HiveQL
Queries and Views

Munteanu Bianca-Ștefania

Ce este HiveQL?

HiveQL (Hive Query Language) este un limbaj de interogare similar cu SQL, utilizat pentru a interoga și gestiona datele stocate în Apache Hive, un sistem de data warehousing construit pe Apache Hadoop.

SQL vs HiveQL

01

Sintaxă Familiară

Utilizatorii familiarizați cu SQL se adapteaza usor

02

Operațiuni de Bază

SELECT, INSERT, UPDATE, DELETE...

03

Funcții de Agregare

COUNT, SUM, AVG, MIN, MAX...

04

Tipuri de Date Complexe

ARRAY, MAP și STRUCT

05

Optimizare pentru Big Data

Volume mari de date stocate în HDFS

06

Execuție prin MapReduce

Transformă interogările HiveQL în joburi MapReduce

Utilizare Colecții - ARRAY

Array: Colecție ordonată de elemente de același tip.

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

```
hive> SELECT name, subordinates FROM employees;
John Doe     ["Mary Smith","Todd Jones"]
Mary Smith   ["Bill King"]
Todd Jones   []
Bill King    []
```

Utilizare Colecții - MAP

Map: Colecție de perechi cheie-valoare, unde cheile sunt unice.

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

```
hive> SELECT name, deductions FROM employees;
John Doe     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Mary Smith   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}
Todd Jones   {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}
Bill King    {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}
```

Utilizare Colecții - STRUCT

Struct: Colecție de elemente eterogene, fiecare având un nume și un tip specific.

```
CREATE TABLE employees (
  name         STRING,
  salary       FLOAT,
  subordinates ARRAY<STRING>,
  deductions   MAP<STRING, FLOAT>,
  address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

```
hive> SELECT name, address FROM employees;
John Doe    {"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith  {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Todd Jones  {"street":"200 Chicago Ave.","city":"Oak Park","state":"IL","zip":60700}
Bill King   {"street":"300 Obscure Dr.","city":"Obscuria","state":"IL","zip":60100}
```

Funcții

Funcții Matematice:

- ROUND
- FLOOR
- CEIL (CEILING)

```
hive> SELECT upper(name), salary, deductions["Federal Taxes"],
    > round(salary * (1 - deductions["Federal Taxes"])) FROM employees;

JOHN DOE     100000.0  0.2   80000
MARY SMITH    80000.0  0.2   64000
TODD JONES    70000.0  0.15  59500
BILL KING     60000.0  0.15  51000
```

Funcții de Agregare:

- COUNT
- AVG
- MIN/MAX

```
hive> SET hive.map.aggr=true;

hive> SELECT count(*), avg(salary) FROM employees;

4  77500.0
```

Clauza WHERE

Sintaxa:

SELECT coloane
FROM tabel
WHERE condiție;

- **Utilizarea operatorilor logici**

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'CA';
```

- **Utilizarea funcțiilor în WHERE**

```
hive> SELECT name, salary, deductions["Federal Taxes"],
    >   salary * (1 - deductions["Federal Taxes"])
    > FROM employees
    > WHERE round(salary * (1 - deductions["Federal Taxes"])) > 70000;
John Doe    100000.0  0.2   80000.0
```

LIKE și RLIKE

- **LIKE**: Permite potrivirea stringurilor care încep, se termină sau conțin un anumit substring

```
hive> SELECT name, address.street FROM employees WHERE address.street LIKE '%Ave.';
John Doe        1 Michigan Ave.
Todd Jones      200 Chicago Ave.
```

- **RLIKE (REGEXP):** O extensie Hive care permite utilizarea expresiilor regulare Java

```
hive> SELECT name, address.street
    > FROM employees WHERE address.street RLIKE '.*(Chicago|Ontario).*';
Mary Smith      100 Ontario St.
Todd Jones      200 Chicago Ave.
```

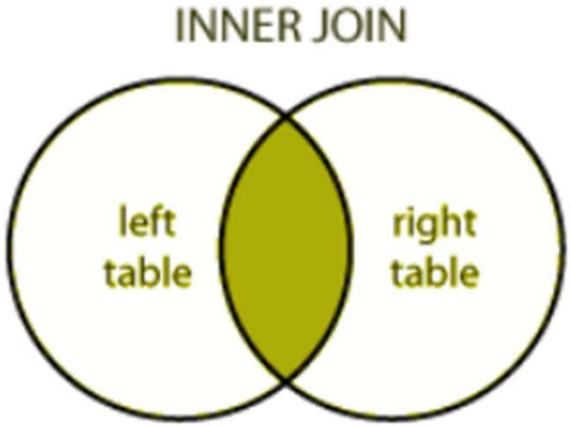
GROUP BY și HAVING

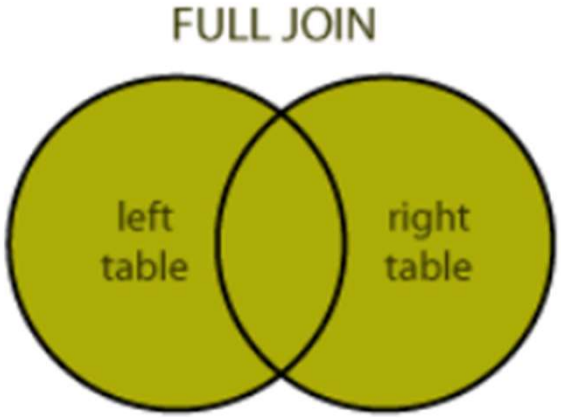
- **GROUP BY:** Utilizat pentru a grupa rândurile care au aceleași valori într-una sau mai multe coloane. Acesta este adesea folosit împreună cu funcții de agregare

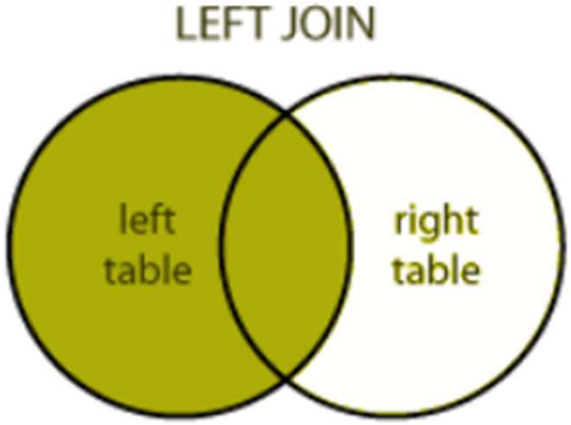
- **HAVING:** Utilizat pentru a filtra grupurile de rânduri create de clauza GROUP BY. Spre deosebire de WHERE, HAVING poate filtra rezultatele pe baza funcțiilor de agregare.

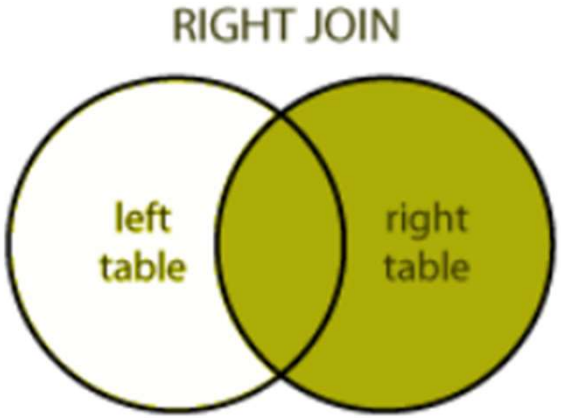
```
SELECT coloane, funcții_agregare
FROM tabel
WHERE condiție
GROUP BY coloane
HAVING condiție_agregare;
```

```
hive> SELECT year(ymd), avg(price_close) FROM stocks
    > WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'
    > GROUP BY year(ymd)
        > HAVING avg(price_close) > 50.0;
1987    53.88968399108163
1991    52.49553383386182
1992    54.80338610251119
1999    57.77071460844979
2000    71.74892876261757
2005    52.401745992993554
...
```









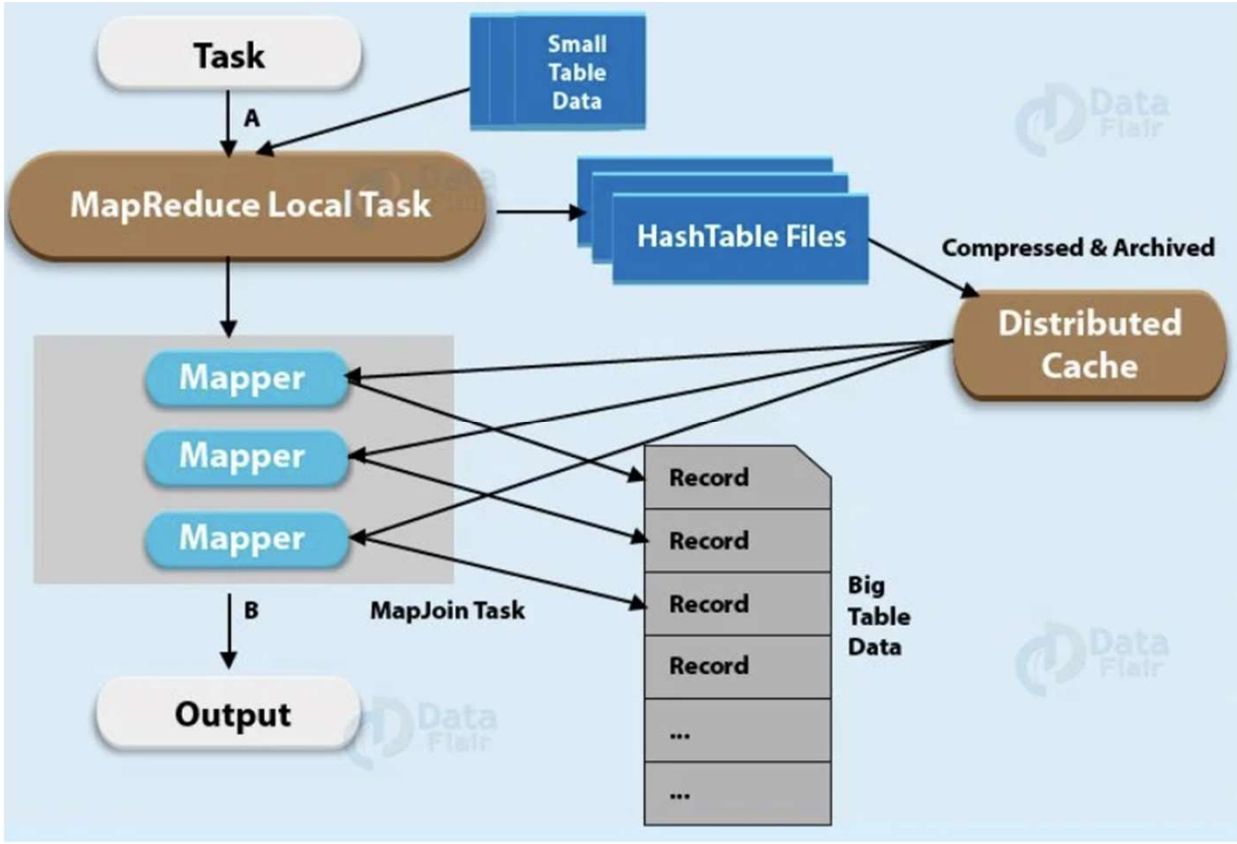
JOIN

- Utilizat pentru a combina rânduri din două sau mai multe tabele pe baza unei coloane comune.
- Hive suportă doar equi-join-uri.

Map-side JOIN

- Optimizare care permite efectuarea operațiilor de join în faza de mapare atunci când toate tabelele, cu excepția unuia, sunt suficient de mici pentru a fi încărcate în memorie.
- Elimină necesitatea fazei de reduce, accelerând execuția
- Se poate seta marimea tabelului mic

```
hive.mapjoin.smalltable.filesize=25000000
```



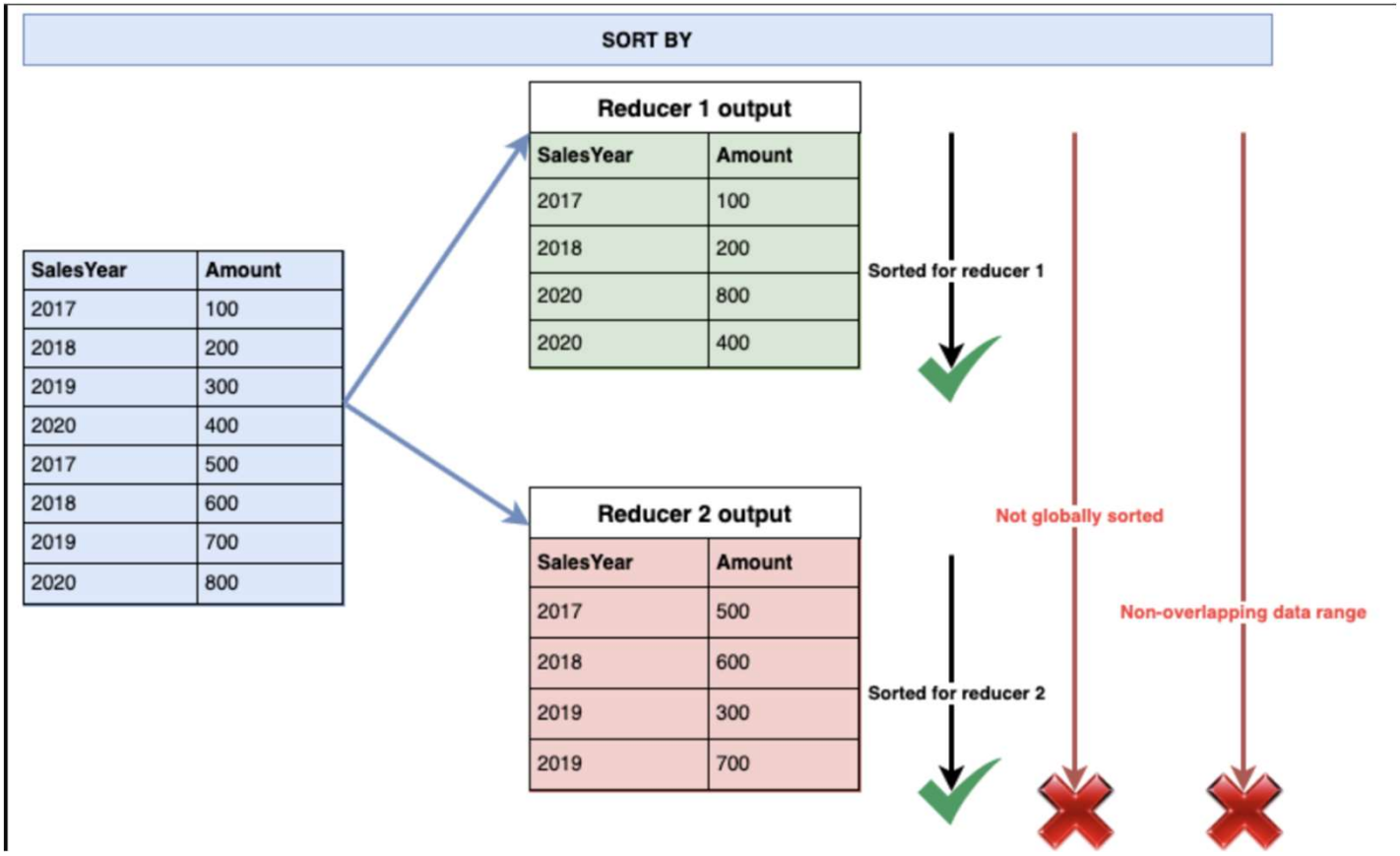

```
hive> set hive.auto.convert.join=true;

hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM stocks s JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'AAPL';
```

SORT BY

- sortează datele pe fiecare reducer în parte

SELECT coloane
FROM tabel
SORT BY coloană ASC | DESC;

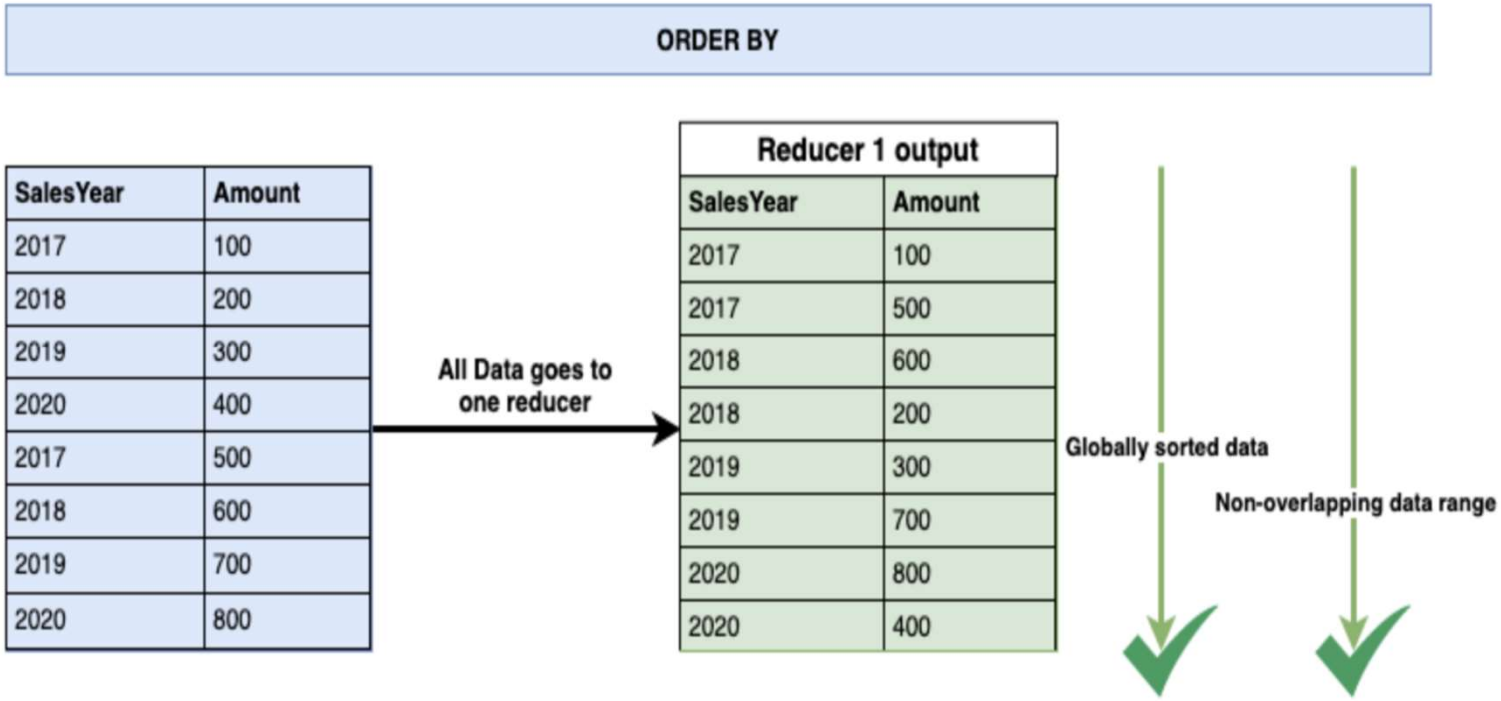



```
1  SELECT SalesYear, Amount
2  FROM tbl_Sales
3  SORT BY SalesYear;
```

ORDER BY

- asigură ordonarea globală a datelor.
- toate datele sunt trecute printr-un singur reducer pentru a obține un set de date sortat global

SELECT coloane
FROM tabel
ORDER BY coloană ASC | DESC;

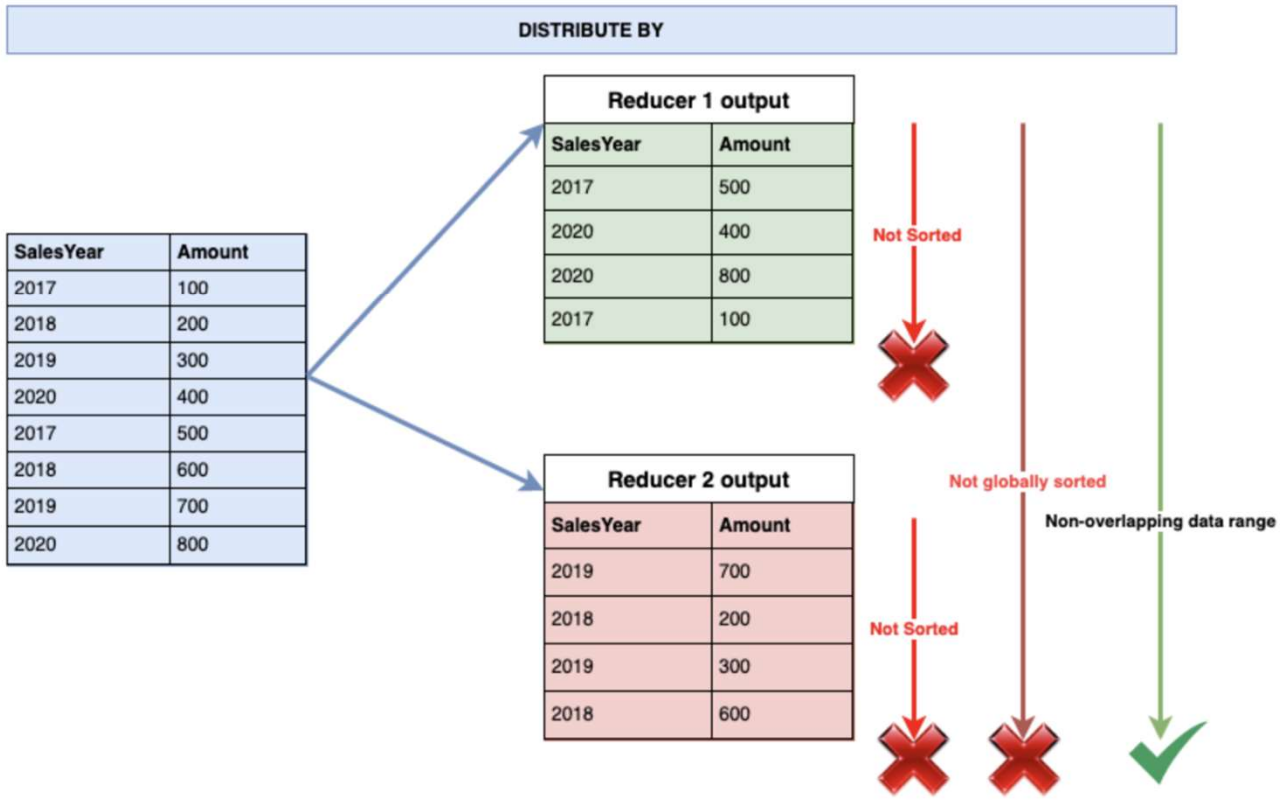



```
SELECT SalesYear, Amount
FROM tbl_Sales
ORDER BY SalesYear;
```

DISTRIBUTE BY

```
1  SELECT SalesYear, Amount
2  FROM tbl_Sales
3  DISTRIBUTE BY SalesYear;
```

SELECT coloane
FROM tabel
DISTRIBUTE BY coloană;

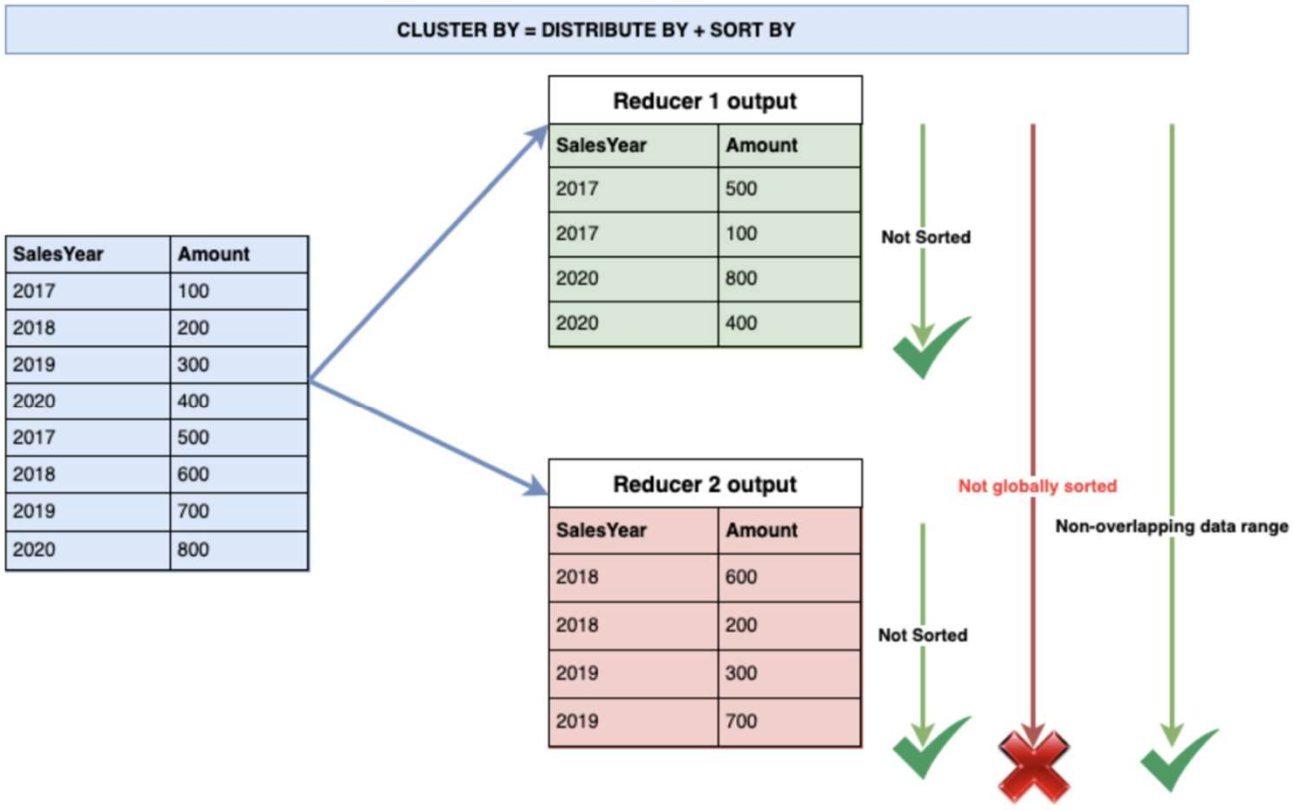


- controlează modul în care datele sunt distribuite între reduceri, bazat pe valorile coloanelor specificate.

CLUSTER BY

```
1  SELECT SalesYear, Amount
2  FROM tbl_Sales
3  CLUSTER BY SalesYear;
```

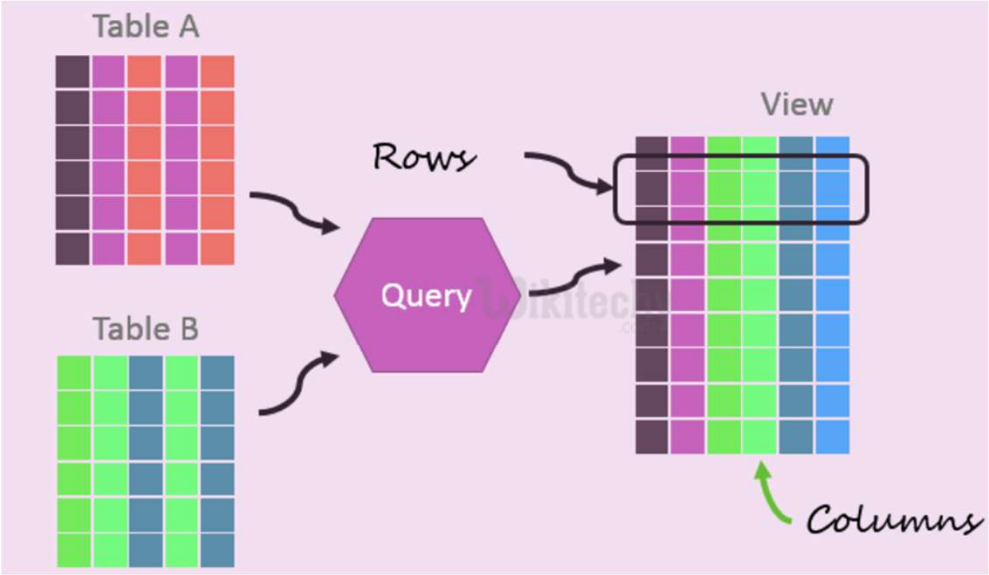
SELECT coloane
FROM tabel
CLUSTER BY coloană;



- combinație de DISTRIBUTE BY și SORT BY, folosind aceleași coloane pentru ambele operații.

View

- permite salvarea unei interogări și tratarea acesteia ca un tabel
- un construct logic care nu stochează date
- Hive nu suportă view-uri materializate
- Reducerea Complexității Interogărilor
- Restricționarea Accesului




```
CREATE VIEW shorter_join AS
SELECT * FROM people JOIN cart
ON (cart.people_id=people.id) WHERE firstname='john';
```

- Stergere View

```
DROP VIEW IF EXISTS shipments;
```

- Modificare View

```
ALTER VIEW shipments SET TBLPROPERTIES ('created_at' = 'some_timestamp');
```

View pe tabel dinamic

- Un tabel dinamic utilizează tipuri de date complexe, cum ar fi MAP, ARRAY și STRUCT, pentru a stoca date flexible.

```
CREATE EXTERNAL TABLE dynamictable(cols map<string,string>)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\004'
  COLLECTION ITEMS TERMINATED BY '\001'
  MAP KEYS TERMINATED BY '\002'
STORED AS TEXTFILE;
```

```
CREATE VIEW orders(state, city, part) AS
SELECT cols["state"], cols["city"], cols["part"]
FROM dynamictable
WHERE cols["type"] = "request";
```

THANK YOU!