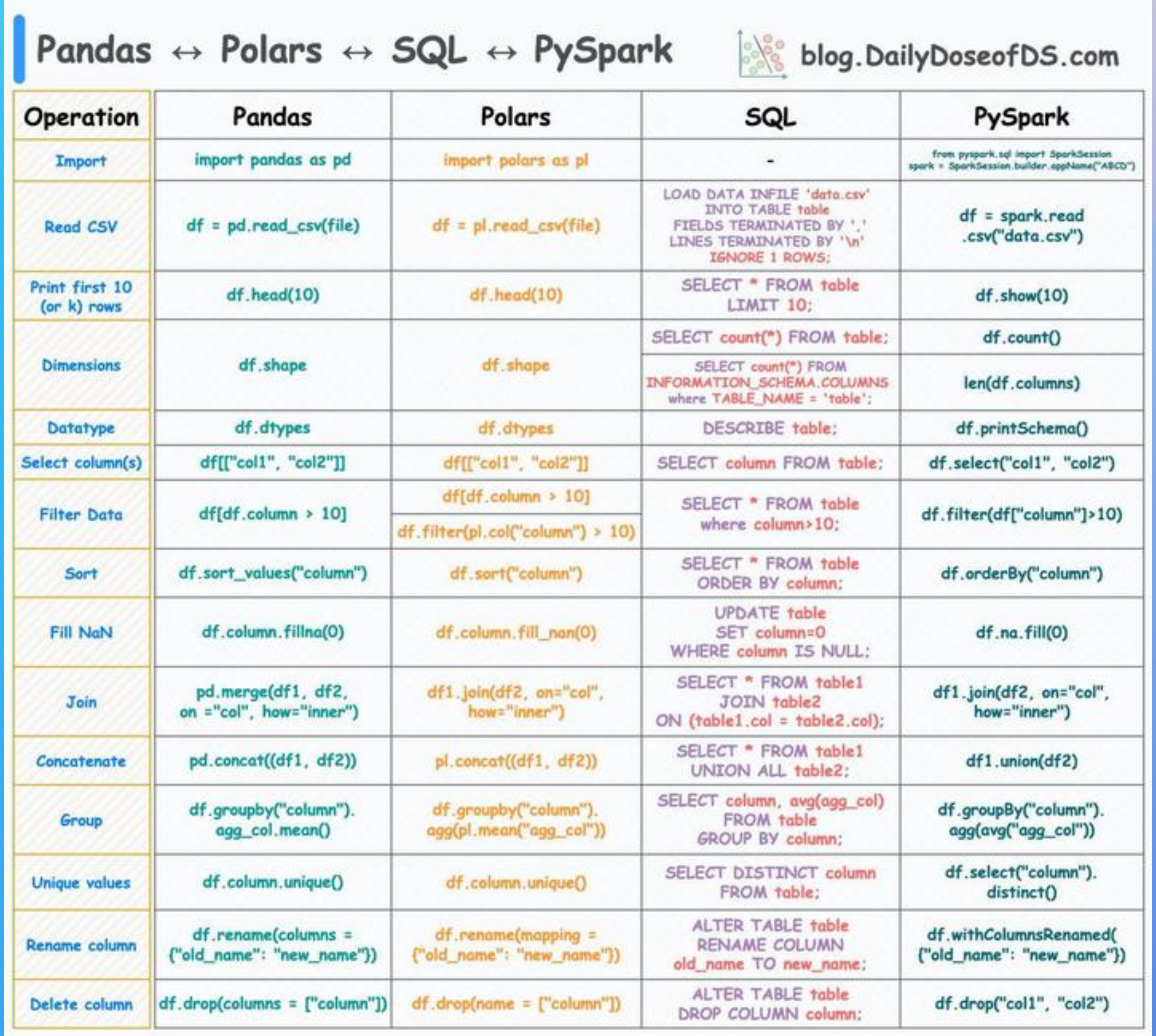
JOINS JOINS



Pandas ↔ Polars ↔ SQL ↔ PySpark

blog.DailyDoseofDS.com

Operation	Pandas	Polars	SQL	PySpark

Import	import pandas as pd	import polars as pl	-	from pyspark.sql import SparkSession
				spark = SparkSession.builder.appName("ABCD")

Read CSV	df = pd.read_csv(file)	df = pl.read_csv(file)	LOAD DATA INFILE 'data.csv'
			INTO TABLE table
			FIELDS TERMINATED BY ','
			LINES TERMINATED BY '\n'
			IGNORE 1 ROWS;
				df = spark.read
				.csv("data.csv")

Print first 10 (or k) rows	df.head(10)	df.head(10)	SELECT * FROM table
			LIMIT 10;
				df.show(10)

Dimensions	df.shape	df.shape	SELECT count(*) FROM table;	df.count()

			SELECT count(*) FROM
			INFORMATION_SCHEMA.COLUMNS
			where TABLE_NAME = 'table';
				len(df.columns)

Datatype	df.dtypes	df.dtypes	DESCRIBE table;	df.printSchema()
Select column(s)	df[["col1", "col2"]]	df[["col1", "col2"]]	SELECT column FROM table;	df.select("col1", "col2")

Filter Data	df[df.column > 10]	df[df.column > 10]
		df.filter(pl.col("column") > 10)
			SELECT * FROM table
			where column>10;
				df.filter(df["column"]>10)

Sort	df.sort_values("column")	df.sort("column")	SELECT * FROM table
			ORDER BY column;
				df.orderBy("column")

Fill NaN	df.column.fillna(0)	df.column.fill_nan(0)	UPDATE table
			SET column=0
			WHERE column IS NULL;
				df.na.fill(0)

Join	pd.merge(df1, df2,
	on ="col", how="inner")
		df1.join(df2, on="col",
		how="inner")
			SELECT * FROM table1
			JOIN table2
			ON (table1.col = table2.col);
				df1.join(df2, on="col",
				how="inner")

Concatenate	pd.concat((df1, df2))	pl.concat((df1, df2))	SELECT * FROM table1
			UNION ALL table2;
				df1.union(df2)

Group	df.groupby("column").
	agg_col.mean()
		df.groupby("column").
		agg(pl.mean("agg_col"))
			SELECT column, avg(agg_col)
			FROM table
			GROUP BY column;
				df.groupBy("column").
				agg(avg("agg_col"))

Unique values	df.column.unique()	df.column.unique()	SELECT DISTINCT column
			FROM table;
				df.select("column").
				distinct()

Rename column	df.rename(columns = {"old_name": "new_name"})	df.rename(mapping = {"old_name": "new_name"})	ALTER TABLE table RENAME COLUMN old_name TO new_name;	df.withColumnsRenamed({"old_name": "new_name"})
Delete column	df.drop(columns = ["column"])	df.drop(name = ["column"])	ALTER TABLE table DROP COLUMN column;	df.drop("col1", "col2")

PYTHON	SQL
head	limit
unique	distinct
nunique	count distinct
sort_values	order by
groupby	group by
merge	join
map	case when then

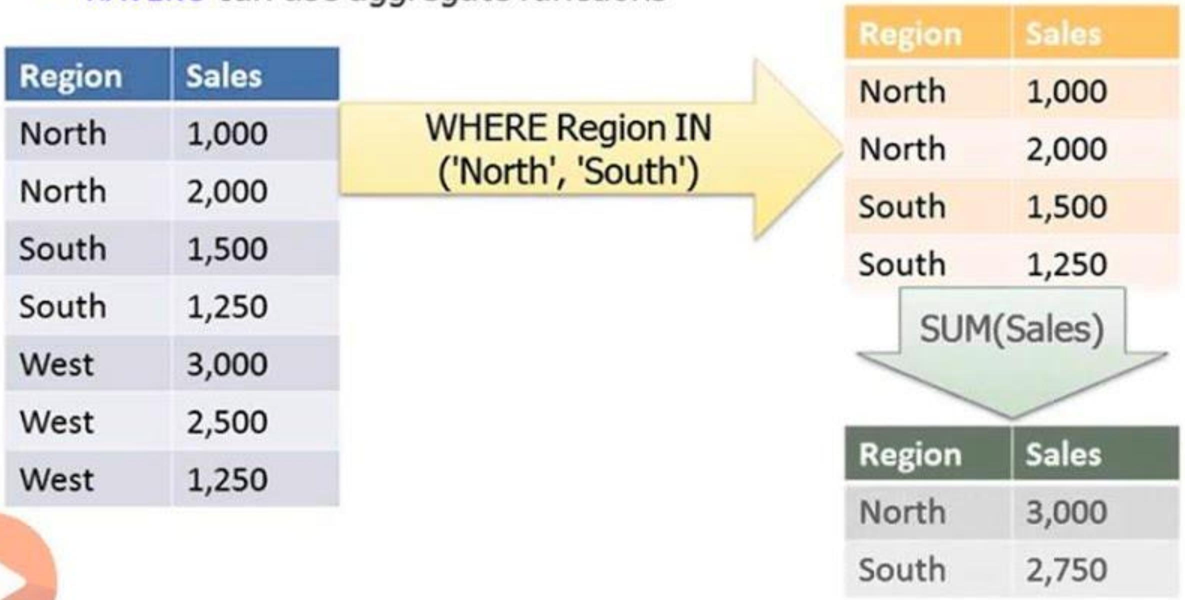
HAVING vs WHERE

HAVING vs. WHERE

WHERE filters rows

HAVING filters groups

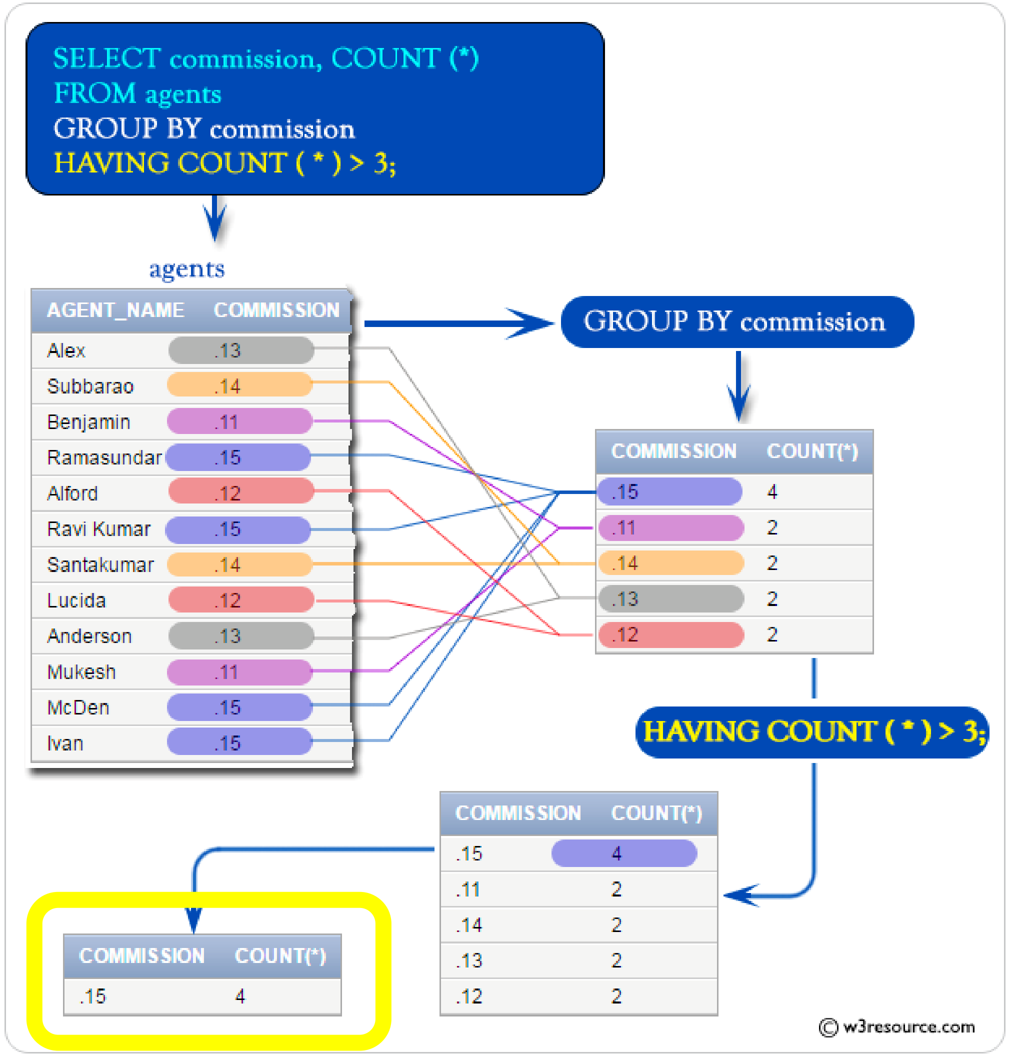
- HAVING can use aggregate functions



Region	Sales
North	1,000
North	2,000
South	1,500
South	1,250
West	3,000
West	2,500
West	1,250

Region	Sales
North	1,000
North	2,000
South	1,500
South	1,250

Region	Sales
North	3,000
South	2,750



```
SELECT commission, COUNT (*)
FROM agents
GROUP BY commission
HAVING COUNT ( * ) > 3;
```

agents

AGENT_NAME	COMMISSION
Alex	.13
Subbarao	.14
Benjamin	.11
Ramasundar	.15
Alford	.12
Ravi Kumar	.15
Santakumar	.14
Lucida	.12
Anderson	.13
Mukesh	.11
McDen	.15
Ivan	.15

GROUP BY commission

COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

HAVING COUNT (*) > 3;

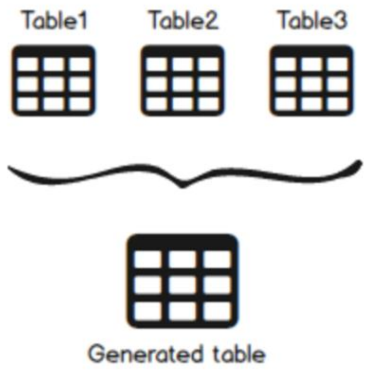
COMMISSION	COUNT(*)
.15	4
.11	2
.14	2
.13	2
.12	2

COMMISSION	COUNT(*)
.15	4

JOINS

JOINING TWO TABLES: JOIN

- Unfortunately, after normalization we end up with the information split in many tables. Thus, sometimes we want to gather info which is split in several tables.



JOINING TWO TABLES: JOIN

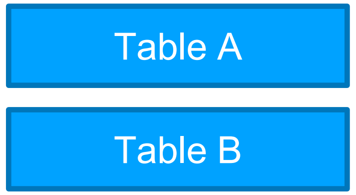
- Joining two tables in not so easy as it may look like. We need to answer some questions before:

- How, vertically or horizontally? (concat, merge in Python)?

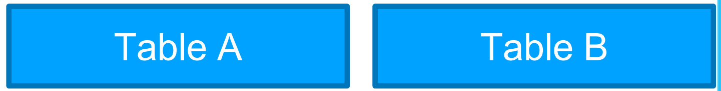
- What about the records inside in each table, should be on both or only in one?

- Join is a method to **combine tables horizontally**

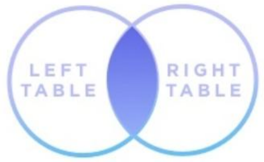
Combining Data Vertically



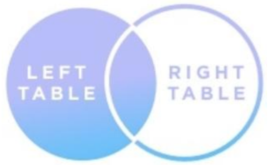
Combining Data Horizontally



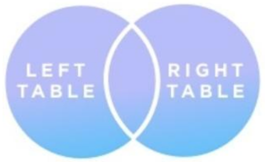
Types of joins



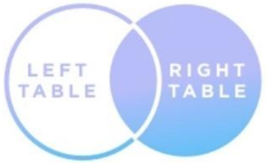
Inner Join



Left Join



Full Join



Right Join

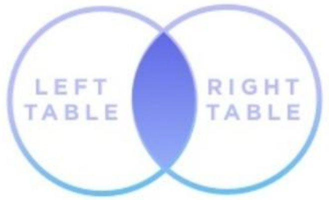
JOIN SYNTAX

- Join syntax is:

```
SELECT ta.col_n, …., t_b.col_m FROM table_a AS ta
TYPE_OF JOIN table_b AS tb
ON ta.col_x = tb.col_y
```

Where TYPEOF = (INNER, LEFT, RIGHT, FULL)

TYPES OF JOINS: INNER JOIN

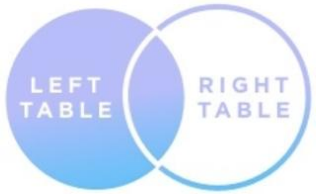


- INNER JOIN (default) returns only the matching rows in both tables.

SELECT ta.col_n, …., t_b.col_m FROM table_a AS ta
INNER JOIN table_b AS tb
ON ta.col_x = tb.col_y

```
select * from bank.account as a
inner join bank.loan as l on
a.account_id = l.account_id
```

TYPES OF JOINS: LEFT JOIN

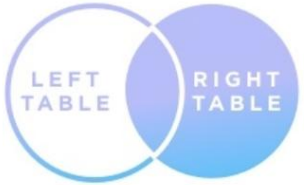


- LEFT JOIN returns a table with the all the rows on the left table and the matching ones on the right.
- For non matching rows, NULL values are returned.

SELECT ta.col_n, ..., t_b.col_m FROM table_a AS ta
LEFT JOIN table_b AS tb
OB ta.col_x = tb.col_y

```
select * from bank.account as a
left join bank.loan as l
on a.account_id = l.account_id
```

TYPES OF JOINS: RIGHT JOIN



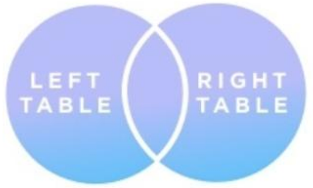
- RIGHT JOIN returns a table with the all the rows on the right table and the matching ones on the right.

- For non matching rows, NULL values are returned.

SELECT ta.col_n, …., t_b.col_m FROM table_a AS ta
RIGHT JOIN table_b AS tb
OB ta.col_x = tb.col_y

```
select * from bank.account as a
right join bank.loan as l
on a.account_id = l.account_id
```

TYPES OF JOINS: FULL JOIN



- FULL JOIN returns a table with the all the rows both tables
- In MySQL full join needs to UNION a LEFT join with a RIGHT join

```
SELECT ta.col_n, …., t_b.col_m FROM table_a AS ta
LEFT JOIN table_b AS tb
ON ta.col_x = tb.col_y
UNION
SELECT ta.col_n, …., t_b.col_m FROM table_a AS ta
RIGHT  JOIN table_b AS tb
ON ta.col_x = tb.col_y
```

```
select * from bank.account as a
left join bank.loan as l
on a.account_id = l.account_id
union
select * from bank.account as a
right join bank.loan as l
on a.account_id = l.account_id;
```

IRON HACK

THANKS !