SQL:** INSERT INTO table (Name, Age) VALUES ('John', 30);**Pandas:** df.append({'Name': 'John', 'Age': 30}, ignore_index=True)**PySpark:** df.union(spark.createDataFrame([{'Name': 'John', 'Age': 30}]))
UPDATE	df.loc[df['column'] condition, 'column'] = new_value	df.withColumn('column', F.when(df['column'] condition, new_value).otherwise(df['column']))	Update 'Category' based on condition:**SQL:** UPDATE table SET Category = 'Senior' WHERE Age > 30;**Pandas:** df.loc[df['Age'] > 30, 'Category'] = 'Senior'**PySpark:** df.withColumn('Category', F.when(df['Age'] > 30, 'Senior').otherwise(df['Category']))
DELETE FROM	df = df[~(df['column'] condition)]	df.filter(df['column'] condition)	Delete rows where age is less than 18:**SQL:** DELETE FROM table WHERE Age < 18;**Pandas:** df = df[~(df['Age'] < 18)]**PySpark:** df.filter(df['Age'] >= 18)
LIMIT	df.head(n)	df.limit(n)	Select the first 5 rows:**SQL:** SELECT * FROM table LIMIT 5;**Pandas:** df.head(5)**PySpark:** df.limit(5)

COUNT	df['column'].count()	df.select(F.count('column'))	Count non-null entries:**SQL:** SELECT COUNT(Age) FROM table;**Pandas:** df['Age'].count()**PySpark:** df.select(F.count('Age'))
SUM	df['column'].sum()	df.select(F.sum('column'))	Sum of 'Sales' column:**SQL:** SELECT SUM(Sales) FROM table;**Pandas:** df['Sales'].sum()**PySpark:** df.select(F.sum('Sales'))
AVG (Average)	df['column'].mean()	df.select(F.avg('column'))	Average of 'Price' column:**SQL:** SELECT AVG(Price) FROM table;**Pandas:** df['Price'].mean()**PySpark:** df.select(F.avg('Price'))
MIN (Minimum)	df['column'].min()	df.select(F.min('column'))	Minimum value of 'Age':**SQL:** SELECT MIN(Age) FROM table;**Pandas:** df['Age'].min()**PySpark:** df.select(F.min('Age'))
MAX (Maximum)	df['column'].max()	df.select(F.max('column'))	Maximum value of 'Age':**SQL:** SELECT MAX(Age) FROM table;**Pandas:** df['Age'].max()**PySpark:** df.select(F.max('Age'))
HAVING	df.groupby('column').filter(lambda x: condition)	df.groupBy('column').agg(F.sum('other_column').alias('total')).filter('total > value')	Filter groups with total sales greater than 1000:**SQL:** SELECT Department, SUM(Sales) FROM table GROUP BY Department HAVING SUM(Sales) > 1000;**Pandas:** df.groupby('Department').filter(lambda x: x['Sales'].sum() > 1000)**PySpark:** df.groupBy('Department').agg(F.sum('Sales').alias('total_sales')).filter('total_sales > 1000')

CONCATENATE	df['new_column'] = df['column1'] + df['column2']	df.withColumn('new_column', F.concat_ws(' ', df['column1'], df['column2']))	Concatenate 'FirstName' and 'LastName':**SQL:** `SELECT FirstName
LIKE (Pattern Match)	df[df['column'].str.contains('pattern')]	df.filter(df['column'].like('%pattern%'))	Select rows where 'Name' contains 'John':**SQL:** SELECT * FROM table WHERE Name LIKE '%John%';**Pandas:** df[df['Name'].str.contains('John')]**PySpark:** df.filter(df['Name'].like('%John%'))
IN (List)	df[df['column'].isin(['value1', 'value2'])]	df.filter(df['column'].isin(['value1', 'value2']))	Select rows where 'Country' is 'USA' or 'Canada':**SQL:** SELECT * FROM table WHERE Country IN ('USA', 'Canada');**Pandas:** df[df['Country'].isin(['USA', 'Canada'])]**PySpark:** df.filter(df['Country'].isin(['USA', 'Canada']))
BETWEEN	df[df['column'].between(value1, value2)]	df.filter(df['column'].between(value1, value2))	Select rows where 'Age' is between 18 and 30:**SQL:** SELECT * FROM table WHERE Age BETWEEN 18 AND 30;**Pandas:** df[df['Age'].between(18, 30)]**PySpark:** df.filter(df['Age'].between(18, 30))
CASE WHEN THEN ELSE	df['column'].apply(lambda x: 'value_if_true' if condition else 'value_if_false')	df.withColumn('new_column', F.when(df['column'] condition, 'value_if_true').otherwise('value_if_false'))	Assign 'Adult' or 'Minor' based on 'Age':**SQL:** SELECT CASE WHEN Age >= 18 THEN 'Adult' ELSE 'Minor' END AS Category FROM table;**Pandas:** df['Category'] = df['Age'].apply(lambda x: 'Adult' if x >= 18 else 'Minor')**PySpark:**

			df.withColumn('Category', F.when(df['Age'] >= 18, 'Adult').otherwise('Minor'))
SUBSTRING	df['column'].str.slice(start, end)	df.withColumn('new_column', F.substring('column', start, end))	Extract first letter from 'Name':**SQL:** SELECT SUBSTRING(Name, 1, 1) AS Initials FROM table;**Pandas:** df['Initials'] = df['Name'].str.slice(0, 1)**PySpark:** df.withColumn('Initials', F.substring('Name', 1, 1))
LENGTH	df['column'].str.len()	df.withColumn('new_column', F.length('column'))	Length of each string in 'Name':**SQL:** SELECT LENGTH(Name) AS NameLength FROM table;**Pandas:** df['NameLength'] = df['Name'].str.len()**PySpark:** df.withColumn('NameLength', F.length('Name'))
REPLACE	df['column'].str.replace('old', 'new')	df.withColumn('column', F.regexp_replace('column', 'old', 'new'))	Remove dashes from 'Phone':**SQL:** SELECT REPLACE(Phone, '-', '') AS Phone FROM table;**Pandas:** df['Phone'] = df['Phone'].str.replace('-', '')**PySpark:** df.withColumn('Phone', F.regexp_replace('Phone', '-', ''))
DISTINCT COUNT	df['column'].nunique()	df.select(F.countDistinct('column'))	Count unique countries:**SQL:** SELECT COUNT(DISTINCT Country) FROM table;**Pandas:** df['Country'].nunique()**PySpark:** df.select(F.countDistinct('Country'))