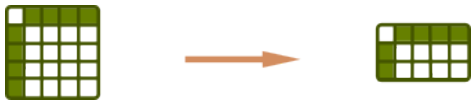
Select

Selecting data with Pandas vs SQL

`df`	**SELECT * FROM tab;**	query data from all columns
`df[['col_1', 'col_2']]`	**SELECT col_1, col_2 FROM tab;**	select subset of columns and get all rows
`df.assign(new_col = df['col_1'] / df['col_2'])`	**SELECT *, col_1/col_2 as new_col FROM tab;**	Aliases - add a calculated column based on other columns
`df[['col 1']]`	**SELECT `col 1` FROM tab;**	Column name with space - `col 1`
`df.sort_values(by='col_1', ascending=True)`	**SELECT * FROM tab ORDER BY col_1 ASC;**	Sort values by col_1 in ascending or descending (DESC) order

Where

Filtering in SQL vs Pandas

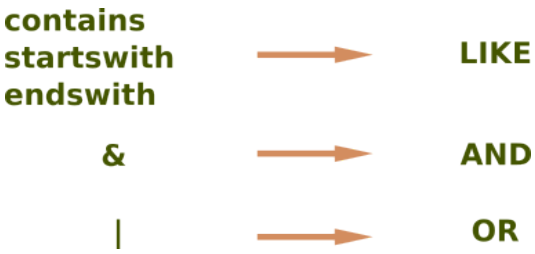


`df[df['col_1'] == '11']`	**SELECT * FROM tab WHERE col_1 = '11';**	Filtering on single condition
`df[(df['col_1'] == 11) & (df['col_2'] > 5)]`	**SELECT * FROM tab WHERE col_1 = 11 AND col_2 > 5;**	Filtering on multiple conditions
`df[df['col_2'].isna()]`	**SELECT * FROM tab WHERE col_2 IS NULL;**	NULL checking is done using the isna()
`df[df['col_1'].notna()]`	**SELECT * FROM tab WHERE col_1 IS NOT NULL;**	NOT NULL checking is done using the notna()
`df[df.col_1 > df.col_2]`	**SELECT * FROM tab WHERE col_1 > col_2;**	Where clause with 2 SQL columns

`df.query('col_1 == col_2')`	**SELECT * FROM tab WHERE col_1 = col_2;**	Filter with Pandas Query
`df.query('col_1 == `col 2`')`	**SELECT * FROM tab WHERE col_1 = `col 2`;**	Column name with space - `col 2` in where clause

Like, and, or

Operators(Text, Logical) in Pandas vs SQL



`df[df['col_1'].str.contains('i', na=False)]`	**WHERE col_1 LIKE '%i%'**	Finds values which contain `i`. Column need to be string - .astype(str)
`df[df['col_1'].str.contains('sh|rd', regex=True, na=True)]`	**WHERE col_1 LIKE '%sh%' OR col_1 LIKE '%rd%'**	Finds any values which contain `sh` or `rd`
`df[df['col_1'].str.startswith('h', na=False)]`	**WHERE col_1 LIKE 'h%'**	Finds any values that start with `h`
`df[df['col_1'].astype(str).str.endswith('k', na=False)]`	**WHERE col_1 LIKE '%k'**	Finds any values that ends with `k`
`(df['col_1'] == '11') & (df['col_2'] > 5)`	**WHERE col_1 = '11' AND col_2 > 5;**	AND = SQL - `and`, Pandas - `&`
`(df['col_1'] == '11') | (df['col_2'] > 5)`	**WHERE col_1 = '11' OR col_2 > 5;**	OR = SQL - `or`, Pandas - `|`
`df[df['col_1'].isin([1,2,3])]`	**SELECT * FROM tab WHERE col_1 in (1,2,3);**	IN operator - find values from list of values
`df[df['col_1'].between(1, 5)]`	**SELECT * FROM tab WHERE col_1 BETWEEN 1 AND 5;**	BETWEEN operator - find values in a range

Group by

Group by operations in SQL vs Pandas

`df.groupby('col_1').size()`
`df.groupby('col_1')['col_2'].count()`
`df.col_1.value_counts()`
	**SELECT col_1, count(*)
	FROM tab**
		count records in each group(3 versions in Pandas)

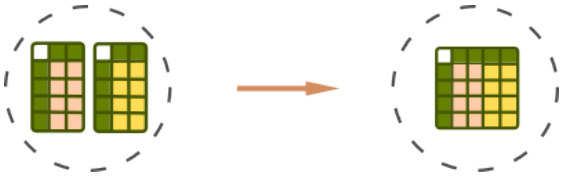
	GROUP BY col_1;	
import numpy as np df.groupby('col_1').agg({'col_2': np.mean, 'col_1': np.size})	SELECT col_1, AVG(col_2), COUNT(*) FROM tab GROUP BY col_1;	Apply multiple statistical functions
df.groupby(['col_1', 'col_2']).agg({'col_3': [np.size, np.mean]})	SELECT col_1, col_2, COUNT(*), AVG(col_3) FROM tab GROUP BY col_1, col_2;	Grouping by multiple columns, multiple functions

# group by 	
g = df.groupby('col_1')
# having count(*) > 10 	
g.filter(lambda x: len(x) > 10)['col_1']
	SELECT col_1, count(*)
	FROM tab
	GROUP BY col_1
	HAVING count(*) > 10;
		HAVING - Group by column and filtering contidion on the groups

df.groupby('col_1').col_2.nunique()	SELECT count(distinct col_2) FROM tab GROUP BY col_1;	count(distinct) - count unique elements in group

Join

Join in SQL and Pandas



pd.merge(df1, df2, on='key')	SELECT * FROM t1 INNER JOIN t2 ON t1.key = t2.key;	Inner join of 2 table/dataframes(1)
pd.merge(df1, df2, on='key', how='left')	SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.key = t2.key;	Left Outer join
pd.merge(df1, df2, on='key', how='right')	SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.key = t2.key;	Right Join
pd.merge(df1, df2, on='key', how='outer')	SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.key = t2.key;	Full Join (not working on MySQL)
pd.merge(df1, df2, left_on= ['col_1', 'col_2'], right_on= ['col_3', 'col_4'], how = 'right')	SELECT * FROM t1 INNER JOIN t2 ON t1.col_1 = t2.col_3 AND t1.col_2 = t2.col_4;	Join on columns with different names

`m = pd.merge(df1, df2, how='left', on=['col_1', 'col_2'])` `pd.merge(m, df3[['col_1', 'col_2', 'col_3']], how='left', on=['col_1', 'col_3'])`	**SELECT t1.col_a, t2.col_b, t3.col_c** **FROM t1** **LEFT OUTER JOIN t2** **ON t1.col_1 = t2.col_1** **AND t1.col_2 = t2.col_2** **LEFT OUTER JOIN t3** **ON t1.col_1 = t3.col_1** **AND t1.col_3 = t3.col_3**	join multiple dataframes on multiple columns

Union

Union in SQL and Pandas

`pd.concat([df1, df2])`	**SELECT *** **FROM df1** **UNION ALL** **SELECT *** **FROM df2;**	Unioun All(columns must have same number of columns)
`cols= ['col_1', 'col_2']` `pd.concat([df1[cols], df2[cols]])`	**SELECT col_1, col_2** **FROM t1** **UNION ALL** **SELECT col_1, col_2** **FROM t2;**	Unioun All
`pd.concat([df1, df2]).drop_duplicates()`	**SELECT col_1, col_2** **FROM t1** **UNION** **SELECT col_1, col_2** **FROM t1;**	Unioun All (remove duplicate rows)

Limit

Limit in SQL and Pandas

`df.head(10)`	**SELECT * FROM tab** **LIMIT 10;**	Get top rows
`df.tail(10)`	**SELECT * FROM tab** **ORDER BY id DESC**	Get last N rows

```
LIMIT 10;
```

```
lim = 2
offset = 5
df.sort_values('col_1',
ascending=False).iloc[offset:lim+offset]
```

```
SELECT * FROM tab
ORDER BY col_1 DESC
LIMIT 2 OFFSET 5;
```

return only top 2 records, start on record 6 (OFFSET 5)

Update

Update in SQL vs Pandas

```
df.loc[df['col_1'] < 2, 'col_1'] *= 2
```

```
UPDATE tab
SET col_1 = col_1*2
WHERE col_1 < 2;
```

Update all rows for 1 column with condition

```
df1['name'] =
np.where(df2['id']==1,df2['name'],df1['name'])
```

```
UPDATE t1,
(
SELECT * FROM t2 WHERE id = 1
) AS temp
SET t1.name = temp.name
WHERE t1.id = 1;
```

Update based on select from another table / dataframe

Delete

Delete in SQL vs Pandas

```
df = df.loc[df['col_1'] > 9]
```

```
DELETE FROM tab
WHERE col_1 > 9;
```

Delete rows with condition

```
df1.drop(df1[(df1.id.isin(df2.id) &
(df1.id==1))].index)
```

```
DELETE t1
FROM df1 as t1
JOIN df2 as t2 ON t1.id = t2.id
WHERE t2.id = 1;
```

Delete rows with condition based on another table / dataframe

Insert

Insert in SQL vs Pandas

```
data = {'col_1': 1, 'col_2': '11'}
df = df.append(data, ignore_index =
True)
```

```
INSERT INTO tab(col_1, col_2)
VALUES (1, '11');
```

Add new data/rows to a table/dataframe