

Databases, OLAP & Apache Druid

Lakshminarayana Chari

LNC.ADONI@gmail.com

<https://www.linkedin.com/in/lakshminarayana-chari-7547757/>

https://github.com/lnchariadoni/apache_druid_self_learn

TOC

- **Databases**

- RDBMS
 - Row, Columnar Databases
- Distributed Databases
 - HBase, Cassandra & MongoDB
- Columnar File formats
 - Parquet vs ORC

- **Apache Druid**

- Apache Druid
- Ingestion
- Segments
- Transformations
- Probabilistic Algorithms – HLL, DataSketch
- Native, SQL Querying
- Lookups, Multi Column Values

Databases

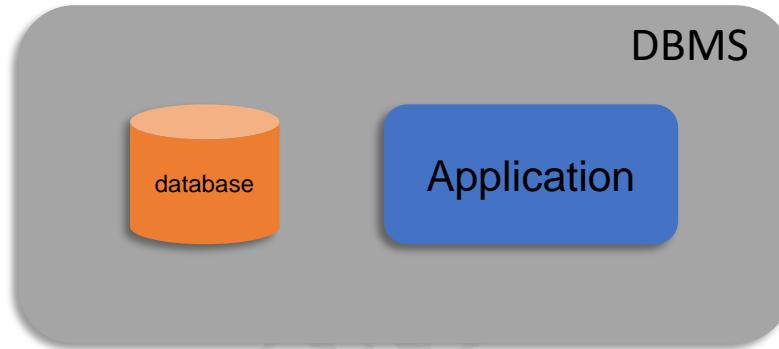
- **Evolution of Databases**
 - RDBMS – 70's, 80's, 90's and 21st century
- **Different types of DBMS**
 - Storage Pattern
 - Read/Write Pattern
 - Row based vs Column based
- **Why other types of databases**
 - CAP Theorem
 - Key-Value Pair – HBase , Cassandra
 - Storage Pattern
 - Read/Write Pattern
 - Document Store – Mongo
 - Example Use case
 - HBase vs Cassandra vs Mongo
- **Columnar File formats**
 - Need for different file formats
 - Parquet vs ORC

Evolution of Databases

- RDBMS – First proposed by E.F. Codd in 70's.

Database

- Organized collection of data
- Defined data types



DBMS

- Responsible for database
- ACID
- User interaction and querying.

- Codd proposed rules for RDBMS – Relational Database Management System.
- 80s
 - DB2, Oracle, Ingres, Sybase...
 - First version of SQL-89 specification released.
- Late 80s - 90s
 - Birth of enterprise class OLTP applications – CRM, SCM.
 - Standardization of SQL across RDBMS (DB2, Oracle, Sybase, SQL Server...)
 - Features & Functionality driven by OLTP applications. (ex: Views, Materialized Views...)

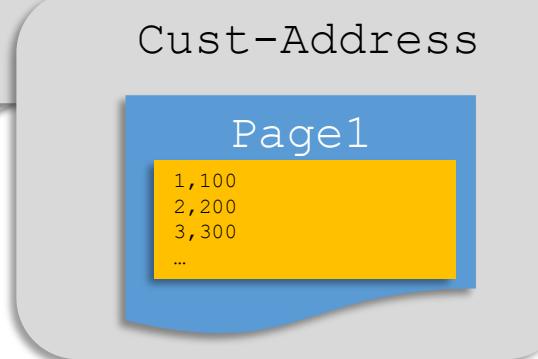
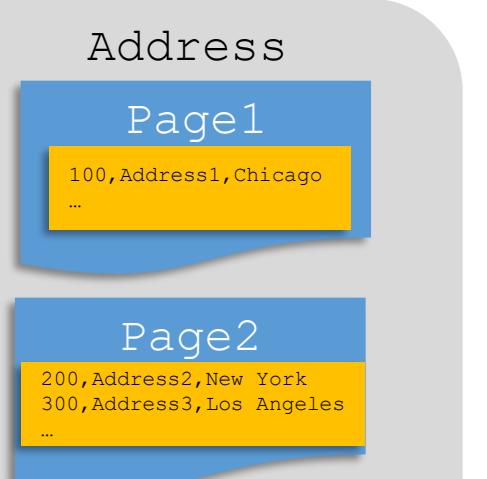
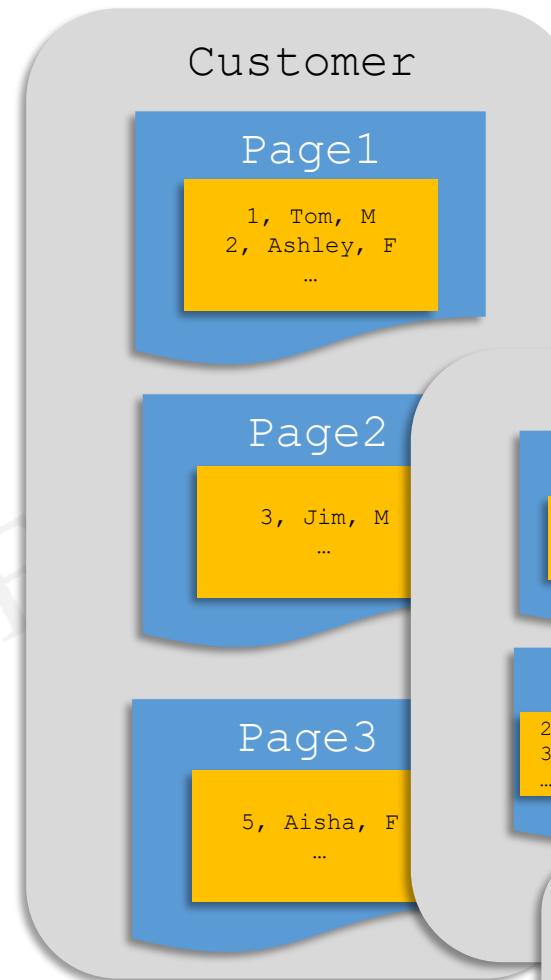
Row Based RDBMS

- Rows are stored in chunks/blocks/pages
 - Horizontally splitting table
 - Number of Pages ~ table size/rows
- Used for OLTP applications
 - Measurement of performance in IOPS – write latency
 - Normalized Schema – 3NF
- Access is through DML and DDL
 - Standard SQL, with variations
 - Designed for consistency. i.e in-built integrity constraints
- Examples – Oracle (1979), DB2 (1983), SQL-Server (1993), Sybase (1988)...

CID	Name	Gender
1	Tom	M
2	Ashley	F
3	Jim	M

AID	Street	City
100	Address1	Chicago
200	Address2	New York
300	Address3	Los Angeles

CID	SID
1	100
2	200
3	300



Row Based RDBMS – Write

update set
name='Jimmy' where
name = 'Jim'



?

Customer

Page1

1, Tom, M
2, Ashley, F
...

Page2

4, Jim, M
...

Page3

5, Aisha, F
8, Rekha, F
...

insert (8, Rekha, F)



Address

Page1

100,Address1,Chicago
...

Page2

200,Address2>New York
300,new_address,Los
Angeles

update SET address
='new_address' where
AID=300



Cust-Address

Page1

1,100
2,200
3,300
...

Row Based RDBMS – Read

Select Gender, City,
count(CID) from Customer,
Address, Cust-Address where
<join> group by Gender, City



- Results in full tablescan of all tables.
- All columns of all tables will be loaded into memory from disk.
- More tables or attributes of an entity, the Disk i/o increases proportionately.

Customer

Page1

1, Tom, M
2, Ashley, F
...

Page2

4, Jim, M
...

Page3

5, Aisha, F
...

Address

Page1

100,Address1,Chicago
...

Page2

200,Address2,New York
300,Address3,Los Angeles
...

Select AID from Address
where city = 'Chicago'



- Results in full tablescan.
- Can be avoided with indexes. But on which column/how many columns to apply indexes?
- Though only AID is used in select, all columns of the table would be loaded from disk

Cust-Address

Page1

1,100
2,200
3,300
...

Row Based RDBMS

Purpose	<ul style="list-style-type: none">• OLTP
Schema Design	<ul style="list-style-type: none">• 3NF• No redundancy
Access	<ul style="list-style-type: none">• Optimized for Write heavy operations• Write results in specific page access• Read always accesses entire row, irrespective of which columns are required.
Schema, Structure	<ul style="list-style-type: none">• Fixed Schema & Structure• Can't add columns dynamically• Predefined/Fixed Column datatypes
Others	<ul style="list-style-type: none">• Inbuilt support for constraints• Inbuilt support for joins• Access is always through PK/FK. Other column access is inefficient or requires indexing.• Data pages are mutable.

Why
Column
based
RDBMS?

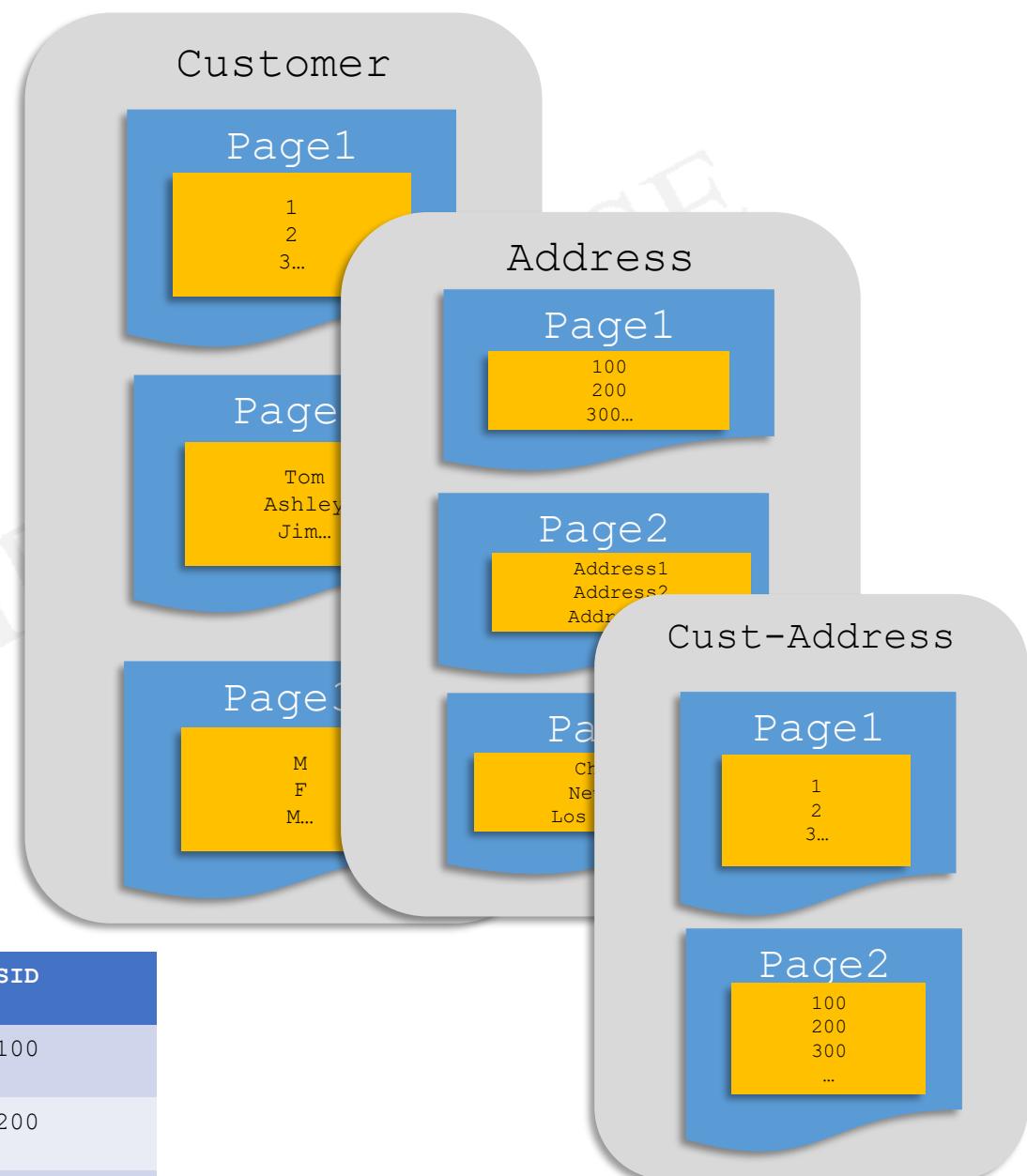
Column Based RDBMS

- Columns are stored in chunks/blocks/pages
 - Vertically splitting table
 - Number of pages ~ number of columns
- Used for OLAP applications
 - Star/Snowflake Schema Design
- Access is through DML and DDL
 - Standard SQL, with variations
 - Supports integrity constraints
- Examples – Vertica (2005), Greenplum(2003), Redshift(2012) ...

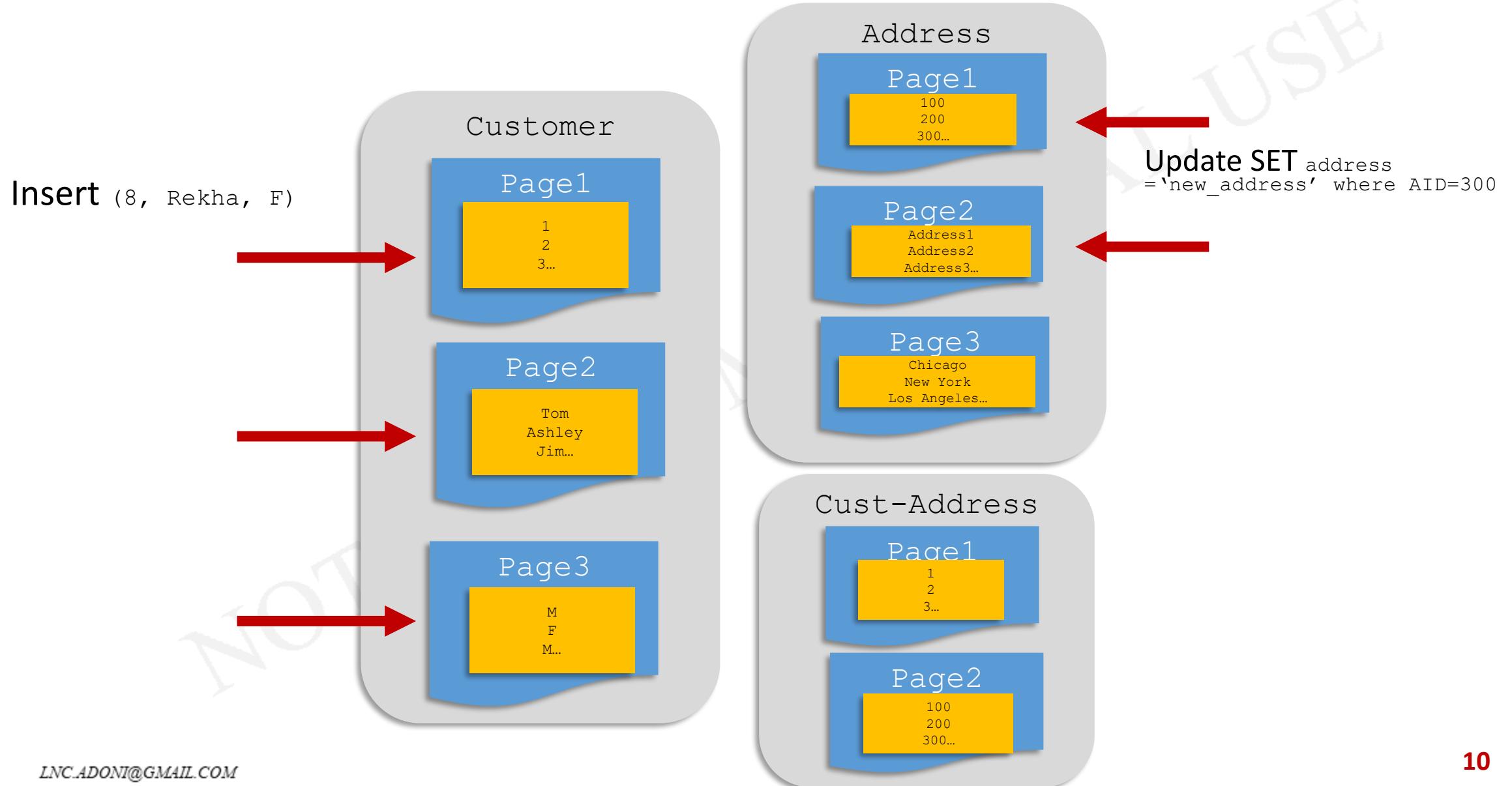
CID	Name	Gender
1	Tom	M
2	Ashley	F
3	Jim	M

SID	Street	City
100	Address1	Chicago
200	Address2	New York
300	Address3	Los Angeles

CID	SID
1	100
2	200
3	300



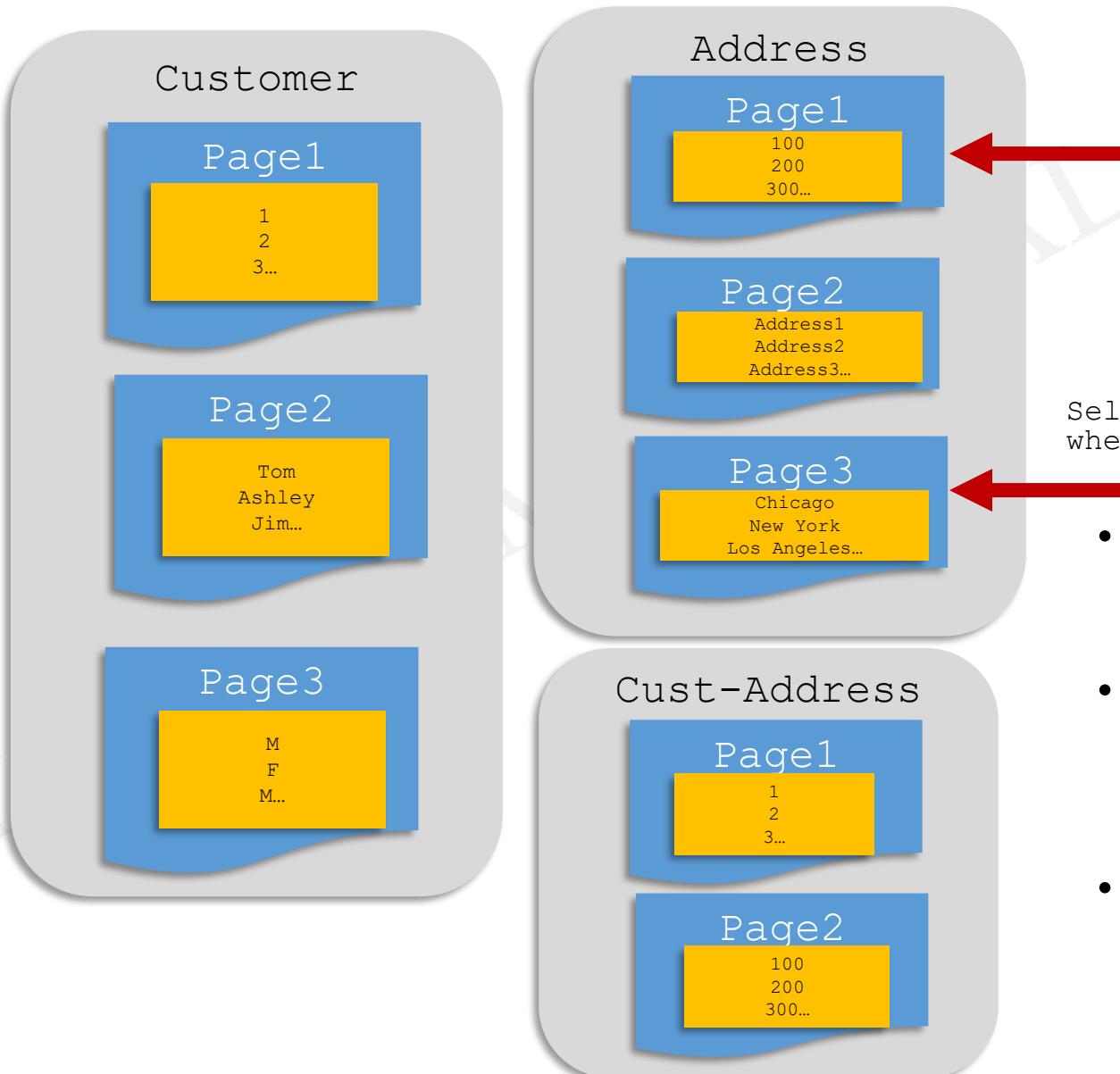
Column Based RDBMS – Write



Column Based RDBMS – Read

Select Gender, City, count(CID) from Customer, Address, Cust-Address where <join> group by Gender, City

- Reads only specific/required columns from all the tables.
- Limited and efficient disk i/o.



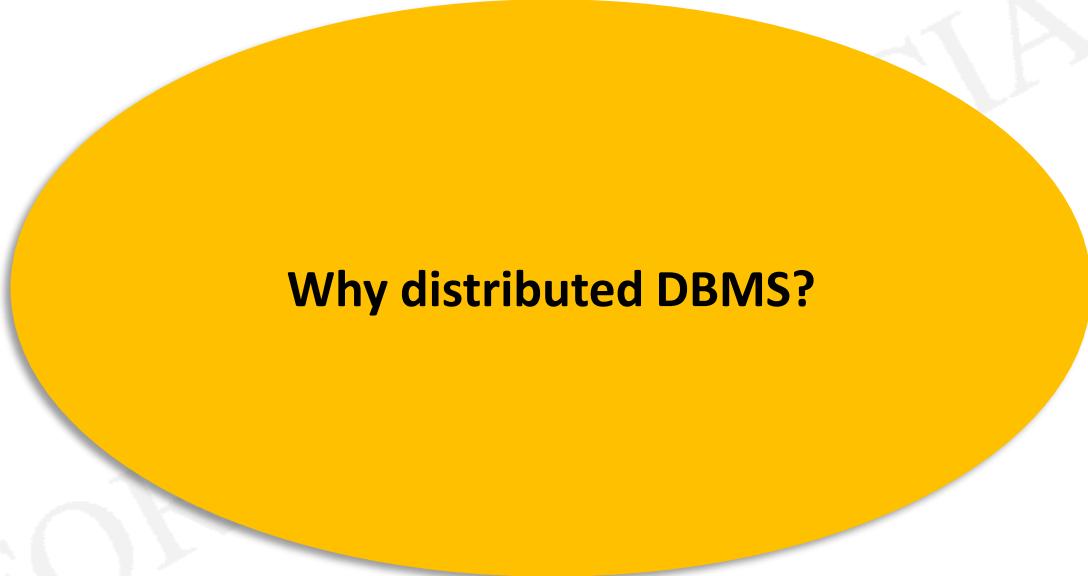
Select AID from Address where city = 'Chicago'

- Reads **Page3** of the Address table to apply filter condition.
- Based on the result of the filter condition it reads **Page1** to retrieve AID column values.
- All column values for each column/page are indexed and stored for efficient access.

Row vs Column RDBMS

	Row	Column
Purpose	<ul style="list-style-type: none">• OLTP	<ul style="list-style-type: none">• OLAP
Design Approach	<ul style="list-style-type: none">• Entity-Relationship	<ul style="list-style-type: none">• Dimensional
Schema Design	<ul style="list-style-type: none">• 3NF• No redundancy	<ul style="list-style-type: none">• Star Schema, Snow-flake Schema• Scope for redundancy
Access	<ul style="list-style-type: none">• Optimized for Write heavy operations.• Write results in specific page access• Read always accesses entire row, irrespective of which columns are required.	<ul style="list-style-type: none">• Optimized for Read heavy operations• Write results in accessing all pages of the table.• Read always accesses specific column pages
Schema, Structure	<ul style="list-style-type: none">• Fixed Schema & Structure• Can't add columns dynamically• Predefined/Fixed Column datatypes	<ul style="list-style-type: none">• Fixed Schema & Structure• Can't add columns dynamically• Predefined/Fixed Column datatypes
Others	<ul style="list-style-type: none">• Inbuilt support for constraints• Inbuilt support for joins• Access is always through PK/FK. Other column access is inefficient or requires indexing.• Data pages are mutable.	<ul style="list-style-type: none">• Inbuilt support for constraints• Inbuilt support for joins• Access through any column is possible. Retrieving all columns is expensive.• Data pages are mutable.

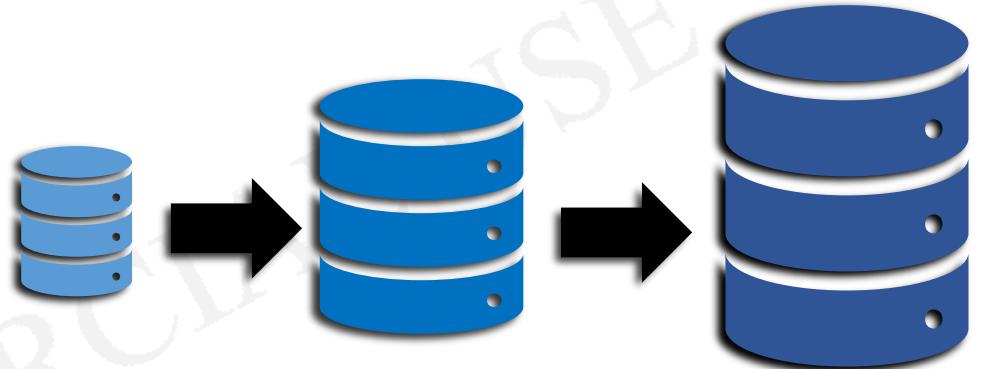
Row vs Column RDBMS ...



Why distributed DBMS?

Limitations of RDBMS

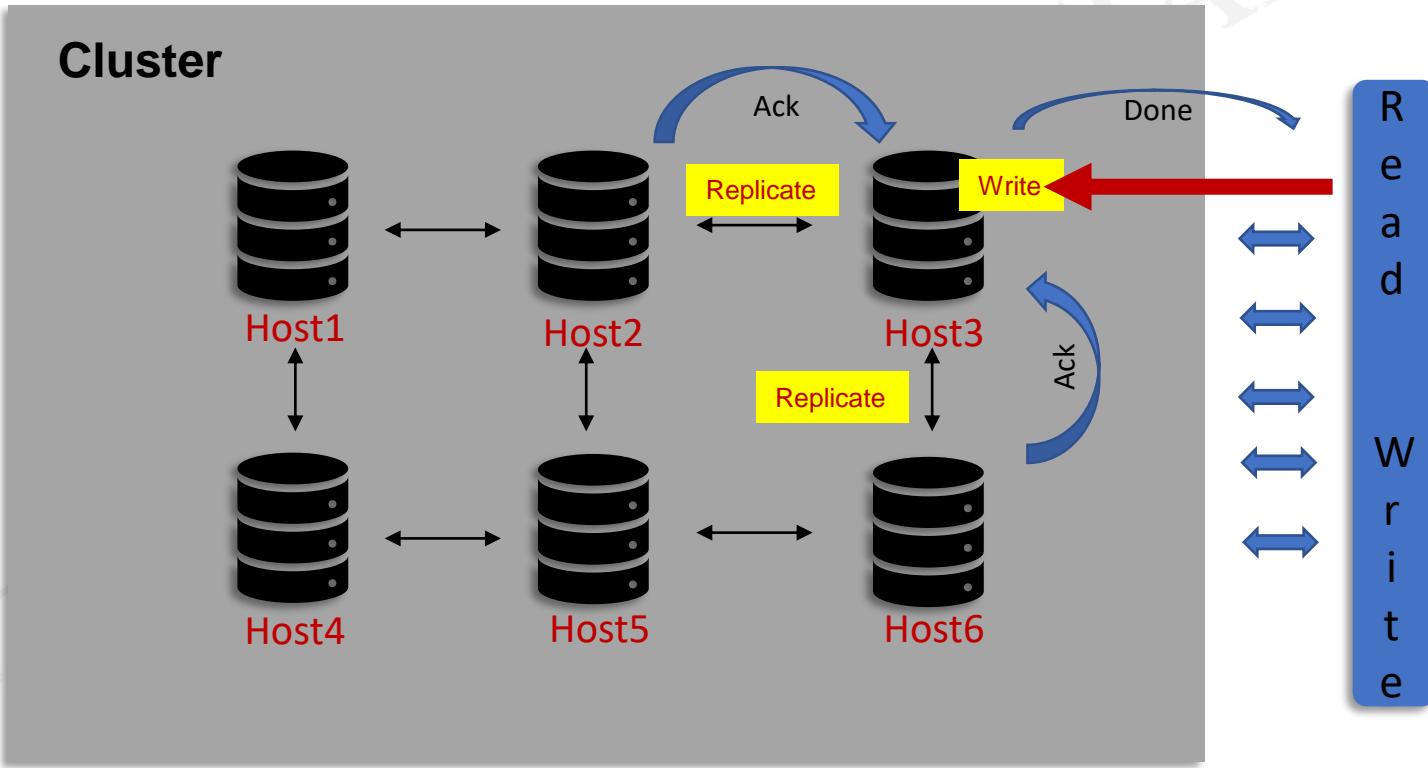
- Not designed for scale
 - As data volume increases, RDBMS can scale only vertically.
 - Single node hardware
- No flexibility in Structure, Schema
 - Structure & Schema must be pre-defined
- Cases where consistency is not a primary goal
 - Eventual consistency is acceptable.



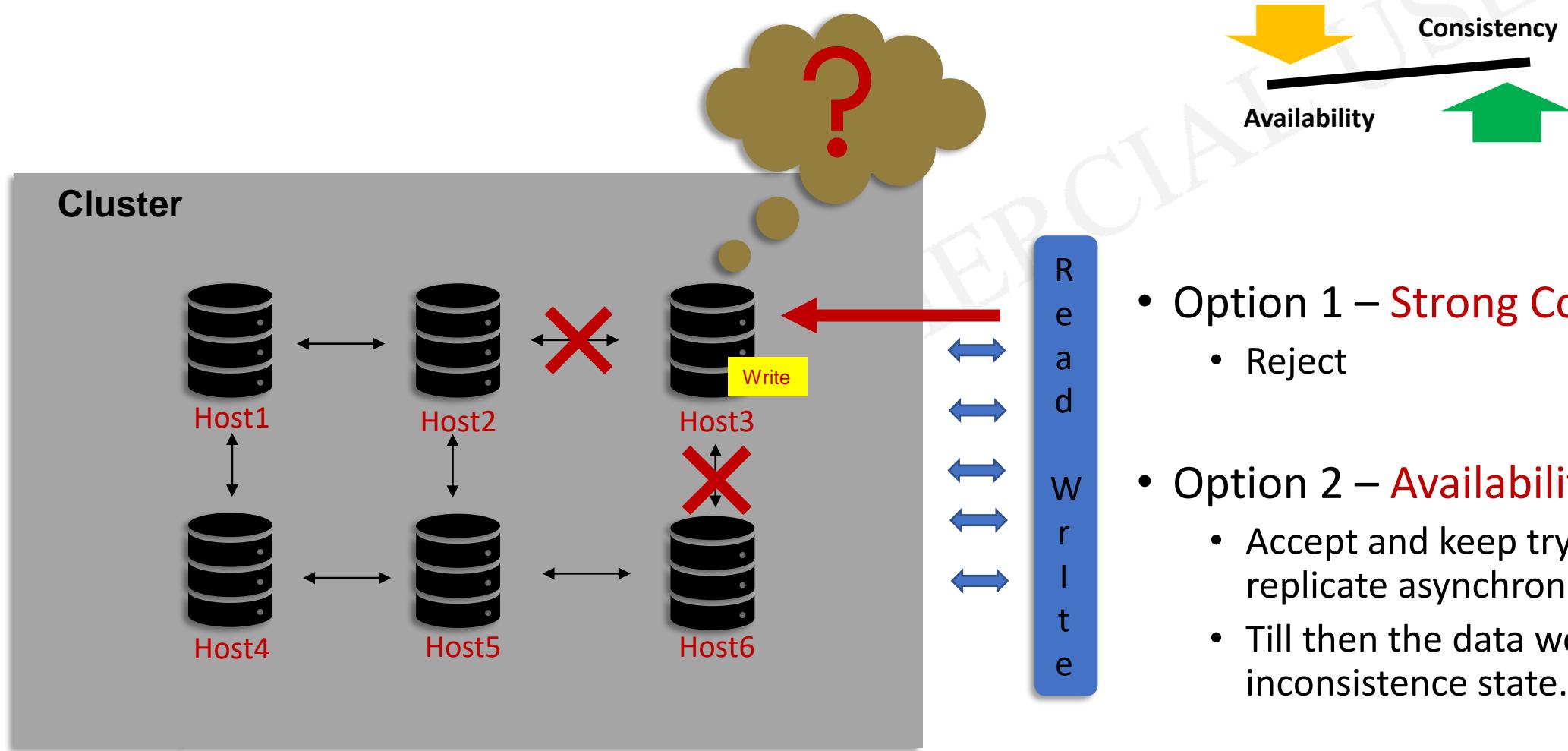
- **Structure:** Customer (ID, Name, Gender)
- **Schema:** ID - bigint,
Name - varchar(250),
Gender - char(1)

Distributed Database Systems

Distributed Databases



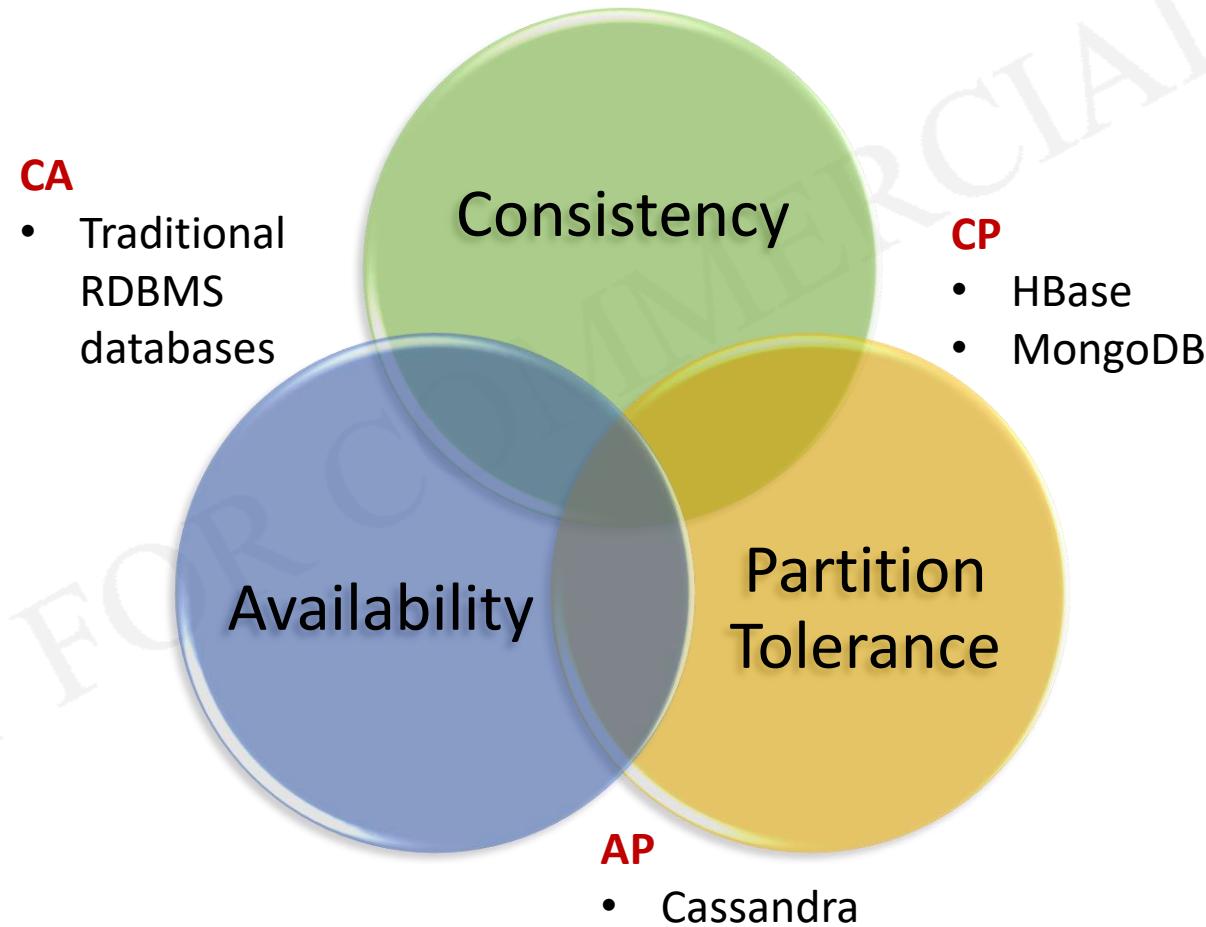
Distributed Databases – Partition Tolerance



Partition Tolerance is fault tolerance of network partition of data across nodes/host in a cluster.
LNC.ADONI@GMAIL.COM

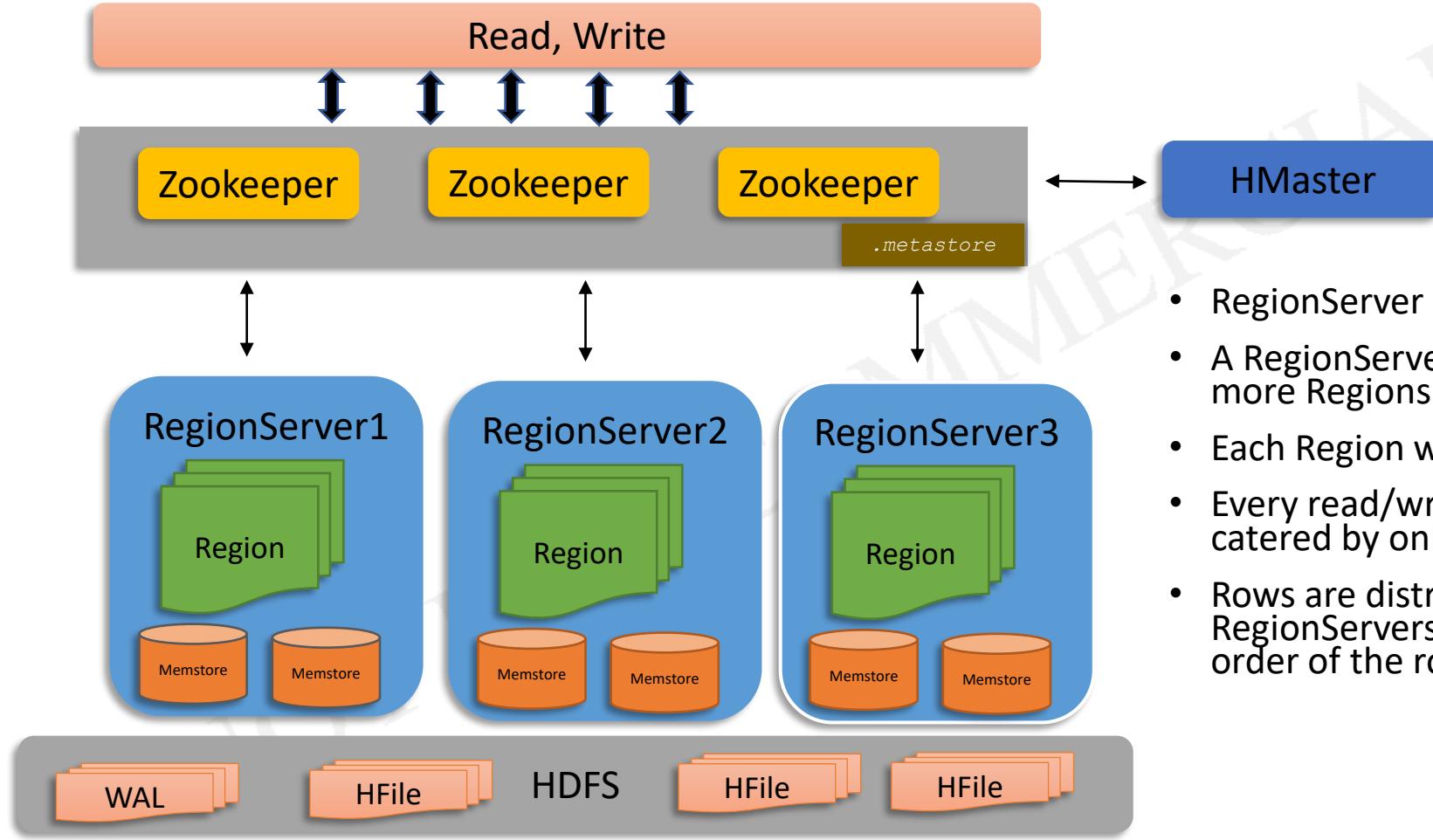
CAP Theorem

- Initially proposed by Dr. Eric Brewer in 1998.



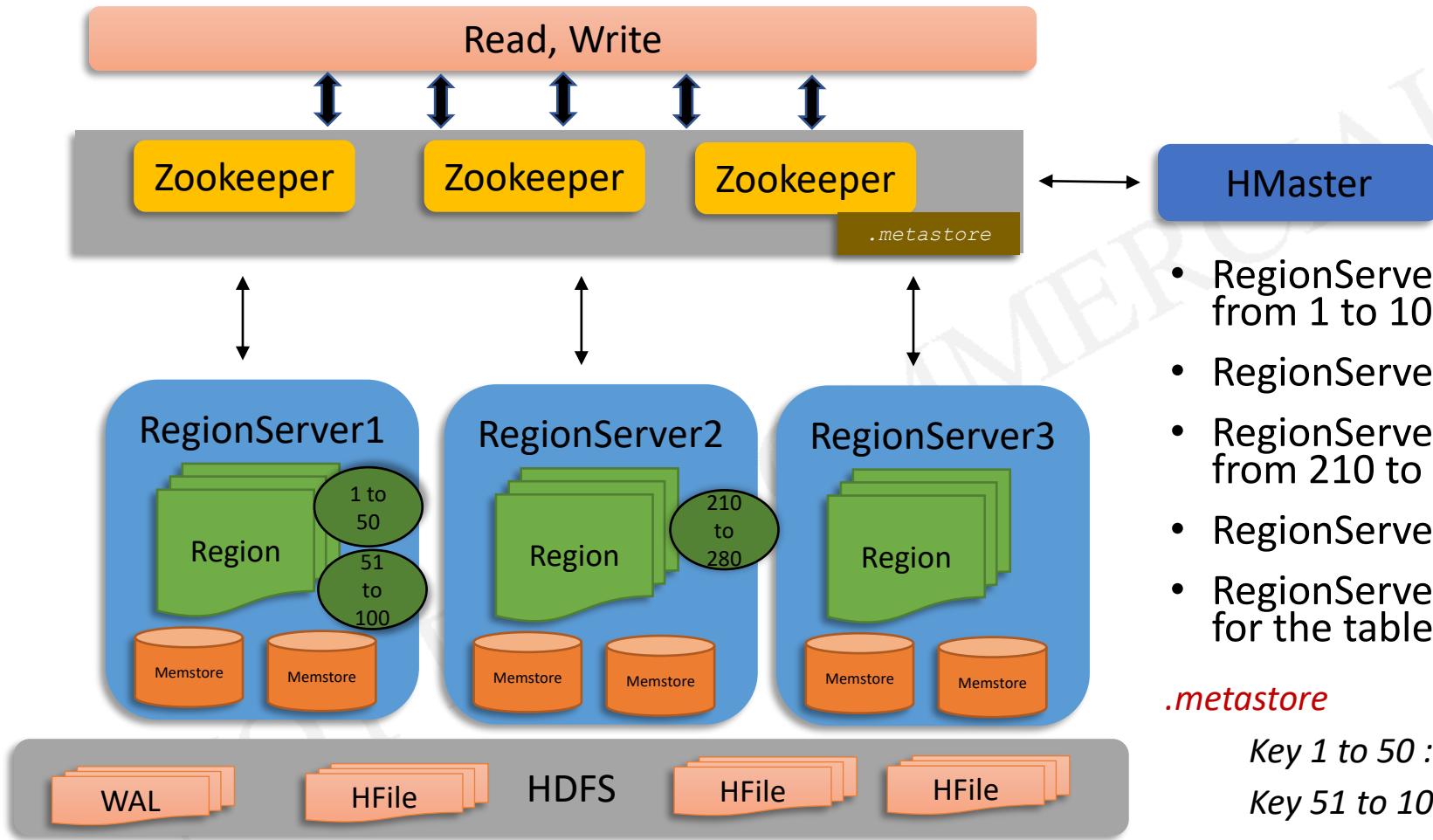


HBase Architecture



- RegionServer is a physical machine.
- A RegionServer is responsible for 1 or more Regions.
- Each Region will have 1 or more Rows.
- Every read/write for a specific row is catered by only one RegionServer.
- Rows are distributed across RegionServers' based on lexicographical order of the rowkey.

HBase Architecture – Region Assignment



- RegionServer1 is responsible for IDs from 1 to 100.
- RegionServer1 has 2 regions.
- RegionServer2 is responsible for IDs from 210 to 280.
- RegionServer2 has only 1 region.
- RegionServer3 is not hosting any data for the table.

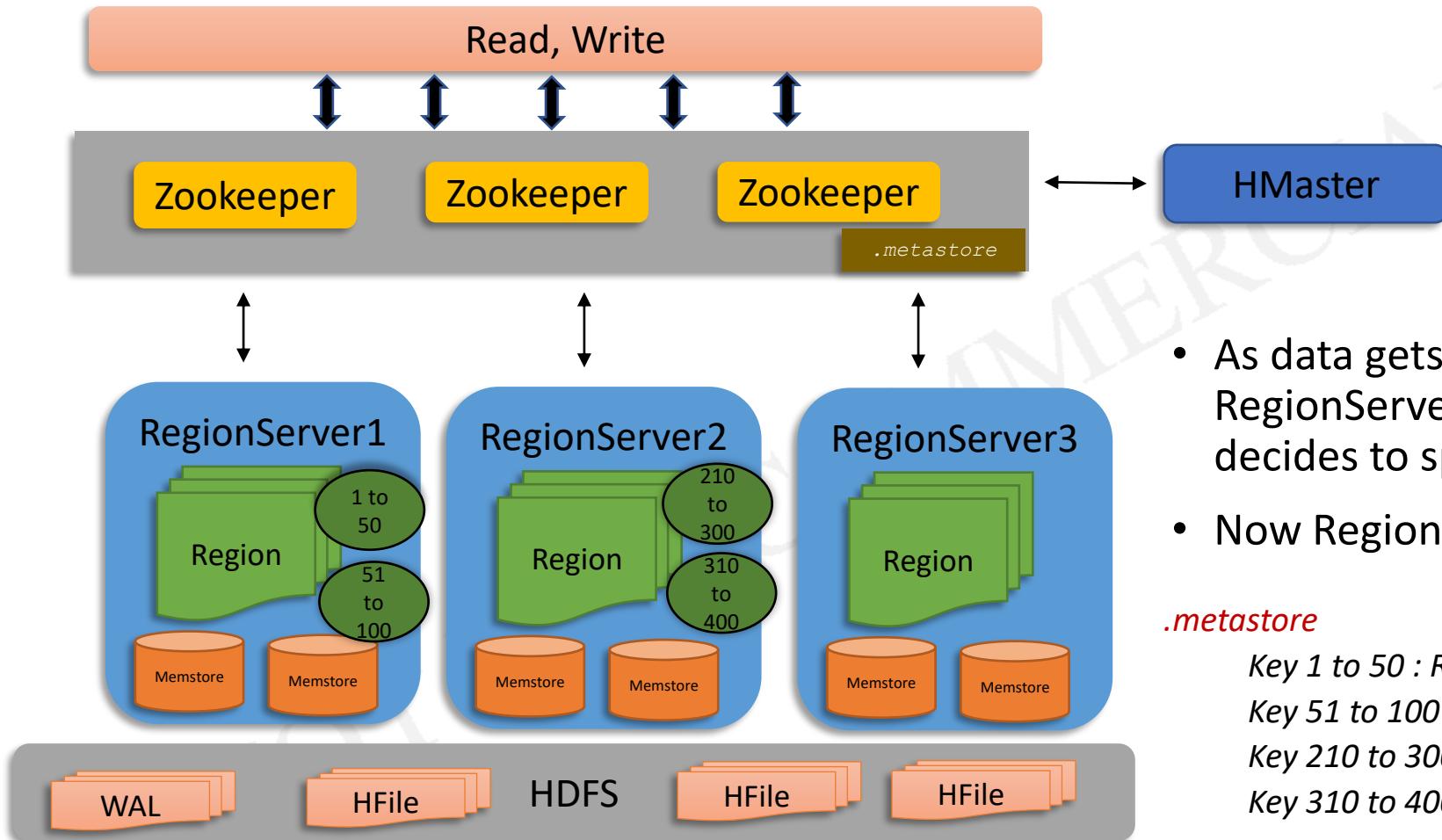
.metastore

Key 1 to 50 : RegionServer1, Region1

Key 51 to 100 : RegionServer1, Region2

Key 210 to 280: RegionServer2, Region1

HBase Architecture – Region Split

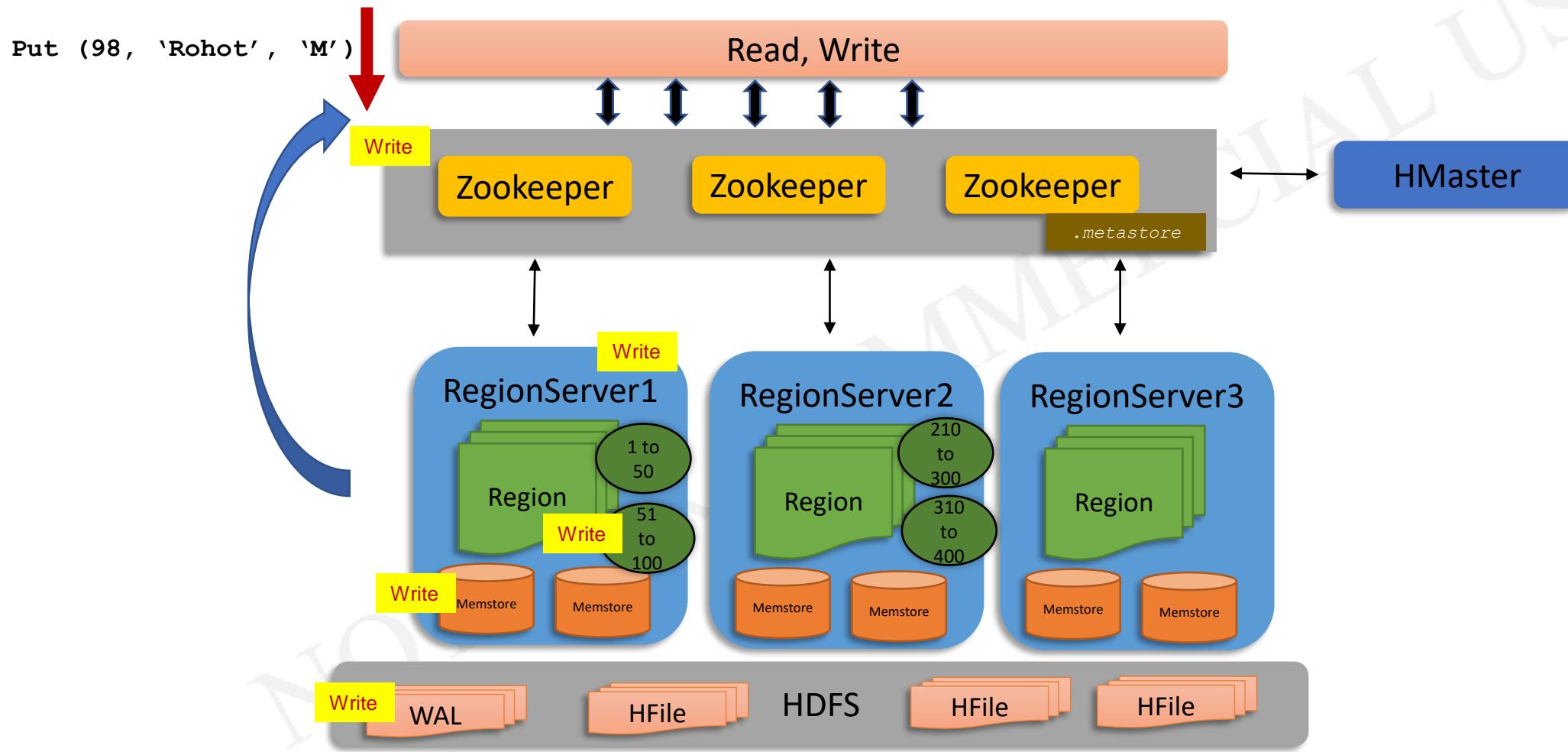


- As data gets ingested into RegionServer2, RegionServer2 decides to split the region.
- Now RegionServer2 has 2 regions.

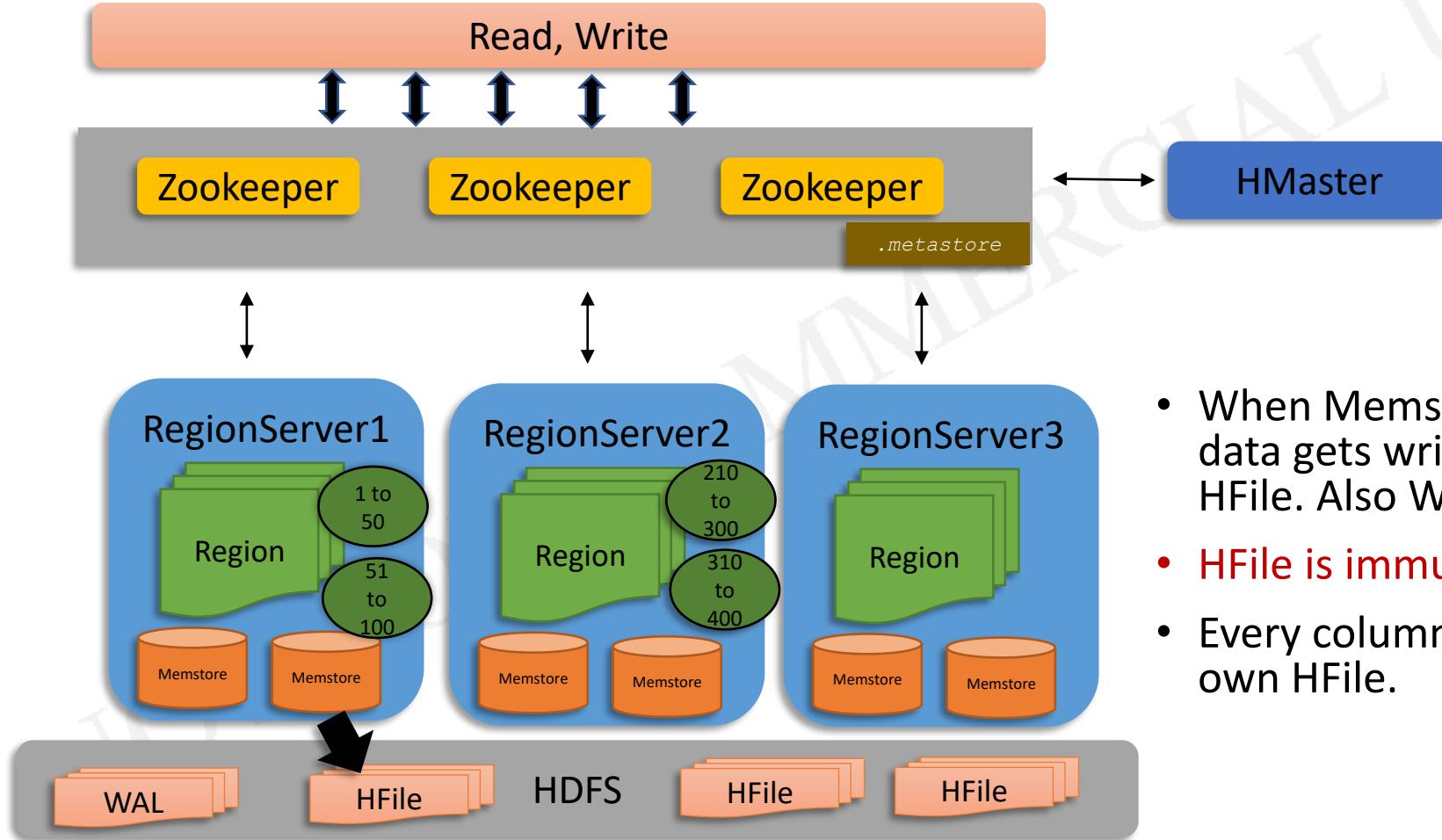
.metastore

Key 1 to 50 : RegionServer1, Region1
Key 51 to 100 : RegionServer1, Region2
Key 210 to 300: RegionServer2, Region1
Key 310 to 400: RegionServer2, Region2

HBase Architecture - Write

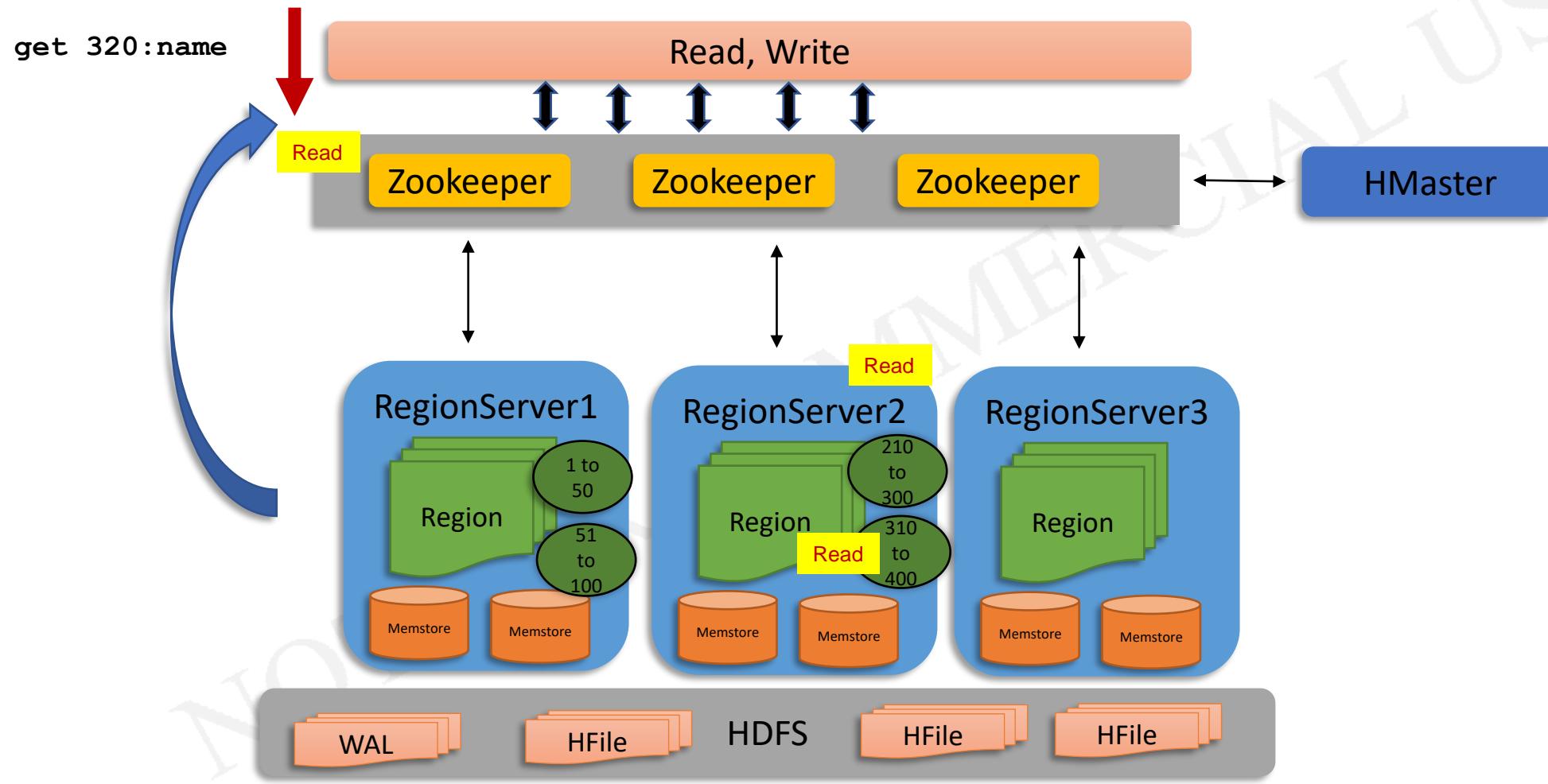


HBase Architecture – Flush

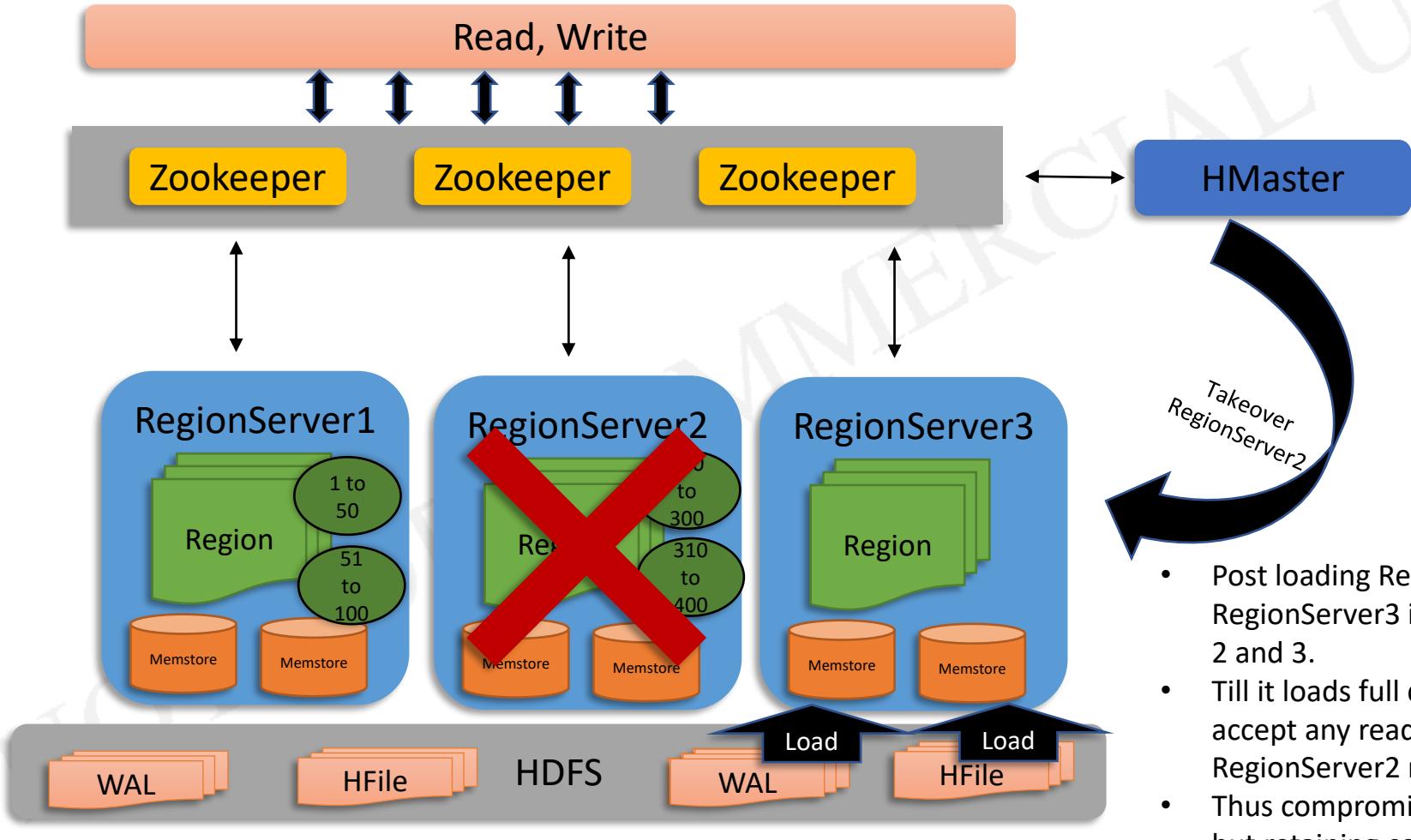


- When Memstore gets full, the data gets written to a new HFile. Also WAL gets purged.
- **HFile is immutable.**
- Every column family has its own HFile.

HBase Architecture - Read



HBase Architecture – Partition Tolerance



HBase Data Storage – Example 1

CID	Name	Gender	Street	City
1	Tom	M	Address1	Chicago
2	Ashley	F	Address2	New York
3	Jim	M	Address3	Los Angeles

RowKey: CID

Column Families: details, address

Key	Value
1:details:Name	Tom
1:details:Gender	M
1:address:Street	Address1
1:address:City	Chicago
2:details:Name	Ashley
2:details:Gender	F
2:address:Street	Address2
2:address:City	New York
3:details:Name	Jim
3:details:Gender	M
3:address:Street	Address3
3:address:City	Los Angeles

- create table 'customer', 'details', 'address'
- 'rowkey' column is an inbuilt column created by HBase, can't be created manually.
- No columns are created as part of DDL.

Key	Value
1:details:Name	Tom
1:details:Gender	M
2:details:Name	Ashley
2:details:Gender	F
3:details:Name	Jim
3:details:Gender	M

Put (1:details:age, 25)

Key	Value
1:details:Age	25

Put (3:address:City, 'Ohio')

Key	Value
3:address:cCity	Ohio

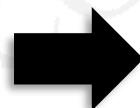
HBase Data Storage – Example 2

CID	Name	Gender	Street	City
1	Tom	M	Address1	Chicago
2	Ashley	F	Address2	New York
3	Jim	M	Address3	Los Angeles

RowKey: CID,Name

Column Families: details, address

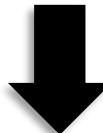
Key	Value
1Tom:details:Gender	M
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:details:Gender	F
2Ashley:address:Street	Address2
2Ashely:address:City	New York
3Jim:details:Gender	M
3Jim:address:Street	Address3
3Jim:address:City	Los Angeles



Key	Value
1Tom:details:Gender	M
2Ashley:details:Gender	F
3Jim:details:Gender	M

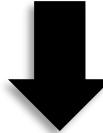
Key	Value
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:address:Street	Address2
2Ashely:address:City	New York
3Jim:address:Street	Address3
3Jim:address:City	Los Angeles

Put (1Tom:details:age, 25)



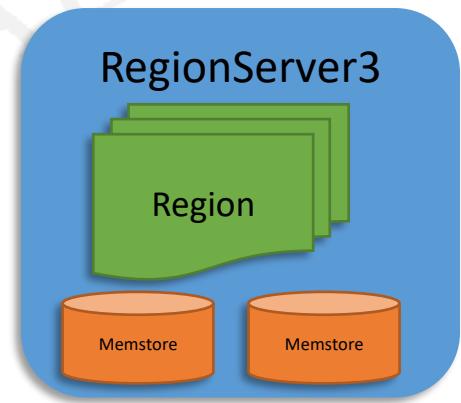
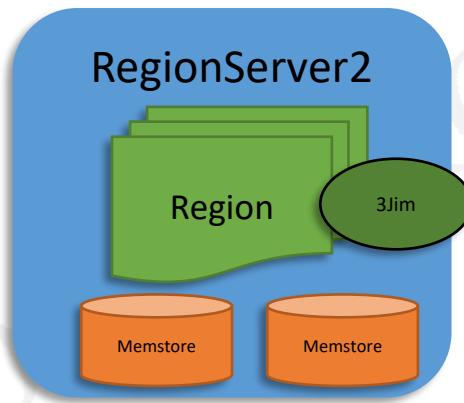
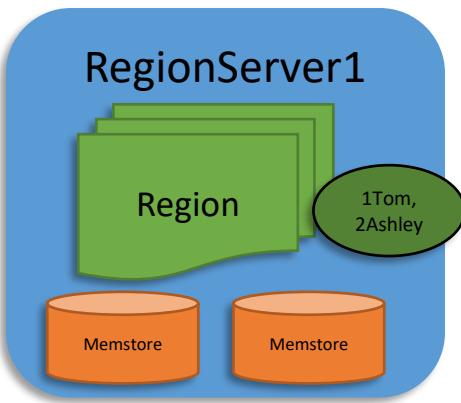
Key	Value
1Tom:details:Age	25

Put (3Jim:address:City, 'Ohio')



Key	Value
3Jim:address:cCity	Ohio

HBase Data Storage – Physical



Key	Value
1Tom:details:Gender	M
2Ashley:details:Gender	F

Key	Value
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:address:Street	Address2
2Ashely:address:City	New York

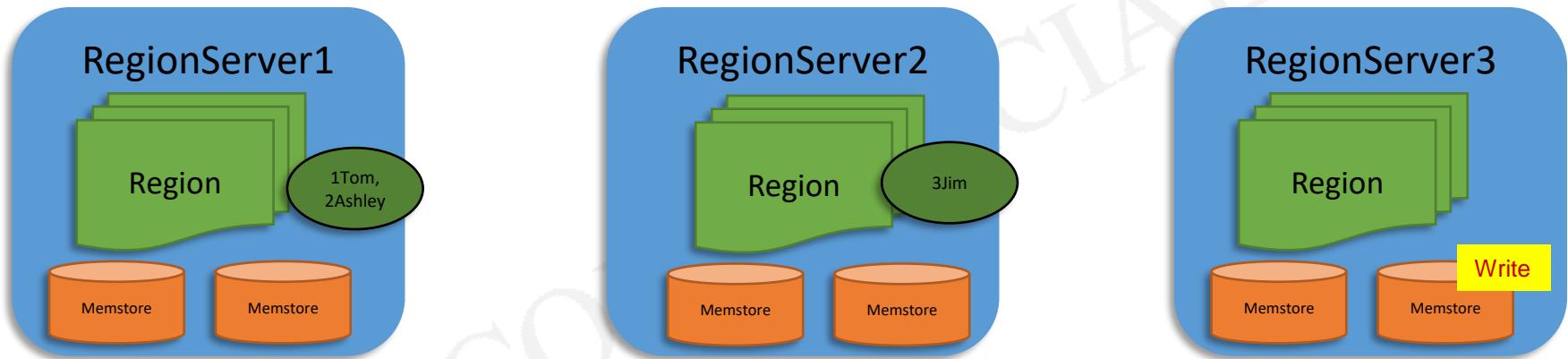
Key	Value
3Jim:details:Gender	M

Key	Value
3Jim:address:City	Ohio

HDFS

HBase Data Storage – Put - 1

Put '4Rekha' 'details:Gender' 'F'
Put '2Ashley' 'details:Age' '25'
Put '1Tom' 'details:email' 'tom@email.com'



Key	Value
1Tom:details:Gender	M
2Ashley:details:Gender	F

Key	Value
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:address:Street	Address2
2Ashely:address:City	New York

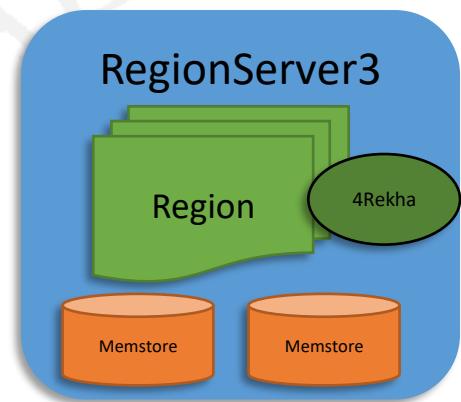
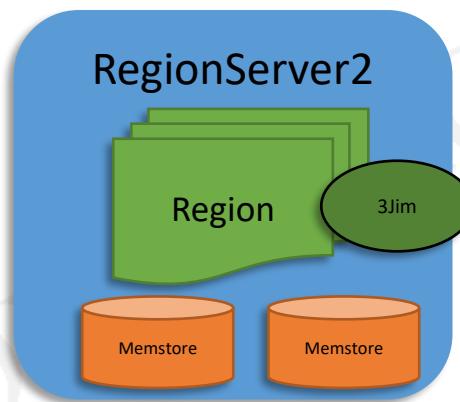
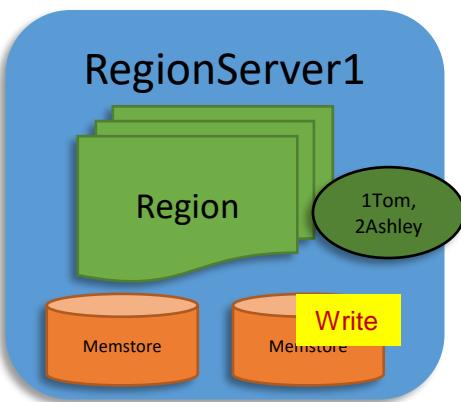
Key	Value
3Jim:details:Gender	M

Key	Value
3Jim:address:cCity	Ohio

HDFS

HBase Data Storage – Put - 2

Put '4Rekha' 'details:Gender' 'F'
Put '2Ashley' 'details:Age' '25'
Put '1Tom' 'details:email' 'tom@email.com'



Key	Value
1Tom:details:Gender	M
2Ashley:details:Gender	F

Key	Value
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:address:Street	Address2
2Ashely:address:City	New York

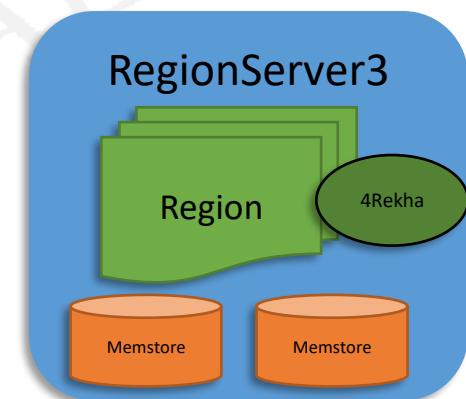
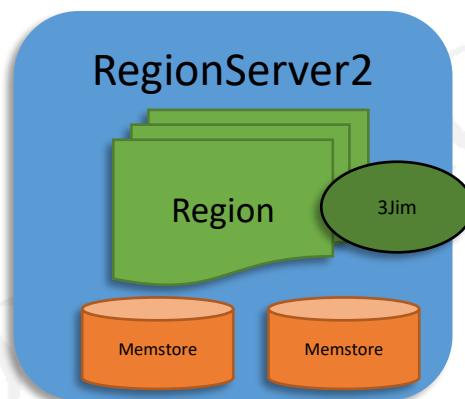
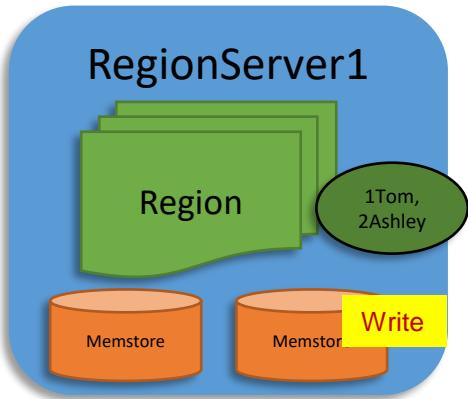
Key	Value
3Jim:details:Gender	M

Key	Value
3Jim:address:cCity	Ohio

HDFS

HBase Data Storage – Put - 3

Put '4Rekha' 'details:Gender' 'F'
Put '2Ashley' 'details:Age' '25'
Put '1Tom' 'details:email' 'tom@email.com'



Key	Value
1Tom:details:Gender	M
2Ashley:details:Gender	F
Key	Value
1Tom:details:Age	25

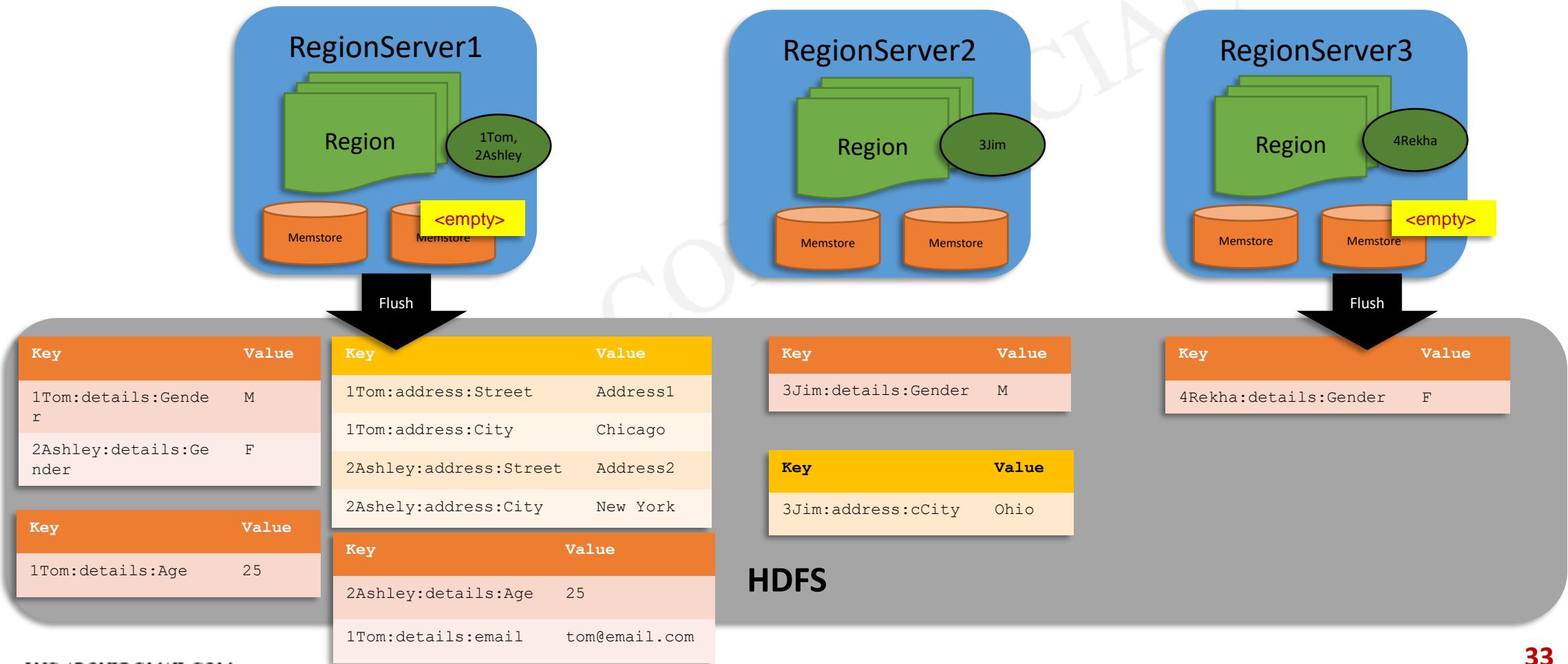
Key	Value
1Tom:address:Street	Address1
1Tom:address:City	Chicago
2Ashley:address:Street	Address2
2Ashely:address:City	New York

Key	Value
3Jim:details:Gender	M
Key	Value
3Jim:address:cCity	Ohio

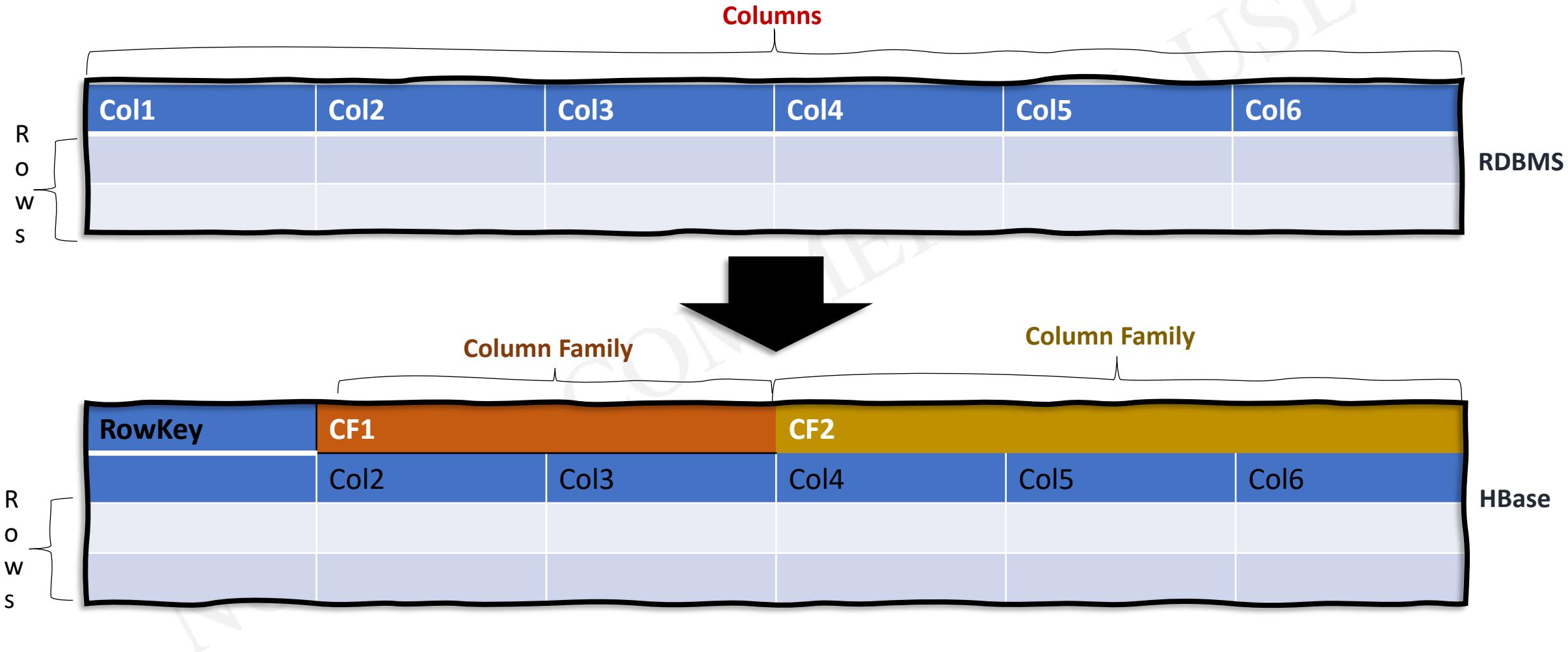
HDFS

HBase Data Storage – Put - Flush

```
Put '4Rekha' 'details:Gender' 'F'  
Put '2Ashley' 'details:Age' '25'  
Put '1Tom' 'details:email' 'tom@email.com'
```



HBase Data Model vs RDBMS



HBase Data Model - Challenge

Customer details – login profile

- | | |
|--|--|
| <ul style="list-style-type: none">• First name• Last Name• DOB• Gender• Address line1• Address line2• Email• Phone number | <ul style="list-style-type: none">• City• State• Zipcode• Send me exciting offers – Yes/No• Use my details for profiling – Yes/No• Can we call you for communication – Yes/No |
|--|--|

What should be the data model?

- Rowkey
- Column Families
- Columns

• What are the assumptions?

• What questions would you ask?

HBase Data Model – Challenge – Approach

Customer details – login profile

- First name
- Last Name
- DOB
- Gender
- Address line1
- Address line2
- Email
- Phone number
- City
- State
- Zipcode
- send me exciting offers – Yes/No
- Use my details for profiling – Yes/No
- Can we call you for communication – Yes/No

```
create table 'customer', 'basic', 'details',  
'address', 'settings'
```

Customer

Email <rowkey>

First Name + Last Name <basic>

Phone_number <basic>

DOB <details>

Gender <details>

Address_line1 <details>

Address_line2 <details>

City <details>

State <details>

Zipcode <details>

Value1 <settings>

Value2 <settings>

Value3 <settings>

HBase Data Model – Challenge

We want to enable login using phone number

HBase Data Model – Challenge – Approach

Customer details – login profile

- First name
- Last Name
- DOB
- Gender
- Address line1
- Address line2
- Email
- Phone number
- City
- State
- Zipcode
- send me exciting offers – Yes/No
- Use my details for profiling – Yes/No
- Can we call you to communication – Yes/No

```
create table 'customer',
'basic', 'details',
'address', 'settings'

customer
Email <rowkey>

First Name + Last Name <basic>
Phone_number <basic>

DOB <details>
Gender <details>
Address_line1 <details>
Address_line2 <details>
City <details>
State <details>
Zipcode <details>

Value1 <settings>
Value2 <settings>
Value3 <settings>
```

```
create table 'customer_mobile',
'map'
```

customer mobile
Phone_number <rowkey>
email <map>

Or

```
create table 'customer_mobile'
customer mobile
Phone_number#email <rowkey>
```

HBase Data Model – Use case

- Show recommended movies list.
 - Recommendations may vary by customer.



RDBMS Data Model – Netflix Recommendation Use case

Customer

Cust_id
Cust_name
Cust_gender

All Movies

Movie_id
Movie_name
Release_date
Budget
Description
Rating
Duration

Cust Movie Recomm

Cust_id
Movie_id

```
Select movie_name  
from  
    <required tables>  
where  
    cust_id=<#> and <join conditions>
```

Can you show cast (actor, actress, director...) also, along with movie name?

RDBMS Data Model– Recommendation Use case

<u>Customer</u>	<u>All Movies</u>	<u>Cust Movie Recomm</u>	<u>All Cast</u>	<u>Movie Cast map</u>
Cust_id	Movie_id	Cust_id	Cast_id	Movie_id
Cust_name	Movie_name	Movie_id	Cast_name	Cast_id
Cust_gender	Release_date		Gender	
	Budget		DOB	
	Description			
	Rating			
	Duration			
	Genre			

```
Select movie_name, cast_name  
from  
    <required tables>  
where  
    cust_id=<#> and <join conditions>
```

Can you show awards information also?

Our data science team generated new
recommendations for customers.
Go update NOW!!!!

HBase Data Model – Recommendation Use case

Create table '**cust_recomm**',
'mv', 'others'

Rowkey	Value
Customer1:mv:1	1001
Customer1:mv:2	1002
Customer2:mv:1	1003
Customer3:mv:1	5001
Customer3:mv:2	5002
Customer3:mv:3	9001

Rowkey	Value
Customer1:mv:1	9001
Customer1:mv:2	1003
Customer1:mv:3	5002

Create table '**movies**',
'details', 'others'

Rowkey	Value
1001:details:name	Forest Gump
1002:details:name	Avataar
1003:details:name	The Angel
1004:details:name	GoodFellas
5001:details:name	GodFather 1
5002:details:name	GodFather 2
9001:details:name	GodFather 3
9001:details:c1	Al Pachino
1001:details:c1	Tom Hanks
1001:details:c1	Sally Fields
1001:details:oscaractor	Tom Hanks
1001:details:oscardirector	Robert Zemeckis

Can you show cast
also, along with movie
name?

Can you show awards
information also?

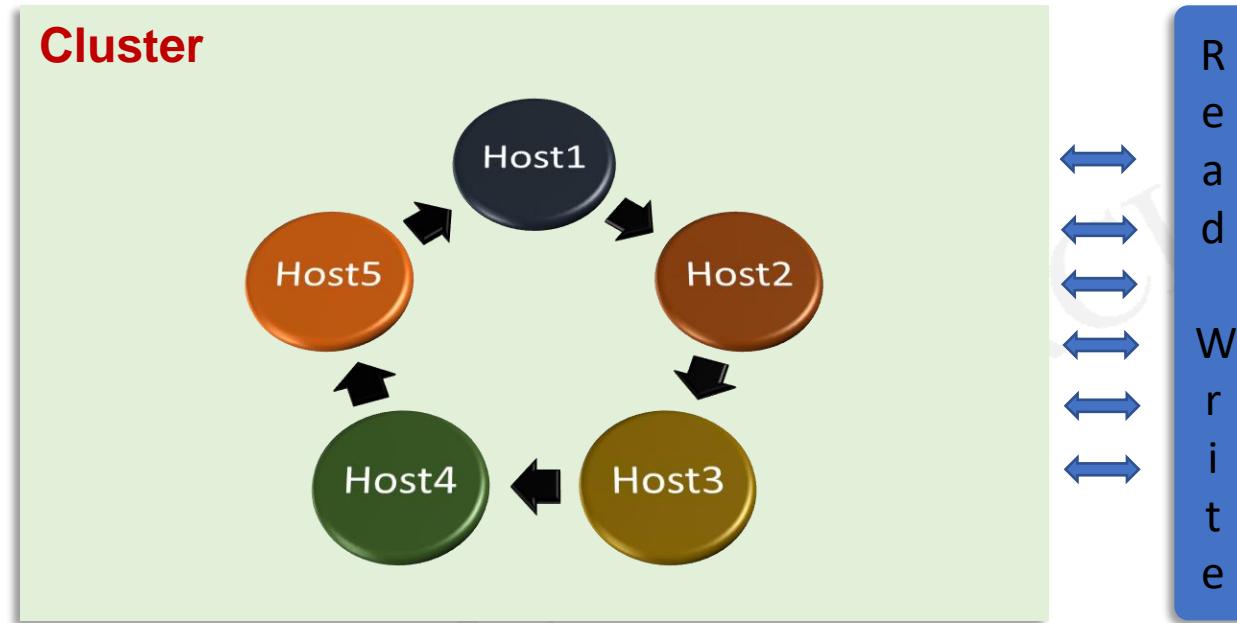
Our data science team
generated new
recommendations for
customers.
Go update NOW!!!!

Example:
Customer1- 9001, 1003,
5002



Cassandra

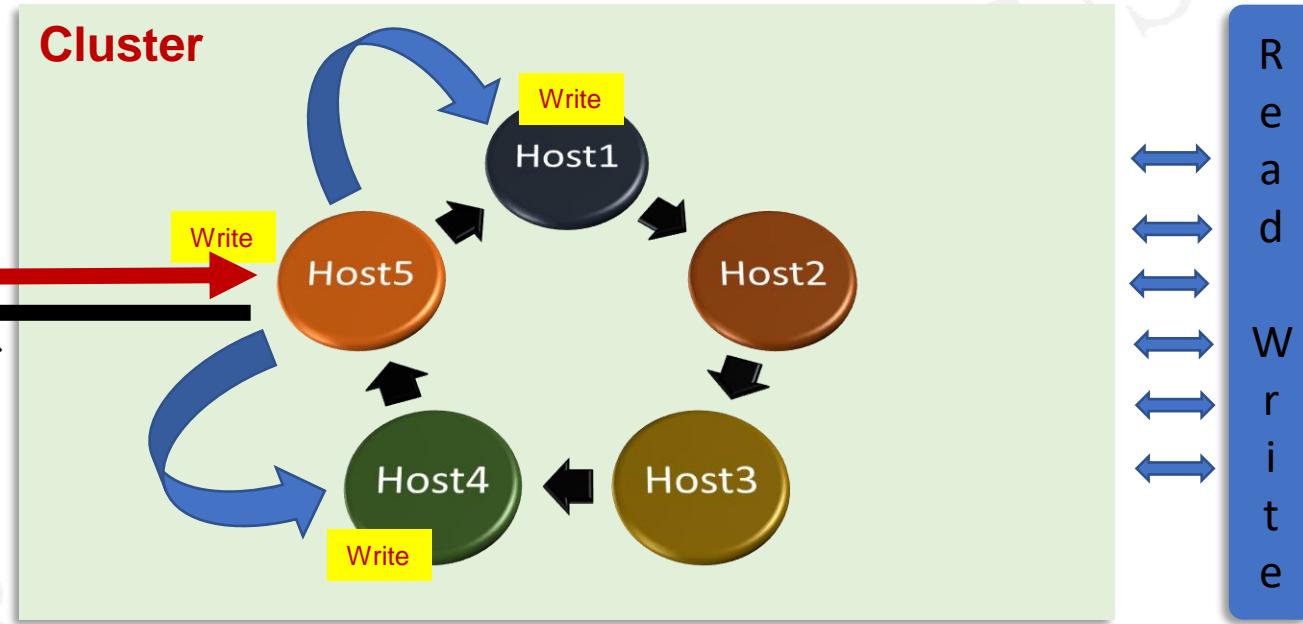
Cassandra Architecture



- Cassandra implements **ring** Architecture.
- Node (host) is a physical machine. Every node (host) acts as its own master. (i.e there is no master).
- The nodes (hosts) talk to each other using Gossip protocol.
- Clients can talk to any host to read/write data.
- Every node(host) is responsible for the honoring query for any segment of data.
- Every node is responsible for replicating data for the segment it owns.

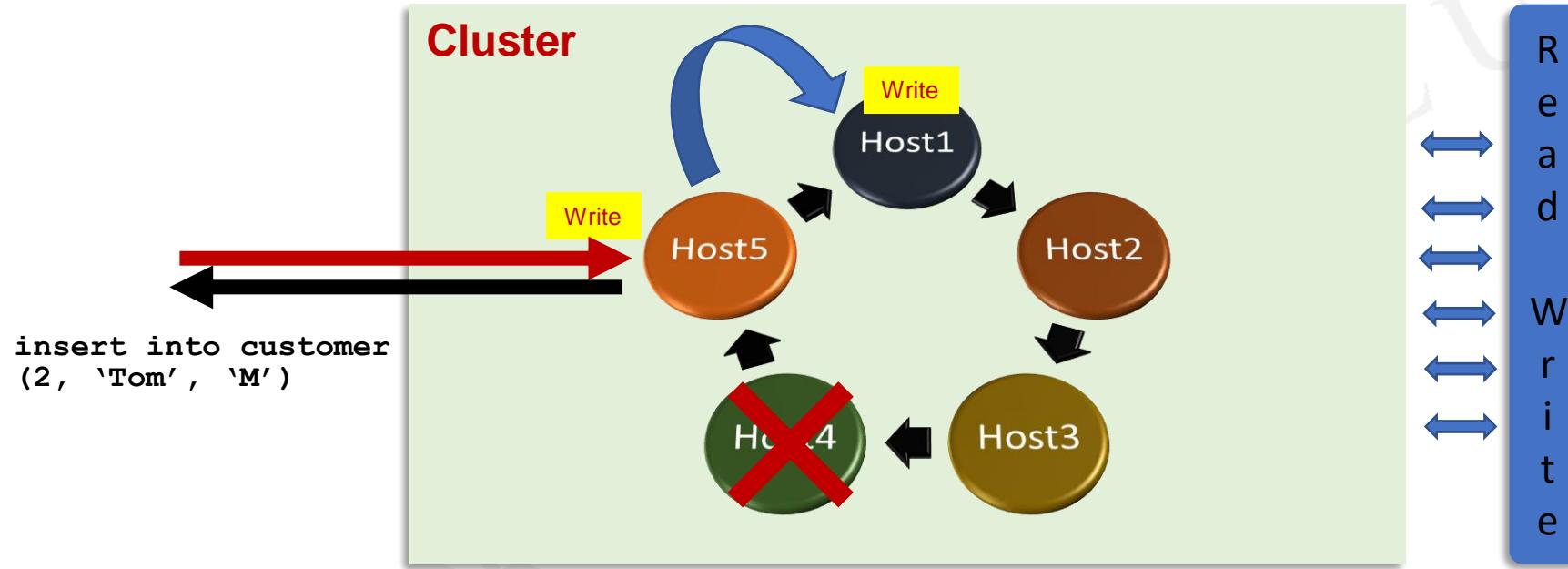
Cassandra Architecture - Write

```
create table customer(  
    cust_id varint PRIMARY KEY,  
    cust_name text,  
    gender text)  
  
insert into customer  
(1, 'Julie', 'F')
```



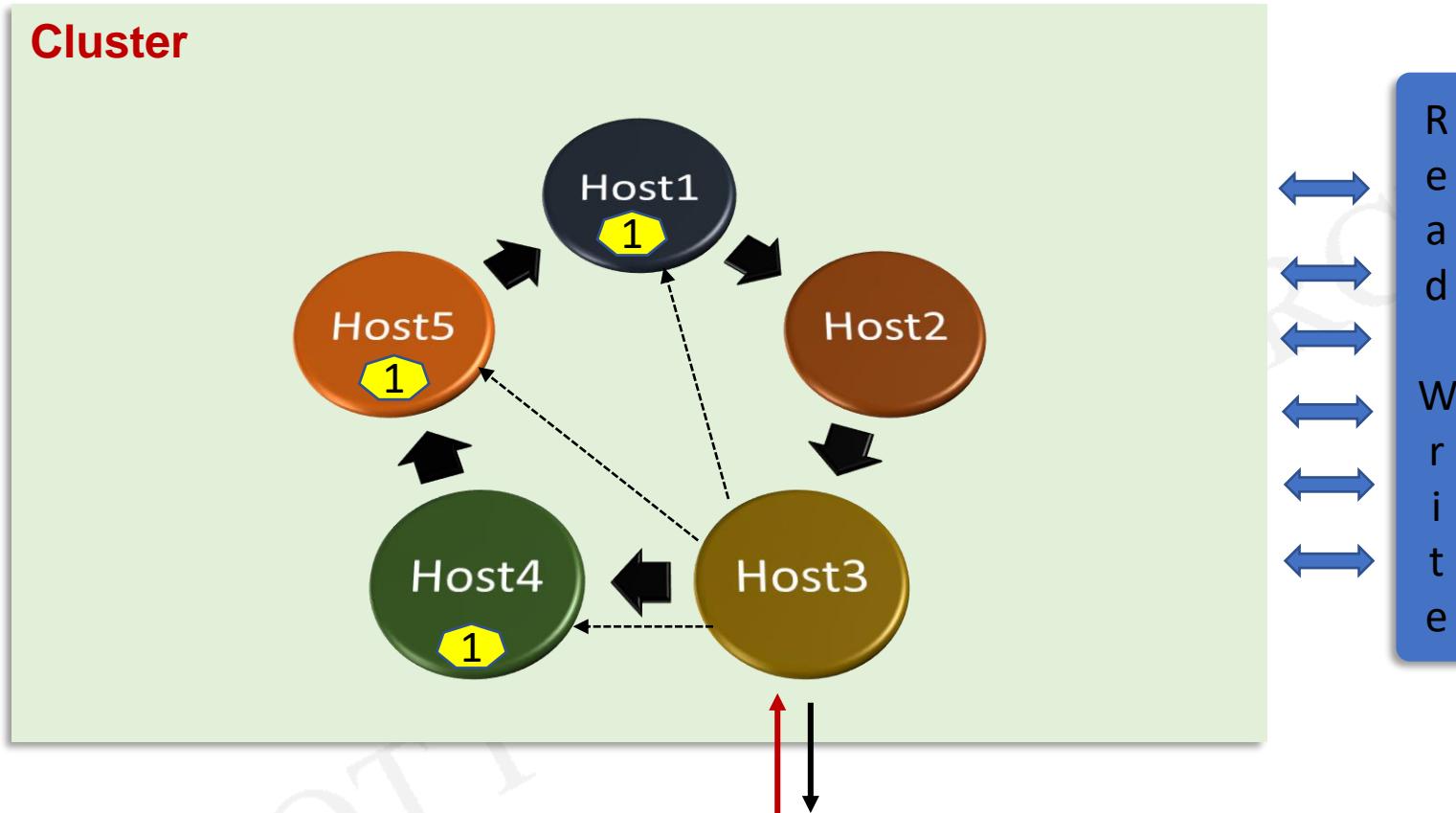
- Replication factor is configurable.
- Replication strategy (Simple, Old Network Topology...) can be defined.
- Based on the defined consistency, if the replication fails, the primary host continues with the operation. **This results in temporary inconsistency.**

Cassandra Architecture – Write CAP



- Based on the defined consistency, if the replication fails, the primary host continues with the operation.
- This results in temporary inconsistency.
- When the data is read for the specific record, the coordinator node notifies Host4 that the data is out of sync.
- If Host5, Host1 go out of the cluster and Host4 recovers, then Host4 will continue to return inconsistent data.

Cassandra Architecture - Read



- Host3 acts as coordinator for the client.
- Based on the primary key (partition key), Host3 contacts the respective hosts to get the required data.
- Host3, the coordinator, identifies the most recent data based on timestamp and returns the data to client.
- If the hosts are out of sync, Host3 communicates the replica hosts to update themselves.

Cassandra Data Storage

CID	Name	Gender	Street	City
1	Tom	M	Address1	Chicago
2	Ashley	F	Address2	New York
3	Jim	M	Address3	Los Angeles

Primarykey: CID

Primarykey	Value
1	{name:'Tom', Gender: 'M', Street: 'Address1', City: 'Chicago'}
2	{name:'Ashley', Gender: 'F', Street: 'Address2', City:'New York'}
3	{name:'Jim', Gender:'M', Street:'Address3', City:'Los Angeles'}

Cassandra – Collection Data Types

List

```
create table customer(  
    cust_id varint PRIMARY KEY,  
    cust_name text,  
    gender text,  
    phone_number list<int>)
```

```
insert into  
customer(cust_id,  
cust_name, gender,  
phone_number)  
values(1,'a','M',  
[12345,98657,21345])
```

- Duplicate values are allowed
- Returns results as inserted.

Set

```
create table customer(  
    cust_id varint PRIMARY KEY,  
    cust_name text,  
    gender text,  
    phone_number set<int>)
```

```
insert into  
customer(cust_id,  
cust_name, gender,  
phone_number)  
values(1,'a','M',  
{12345,98657,21345})
```

- Unique values
- Returns results in sorted order

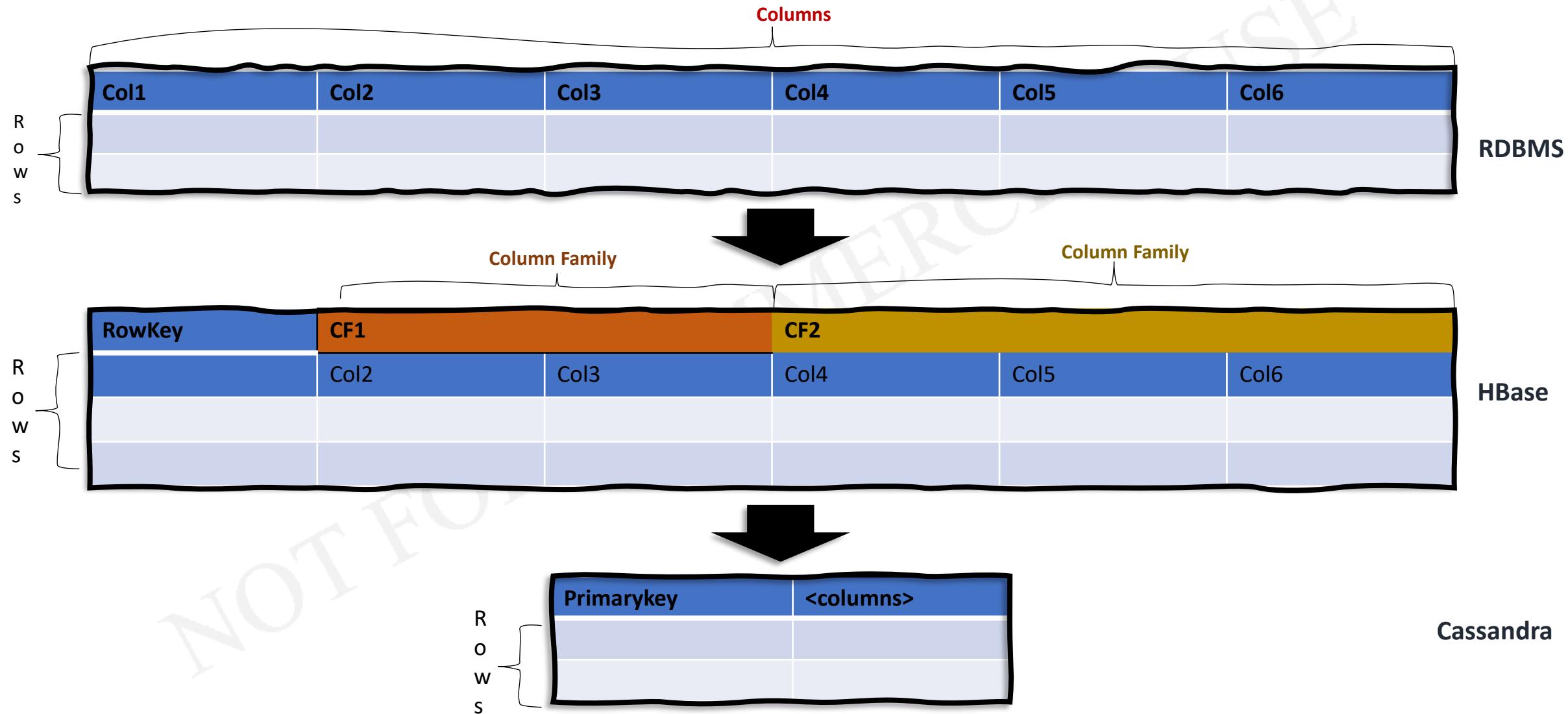
Map

```
create table customer(  
    cust_id varint PRIMARY KEY,  
    cust_name text,  
    gender text,  
    phone_number map<text, int>)
```

```
insert into  
customer(cust_id,  
cust_name, gender,  
phone_number)  
values(1,'a','M', { 'home' :  
12345,'office' :98657,'othe  
rs' :21345})
```

- Key-Value pair
- Secondary index can be created on either key or value, but not on both.

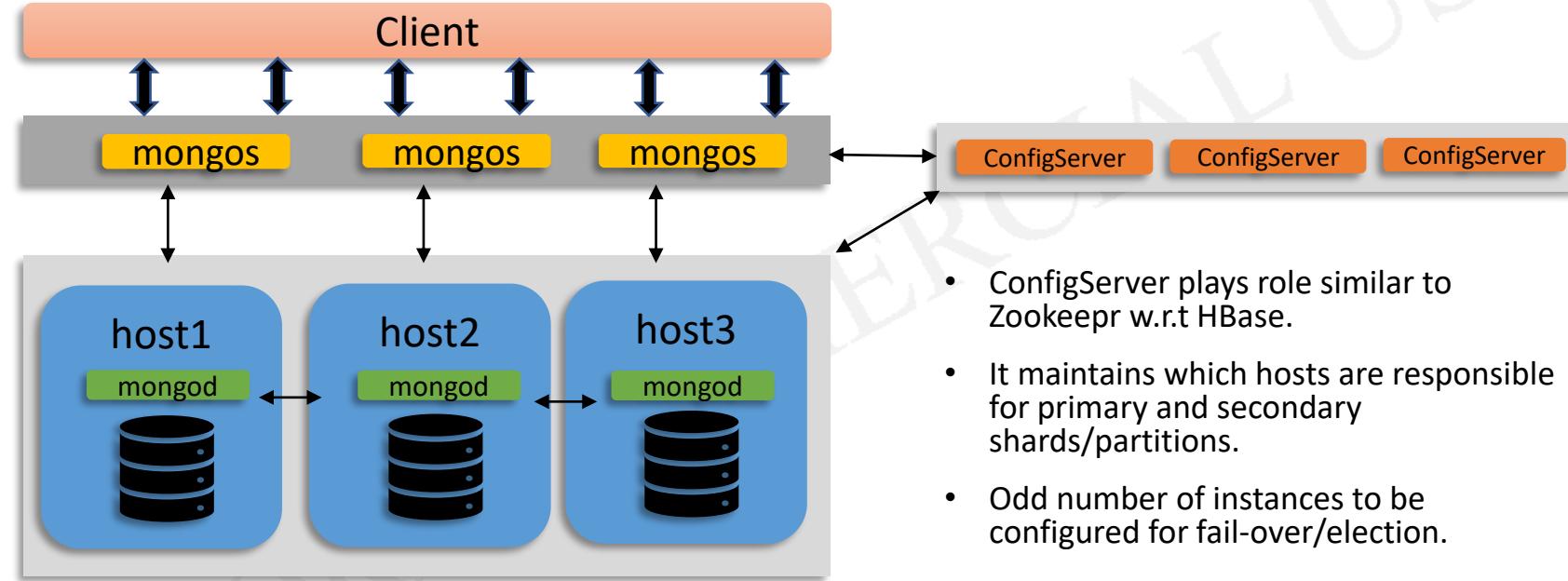
HBase Data Model vs Cassandra vs RDBMS





mongoDB

mongoDB Architecture

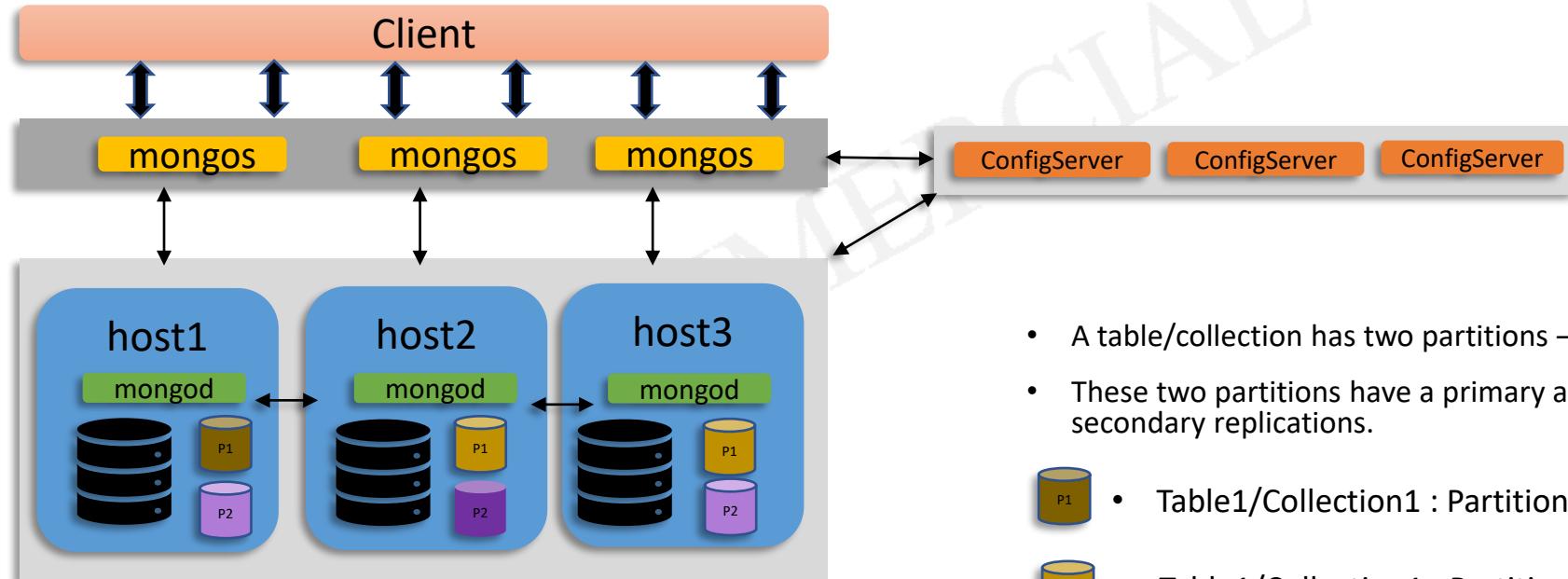


- Host machines which run ***mongod component, responsible for data.***
- All the hosts talk to each other and maintain heartbeat.
- One of the hosts acts as primary and all write operations are directed to only this primary.
- Primary takes responsibility to replicate the data to secondary hosts.

- ConfigServer plays role similar to Zookeeper w.r.t HBase.
- It maintains which hosts are responsible for primary and secondary shards/partitions.
- Odd number of instances to be configured for fail-over/election.

mongoDB Architecture – Replication, Shards

- Partition/Shards are created on an index.
- Multiple indexes can be created, but only one index can be used for sharding/partition.
- Primary partition maintains “oplog”, for write operations. And “oplog” is transmitted to secondary shards.



- All writes to Table1/Collection1 – Partition 1 are directed to host1
- All writes to Table1/Collection1 – Partition 2 are directed to host2

- A table/collection has two partitions – P1, P2.
 - These two partitions have a primary and multiple secondary replications.
- Table1/Collection1 : Partition 1 Primary
 - Table1/Collection1 : Partition 1 Secondary
 - Table1/Collection1 : Partition 2 Primary
 - Table1/Collection1 : Partition 2 Secondary

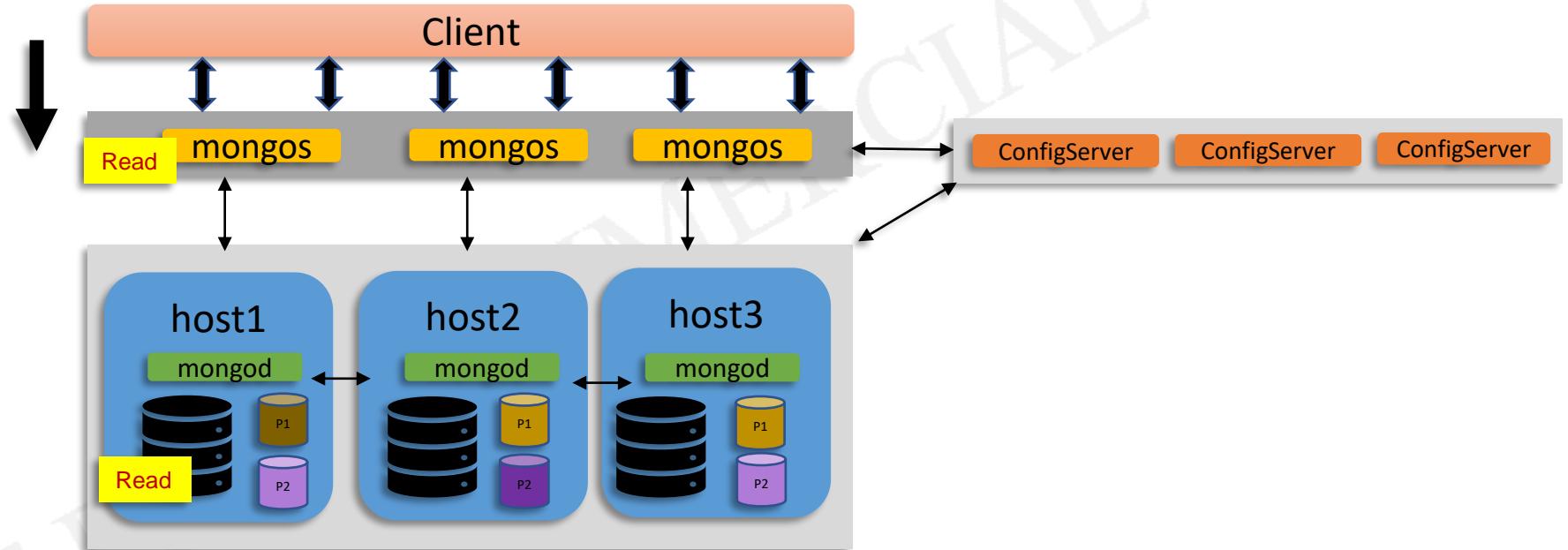
mongoDB Data Model - Example

```
{  
    _id: ObjectId('7df78ad8902c'),  
    cust_id: 1001,  
    cust_name: 'Tim',  
    last_updated: '2019-01-01',  
    count: 2,  
    movies: [  
        {  
            movie_id: '1',  
            name: 'Forest Gump',  
            cast: ['Tom Hanks'],  
            collections: 50000000,  
            recognitions: 'Oscar'  
        },  
        {  
            movie_id: '2',  
            name: 'As Good as It Gets',  
            cast: ['Helen Hunt', 'Jack Nichaleson'],  
            budget: 1000000  
        }  
    ]  
}
```

- **_id is created by mongoDB, this acts as unique identifier.**
- **cust_id can be used for indexing and partition/shards.**
- **Any number of secondary indexes can be created. Ex: movie name.**
- **The data representation is like JSON.**
- **The fields across documents (rows) or within nested fields can be dynamic/unstructured.**

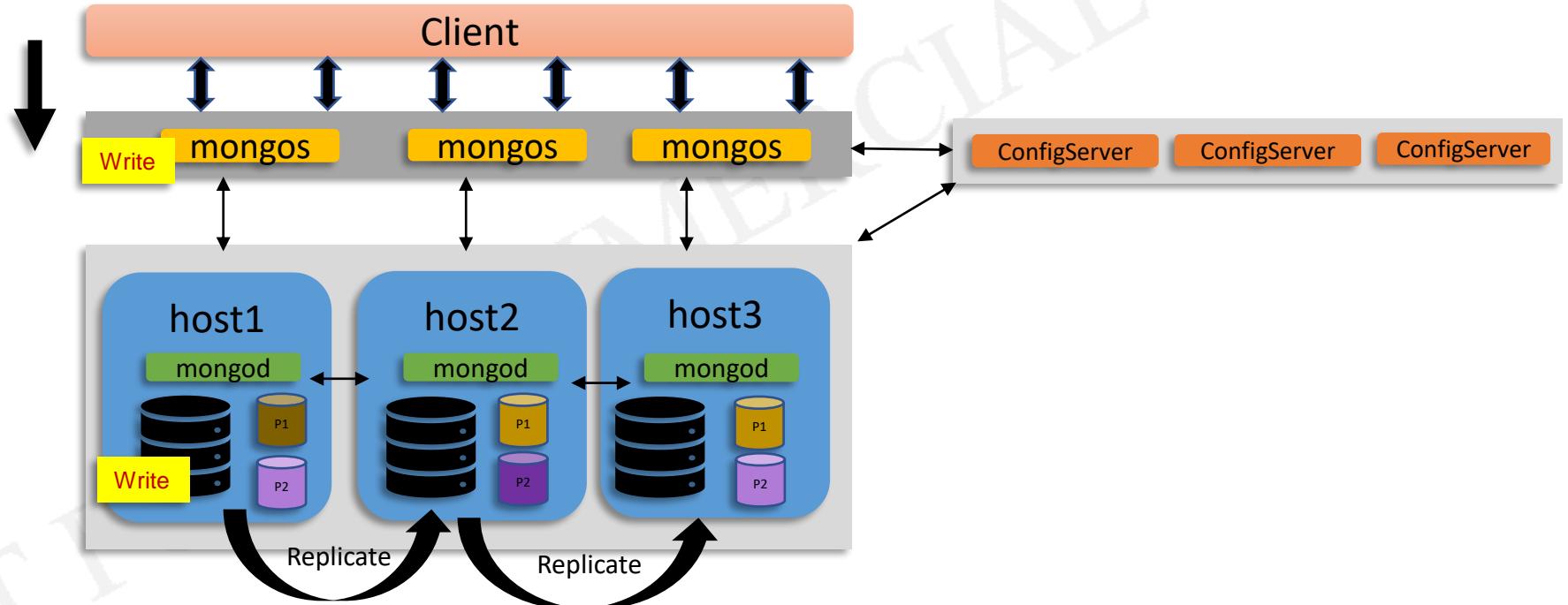
mongoDB Architecture – Read

- Read preference can be set to read always from Primary or secondary.

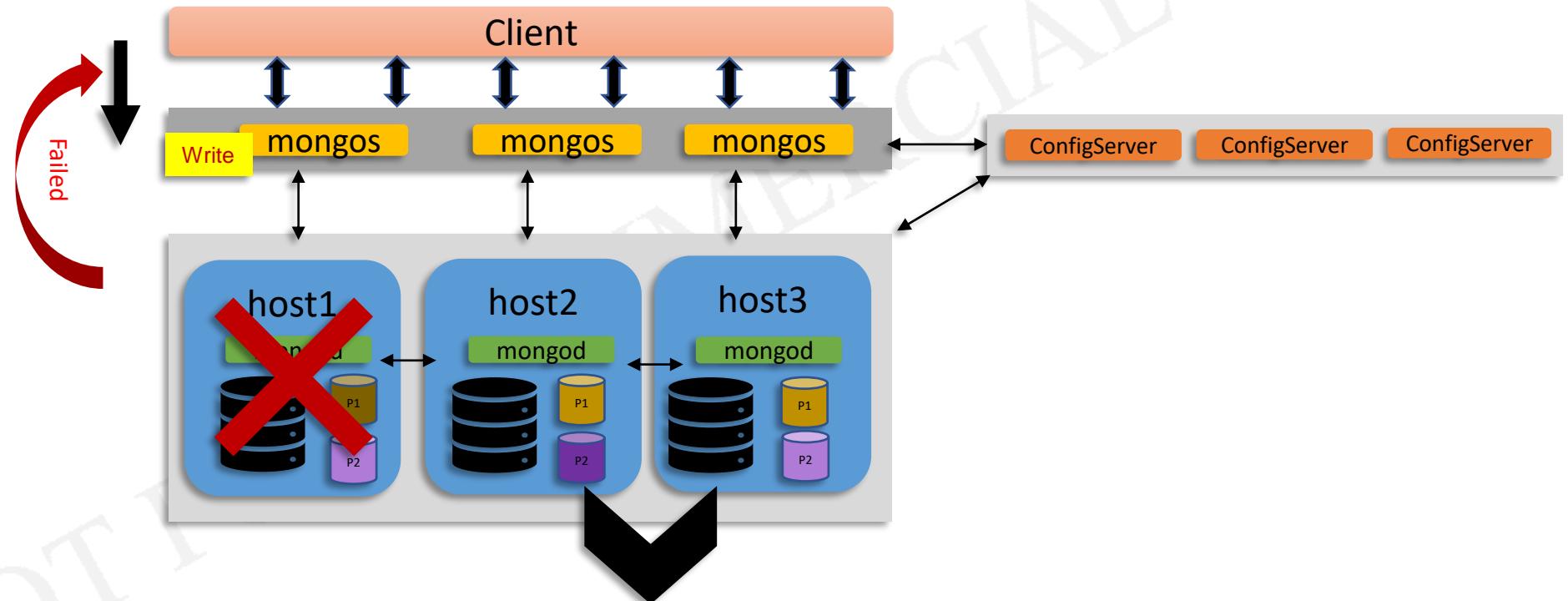


mongoDB Architecture – Write

- Write operation is always done on Primary.
- Shard information for Primary is obtained from ConfigServer and propagated to the corresponding Primary.
- Based on replication factor, Primary triggers replication to Secondary.

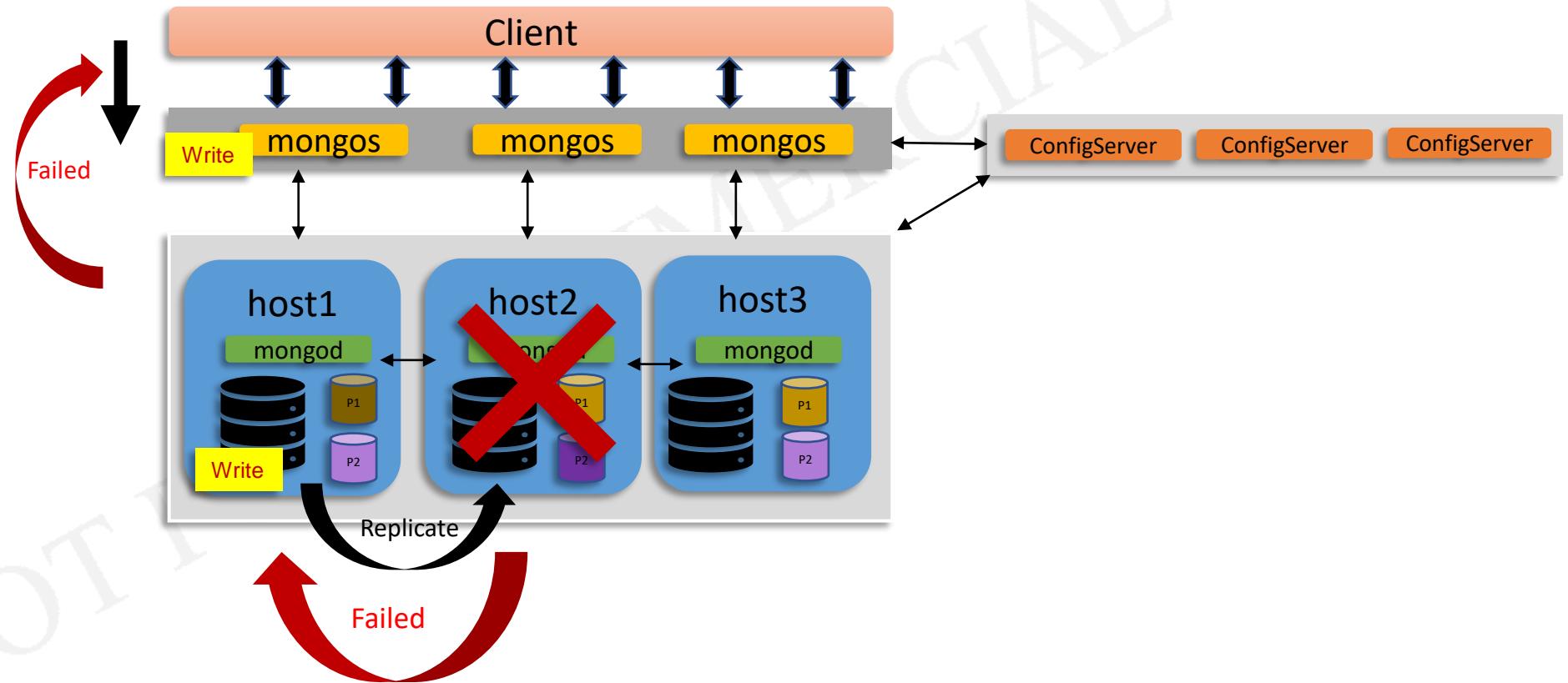


mongoDB Architecture – Write CAP



- Polling happens among all secondary shards and a new Primary is elected.
- Till then no write operations are allowed.
- Thus guaranteeing consistency but compromising on availability

mongoDB Architecture – Write CAP



SQL, NoSQL/Distributed Databases

	SQL/RDBMS	HBase	Cassandra	MongoDB
Purpose	<ul style="list-style-type: none"> OLTP, OLAP 	<ul style="list-style-type: none"> OLTP 	<ul style="list-style-type: none"> OLTP 	<ul style="list-style-type: none"> OLTP
Schema Design	<ul style="list-style-type: none"> OLTP – Normalized OLAP – Star, Snowflake Schema Row/Column based storage Structured with Schema 	<ul style="list-style-type: none"> Column Family based Key-Value based storage Dynamic column creation at write Semi-Structured, Schema less 	<ul style="list-style-type: none"> Column Family based Key-Value based storage Semi-Structured, defined Schema 	<ul style="list-style-type: none"> Document based Key-Value based storage. Value being simple/nested json. Semi-Structured, Schema less
Terminology	<ul style="list-style-type: none"> Database, Table, Row, Column 	<ul style="list-style-type: none"> Database, Table, Row, Column Family : Column 	<ul style="list-style-type: none"> Keyspace, Table/ColumnFamily, Row, Column 	<ul style="list-style-type: none"> Database, Collections, Document, Field
Access	<ul style="list-style-type: none"> OLTP – Write heavy OLAP – Read heavy SQL Support All types of joins support Any number of secondary indexing. Inbuilt support for integrity constraints. 	<ul style="list-style-type: none"> Write & Read heavy Access through row-key Inbuilt versioning Replication is based on HDFS. APIs No support for SQL Does not support joins Does not support integrity constraints across tables. Not support for secondary indexes 	<ul style="list-style-type: none"> Write & Read heavy Access through key Inbuilt versioning Configurable replication CQL – Cassandra Query Language No support for SQL Does not support joins Does not support integrity constraints across tables Supports secondary indexes 	<ul style="list-style-type: none"> Write & Read heavy Access through key APIs No support for SQL Does not support joins Does not support integrity constraints across collectitons Supports indexing Uses BSON format to store data, similar to JSON like format.
CAP	<ul style="list-style-type: none"> CA – Consistency, Availability Single node architecture. 	<ul style="list-style-type: none"> CP – Consistency, Partition Tolerance 	<ul style="list-style-type: none"> AP – Availability, Partition Tolerance Supports eventual consistency 	<ul style="list-style-type: none"> CP – Consistency, Partition Tolerance
Others	<ul style="list-style-type: none"> Inspired by relational theory. Conforms to Codd's rules. 	<ul style="list-style-type: none"> Inspired by Google BigTable 	<ul style="list-style-type: none"> Initially developed by Facebook 	<ul style="list-style-type: none"> Initially Developed by DoubleClick (acquired by Google)

File Formats

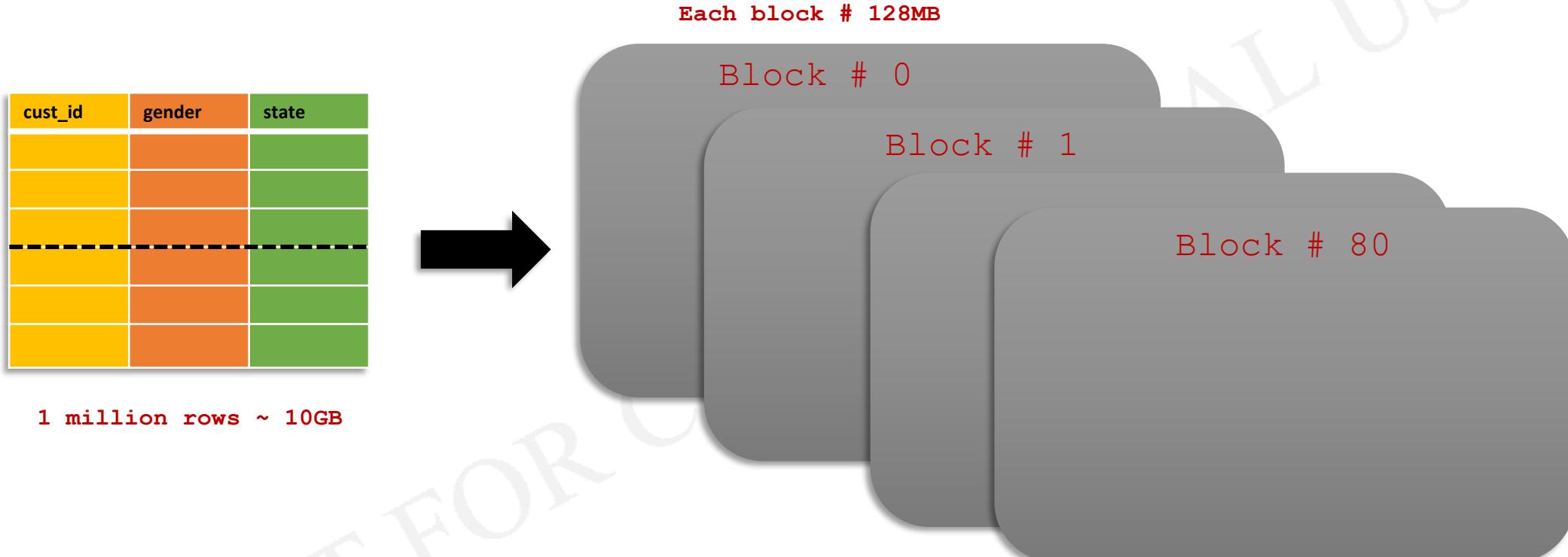


Parquet



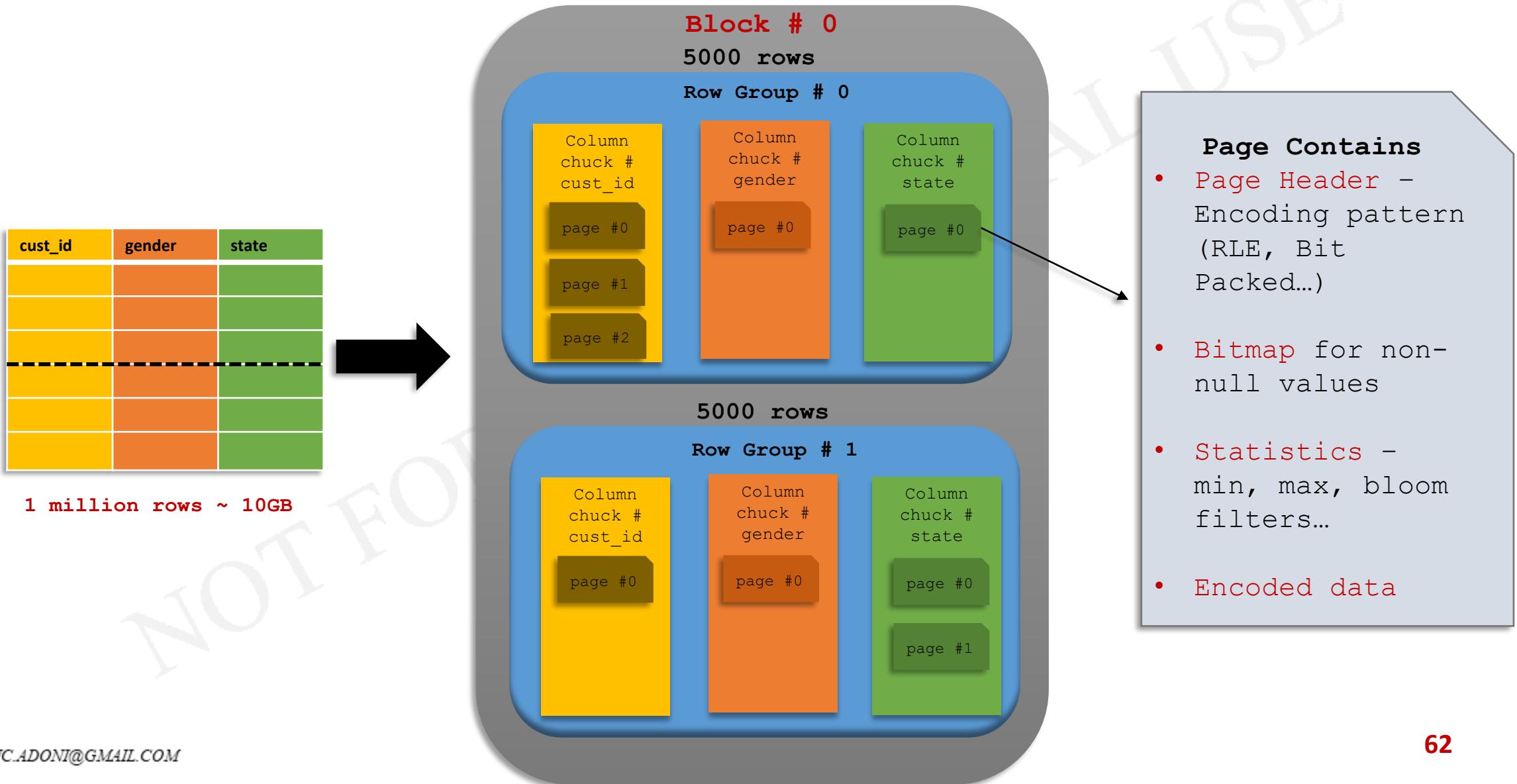
Apache
orcTM

Storage format - Blocks



- Each block size would be exactly 128MB if stored on HDFS (assuming HDFS block size is 128MB).
- If stored using Spark, the number of blocks may be more, based on number of tasks. But each block would be maximum of 128MB in size.

Storage format – Granular View



Encoding

- Run-length Encoding
 - Column Values: [Bangalore, Chennai, Delhi, Bangalore, Bangalore, Mumbai]
 - Encoded Column Values: [Bangalore, Chennai, Delhi, (2,Bangalore),Mumbai]
- Incremental Encoding
 - Column Values: [100, 101, 108, 111, 112]
 - Encoded Column Values: [100, 101, 108, 111] => [100, +1, **+8**, +3, +4]
- Dictionary Encoding
 - Unique Column Values: {Male, Female, Unknown}
 - Dictionary: 1 -> Male , 2 -> Female, 3 – Unknown
 - Encoded Column Values: [Male, Male, Female, Unknown] => [1,1,2,3]

Parquet vs ORC

Parquet		ORC (Optimized Row Columnar)
Inspiration	<ul style="list-style-type: none"> Based on Google Dremel Promoted by Cloudera, Twitter in 2013 Primary objective: Store lots of data and read fast (CPU & Disk I/O). 	<ul style="list-style-type: none"> Evolved from RCFfile Promoted by Hortonworks, Facebook in 2013 Primary objective: Make Hive Fast
Format	<ul style="list-style-type: none"> Columnar store format Optimized read at the expense of write Structured data Schema is embedded with the data 	<ul style="list-style-type: none"> Columnar store format Optimized read at the expense of write Structured data
Compression	<ul style="list-style-type: none"> Supports snappy, gzip2, LZO Snappy is preferred for reads priorities. Gzip2 is preferred for storage priorities 	<ul style="list-style-type: none"> Supports snappy, gzip2 Snappy is preferred for reads priorities. Gzip2 is preferred for storage priorities
Usage	<ul style="list-style-type: none"> Supports and efficient for nested data structure Efficient for wide column data (hundreds of columns) 	<ul style="list-style-type: none"> Efficient for flat data structure Efficient for narrow column data (less than hundred columns)
Others	<ul style="list-style-type: none"> Supports predicate push down. Supports column pruning and predicate pushdown. No support for ACID 	<ul style="list-style-type: none"> Limited support for nesting Supports column pruning and predicate pushdown. Hive supports ACID on ORC store format

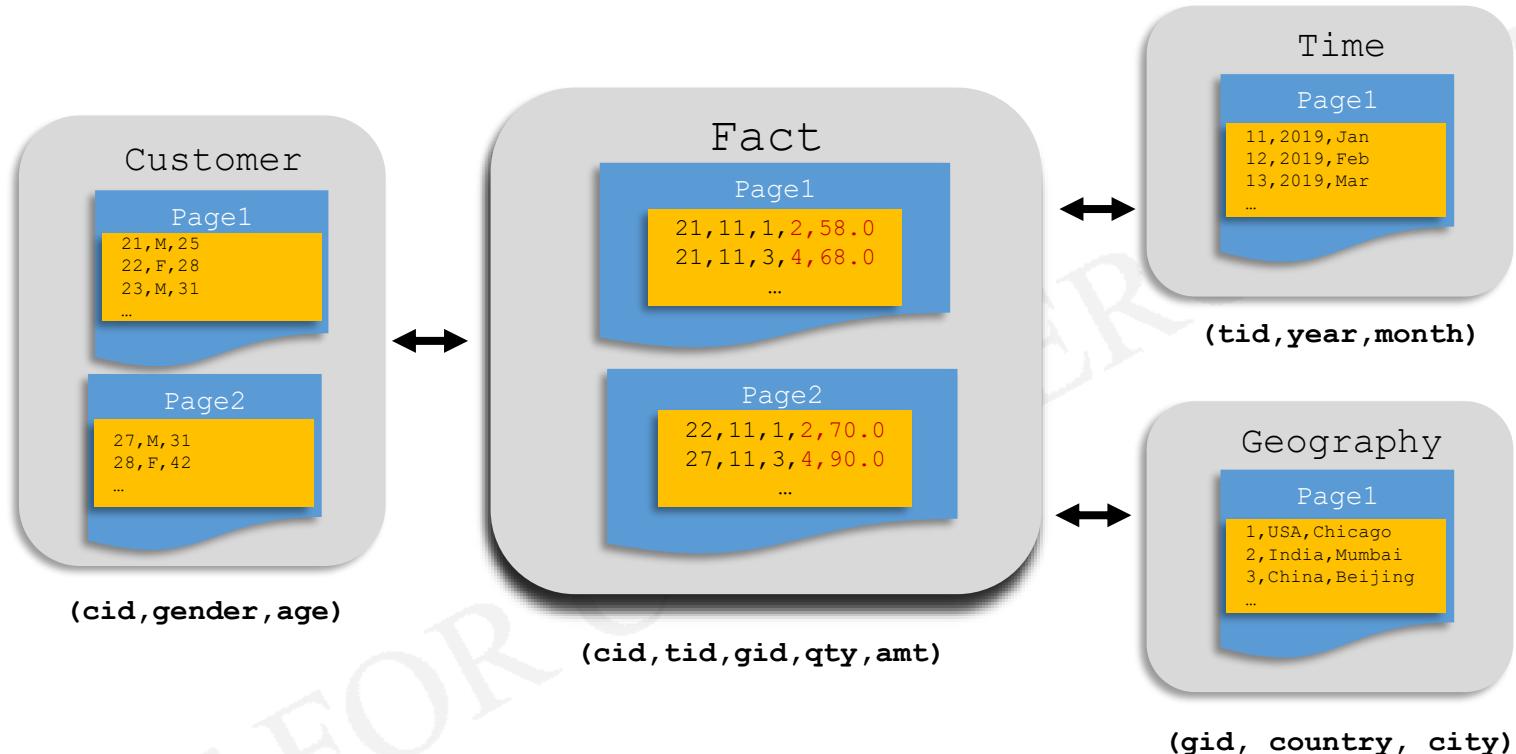
```
{
    "name": "john",
    "phone" : {
        "office_phone" : 123,
        "personal_phone" : 456,
        "office_desk_direct": 9999
    }
}
```

```
{
    "name": "john",
    "office_phone" : 123,
    "personal_phone" : 456,
    "office_desk_direct": 9999
}
```

Apache Druid

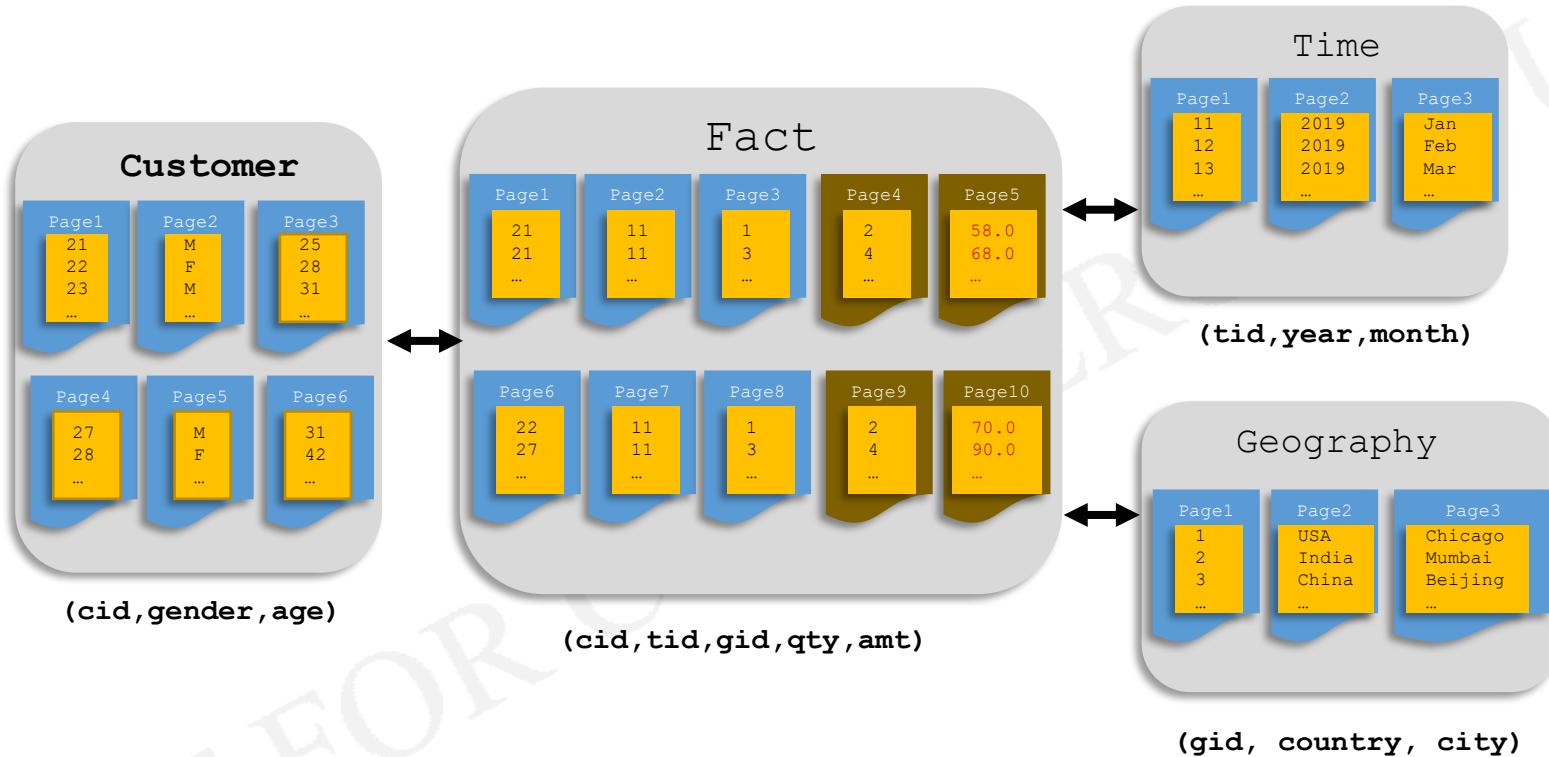
- **Why Distributed OLAP?**
 - Limitations of distributed DBMS – Why OLAP Cube?
 - Introduction to Druid
- **Druid**
 - **Architecture**
 - **Components of Druid**
 - Master
 - Data
 - Query
 - **Data storage in Druid**
 - Segments
 - Granularities – Query, Segment
 - **Ingestion**
 - Ingestion – Rollup, Indexing
 - Different types of Ingestion – Batch, Hadoop Based, Realtime (Kafka)
 - Transformations - JavaScript, Expression
 - Filter
 - Lookup
 - Multi Value Column Dimension
 - Versioning – Reload of data, Drop, Kill

Why OLAP Cube? – Star Schema, Row RDBMS



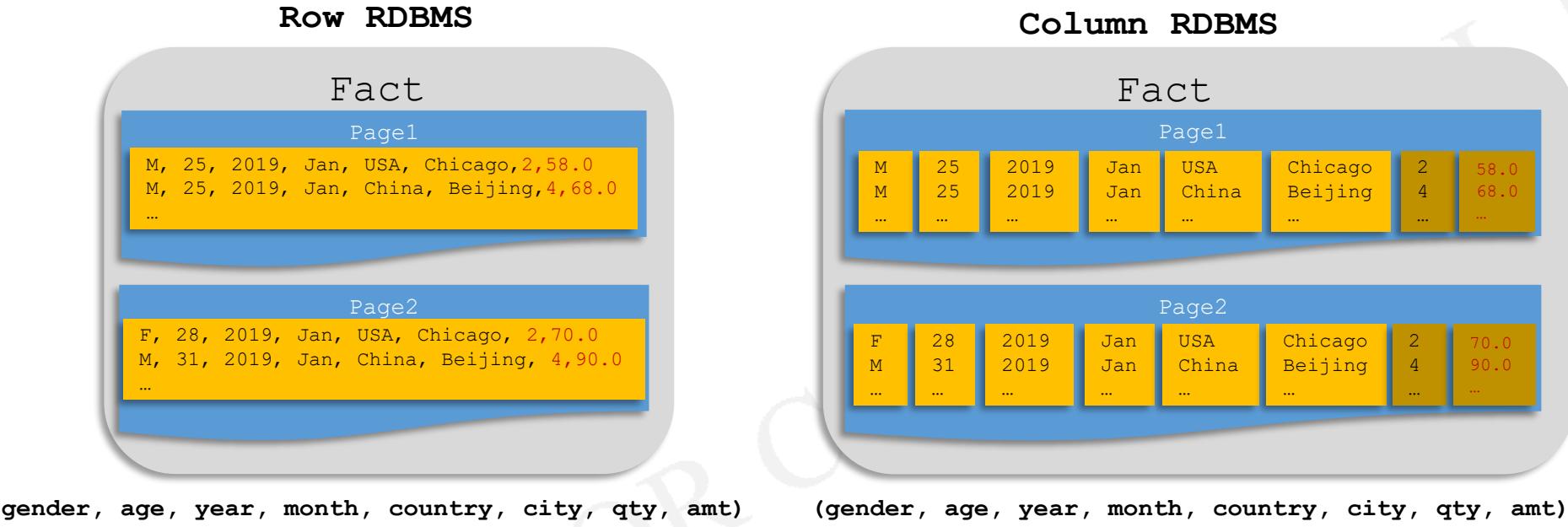
```
select gender,sum(qty) from fact where gender='M' and city='Mumbai'
```

Why OLAP Cube? – Star Schema, Column RDBMS



```
select gender,sum(qty) from fact where gender='M' and city='Mumbai'
```

Why OLAP Cube? – Flat Structure (RDBMS)

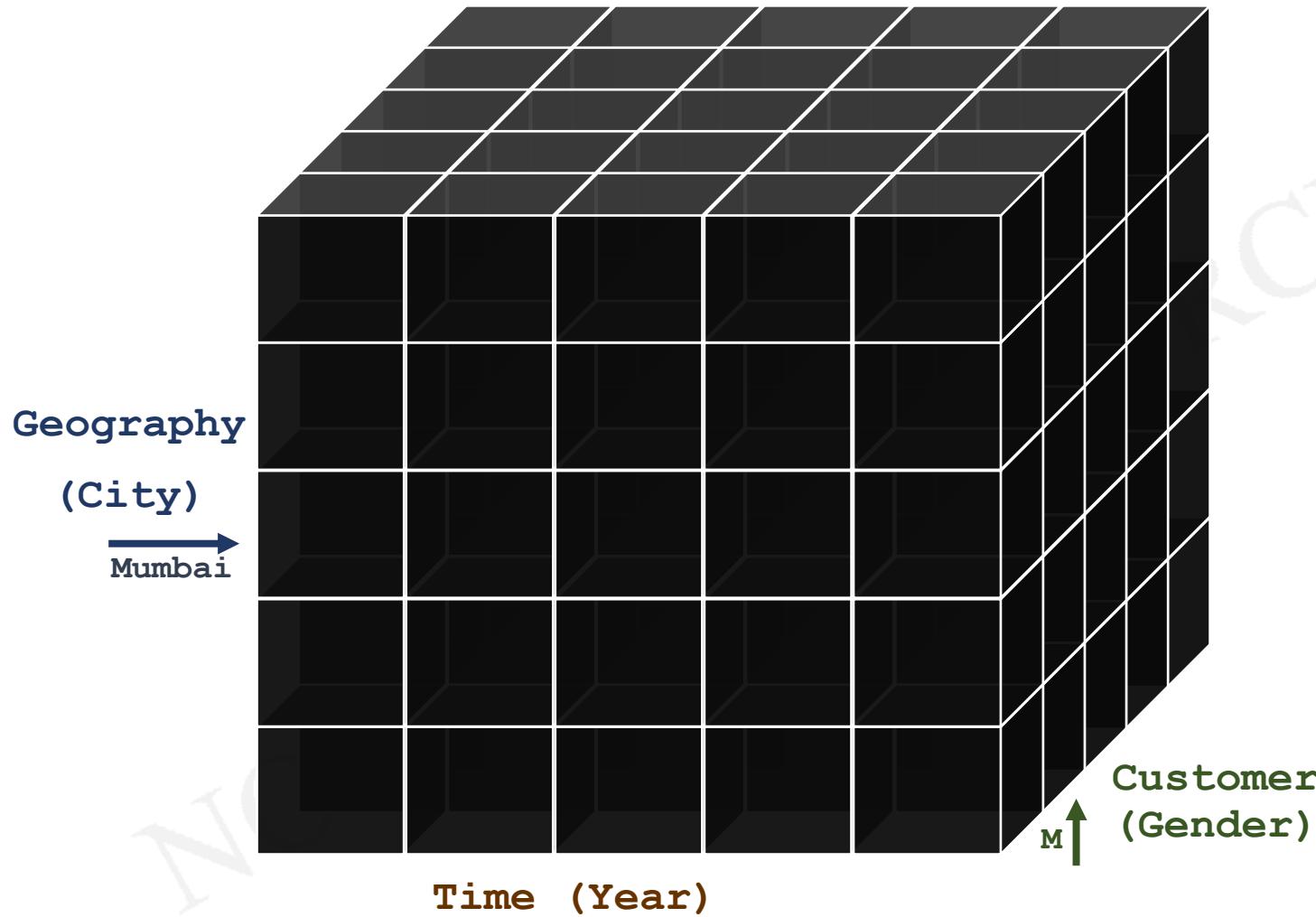


- Every dimension, attribute is stored/treated as an independent unit of data.

Is there a any other way to represent?

select gender,sum(qty) from fact where gender='M' and city='Mumbai'

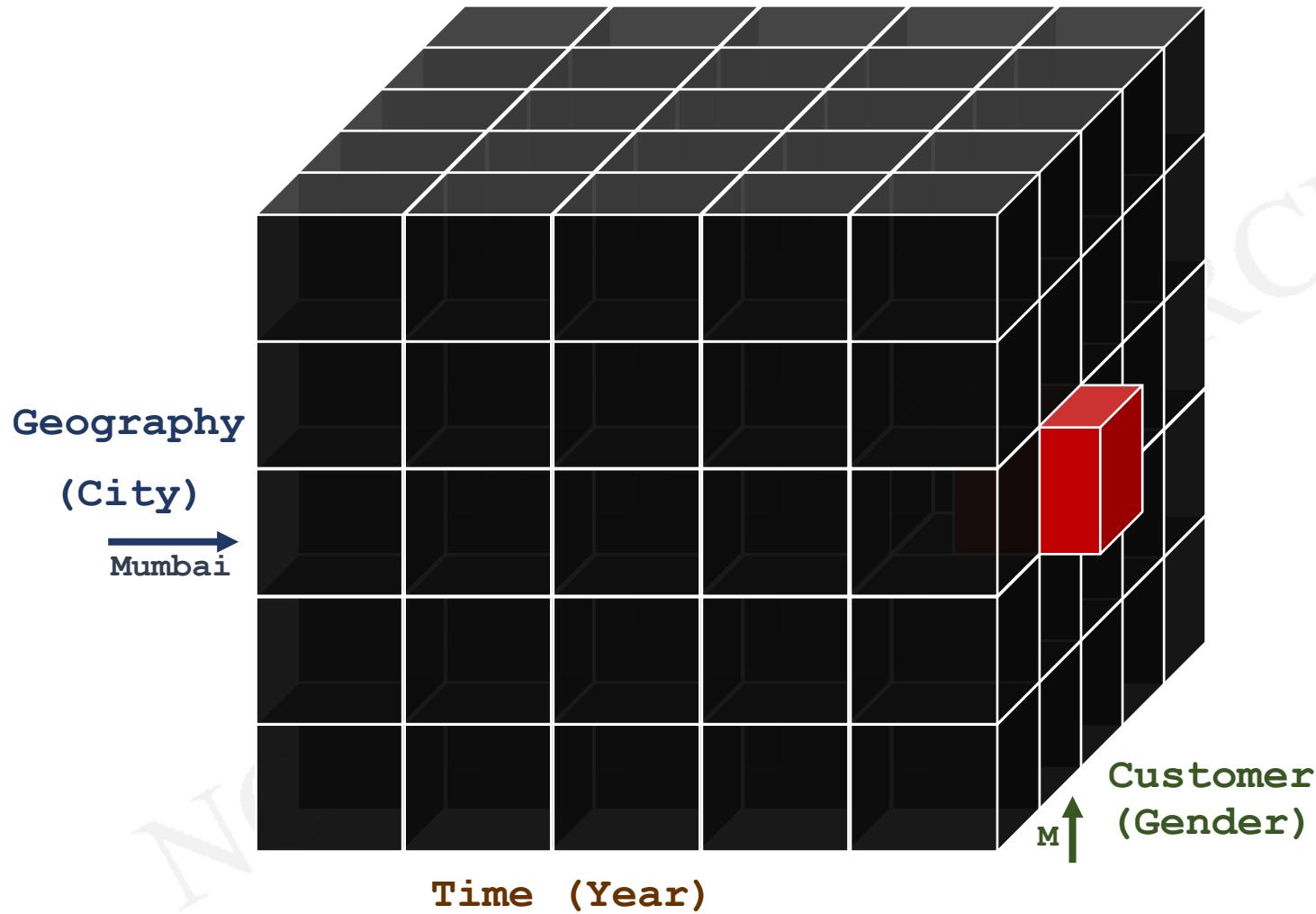
OLAP Cube



- Every attribute (Age, Gender, Marital Status, City, Country...) will be treated as an independent dimension.
- Intersection of multiple dimensions is qualified by cell/metric.

```
select  
gender,sum(qty)  
from fact where  
gender='M' and  
city='Mumbai'
```

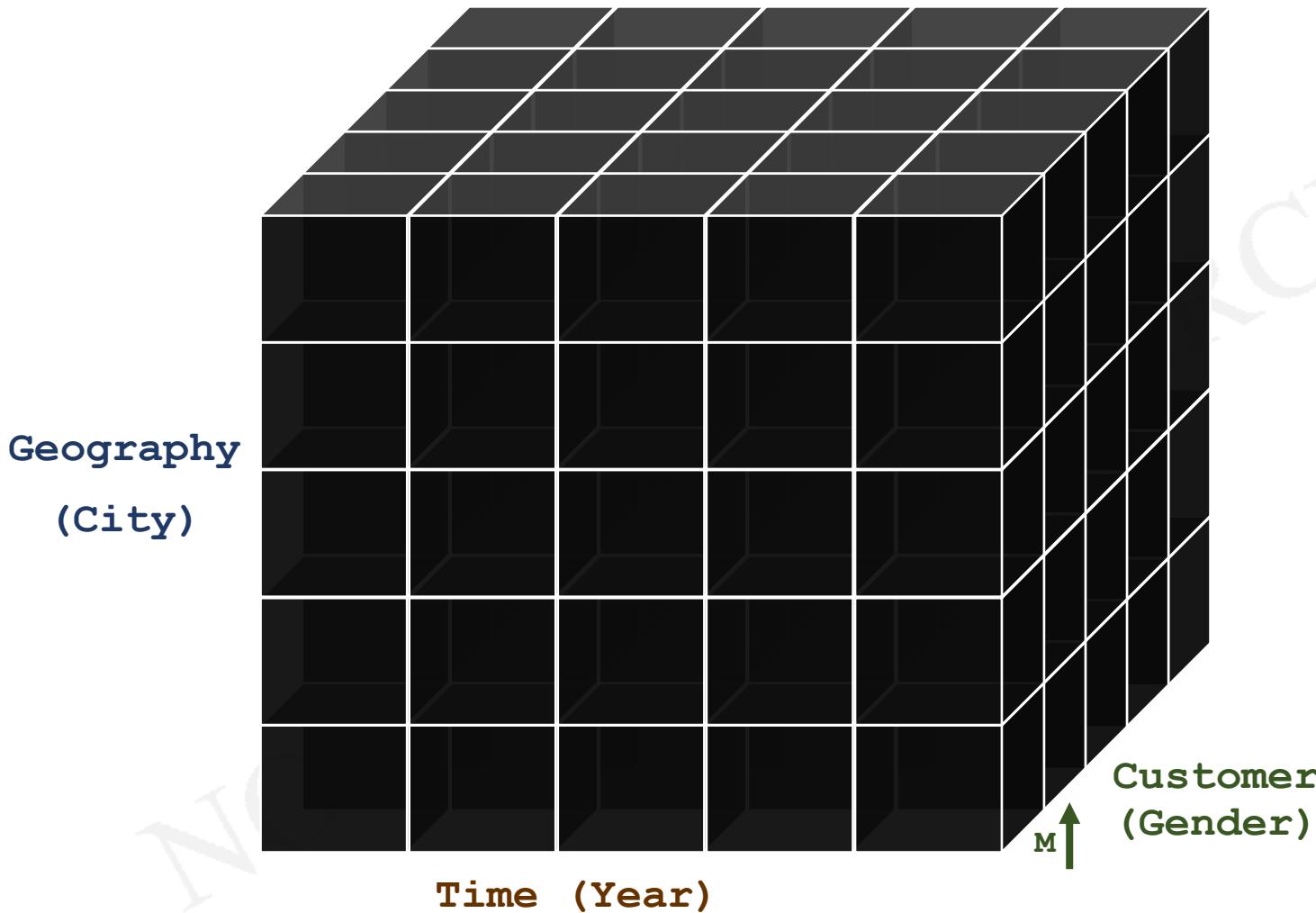
OLAP Cube



- Every attribute (Age, Gender, Marital Status, City, Country...) will be treated as an independent dimension.
- Intersection of multiple dimensions is qualified by cell/metric.

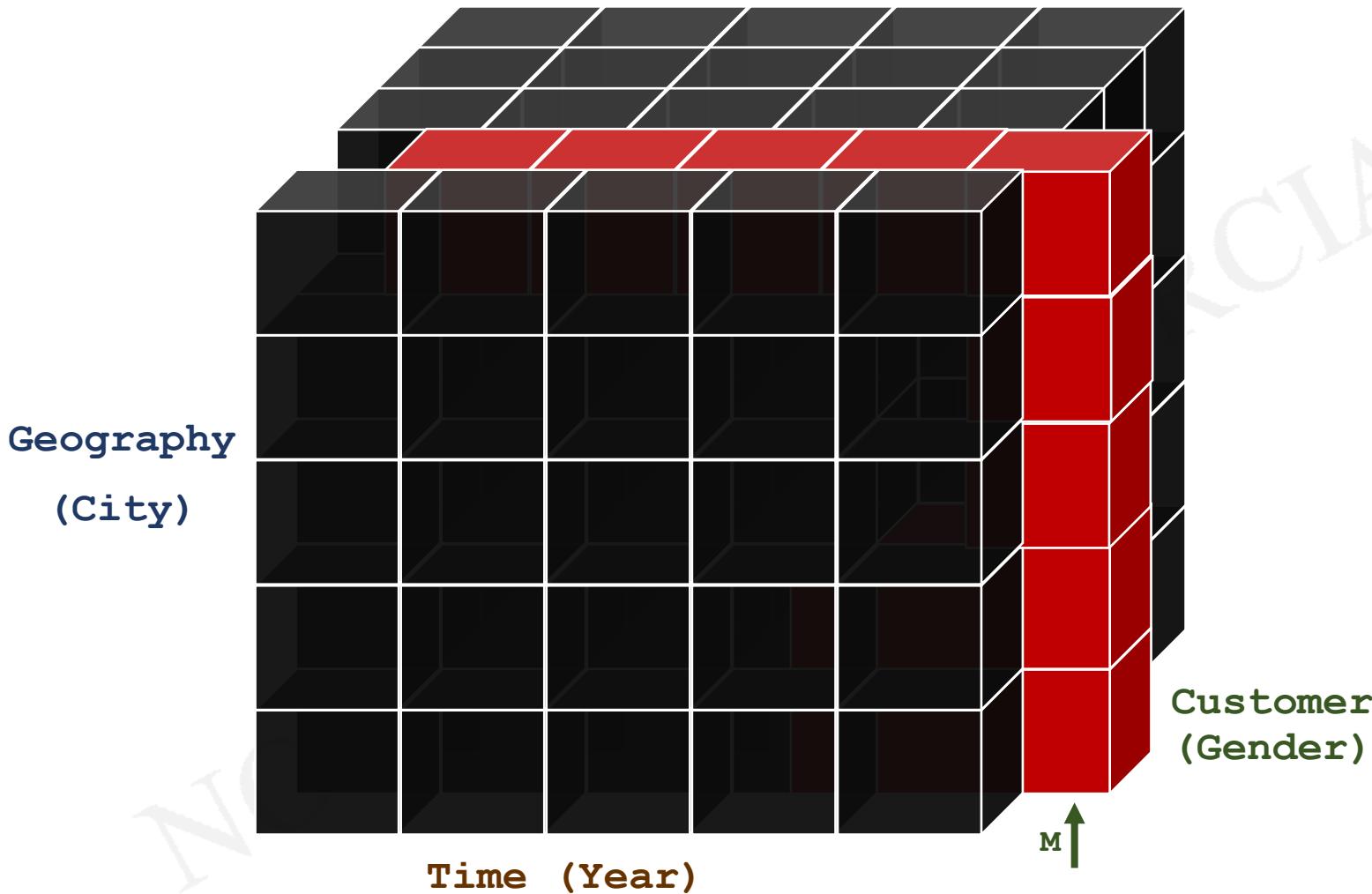
```
select  
gender,sum(qty)  
from fact where  
gender='M' and  
city='Mumbai'
```

OLAP Cube



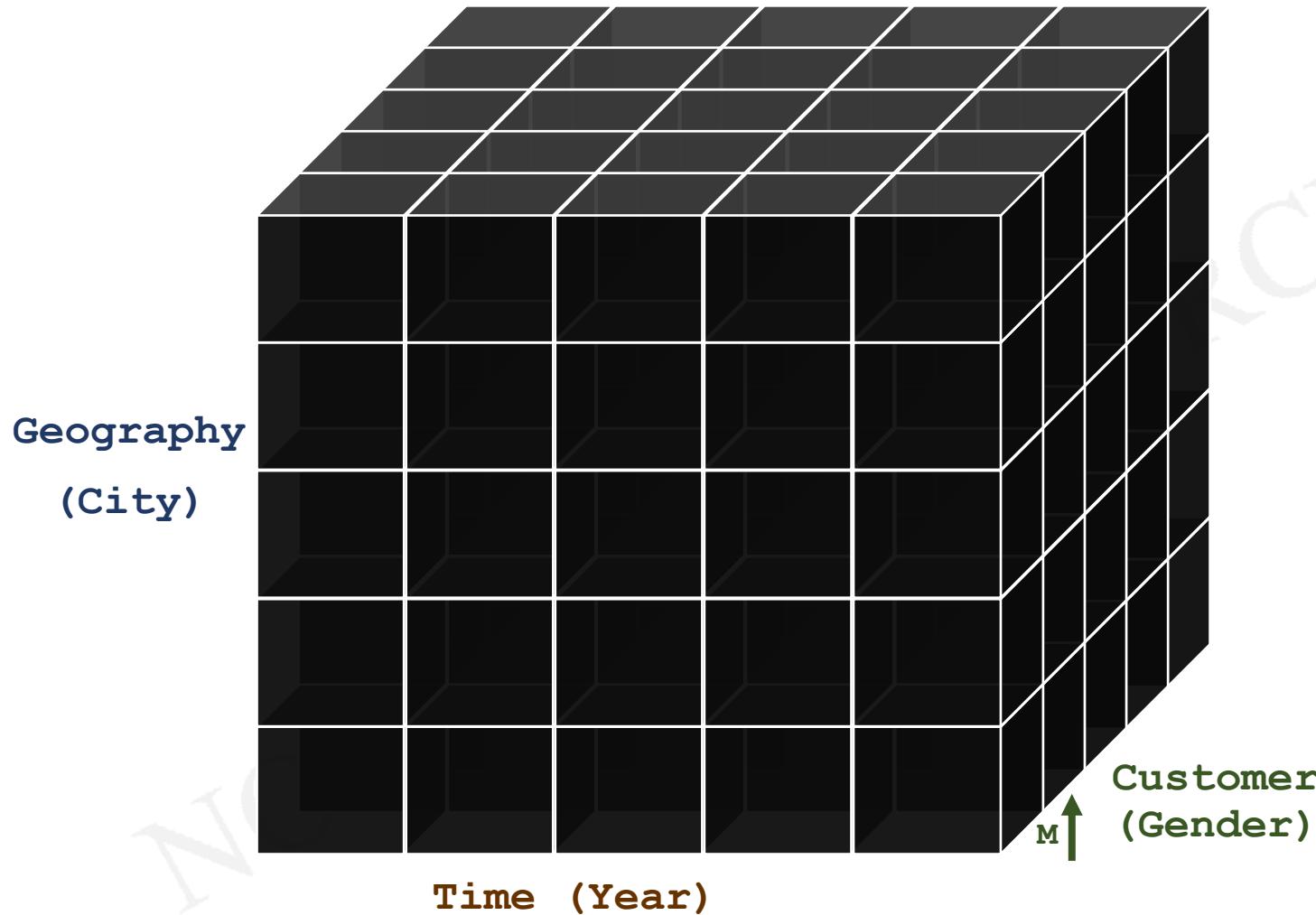
```
select  
gender,sum(qty)  
from fact where  
gender='M'
```

OLAP Cube



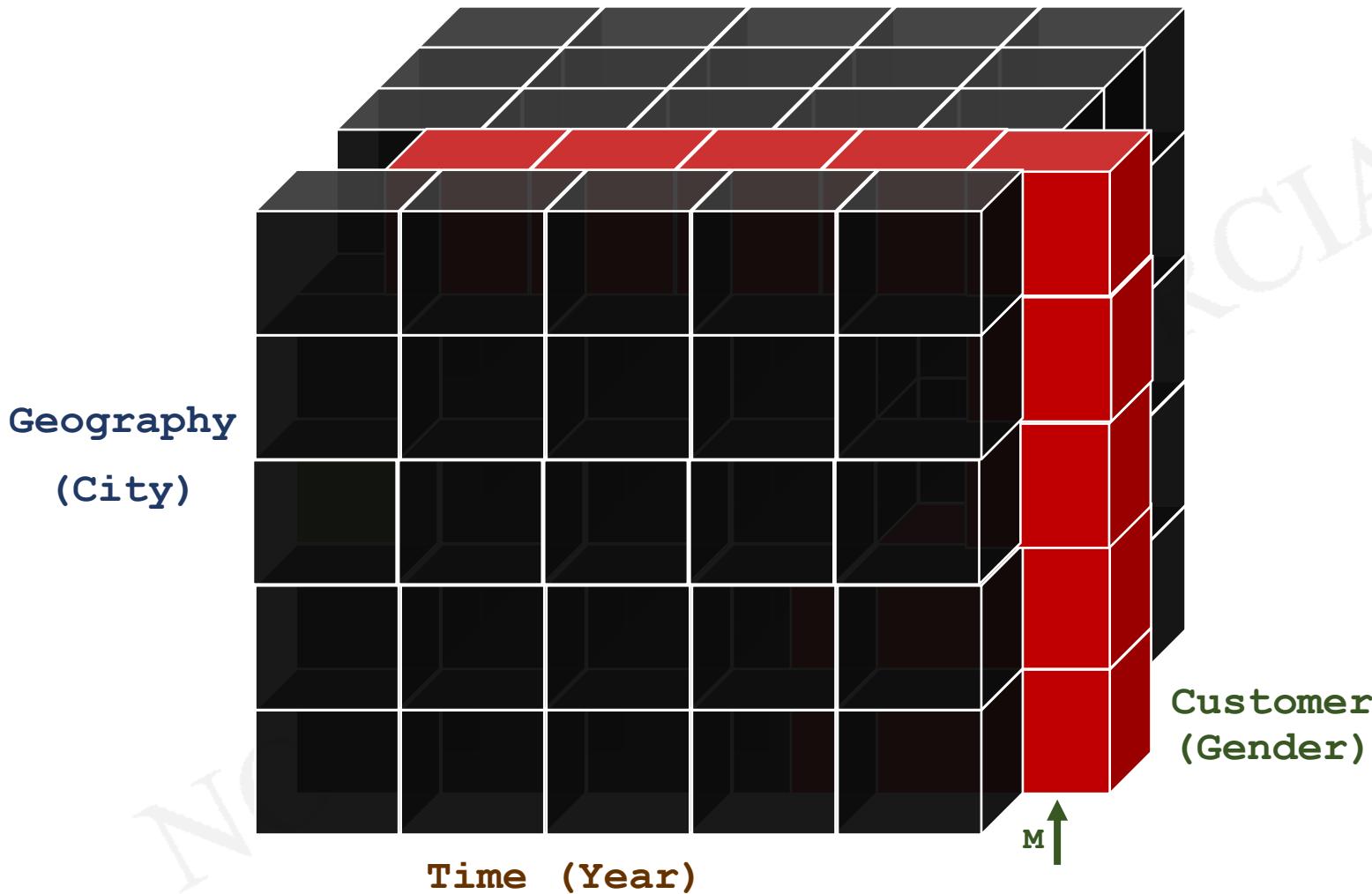
```
select  
gender,sum(qty)  
from fact where  
gender='M'
```

OLAP Cube



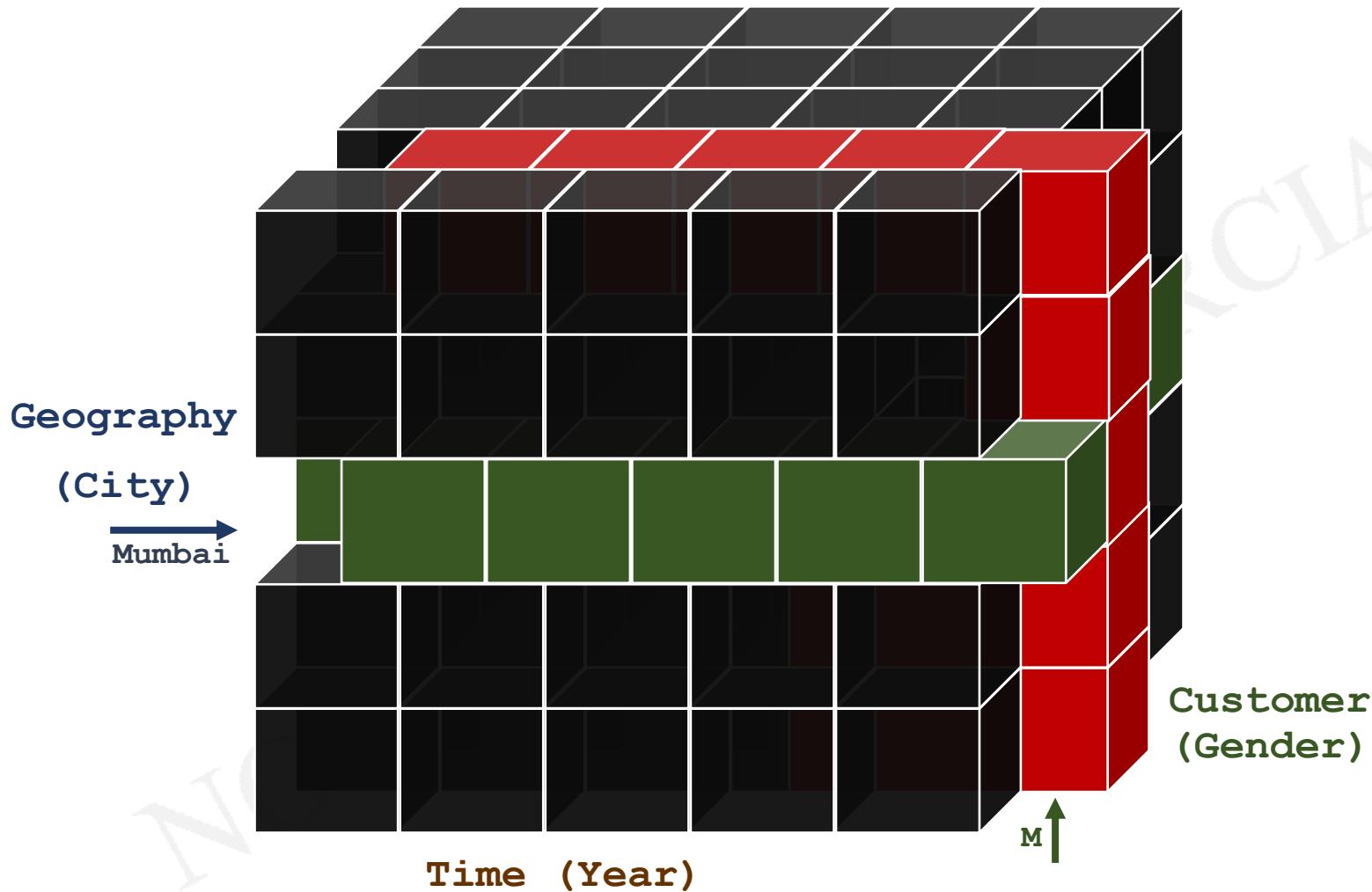
```
select  
gender,sum(qty)  
from fact where  
gender='M' or  
city='Mumbai'
```

OLAP Cube



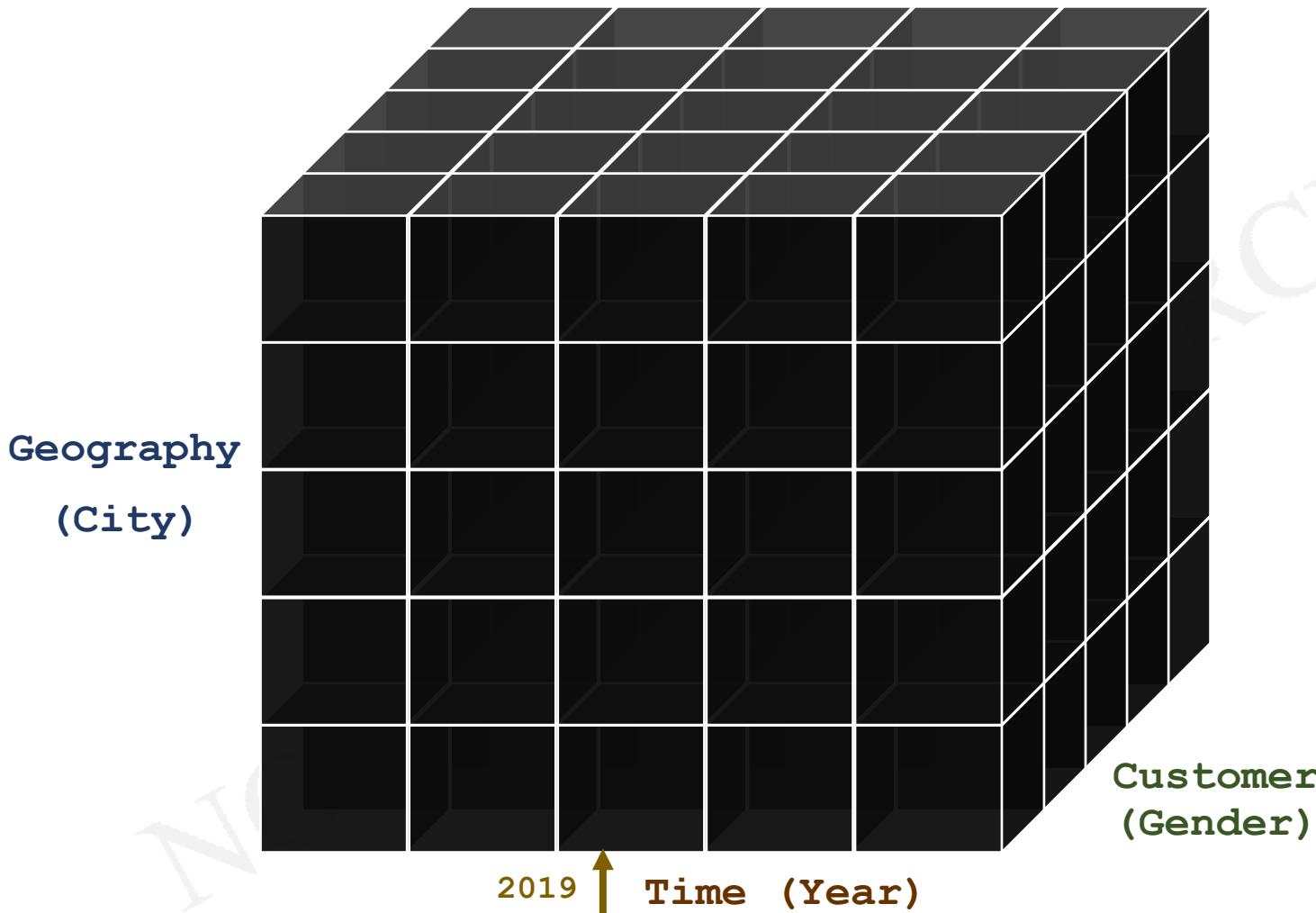
```
select  
gender,sum(qty)  
from fact where  
gender='M' or  
city='Mumbai'
```

OLAP Cube



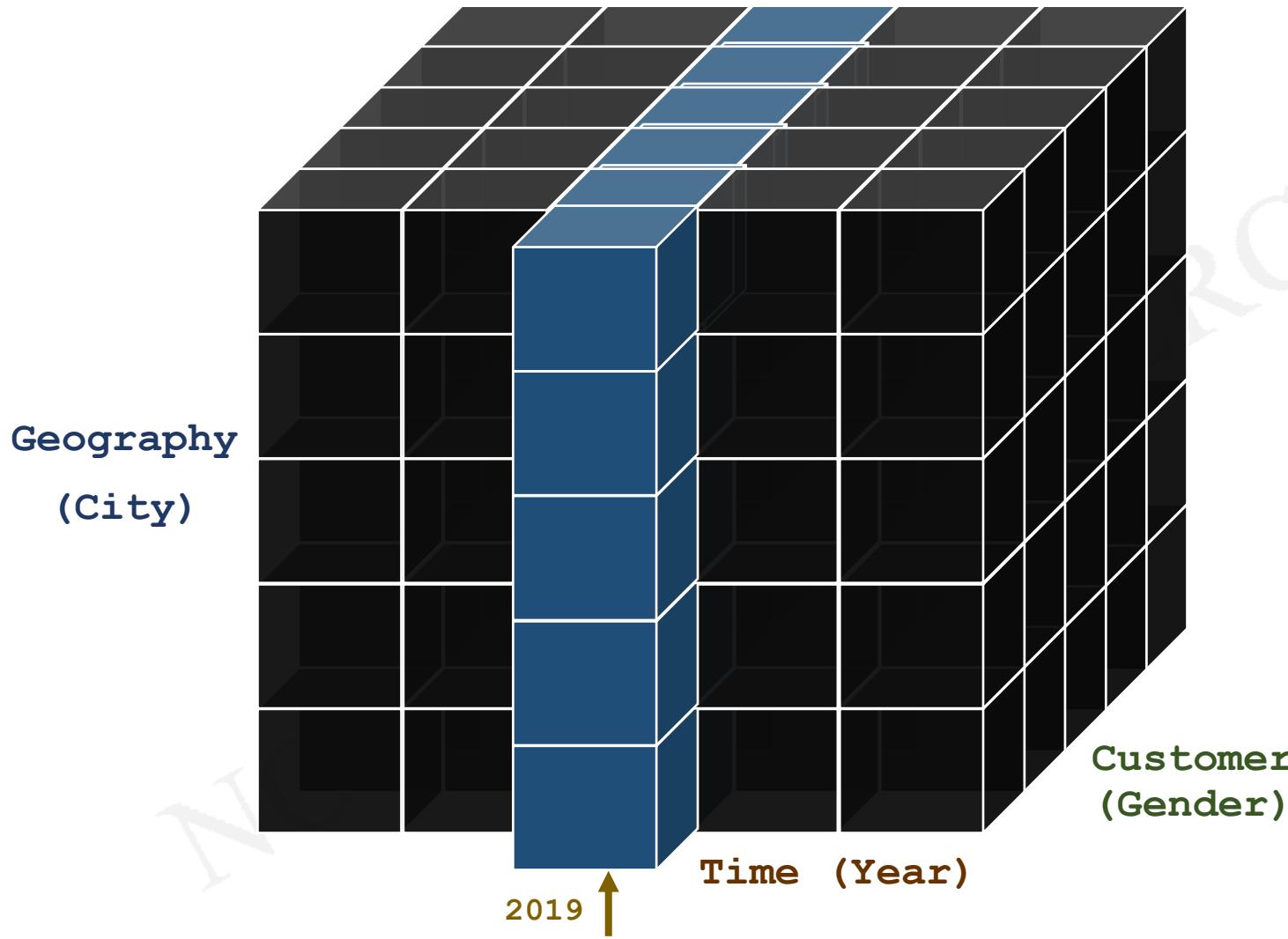
```
select  
gender,sum(qty)  
from fact where  
gender='M' or  
city='Mumbai'
```

OLAP Cube



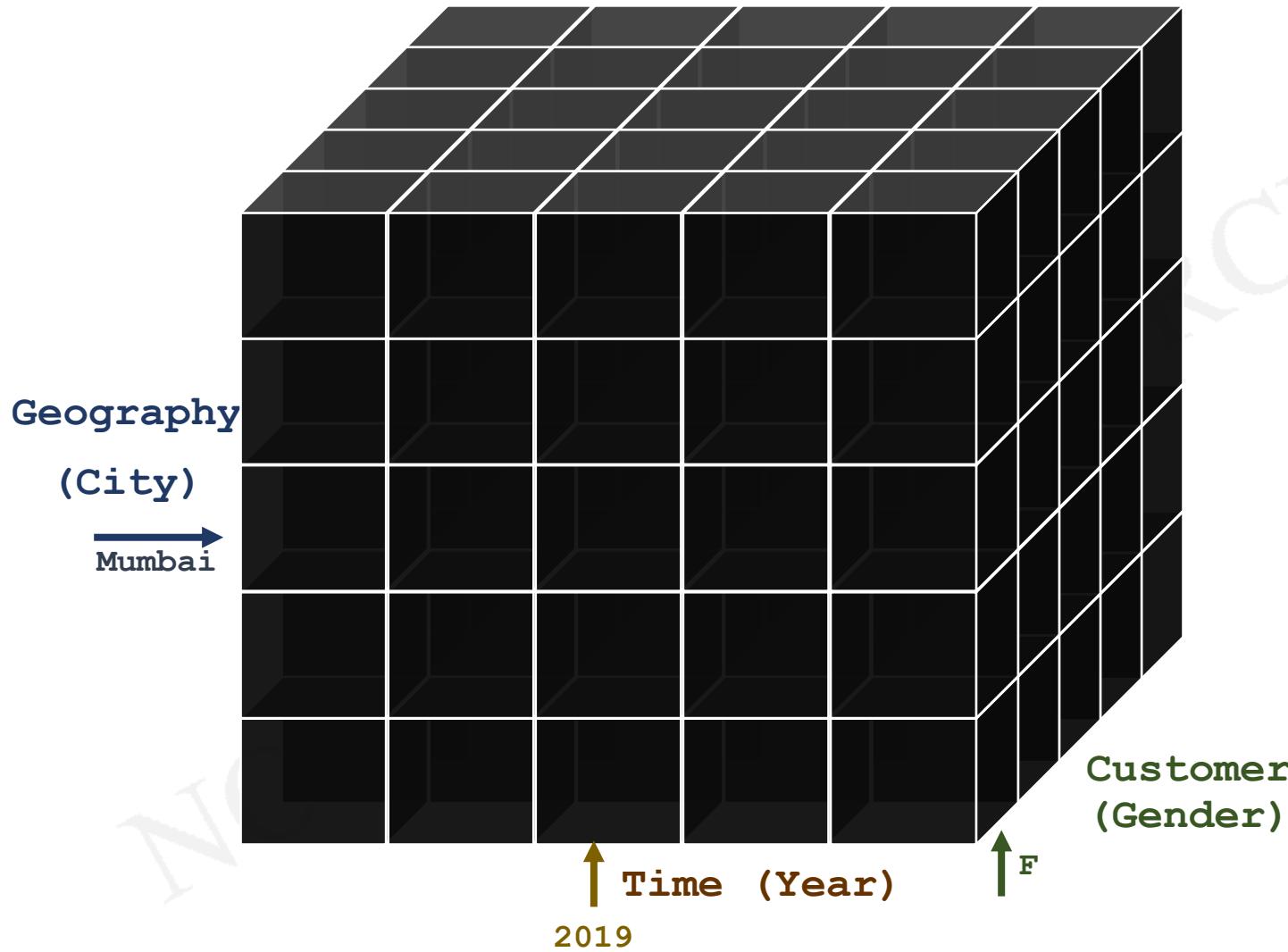
```
select  
gender,sum(qty)  
from fact where  
year=2019
```

OLAP Cube



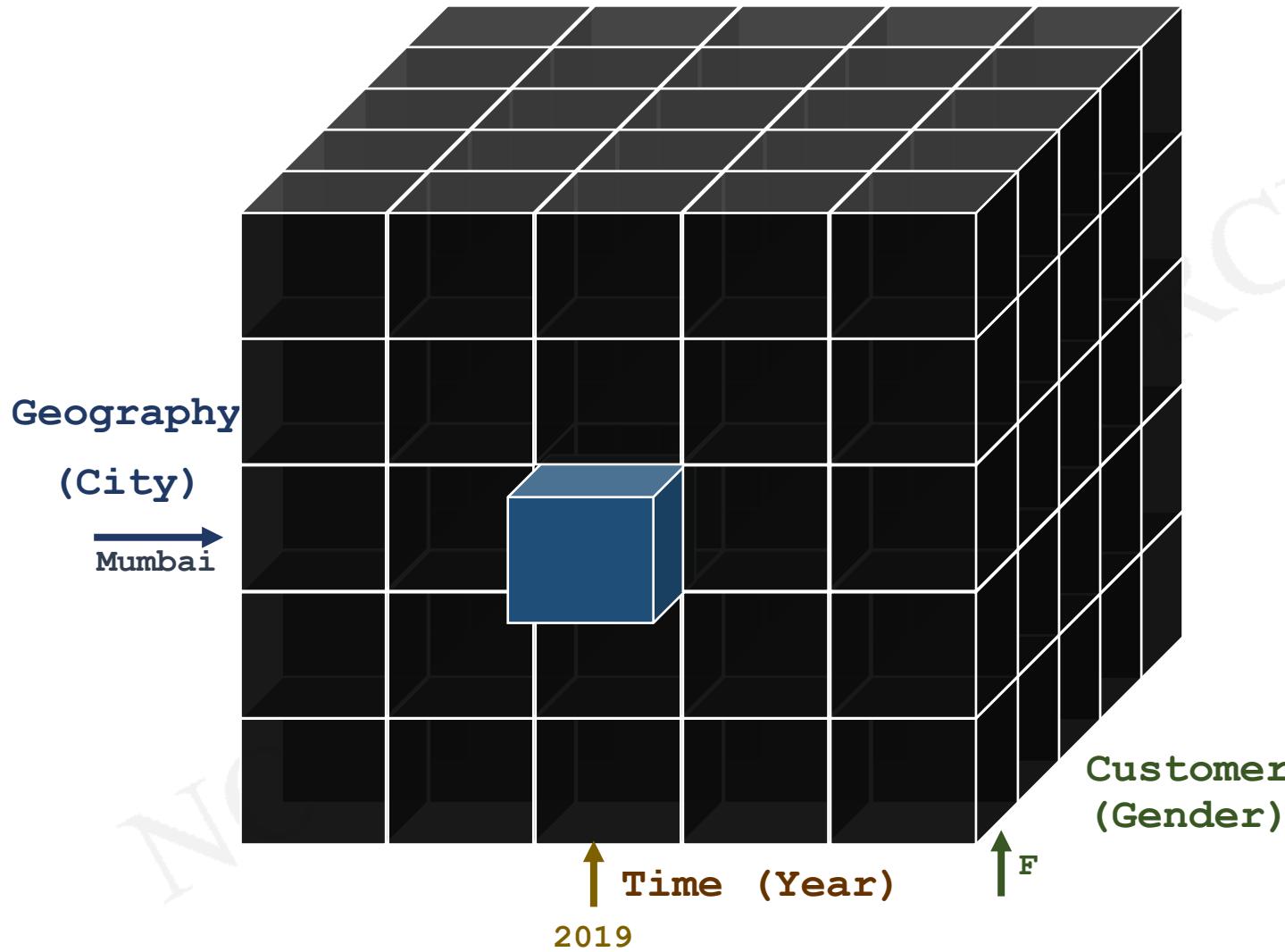
```
select  
gender,sum(qty)  
from fact where  
gender='F'
```

OLAP Cube



```
select  
gender,sum(qty)  
from fact where  
year=2019 and  
gender = 'F' and  
city= 'Mumbai'
```

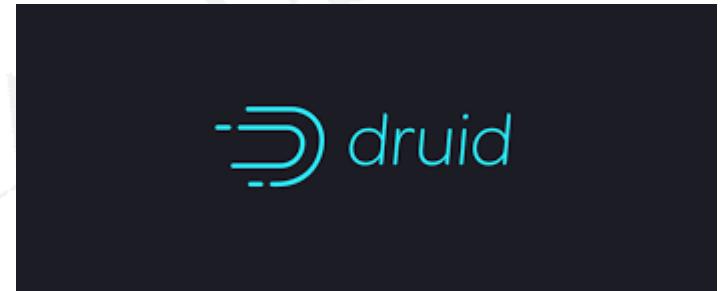
OLAP Cube



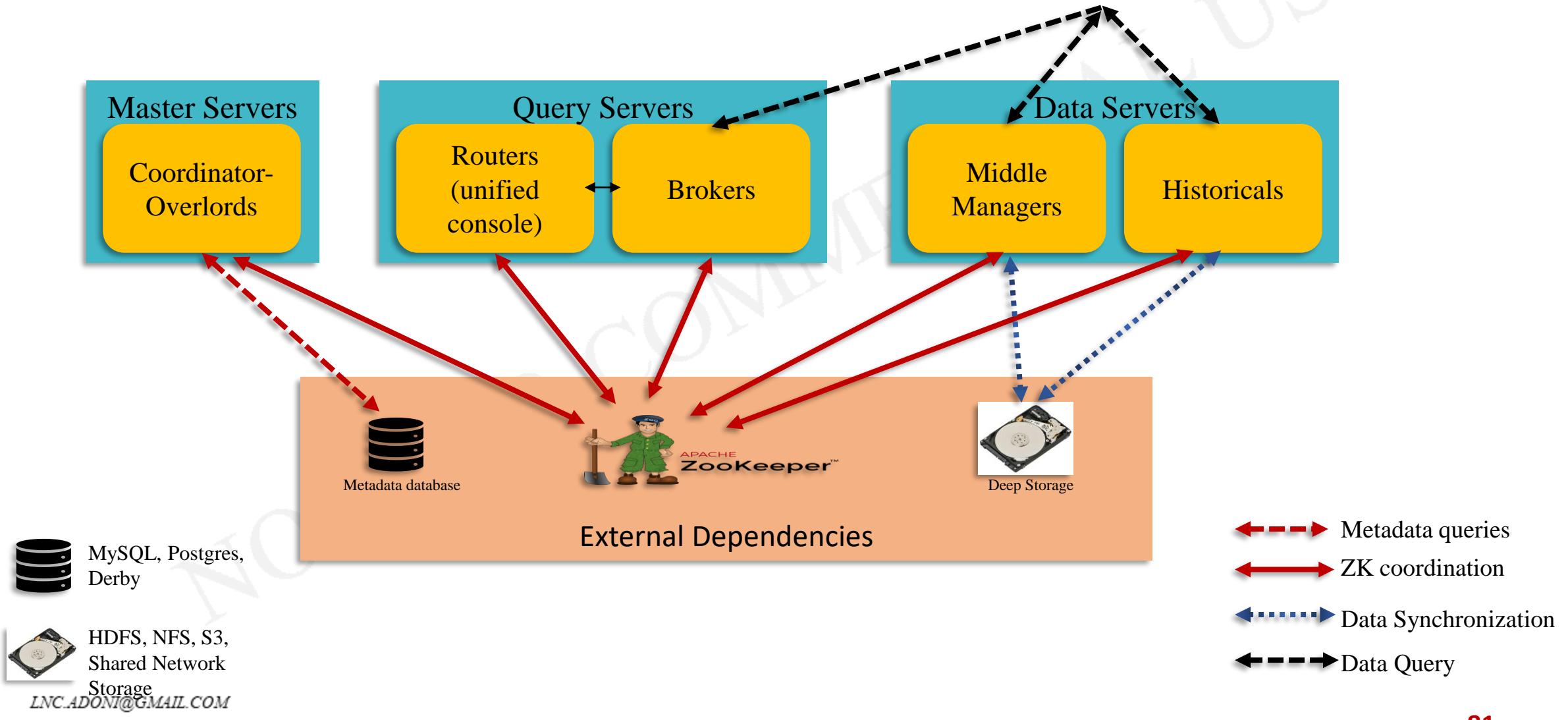
```
select  
gender,sum(qty)  
from fact where  
year=2019 and  
gender = 'F' and  
city= 'Mumbai'
```

Introduction to Apache Druid

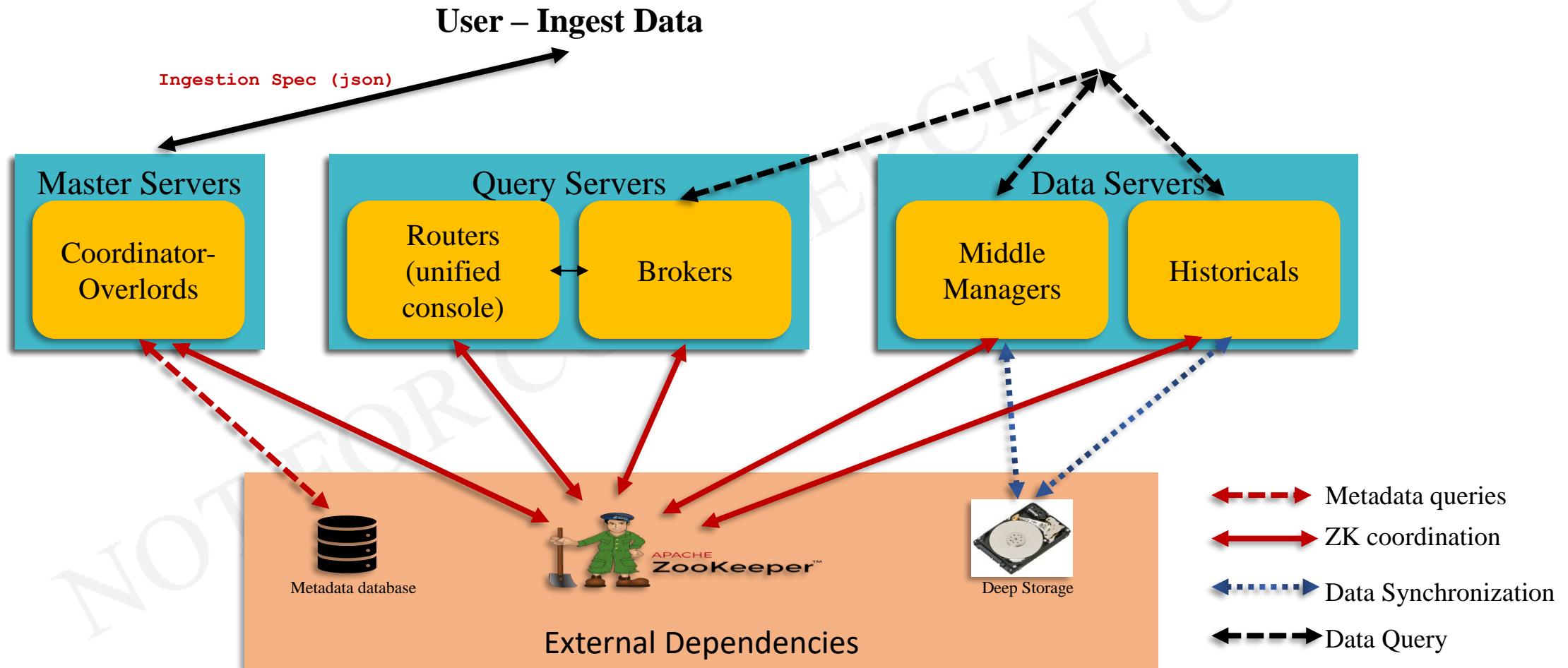
- Druid was developed by Metamarkets in 2011 and opensourced in 2012.
- Druid is – Column Oriented, Distributed Timeseries Data Store, OLAP Engine
 - Supports both realtime and batch ingestion
- The architecture follows **share nothing** design.
- No dependency of any Hadoop components (except Zookeeper).
- Used across industries – **Alibaba, Airbnb, Netflix, Twitter, Walmart, DBS...**
- Latest version is 0.16.0



Apache Druid : Architecture



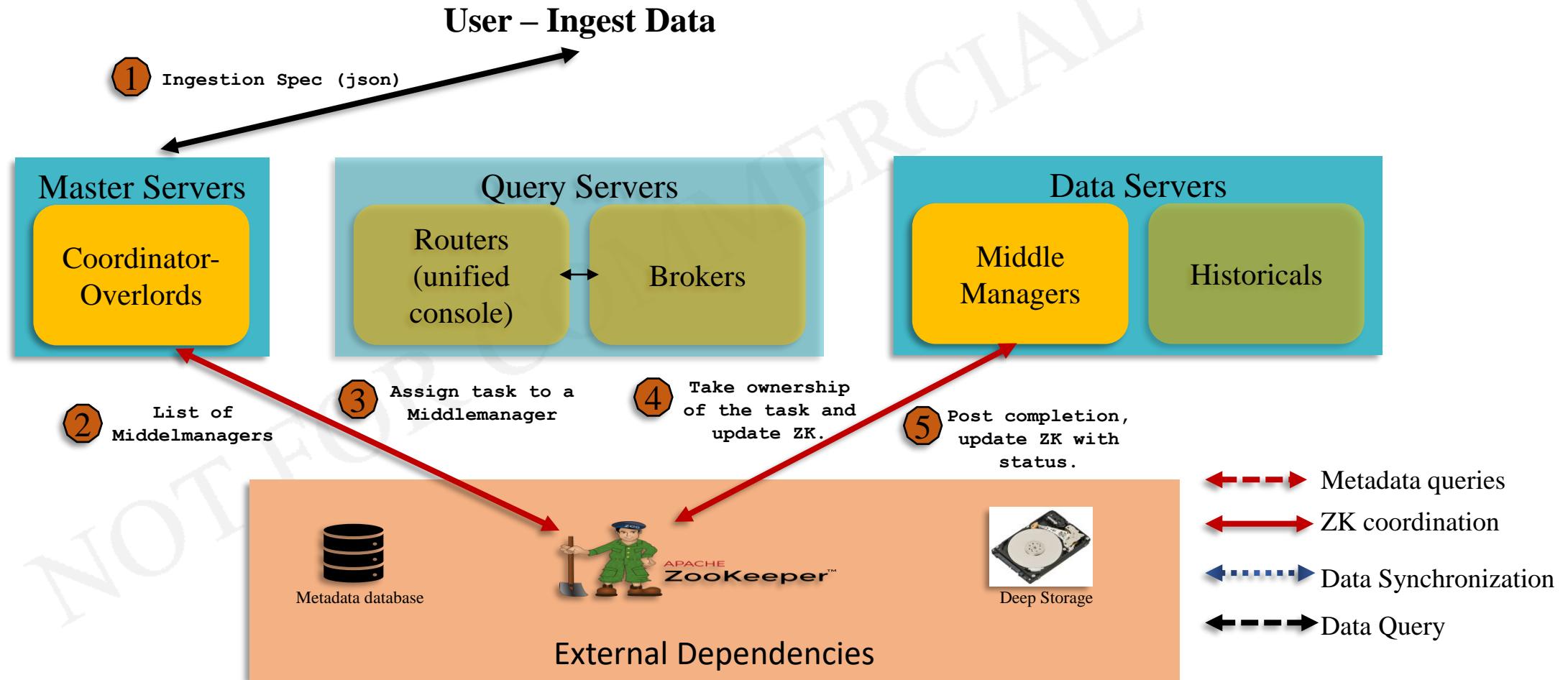
Apache Druid : Ingestion



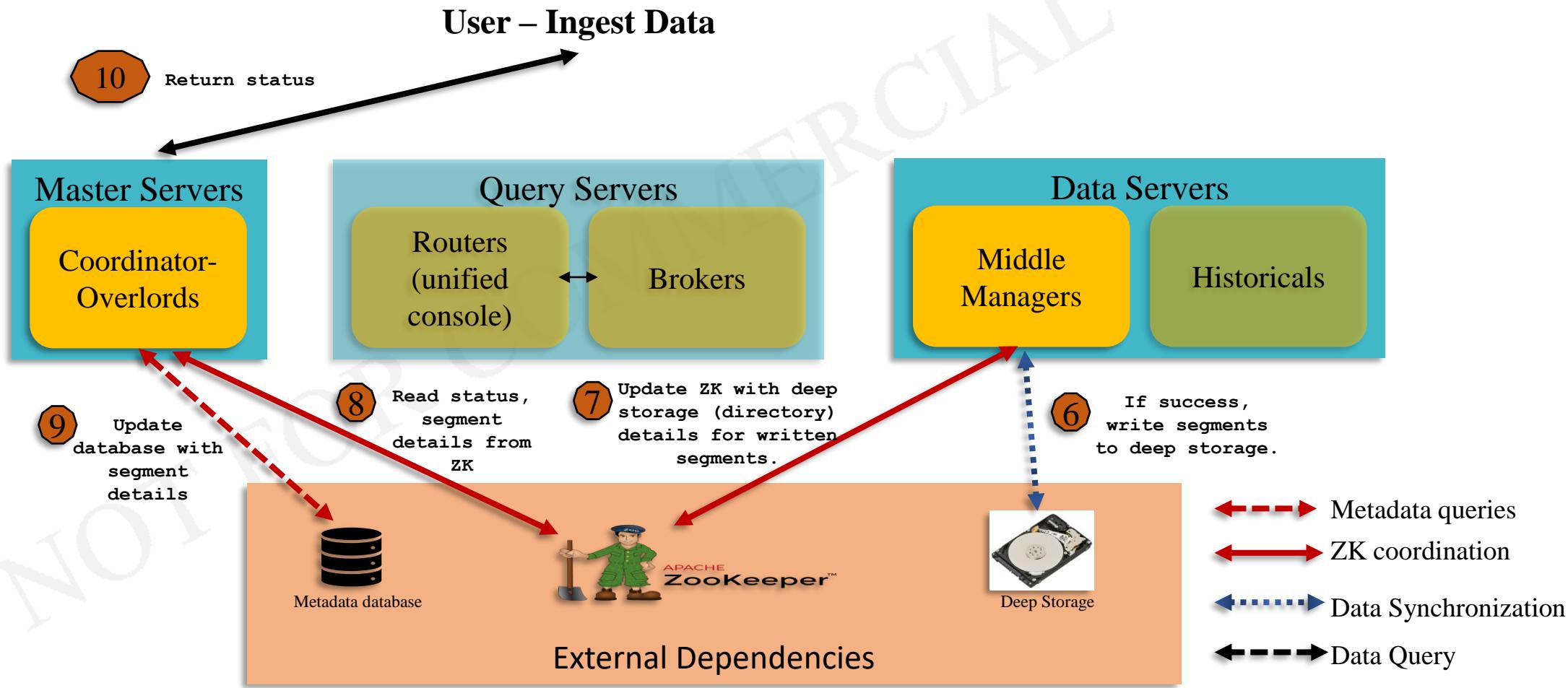
Apache Druid : Ingestion

- Components Involved
 - Coordinator-Overlord
 - Zookeeper
 - Metadata Database
 - Middlemanager
 - Deep Storage
 - Historicals
- Ingestion is always through Coordinator-Overlord.

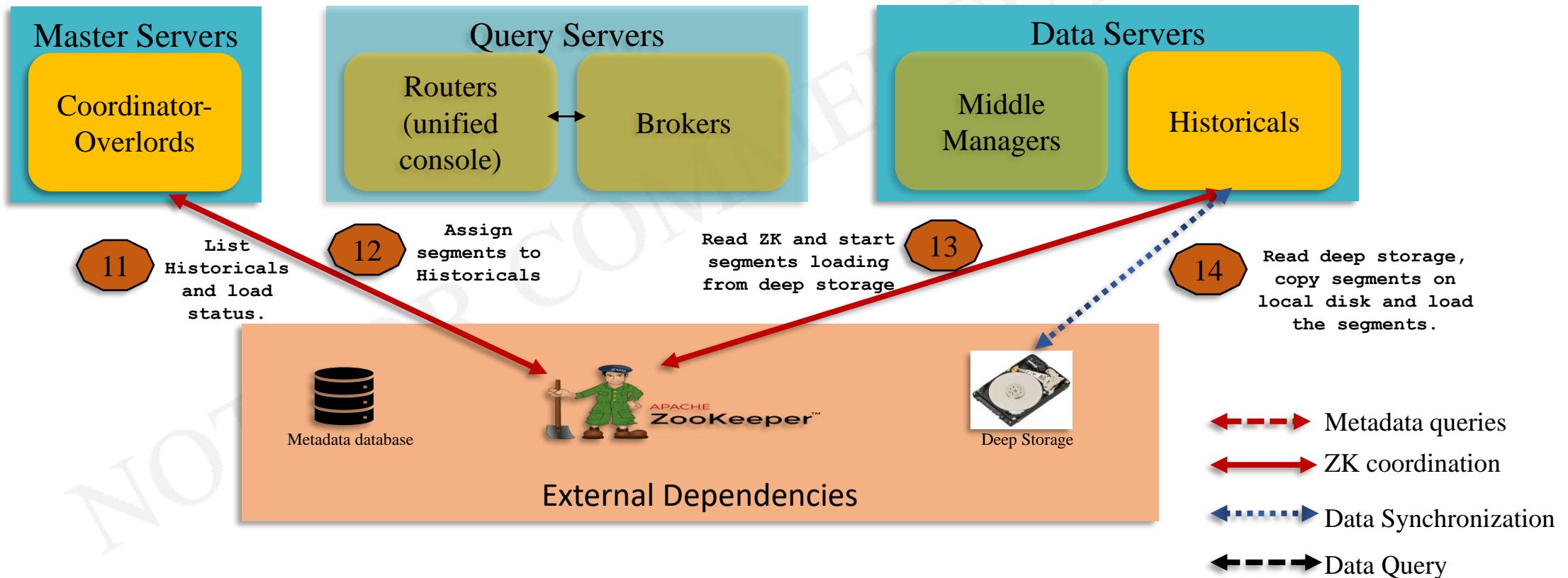
Apache Druid : Ingestion - Workflow



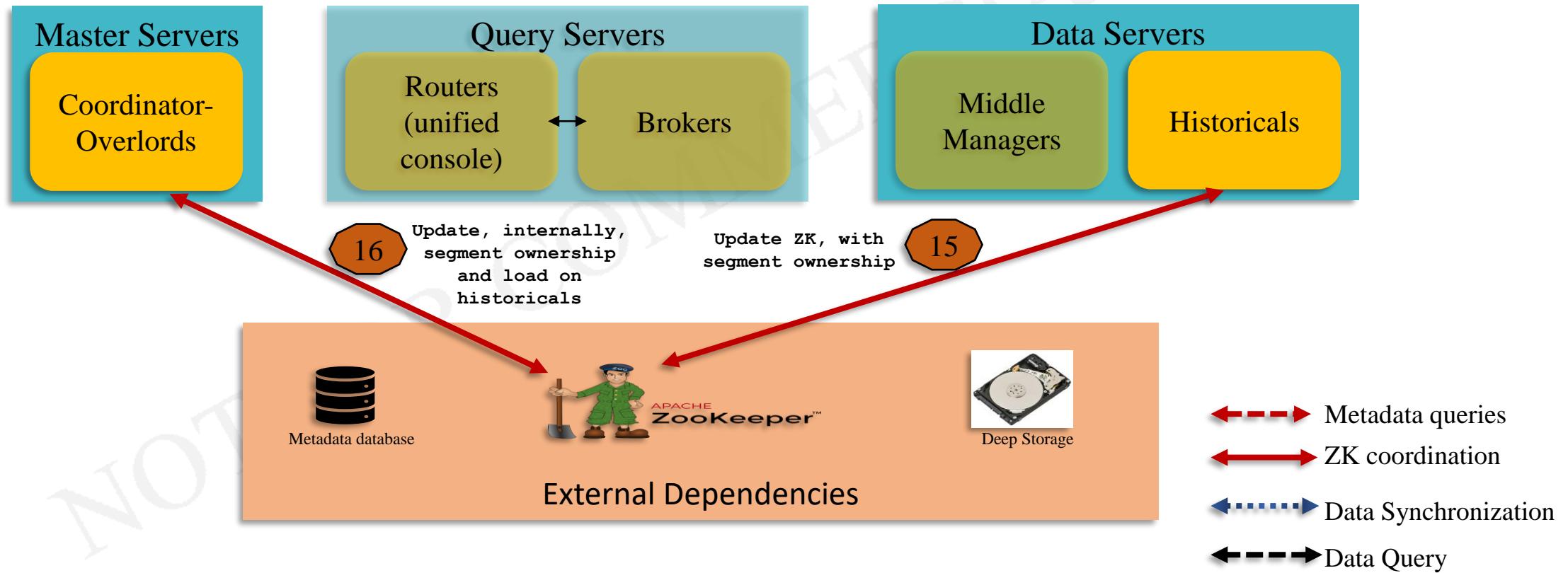
Apache Druid : Ingestion - Workflow



Apache Druid : Ingestion - Workflow



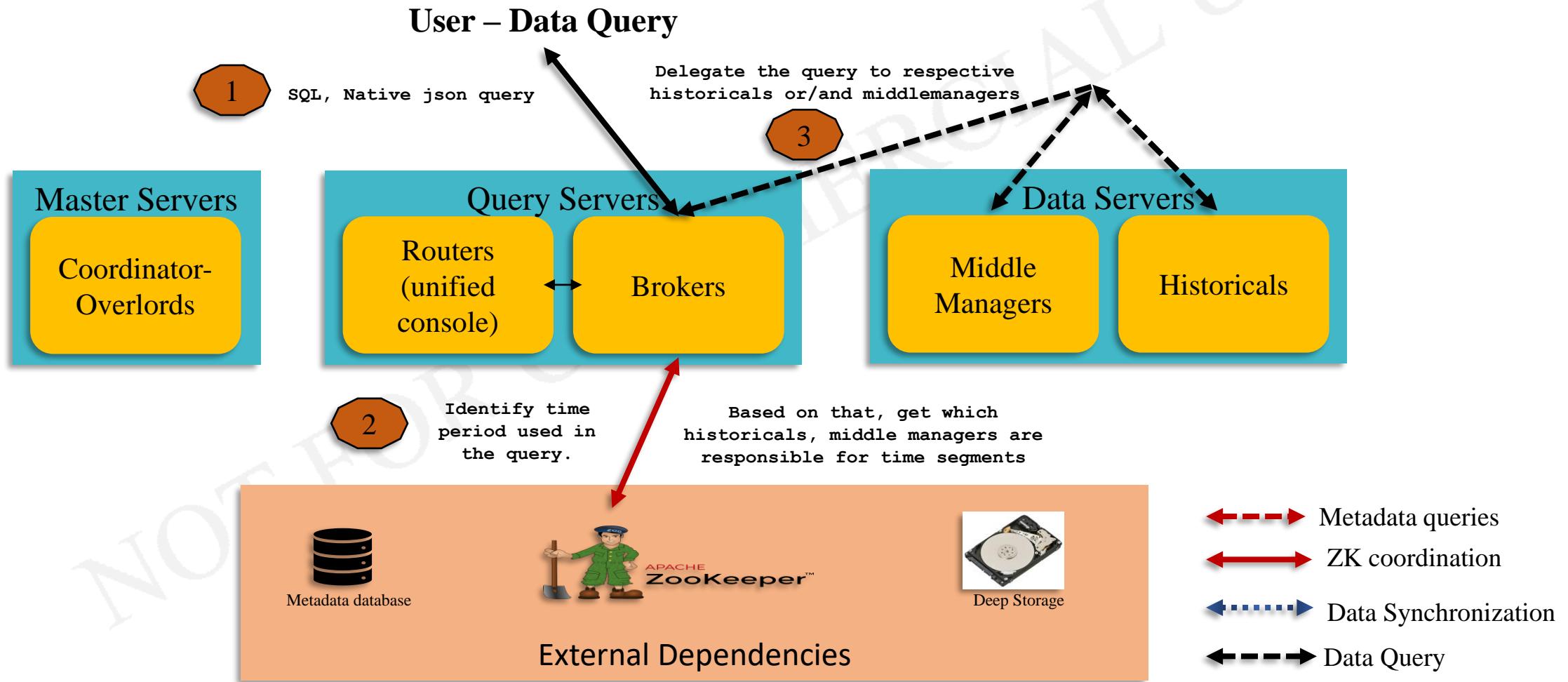
Apache Druid : Ingestion - Workflow



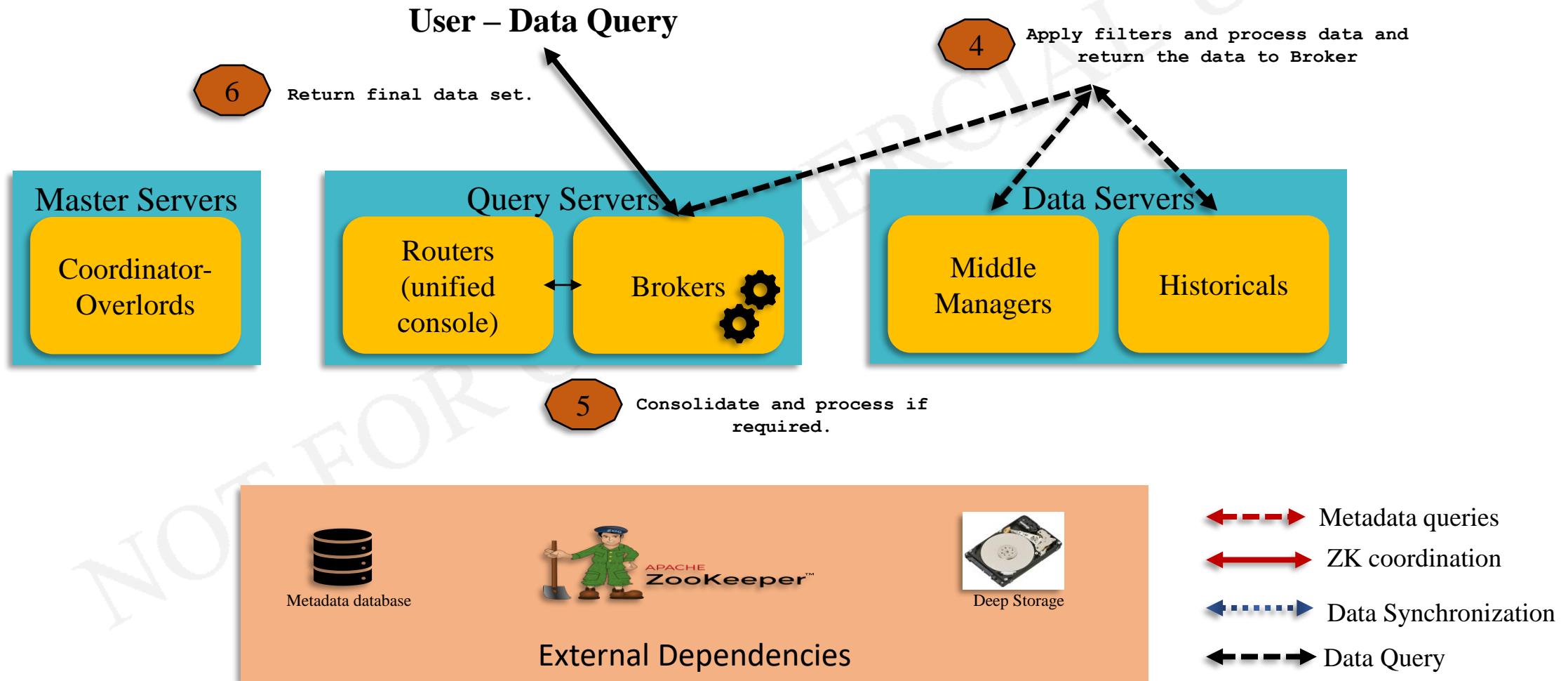
Apache Druid : Query

- Components Involved
 - Broker
 - Zookeeper
 - Historicals
 - Middlemanagers (for realtime data).
- Query is always through Broker.
- Historicals work only on the data/segments they are responsible for.
- Broker collects data from all Historicals and applies further processing logic before returning the final data to client.
- Workflow is same for by Native JSON query and SQL.

Apache Druid : Query - Workflow

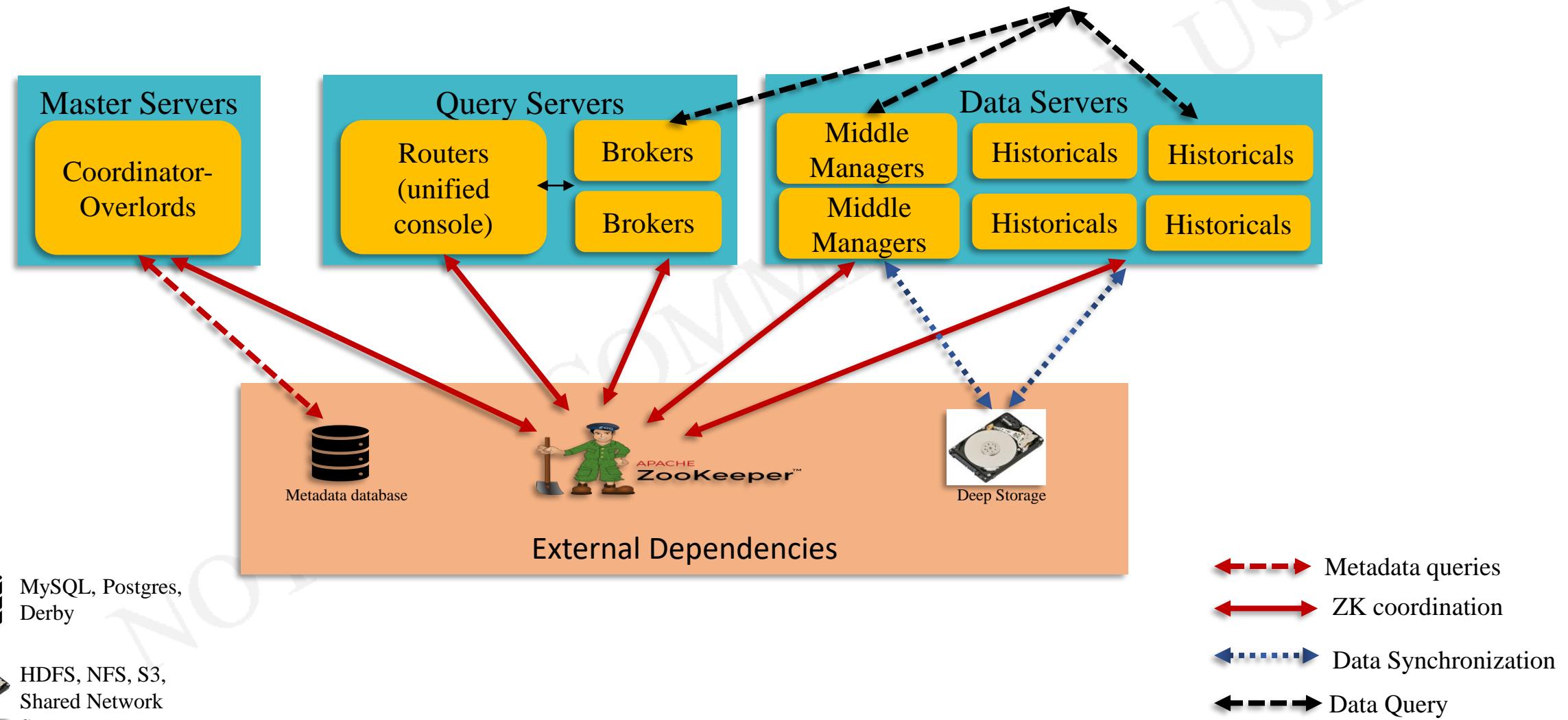


Apache Druid : Query - Workflow



Druid – Typical Cluster Implementation

Apache (Incubating) Druid : Cluster



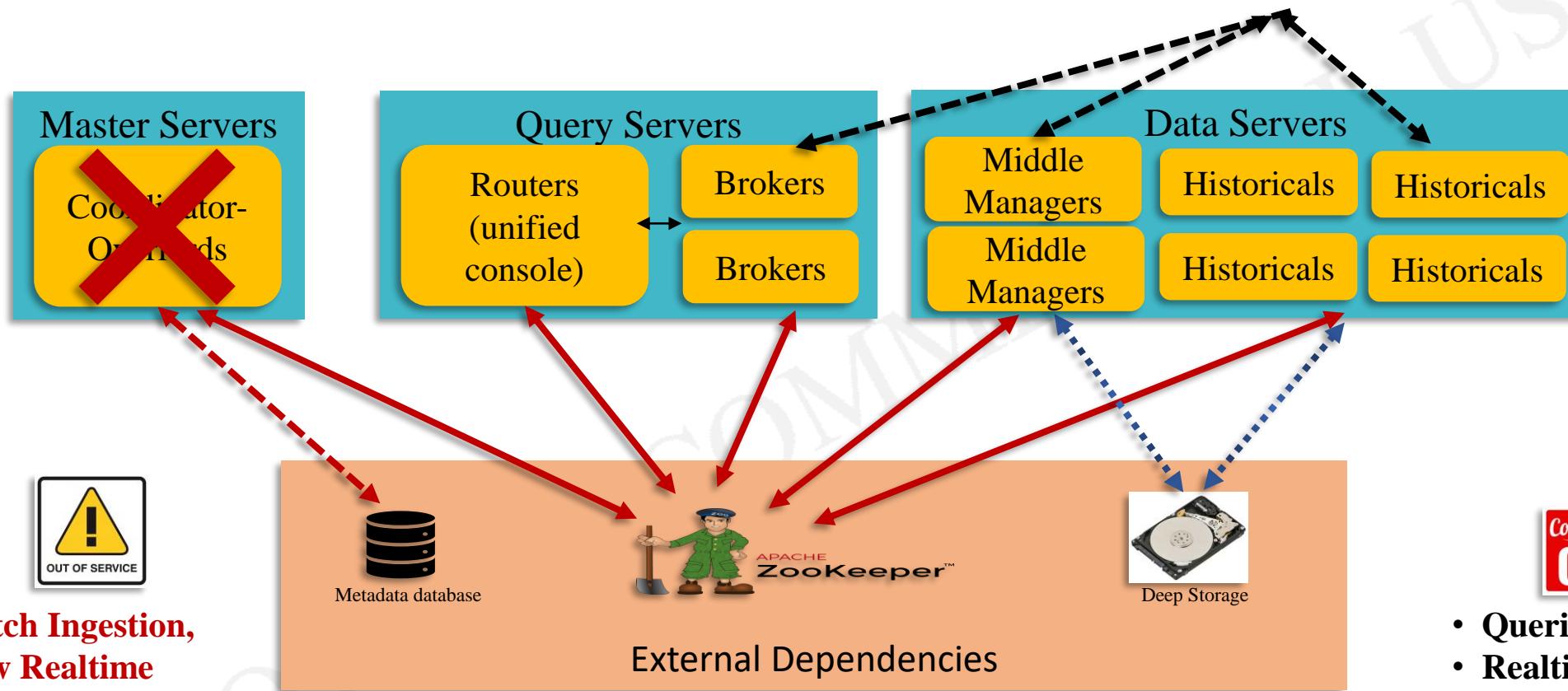
MySQL, Postgres,
Derby



HDFS, NFS, S3,
Shared Network
Storage

What is shared nothing?

Apache Druid : Cluster

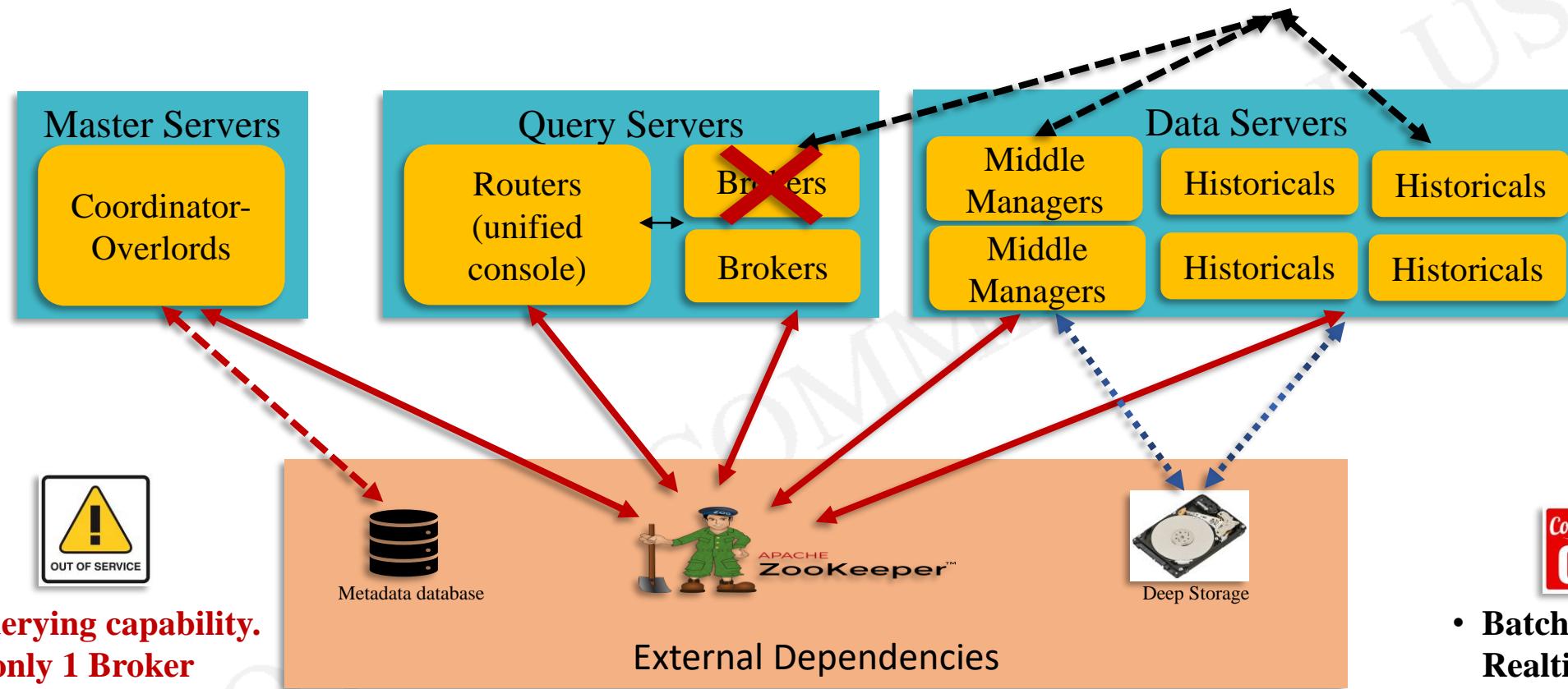


- **Batch Ingestion, new Realtime ingestion**
- **Segment Balancing**
- **Other administration activities**

LNC.ADONI@GMAIL.COM

- **Queries**
- **Realtime Ingestion through Middlemanagers**

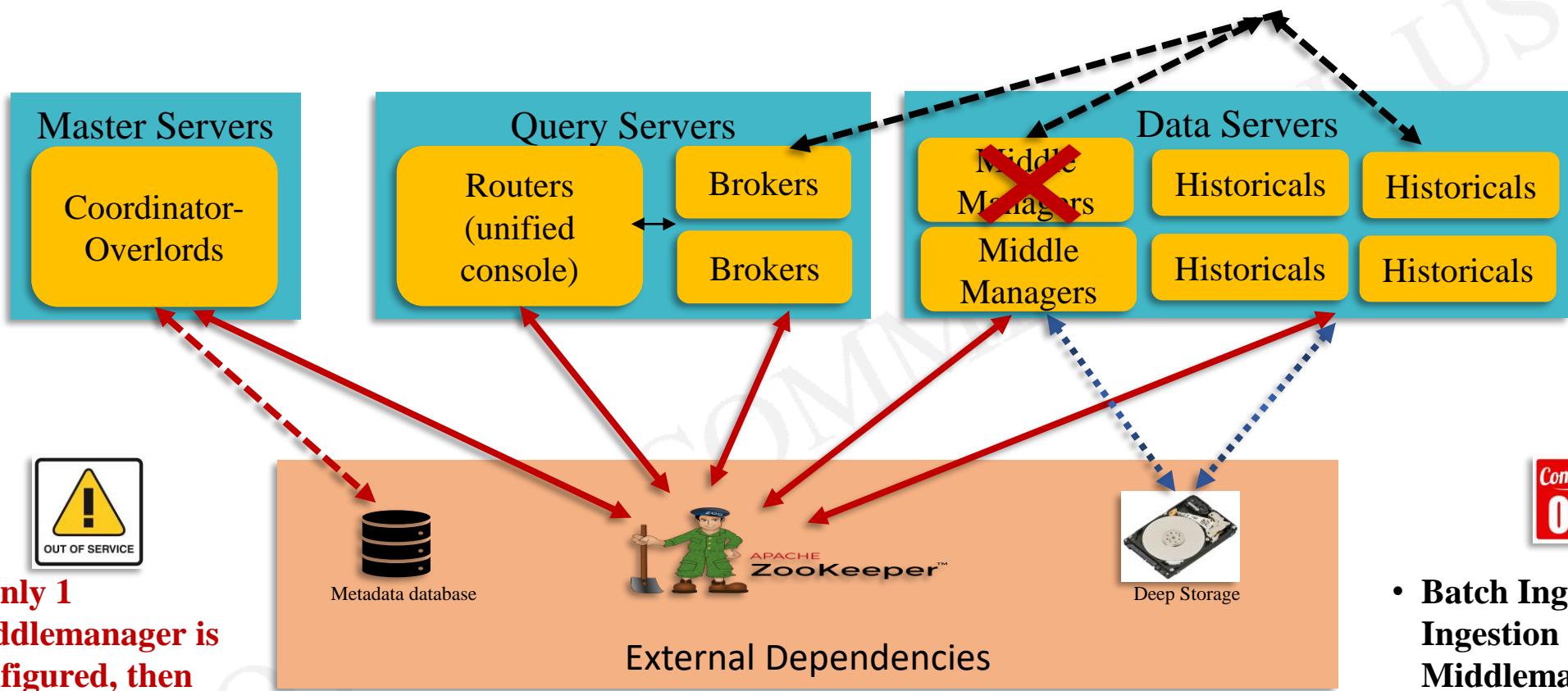
Apache Druid : Cluster



- **Querying capability.**
- **If only 1 Broker exists in the Cluster, then querying capability.**

- **Batch Ingestion, new Realtime Ingestion**
- **Segment Balancing.**
- **Realtime Ingestion through Middlemanagers**

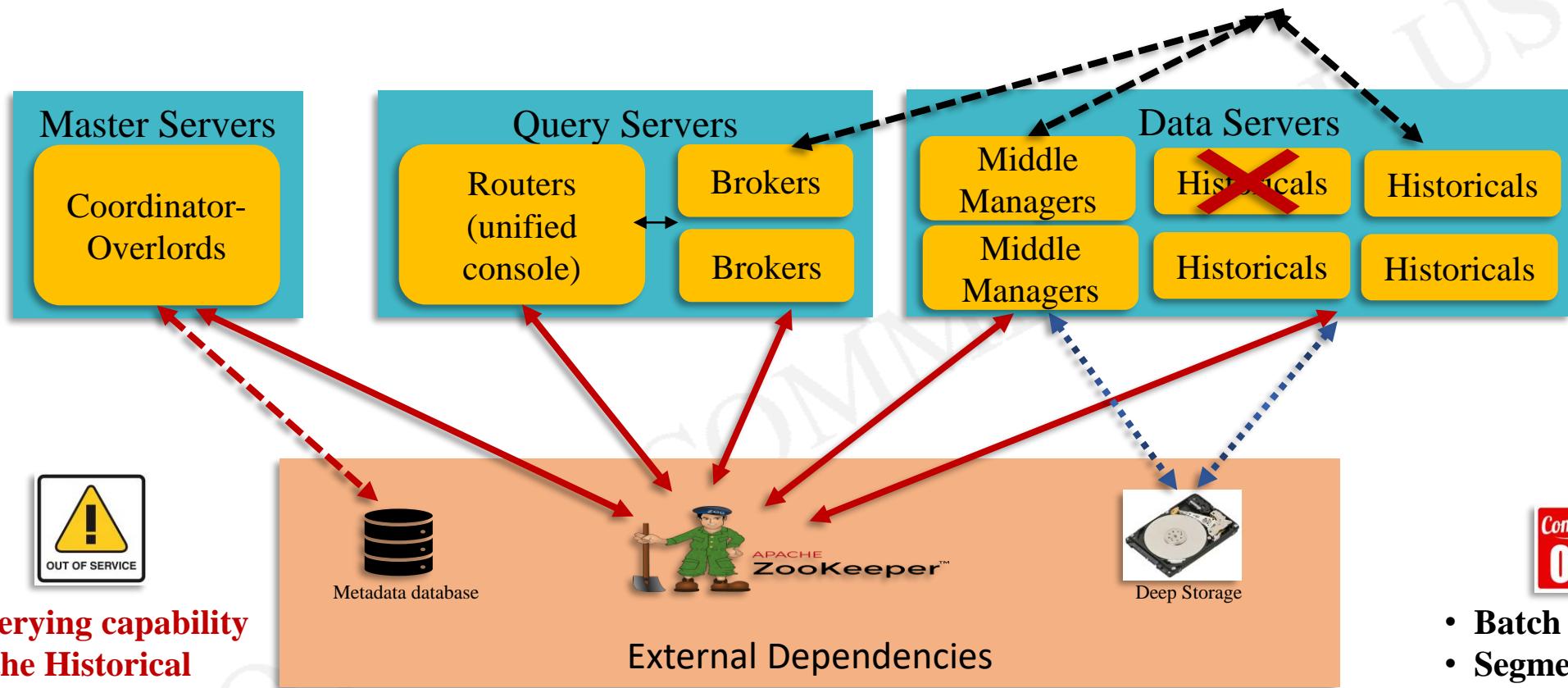
Apache Druid : Cluster



- If only 1 Middlemanager is configured, then batch Ingestion and realtime ingestion.

- **Batch Ingestion, Realtime Ingestion (if atleast 1 Middlemanager is alive)**
 - **Segment Balancing.**
 - **Queries**
 - **Realtime Ingestion through Middlemanagers**

Apache Druid : Cluster



- **Querying capability of the Historical node. If only 1 Historical node, then no queries would be executed.**

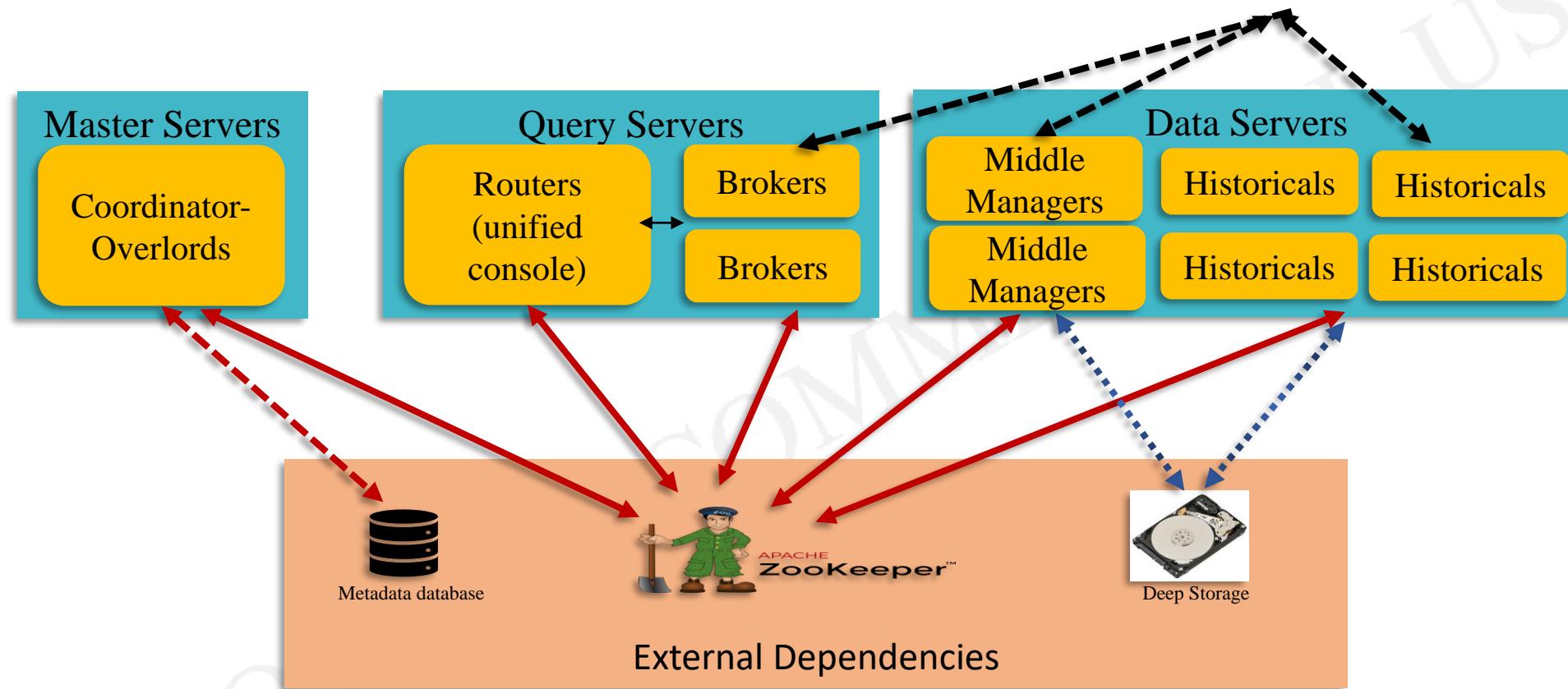
When Historical node goes down, Coordinator-Overlord immediately & automatically assigns the respective segments to other Historicals (auto balancing).

- **Batch Ingestion**
- **Segment Balancing.**
- **Queries**
- **Realtime Ingestion through Middlemanagers**



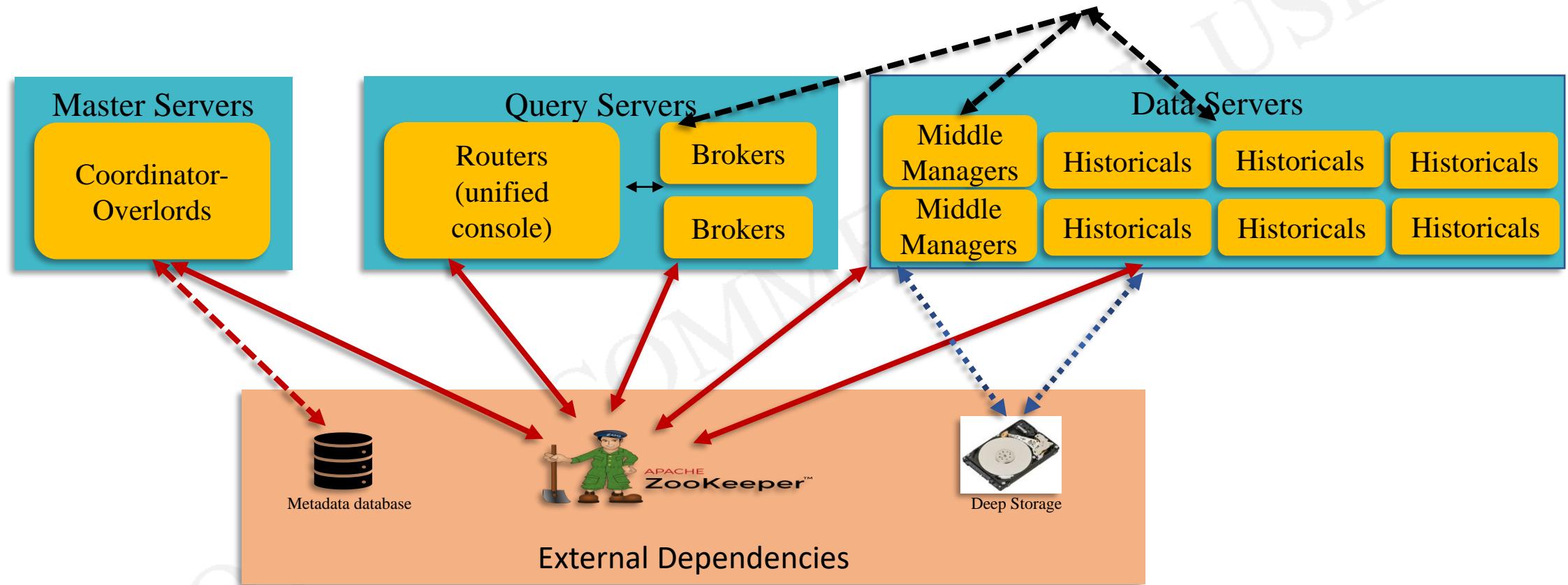
How Druid scale Horizontally?

Apache Druid : Cluster



Existing Historicals are getting overloaded, how to add more Historicals?

Apache Druid : Horizontal Scaling



Existing Historicals are getting overloaded, how to add more Historicals?

Just add another machine, Install Druid and start Historical service. Coordinator-Overlord will take care of everything else. It is that simple. Same for all other components of Druid.

What is Timeseries data store

Timeseries Data Store

- Lets' start with RDBMS

```
create table customer (
  col1 integer,
  col2 float,
  col3 varchar(50)
);
```

What is wrong with this table?

```
create table transactions (
  col11 integer,
  col12 float,
  col13 varchar(50),
  col11 references customer(col1)
);
```

Can we insert data into transactions?

Timeseries data store treats “time” as mandatory data qualifier and gives special privileges to “time” dimension.

Druid - Timeseries Data Store

- Time dimension is mandatory to ingest data into Druid.
 - Any dimension can have timestamp data, but that dimension should be mapped to **_time**.
 - **_time** is reserved keyword in druid. The granularity of **_time** could be anything – Year/Qtr/Month/Day/Hour/Minutes/Seconds
- Druid shards data automatically based on **_time**, called segments.
 - Segment granularity can be defined at the time of ingestion.
 - Each segment is the most granular data store (unlike rows/columns in RDBMS).
 - **Segments are immutable**. Any modifications (delete, add, update) can be done only at segment level.
 - All segments are stored separately in a different folders/directories in deep storage.

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-01-01h

trans_time	pickup	drop	distance	amount
2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2019-01-01 00:27:11	Queens	Manhattan	3	18.20
2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
2019-01-01 00:50:10	Queens	Bronx	4	52.00

```
rollup : false
segmentGranularity: "hour"
queryGranularity: "none"
```

segment-2019-01-02-14h_to_2019-01-02-15h

trans_time	pickup	drop	distance	amount
2019-01-02 14:20:11	Queens	Bronx	4	49.50

segment-2019-01-01-02h_to_2019-01-01-03h

trans_time	pickup	drop	distance	amount
2019-01-01 02:10:25	Queens	Manhattan	3	19.00

segment-2019-01-01-01h_to_2019-01-01-02h

trans_time	pickup	drop	distance	amount
2019-01-01 01:01:56	Manhattan	EWR	6	45.00
2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00

- If **rollup:false**, then no aggregations are performed. Input row count = ingested row count.
- Segments are created based on **segmentGranularity** and ingested data.

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

trans_time	pickup	drop	distance	amount
2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2019-01-01 00:27:11	Queens	Manhattan	3	18.20
2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
2019-01-01 00:50:10	Queens	Bronx	4	52.00
2019-01-01 01:01:56	Manhattan	EWR	6	45.00
2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
2019-01-01 02:10:25	Queens	Manhattan	3	19.00

segment-2019-01-02-00h_to_2019-01-03-00h

trans_time	pickup	drop	distance	amount
2019-01-02 14:20:11	Queens	Bronx	4	49.50

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-00-00h_to_2019-02-01-00h

trans_time	pickup	drop	distance	amount
2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2019-01-01 00:27:11	Queens	Manhattan	3	18.20
2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
2019-01-01 00:50:10	Queens	Bronx	4	52.00
2019-01-01 01:01:56	Manhattan	EWR	6	45.00
2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
2019-01-01 02:10:25	Queens	Manhattan	3	19.00
2019-01-02 14:20:11	Queens	Bronx	4	49.50

```
rollup : false  
segmentGranularity: "month"  
queryGranularity: "none"
```

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-01h

trans_time	pickup	drop	distance	amount	count
2019-01-01 00:00:00	Bronx	Brooklyn	4	53.45	2
2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
2019-01-01 00:00:00	Queens	Bronx	4	52.00	1

rollup : true

segmentGranularity: "hour"

queryGranularity: "hour"

segment-2019-01-02-14h_to_2019-01-02-15h

trans_time	pickup	drop	distance	amount	count
2019-01-02 14:00:00	Queens	Bronx	4	49.50	1

segment-2019-01-01-02h_to_2019-01-01-03h

trans_time	pickup	drop	distance	amount	count
2019-01-01 02:00:00	Queens	Manhattan	3	19.00	1

segment-2019-01-01-01h_to_2019-01-01-02h

trans_time	pickup	drop	distance	amount	count
2019-01-01 01:00:00	Manhattan	EWR	6	45.00	1
2019-01-01 01:00:00	Bronx	Brooklyn	2	26.00	1

- If **rollup:true**, then aggregations would be performed based on unique rows. Input row count \geq ingested row count.

- Notice, granularity of time is hour; minute & seconds information is lost.

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

_time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

trans_time	pickup	drop	distance	amount	count
2019-01-01 00:00:00	Bronx	Brooklyn	4	53.45	2
2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
2019-01-01 01:00:00	Manhattan	EWR	6	45.00	1
2019-01-01 01:00:00	Bronx	Brooklyn	2	26.00	1
2019-01-01 02:00:00	Queens	Manhattan	3	19.00	1

LNC.ADONI@GMAIL.COM

rollup : true
segmentGranularity: "day"
queryGranularity: "hour"

segment-2019-01-02-00h_to_2019-01-03-00h

trans_time	pickup	drop	distance	amount	count
2019-01-02 14:00:00	Queens	Bronx	4	49.50	1

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

_time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

trans_time	pickup	drop	distance	amount	count
2019-01-01 00:00:00	Bronx	Brooklyn	6	79.45	3
2019-01-01 00:00:00	Queens	Manhattan	6	37.20	2
2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1

rollup : true
segmentGranularity: "day"
queryGranularity: "day"

segment-2019-01-02-00h_to_2019-01-03-00h

trans_time	pickup	drop	distance	amount	count
2019-01-02 00:00:00	Queens	Bronx	4	49.50	1



But wait...I want **id** also...

Druid – Segment

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

rollup : true
segmentGranularity: "day"
queryGranularity: "day"

segment-2019-01-02-00h_to_2019-01-03-00h

id	trans_time	pickup	drop	distance	amount	count
8	2019-01-02 00:00:00	Queens	Bronx	4	49.50	1

How to decide what? - segmentGranularity

- Per day data volume is 50 million records.
 - We need to store 3 years of data.
 - Time granularity is yyyy-MM-dd hh:mm:ss.ms

What should be ideal Segment Granularity?

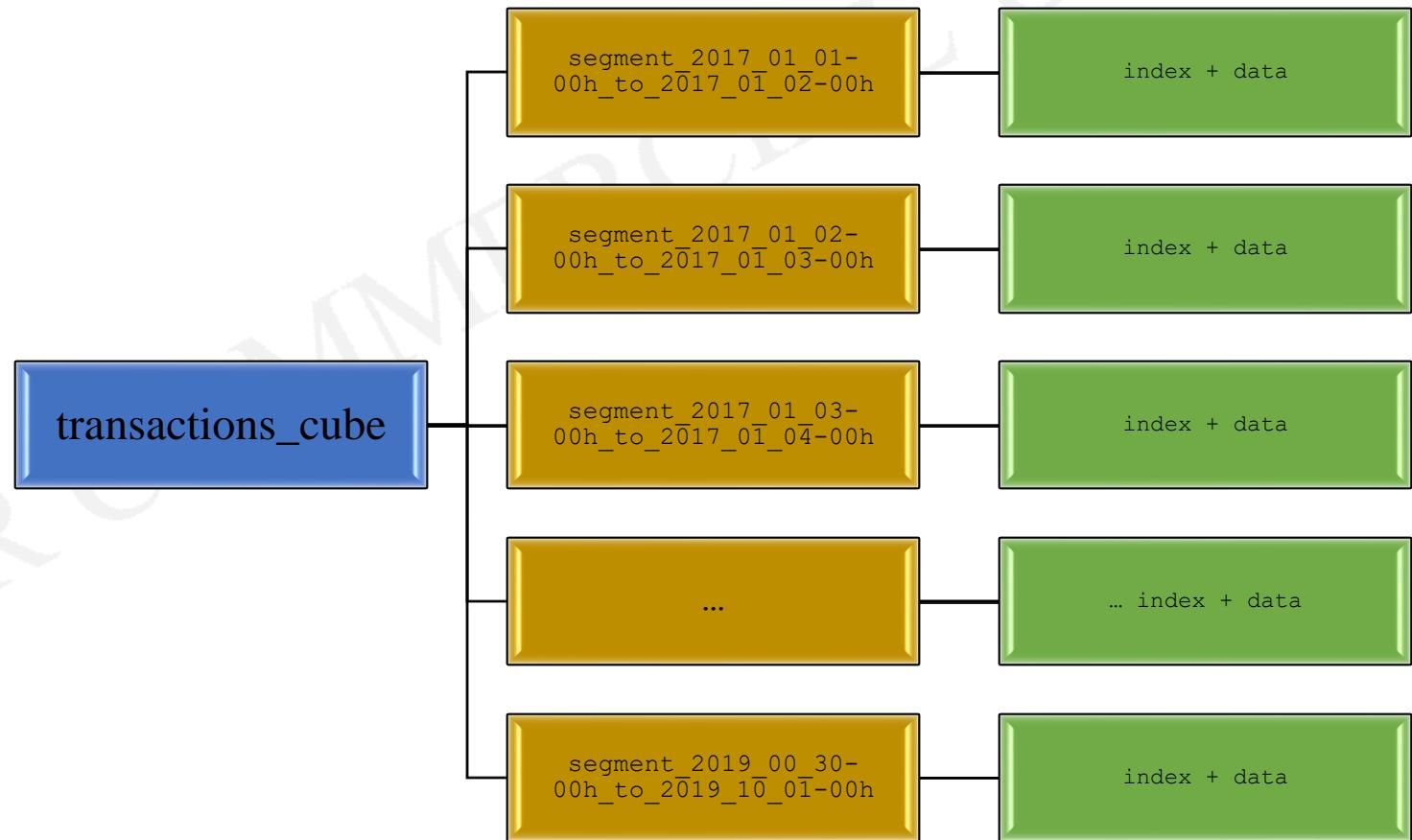
- Total Rows: 50 million/day *30 * 12 *3 ~ **54 billion**
 - Total Hours: 24 * 30 * 12 * 3 = **25,920 hrs**
 - Total Days: **1080 days**
 - Total Months: **36 months**

segmentGranularity	Pros	Cons
Hour	<ul style="list-style-type: none"> Good for real-time ingestion. 	<ul style="list-style-type: none"> Too many segments. Each segment to have only ~2million rows.
Day	<ul style="list-style-type: none"> Manageable number of segments ~50 million rows in each segment 	<ul style="list-style-type: none"> Too many rows in a segment. Segment needs to be further partitioned by other dimension.
Month	<ul style="list-style-type: none"> Daily data ingestion would be a challenge, as segments are immutable. 	<ul style="list-style-type: none"> Too few segments. Each segment to contain more than 1.5 billion rows.

segmentGranularity example

- Per day data volume is 50 million records.
- We need to store 3 years of data.
- Time granularity is yyyy-MM-dd hh:mm:ss.ms

**Assuming
segmentGranularity='day'**

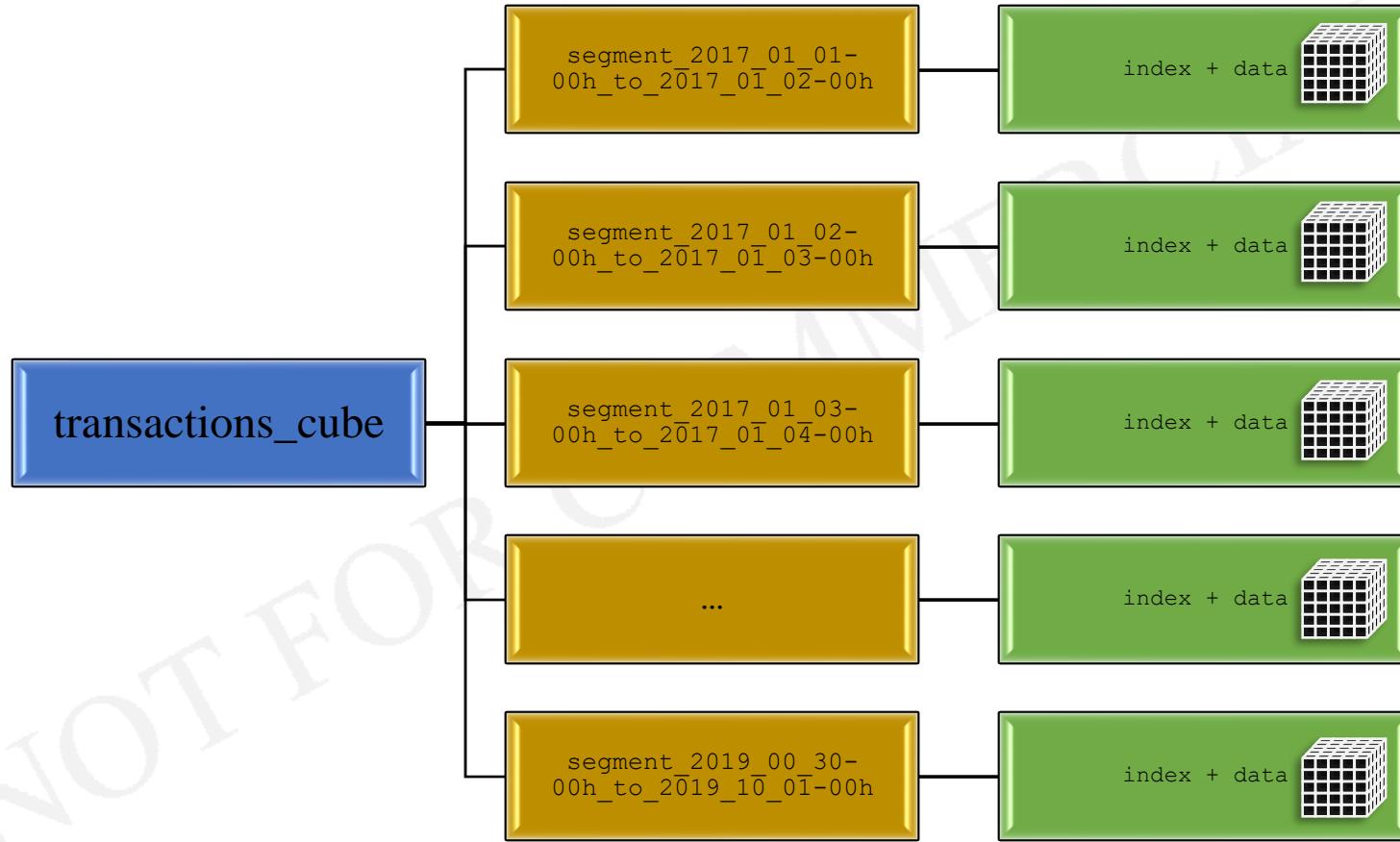


How to decide what? - queryGranularity

- Query granularity is determined by business use cases.
- If the business requirement is to analyze data at hour, then the data should be aggregated to hour.
- Note that at query time, data from lower granularity to higher granularity is possible.
 - If the data is stored as hour, then during query the data can be aggregated to Day or Month...

Fine... but where is the **Cube**?

Where is the Cube?



What is the Definition and Storage of Cube?

- Ingestion task also termed as ‘indexing’ task in Druid.
- It involves following activities
 - Segment rows based on segmentGranularity
 - Dictionary encoding of dimensions in each segment
 - Mapping of column values to dictionary encoded data
 - Compute inverse indexes
 - Dictionary encoding, inverse indexes are computed for only string dimensions
 - Non-string dimensions are left as is.
 - Metrics are stored in a separate data structure i.e indexes are not computed on metric columns

Ingestion: Indexing Task

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

rollup : true
segmentGranularity: "day"
queryGranularity: "day"

segment-2019-01-02-00h_to_2019-01-03-00h

id	trans_time	pickup	drop	distance	amount	count
8	2019-01-02 00:00:00	Queens	Bronx	4	49.50	1

Dictionary Encoding and Mapping

Ingestion: Indexing Task – Dictionary Encoding

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

Dimensions

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{  
    "Bronx" : 0,  
    "Queens" : 1,  
    "Manhattan":2  
}
```

Dimension: drop

```
{  
    "Brooklyn" : 0,  
    "Manhattan" : 1,  
    "Bronx" : 2,  
    "EWR" : 3  
}
```

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{  
    "Queens" :0  
}
```

Dimension: drop

```
{  
    "Bronx" : 0  
}
```

segment-2019-01-02-00h_to_2019-01-03-00h

id	trans_time	pickup	drop	distance	amount	count
8	2019-01-02 00:00:00	Queens	Bronx	4	49.50	1

Dimensions

Ingestion: Indexing Task – Dictionary Encoding, Mapping

segment-2019-01-01-00h_to_2019-01-02-

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

Dimensions

segment-2019-01-02-00h_to_2019-01-03-00h

id	trans_time	pickup	drop	distance	amount	count
8	2019-01-02 00:00:00	Queens	Bronx	4	49.50	1

Dimensions

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{
    "Bronx" : 0,
    "Queens" : 1,
    "Manhattan":2
}
```

Dimension: drop

```
{
    "Brooklyn" : 0,
    "Manhattan" : 1,
    "Bronx" : 2,
    "EWR" : 3
}
```

Column Values Mapping:

Dimension: pickup

```
[
    0,
    1,
    0,
    1,
    2,
    0,
    1 ]
```

Dimension: drop

```
[
    0,
    1,
    0,
    2,
    3,
    0,
    1 ]
```

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{
    "Queens" : 0
}
```

Dimension: drop

```
{
    "Bronx" : 0
}
```

Inverse Indexes

Ingestion: Indexing Task – Dictionary Encoding, Mapping, Inverse Indexes

segment-2019-01-01-00h_to_2019-01-

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

Dimensions

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{
    "Bronx" : 0,
    "Queens" : 1,
    "Manhattan":2
}
```

Dimension: drop

```
{
    "Brooklyn" : 0,
    "Manhattan" : 1,
    "Bronx" : 2,
    "EWR" : 3
}
```

Column Values Mapping:

Dimension: pickup

```
[
    0,
    1,
    0,
    1,
    2,
    0,
    1
]
```

Dimension: drop

```
[
    0,
    1,
    0,
    1,
    0,
    2,
    3,
    0,
    1
]
```

Inverse Indexes:

Dimension: pickup

"Bronx" = [1,0,1,0,0,1,0]
 "Queens" = [0,1,0,1,0,0,1]
 "Manhattan" = [0,0,0,0,1,0,0]

Dimension: drop

"Brooklyn" = [1,0,1,0,0,1,0]
 "Manhattan" = [0,1,0,0,0,0,1]
 "Bronx" = [0,0,0,1,0,0,0]
 "EWR" = [0,0,0,0,1,0,0]

segment-2019-01-02-00h_to_2019-01-03-00h

id	trans_time	pickup	drop	distance	amount	count
8	2019-01-02 00:00:00	Queens	Bronx	4	49.50	1

Dimensions

Dictionary Encoding:

Dimension: id

<no dictionary encoding as it is non-string>

Dimension: pickup

```
{
    "Queens" : 0
}
```

Column Values Mapping:

Dimension: pickup

```
[ 0 ]
```

Dimension: pickup

"Queens" = [1]

Dimension: drop

```
{
    "Bronx" : 0
}
```

Dimension: drop

```
[ 0 ]
```

Dimension: drop

"Bronx" = [1]

What is the use of Inverse indexes?

Usage – Dictionary Encoding, Mapping, Inverse Indexes

segment-2019-01-01-00h_to_2019-01-

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

Dimensions

Dictionary Encoding:

Dimension: id
<no dictionary encoding as it is non-string>

Dimension: pickup
{
 "Bronx" : 0,
 "Queens" : 1,
 "Manhattan":2
}

Dimension: drop
{
 "Brooklyn" : 0,
 "Manhattan" : 1,
 "Bronx" : 2,
 "EWR" : 3
}

Column Values Mapping:

Dimension: pickup
[
 0,
 1,
 0,
 1,
 2,
 0,
 1
]

Dimension: drop
[
 0,
 1,
 0,
 2,
 3,
 0,
 1
]

Inverse Indexes:

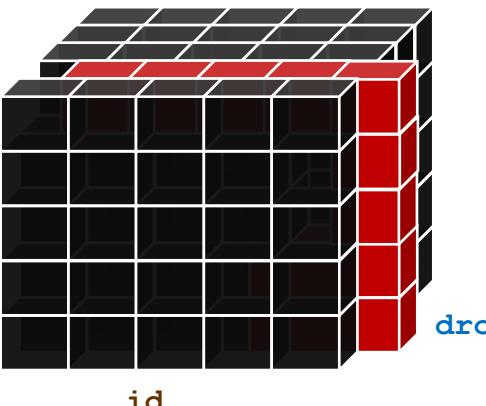
Dimension: pickup
"Bronx" = [1,0,1,0,0,1,0]
"Queens" = [0,1,0,1,0,0,1]
"Manhattan" = [0,0,0,1,0,0]

Dimension: drop
"Brooklyn" = [1,0,1,0,0,1,0]
"Manhattan" = [0,1,0,0,0,0,1]
"Bronx" = [0,0,0,1,0,0,0]
"EWR" = [0,0,0,0,1,0,0]

Select * from where pickup='Queens'

Filtered rows, Metrics: [3,4,3], [18.20,52.00,19.00], [1,1,1]

pickup



Ingestion: Indexing Task – Dictionary Encoding, Mapping, Inverse Indexes

segment-2019-01-01-00h_to_2019-01-

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

Dimensions

Dictionary Encoding:

Dimension: **id**
<no dictionary encoding as it is non-string>

Dimension: **pickup**
{
 "Bronx" : 0,
 "Queens" : 1,
 "Manhattan":2
}

Dimension: **drop**
{
 "Brooklyn" : 0,
 "Manhattan" : 1,
 "Bronx" : 2,
 "EWR" : 3
}

Column Values Mapping:

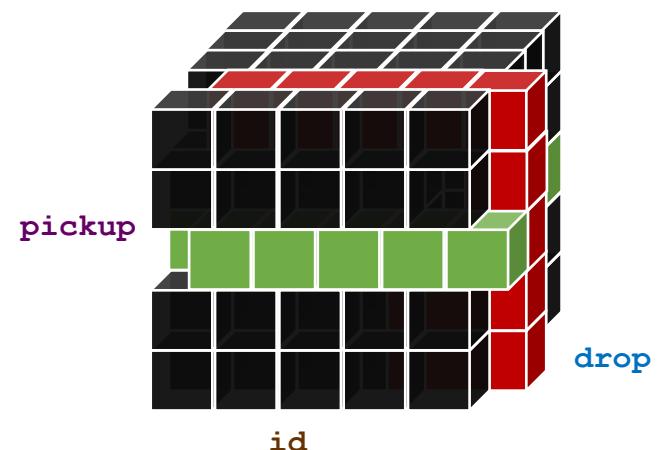
Dimension: **pickup**
[
 0,
 1,
 0,
 1,
 2,
 0,
 1
]

Dimension: **drop**
[
 0,
 1,
 0,
 2,
 3,
 0,
 1
]

Inverse Indexes:

Dimension: **pickup**
"Bronx" = [1,0,1,0,0,1,0]
"Queens" = [0,1,0,1,0,0,1]
"Manhattan" = [0,0,0,1,0,0]

Dimension: **drop**
"Brooklyn" = [1,0,1,0,0,1,0]
"Manhattan" = [0,1,0,0,0,0,1]
"Bronx" = [0,0,0,1,0,0,0]
"EWR" = [0,0,0,0,1,0,0]



Select * from where pickup='Queens' and drop='Bronx'

("Queens" = [0,1,0,1,0,0,1]) & ("Bronx" = [0,0,0,1,0,0,0]) => [0,0,0,1,0,0,0]

Filtered rows, Metrics (distance, amount, count): [4], [52.00], [1]

What about evolving schema?

Evolving Schema

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

_time

Dimensions

Metrics

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

LNC.ADONI@GMAIL.COM

id	trans_time	pickup	drop	vehicle_type	passenger_count	distance	amount
10	2019-06-01 13:20:16	EWR	Queens	Car	2	2	24.95
21	2019-06-02 16:30:11	Queens	Bronx	Van	4	3	18.20

time

Dimensions

Metrics

segment-2019-06-01-00h_to_2019-06-02-00h

i	trans_t	pic	dro	vehi	dis	amou	cou	passen
d	ime	kup	p	cle_	tanc	nt	nt	ger_c
1	2019-06-01 00:00:00	EWR	Queens	Car	2	24.95	1	2

segment-2019-06-02-00h_to_2019-06-03-00h

i	trans_t	pic	dro	vehi	dis	amou	cou	passen
d	ime	kup	p	cle_	tanc	nt	nt	ger_c
2	2019-06-02 00:00:00	Queens	Bronx	Van	4	18.20	1	4

How to correct already indexed
(ingested) data?

Data Correction

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

_time

Dimensions

Metrics

"**appendToExisting**": false

id	trans_time	pickup	drop	distance	amount
2	2019-01-01 00:27:11	Queens	Manhattan	10	100.00
4	2019-01-01 00:50:10	Queens	Bronx	10	100.00

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20
3	2019-01-01 00:00:00	Bronx	Bronx	2	28.50
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00

segment-2019-01-01-00h_to_2019-01-02-00h-v1

id	trans_time	pickup	drop	distance	amount	count
2	2019-01-01 00:00:00	Queens	Manhattan	10	100.00	1
4	2019-01-01 00:00:00	Queens	Bronx	10	100.00	1

Data is lost for rest of the records.

Data Correction

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

_time

Dimensions

Metrics

"appendToExisting": true

id	trans_time	pickup	drop	distance	amount
2	2019-01-01 00:27:11	Queens	Manhattan	10	100.00
4	2019-01-01 00:50:10	Queens	Bronx	10	100.00

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

segment-2019-01-01-00h_to_2019-01-02-00h-v1

id	trans_time	pickup	drop	distance	amount	count
2	2019-01-01 00:00:00	Queens	Manhattan	10	100.00	1
4	2019-01-01 00:00:00	Queens	Bronx	10	100.00	1

Duplicate records for id # 2, 4.
How to correct already ingested data?

Data Correction

"appendToExisting": false

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	10	100.00
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	10	100.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time

Dimensions

Metrics

segment-2019-01-01-00h_to_2019-01-02-00h-v1

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:00:00	Queens	Manhattan	3	18.20
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:00:00	Queens	Bronx	4	52.00
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00

id	trans_time	pickup	drop	distance	amount	count
1	2019-01-01 00:00:00	Bronx	Brooklyn	2	24.95	1
2	2019-01-01 00:00:00	Queens	Manhattan	10	100	1
3	2019-01-01 00:00:00	Bronx	Brooklyn	2	28.50	1
4	2019-01-01 00:00:00	Queens	Bronx	10	100.00	1
5	2019-01-01 00:00:00	Manhattan	EWR	6	45.00	1
6	2019-01-01 00:00:00	Bronx	Brooklyn	2	26.00	1
7	2019-01-01 00:00:00	Queens	Manhattan	3	19.00	1

What happens to old version?

Data Ingestion (Indexing)

- Batch Ingestion
 - Native Ingestion
 - Native Parallel Ingestion
 - Hadoop based ingestion (MapReduce)
- Realtime Ingestion (not yet supported in Data Loader/Unified Console)
 - Kafka
 - Amazon Kinesis
 - http stream

Data Source Used

The screenshot shows a web browser window displaying the NYC Taxi & Limousine Commission website. The URL in the address bar is www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. The page title is "TLC Trip Record Data". On the left sidebar, there is a navigation menu under the "Data" category, which includes "Pilot Programs", "Industry Reports", "Factbook", and "Request Data". Below the sidebar, there are social sharing icons for Facebook, Twitter, and Email, followed by a "Print" button. The main content area features a large heading "TLC Trip Record Data" and two yellow buttons: "Expand All" and "Collapse All". A detailed description follows, explaining that the yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. It notes that the data was collected and provided by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The trip data was not created by the TLC, and the TLC makes no representations as to the accuracy of these data. Another paragraph describes the For-Hire Vehicle (FHV) trip records, mentioning they capture dispatching base license number, pick-up date, time, and taxi zone location ID. These records are generated from FHV Trip Record submissions made by bases. A note states that the TLC publishes base trip record data as submitted by the bases, and they cannot guarantee or confirm their accuracy or completeness. The TLC performs routine reviews of the records and takes enforcement actions when necessary to ensure complete and accurate information. At the bottom, there are three dark grey buttons with white text labeled "2019", "2018", and "2017".

Sample Data

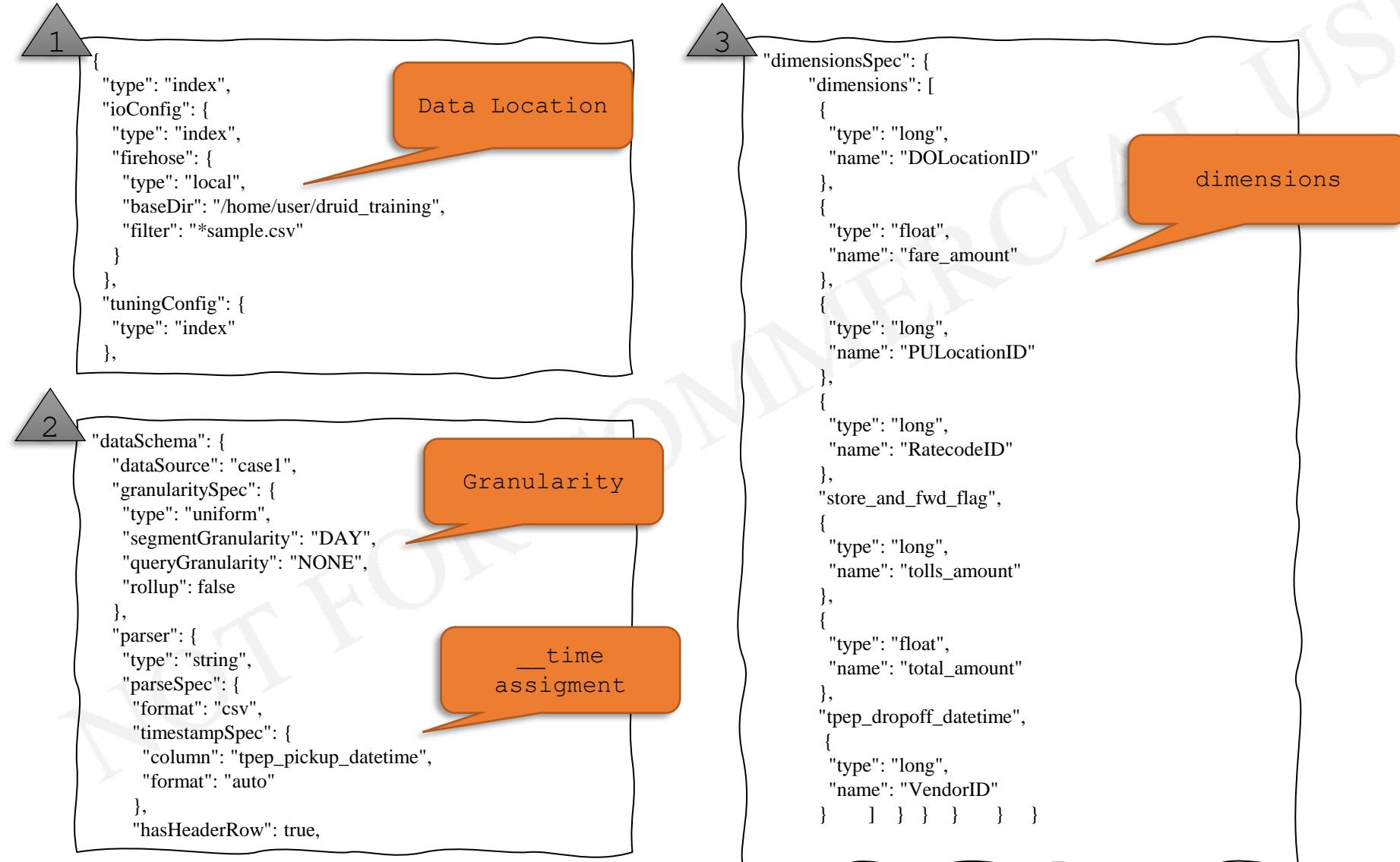
```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,↓  
payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount,congestion_surcharge↓  
↓  
2,2019-01-01 19:09:10,2019-01-01 19:16:42,5,1.82,1,N,238,74,1,8.5,0,0.5,2.32,0,0.3,11.62,↓  
2,2019-01-01 19:24:31,2019-01-01 19:31:01,1,1.21,1,N,237,163,1,7,0,0.5,2.34,0,0.3,10.14,↓  
2,2019-01-01 19:39:05,2019-01-01 19:49:13,5,2.38,1,N,141,238,2,10,0,0.5,0,0,0.3,10.8,↓  
1,2019-01-02 20:24:15,2019-01-02 20:27:53,1,.80,1,N,263,141,1,4.5,0.5,0.5,1.15,0,0.3,6.95,↓  
1,2019-01-02 20:47:39,2019-01-02 20:59:04,1,2.80,1,N,161,263,1,10.5,0.5,0.5,1.5,0,0.3,13.3,↓  
4,2019-01-02 20:29:20,2019-01-02 20:43:49,1,8.74,1,N,138,148,1,25,0.5,0.5,5,0,0.3,31.3,↓  
2,2019-01-02 20:33:28,2019-01-02 20:42:10,1,1.31,1,N,229,234,2,7.5,0.5,0.5,0,0,0.3,8.8,↓  
1,2019-01-03 15:49:53,2019-01-03 15:58:41,1,1.10,1,N,142,237,2,7,0,0.5,0,0,0.3,7.8,↓  
2,2019-01-03 14:55:21,2019-01-03 15:19:44,3,2.93,1,N,244,42,1,17.5,0,0.5,0,0,0.3,18.3,↓  
1,2019-01-03 15:43:44,2019-01-03 15:49:08,1,1.10,1,N,137,229,1,6,0,0.5,1.35,0,0.3,8.15,←
```

What is required for Ingestion Spec

- Source
 - Which path? What files?
- Source Type
 - csv, tsv, json
- Parse Spec
- Filter
- Transformation
- Dimensions
- Metrics
- Segment Granularity, Query Granularity

Batch Ingestion – Native Indexing

Native Ingestion – No Rollup – case1



Native Ingestion – No Rollup – case1

The screenshot shows the Druid Data Sources interface. At the top, there are navigation links: Load data, Datasources (which is selected), Segments, Tasks, Servers, and Query. On the right side of the header are Legacy, Settings, and Help icons. Below the header, the title "Datasources" is displayed, along with "Refresh" and "Actions" buttons. There are two toggle switches: "Show segment timeline" and "Show disabled". A "Columns" dropdown menu is also present. The main table has columns: Datasource, Availability, Segment load/drop, Retention, Replicated size, Size, Compaction, Avg. segment..., Num rows, and Actions. The "case1" row is listed, showing it is "Fully available (3 segments)" with "No segments to load/drop". The retention is set to "Cluster default: loadForever". The replicated size is 12.61 KB, and the size is 12.61 KB. The compaction is set to "None". The average segment size is 4.20 KB, and there are 10 rows. The "Actions" column contains a magnifying glass icon and a wrench icon.

The screenshot shows the Druid Segments interface. At the top, there are navigation links: Load data, Datasources, Segments (selected), Tasks, Servers, and Query. On the right side of the header are Legacy, Settings, and Help icons. Below the header, the title "Segments" is displayed, along with "Refresh" and "Actions" buttons. There are three buttons for grouping: "None" (selected), "Interval", and "...". A "Columns" dropdown menu is also present. The main table has columns: Segment ID, Datasource, Start, End, Version, Partition, Size, Num rows, Replicas, Is published, Is realtime, Is available, Is overshado..., and Actions. The table lists three segments for the "case1" datasource. The first segment starts at 2019-01-03T00:00:00.000Z and ends at 2019-01-04T00:00:00.000Z, with a version of 2019-10-03T09:0... and a partition of 0. It has a size of 4.17 KB, 3 num rows, 1 replicas, and is true for published, realtime, available, and overshadowed. The second segment starts at 2019-01-02T00:00:00.000Z and ends at 2019-01-03T00:00:00.000Z, with a version of 2019-10-03T09:0... and a partition of 0. It has a size of 4.26 KB, 4 num rows, 1 replicas, and is true for published, realtime, available, and overshadowed. The third segment starts at 2019-01-01T00:00:00.000Z and ends at 2019-01-02T00:00:00.000Z, with a version of 2019-10-03T09:0... and a partition of 0. It has a size of 4.18 KB, 3 num rows, 1 replicas, and is true for published, realtime, available, and overshadowed. The "Actions" column for each row contains a magnifying glass icon and a wrench icon.

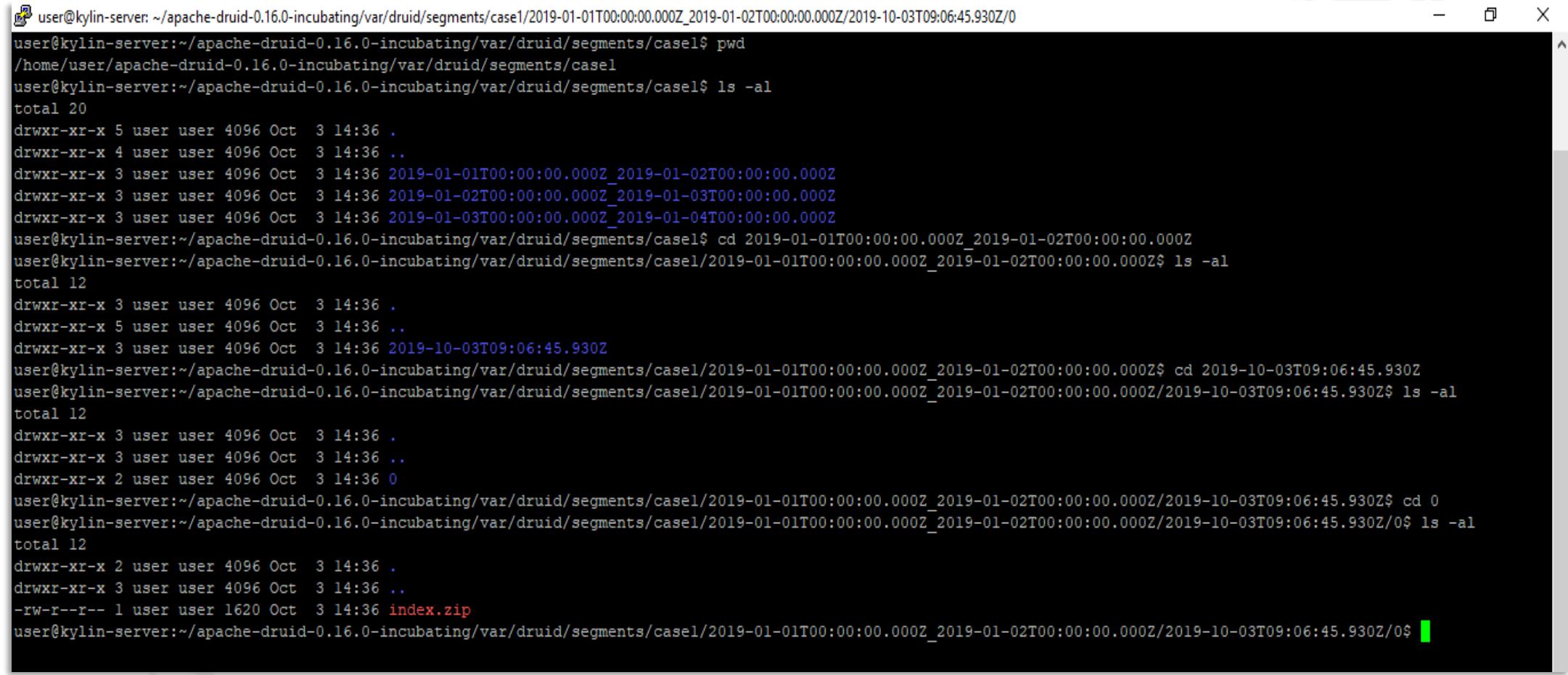
Native Ingestion – No Rollup – case1

The screenshot shows the Druid UI interface. The top navigation bar has tabs for Load data, Datasources, Segments, Tasks, Servers, and Query. The Query tab is selected. Below the navigation is a search bar containing the query: `1 select * from case1`. The left sidebar shows a tree structure of data sources and tables. Under the 'druid' source, the 'case1' table is expanded, showing columns: _time, DOLocationID, extra, fare_amount, improvement_surcharge, mta_tax, passenger_count, payment_type, PUlocationID, RatecodeID, store_and_fwd_flag, tip_amount, tolls_amount, total_amount, tpep_dropoff_datetime, trip_distance, and VendorID. The main panel displays the query results in a table format:

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
2019	1	1	25.5	3
2019	1	2	47.5	4
2019	1	3	30.5	3

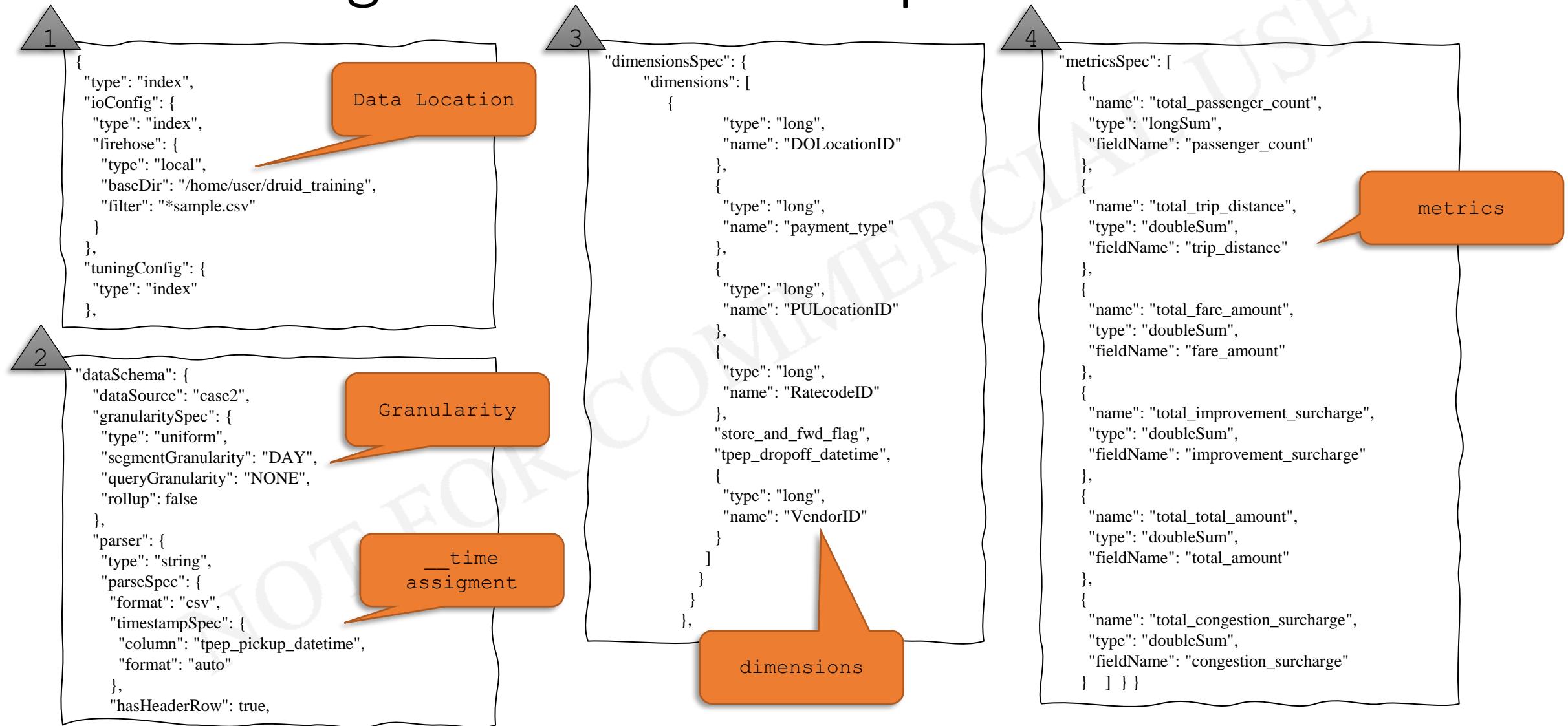
Below the table are buttons for Run, Auto run, and Smart query limit. A status message indicates "3 results in 0.48s".

Native Ingestion – No Rollup – case1



```
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:06:45.930Z/0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1$ pwd
/home/user/apache-druid-0.16.0-incubating/var/druid/segments/case1
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1$ ls -al
total 20
drwxr-xr-x 5 user user 4096 Oct  3 14:36 .
drwxr-xr-x 4 user user 4096 Oct  3 14:36 ..
drwxr-xr-x 3 user user 4096 Oct  3 14:36 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 14:36 2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 14:36 2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1$ cd 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 14:36 .
drwxr-xr-x 5 user user 4096 Oct  3 14:36 ..
drwxr-xr-x 3 user user 4096 Oct  3 14:36 2019-10-03T09:06:45.930Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ cd 2019-10-03T09:06:45.930Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:06:45.930Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 14:36 .
drwxr-xr-x 3 user user 4096 Oct  3 14:36 ..
drwxr-xr-x 2 user user 4096 Oct  3 14:36 0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:06:45.930Z$ cd 0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:06:45.930Z/0$ ls -al
total 12
drwxr-xr-x 2 user user 4096 Oct  3 14:36 .
drwxr-xr-x 3 user user 4096 Oct  3 14:36 ..
-rw-r--r-- 1 user user 1620 Oct  3 14:36 index.zip
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case1/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:06:45.930Z/0$
```

Native Ingestion – No Rollup – case2



Native Ingestion – No Rollup – case2

The screenshot shows the Druid UI's Datasources page. The top navigation bar includes tabs for Load data, Datasources (which is selected), Segments, Tasks, Servers, and Query. On the right side of the header are links for Legacy, Settings, and Help. Below the header, there are buttons for Refresh, Show segment timeline, and Show disabled. A "Columns" dropdown menu is also present. The main table has columns for Datasource, Availability, Segment load/drop, Retention, Replicated size, Size, Compaction, Avg. segment..., Num rows, and Actions. Two data sources are listed: "case1" and "case2". Both entries show "Fully available (3 segments)" under Availability and "No segments to load/drop" under Segment load/drop. The Retention column indicates "Cluster default: loadForever". The Replicated size and Size columns show values like 12.61 KB and 16.27 KB. The Compaction column shows "None". The Avg. segment... and Num rows columns show 4.20 KB and 10 respectively. The Actions column contains icons for viewing and editing.

Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
case1	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	12.61 KB	12.61 KB	None	4.20 KB	10	
case2	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	16.27 KB	16.27 KB	None	5.42 KB	10	

The screenshot shows the Druid UI's Segments page. The top navigation bar includes tabs for Load data, Datasources (selected), Segments (selected), Tasks, Servers, and Query. On the right side of the header are links for Legacy, Settings, and Help. Below the header, there are buttons for Refresh, Group by (set to None), Interval, and a three-dot menu. A "Columns" dropdown menu is also present. The main table has columns for Segment ID, Datasource, Start, End, Version, Partition, Size, Num rows, Replicas, Is published, Is realtime, Is available, and Is overshadowed. Three segments are listed for the datasource "case2": "case2_2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z", "case2_2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z", and "case2_2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z". Each segment has a size between 5.39 KB and 5.49 KB, 1 replica, and is marked as published and realtime. The "Start" and "End" columns have a search input field with the value "case2".

Segment ID	Datasource	Start	End	Version	Partition	Size	Num rows	Replicas	Is published	Is realtime	Is available	Is overshadowed
case2_2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z	case2	2019-01-03T00:00:00.000Z	2019-01-04T00:00:00.000Z	2019-10-03T09:30:00.000Z	0	5.39 KB	3	1	true	false	true	false
case2_2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z	case2	2019-01-02T00:00:00.000Z	2019-01-03T00:00:00.000Z	2019-10-03T09:30:00.000Z	0	5.49 KB	4	1	true	false	true	false
case2_2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z	case2	2019-01-01T00:00:00.000Z	2019-01-02T00:00:00.000Z	2019-10-03T09:30:00.000Z	0	5.40 KB	3	1	true	false	true	false

Native Ingestion – No Rollup – case2

The screenshot shows the Druid UI interface with two separate query panes.

Top Query:

```
1 select * from case2
```

Bottom Query:

```
1 select TIME_EXTRACT(_time, 'year'), TIME_EXTRACT(_time, 'month'), TIME_EXTRACT(_time, 'day'), sum(total_fare_amount), count(*)
2 from case2 group by 1, 2, 3
```

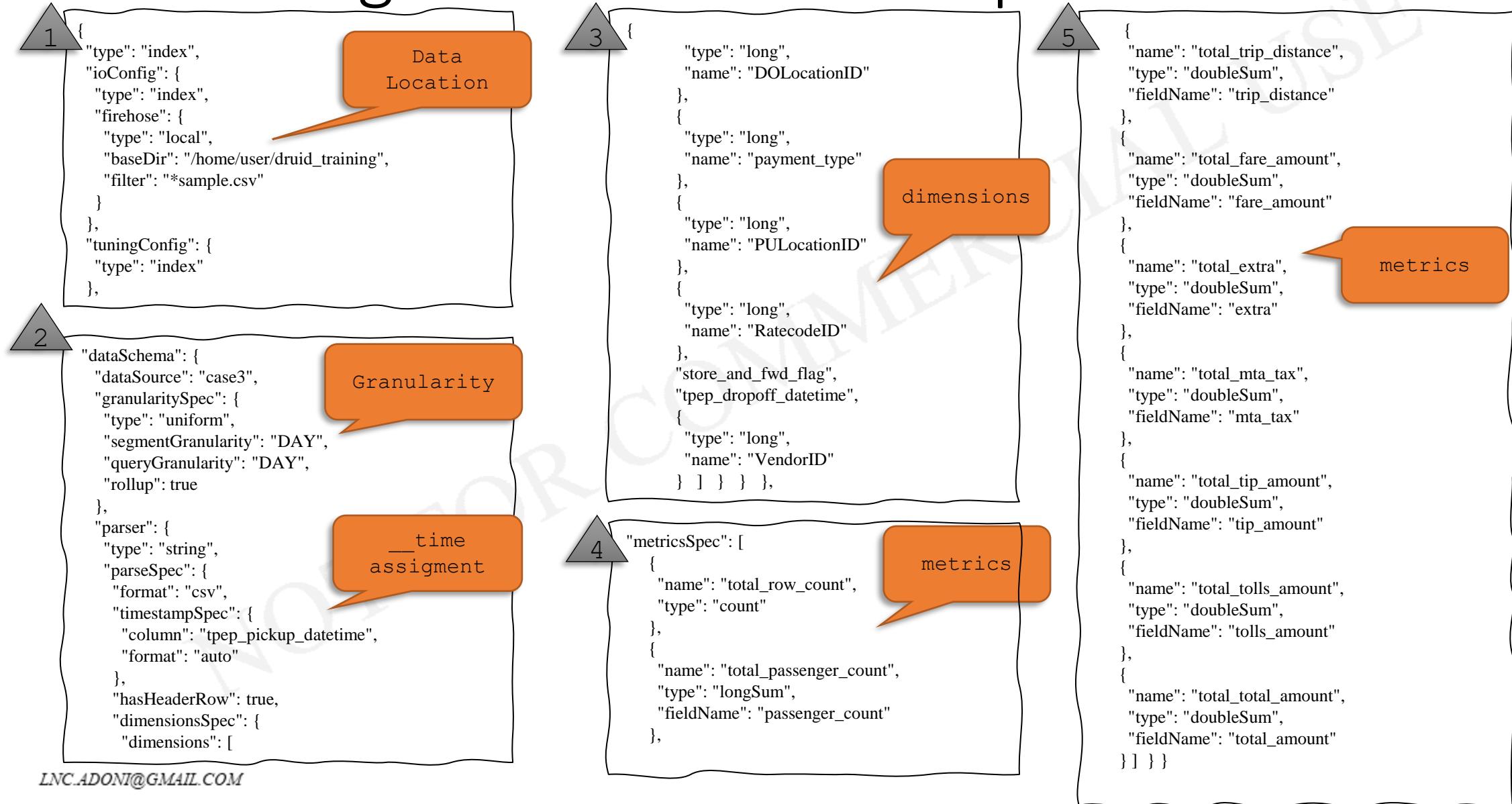
Results of the Bottom Query:

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
2019	1	1	25.5	3
2019	1	2	47.5	4
2019	1	3	30.5	3

Native Ingestion – No Rollup – case2

```
user@kylin-server: ~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z/0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2$ pwd
/home/user/apache-druid-0.16.0-incubating/var/druid/segments/case2
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2$ ls -al
total 20
drwxr-xr-x 5 user user 4096 Oct  3 15:05 .
drwxr-xr-x 5 user user 4096 Oct  3 15:05 ..
drwxr-xr-x 3 user user 4096 Oct  3 15:05 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 15:05 2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 15:05 2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2$ cd 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 15:05 .
drwxr-xr-x 5 user user 4096 Oct  3 15:05 ..
drwxr-xr-x 3 user user 4096 Oct  3 15:05 2019-10-03T09:35:13.755Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ cd 2019-10-03T09\::35\::13.755Z/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 15:05 .
drwxr-xr-x 3 user user 4096 Oct  3 15:05 ..
drwxr-xr-x 2 user user 4096 Oct  3 15:05 0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z$ cd 0/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z/0$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z/0$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z/0$ ls -al
total 12
drwxr-xr-x 2 user user 4096 Oct  3 15:05 .
drwxr-xr-x 3 user user 4096 Oct  3 15:05 ..
-rw-r--r-- 1 user user 1876 Oct  3 15:05 index.zip
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case2/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z/2019-10-03T09:35:13.755Z/0$
```

Native Ingestion – With Rollup – case3



Native Ingestion – With Rollup – case3

The screenshot shows the Druid Data Sources interface. At the top, there are navigation links: druid, Load data, Datasources (which is selected), Segments, Tasks, Servers, and Query. On the right side of the header are Legacy, Settings, and Help icons. Below the header, the title "Datasources" is displayed, along with Refresh, Show segment timeline, and Show disabled buttons. A "Columns" dropdown menu is also present. The main table has columns: Datasource, Availability, Segment load/drop, Retention, Replicated size, Size, Compaction, Avg. segment..., Num rows, and Actions. Three data sources are listed:

Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
case1	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	12.61 KB	12.61 KB	None	4.20 KB	10	
case2	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	16.27 KB	16.27 KB	None	5.42 KB	10	
case3	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	17.08 KB	17.08 KB	None	5.69 KB	10	

The screenshot shows the Druid Segments interface. At the top, there are navigation links: druid, Load data, Datasources, Segments (which is selected), Tasks, Servers, and Query. On the right side of the header are Legacy, Settings, and Help icons. Below the header, the title "Segments" is displayed, along with Refresh, Group by, and a dropdown menu. The main table has columns: Segment ID, Datasource, Start, End, Version, Partition, Size, Num rows, Replicas, Is published, Is realtime, Is available, and Is overshadowed. A search bar at the top allows filtering by "case3". The table displays three segments for the "case3" datasource:

Segment ID	Datasource	Start	End	Version	Partition	Size	Num rows	Replicas	Is published	Is realtime	Is available	Is overshadowed
case3_2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z	case3	2019-01-03T00:00:00.000Z	2019-01-04T00:00:00.000Z	2019-10-03T12:5...	0	5.66 KB	3	1	true	false	true	false
case3_2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z	case3	2019-01-02T00:00:00.000Z	2019-01-03T00:00:00.000Z	2019-10-03T12:5...	0	5.75 KB	4	1	true	false	true	false
case3_2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z	case3	2019-01-01T00:00:00.000Z	2019-01-02T00:00:00.000Z	2019-10-03T12:5...	0	5.67 KB	3	1	true	false	true	false

Native Ingestion – With Rollup – case3

The screenshot shows the Druid UI interface with two stacked queries. The top query is:

```
1 select * from case3
```

The bottom query is:

```
1 select TIME_EXTRACT(__time, 'year'), TIME_EXTRACT(__time, 'month'), TIME_EXTRACT(__time, 'day'), TIME_EXTRACT(__time, 'hour'),  
2 sum(total_fare_amount), count(*)  
3 from case3 group by 1, 2, 3,4
```

The results of the bottom query are displayed in a table:

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4	EXPR\$5
2019	1	1	0	25.5	3
2019	1	2	0	47.5	4
2019	1	3	0	30.5	3

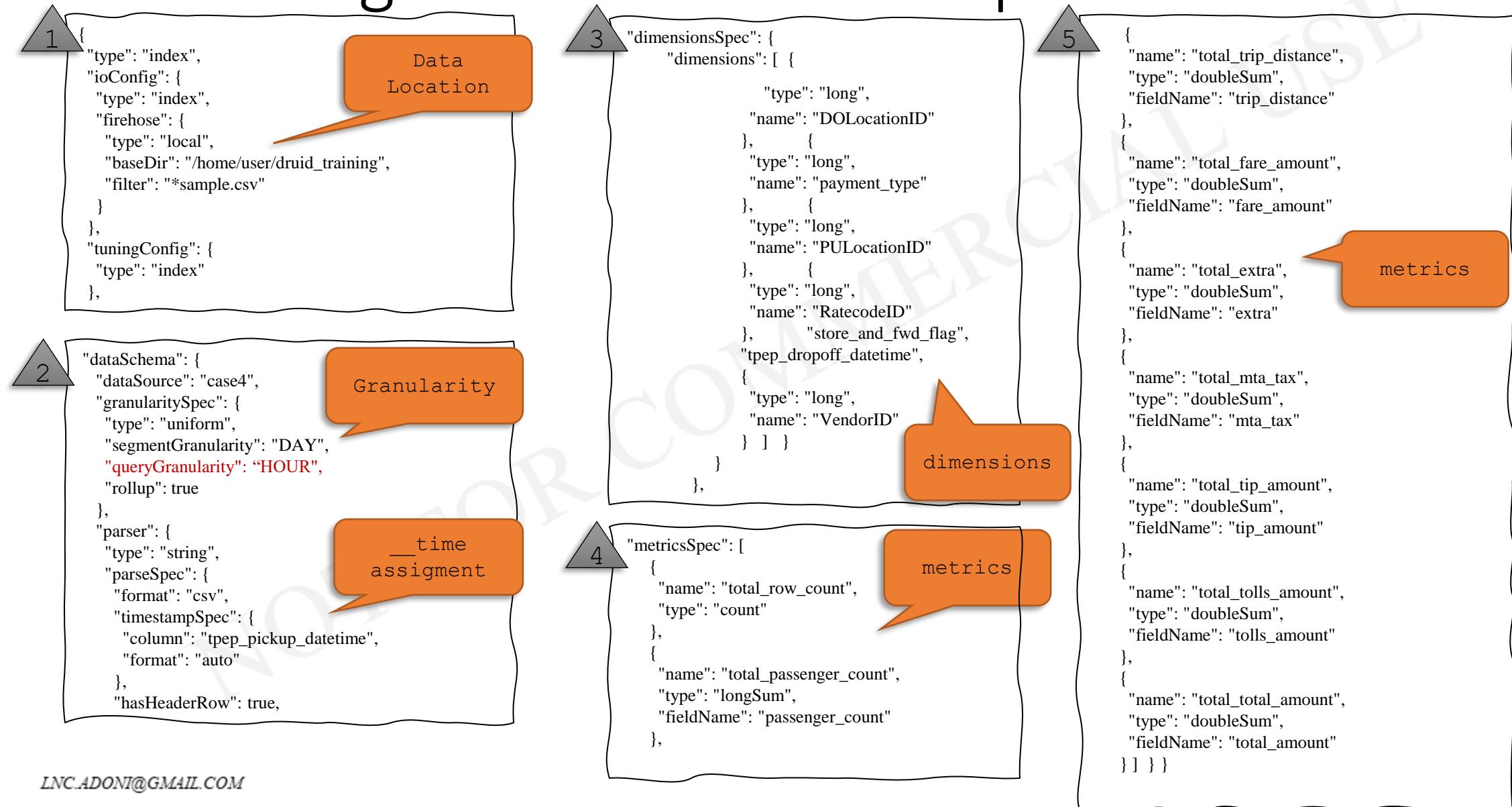
Query controls at the bottom include Run, Auto run, Smart query limit, and a result count of 3 results in 0.12s.

Native Ingestion – With Rollup – case3

```
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ ls -al  
total 20  
drwxr-xr-x 5 user user 4096 Oct  3 18:26 .  
drwxr-xr-x 6 user user 4096 Oct  3 18:26 ..  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ cd 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ ls -al  
total 12  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 .  
drwxr-xr-x 5 user user 4096 Oct  3 18:26 ..  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 2019-10-03T12:56:46.921Z  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ cd 2019-10-03T12\:56\:46.921Z/  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$ ls -al  
total 12  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 .  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 ..  
drwxr-xr-x 2 user user 4096 Oct  3 18:26 0  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$ cd 0  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$ 0$ ls -al  
total 12  
drwxr-xr-x 2 user user 4096 Oct  3 18:26 .  
drwxr-xr-x 3 user user 4096 Oct  3 18:26 ..  
-rw-r--r-- 1 user user 1887 Oct  3 18:26 index.zip  
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case3$ 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T12:56:46.921Z$ 0$
```

queryGranularity : HOUR

Native Ingestion – With Rollup – case4



Native Ingestion – With Rollup – case4

Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
case1	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	12.61 KB	12.61 KB	None	4.20 KB	10	
case2	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	16.27 KB	16.27 KB	None	5.42 KB	10	
case3	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	17.08 KB	17.08 KB	None	5.69 KB	10	
case4	• Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	17.09 KB	17.09 KB	None	5.70 KB	10	

Segment ID	Datasource	Start	End	Version	Partition	Size	Num rows	Replicas	Is published	Is realtime	Is available	Is overshado...
"case4" X	case4	2019-01-03T00:00:00Z	2019-01-04T00:00:00Z	2019-10-03T13:2...	0	5.66 KB	3	1	true	false	true	false
case4_2019-01-02T00:00:00.000Z_2019-01-03T00:00:00Z	case4	2019-01-02T00:00:00Z	2019-01-03T00:00:00Z	2019-10-03T13:2...	0	5.76 KB	4	1	true	false	true	false
case4_2019-01-01T00:00:00.000Z_2019-01-02T00:00:00Z	case4	2019-01-01T00:00:00Z	2019-01-02T00:00:00Z	2019-10-03T13:2...	0	5.67 KB	3	1	true	false	true	false

Native Ingestion – With Rollup – case4

The screenshot shows the Druid UI interface. The top navigation bar has tabs for 'druid', 'Load data', 'Datasources', 'Segments', 'Tasks', 'Servers', and 'Query'. The 'Query' tab is active. On the left, there's a sidebar with a tree view of data sources: 'druid' (selected), 'case4' (selected), and various tables like '_time', 'DO', 'payment_type', etc. The main area displays a query result table.

Query:

```
1 select TIME_EXTRACT(__time, 'year'), TIME_EXTRACT(__time, 'month'), TIME_EXTRACT(__time, 'day'), TIME_EXTRACT(__time, 'hour'),
2 sum(total_fare_amount), count(*)
3 from case4 group by 1, 2, 3,4
```

Run button, Auto run, Smart query limit, 4 results in 0.21s.

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4	EXPR\$5
2019	1	1	19	25.5	3
2019	1	2	20	47.5	4
2019	1	3	14	17.5	1
2019	1	3	15	13	2

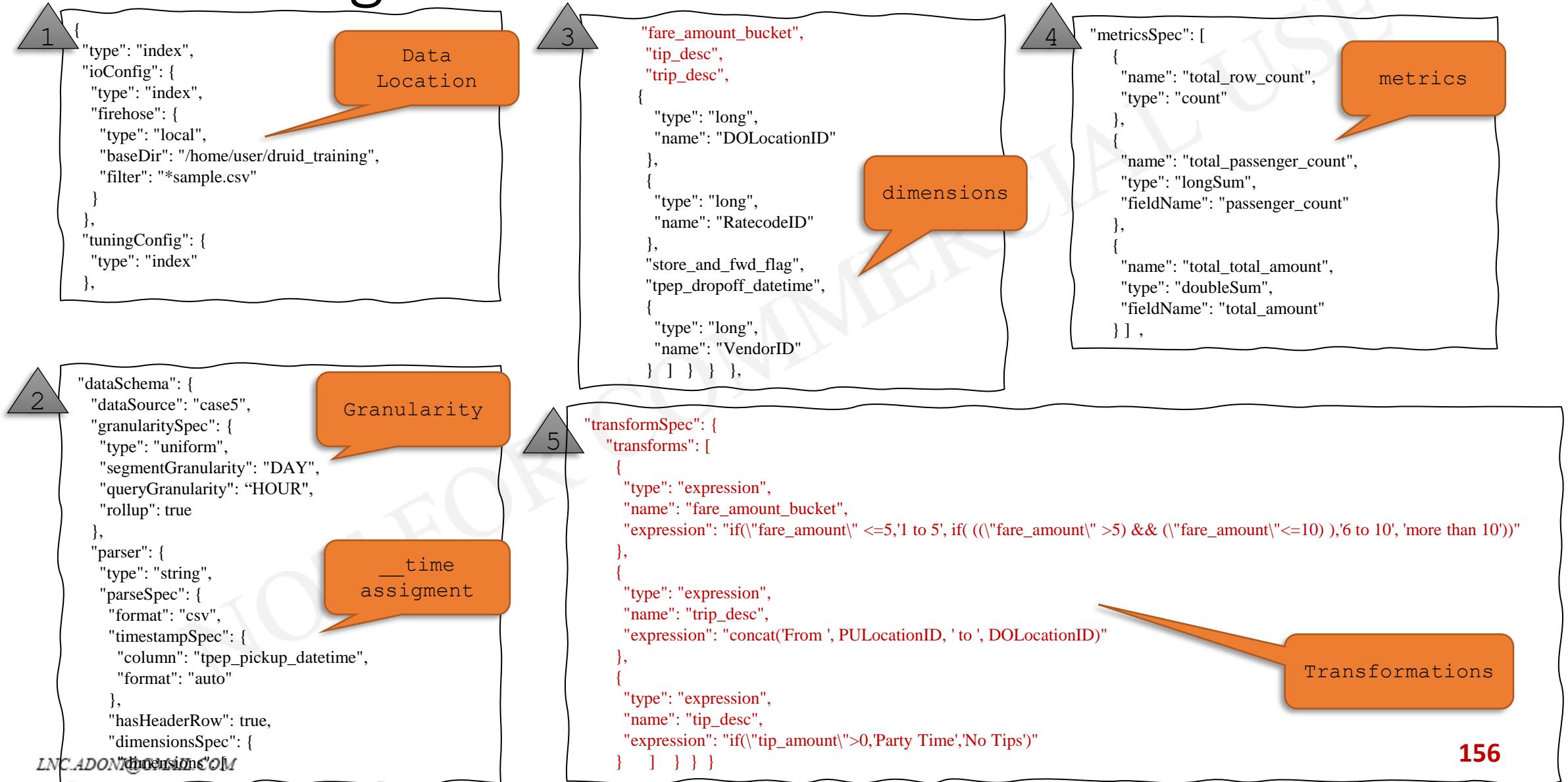
Previous, Page 1 of 1, 20 rows, Next.

Native Ingestion – With Rollup – case4

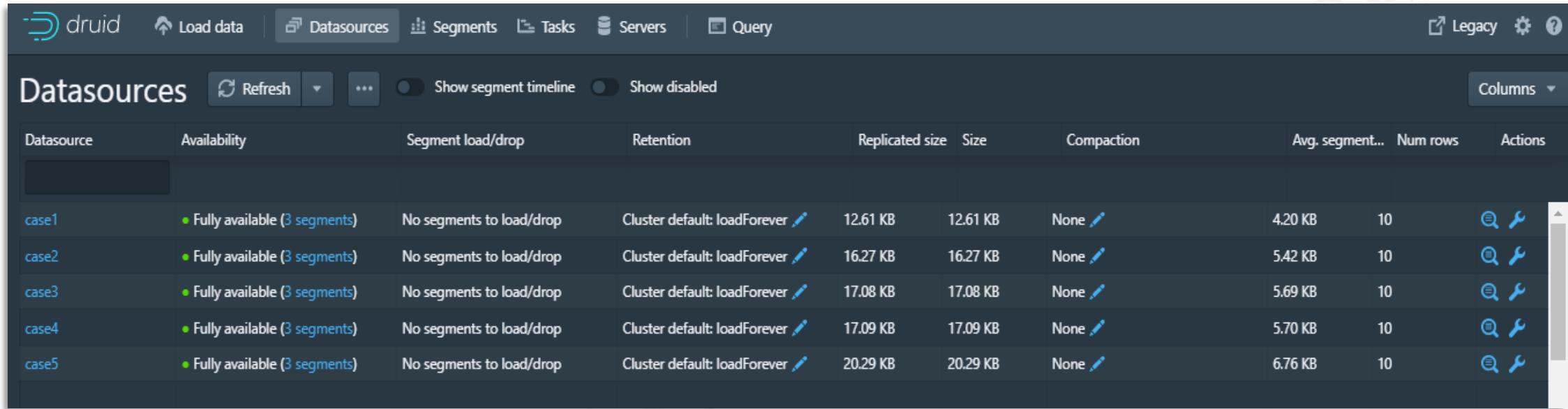
```
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4$ pwd
/home/user/apache-druid-0.16.0-incubating/var/druid/segments/case4
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4$ ls -al
total 20
drwxr-xr-x 5 user user 4096 Oct  3 18:52 .
drwxr-xr-x 7 user user 4096 Oct  3 18:52 ..
drwxr-xr-x 3 user user 4096 Oct  3 18:52 2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 18:52 2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z
drwxr-xr-x 3 user user 4096 Oct  3 18:52 2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4$ cd 2019-01-01T00\::00\::00.000Z_2019-01-02T00\::00\::00.000Z/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 18:52 .
drwxr-xr-x 5 user user 4096 Oct  3 18:52 ..
drwxr-xr-x 3 user user 4096 Oct  3 18:52 2019-10-03T13:22:19.124Z
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ cd 2019-10-03T13\::22\::19.124Z/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ ls -al
total 12
drwxr-xr-x 3 user user 4096 Oct  3 18:52 .
drwxr-xr-x 3 user user 4096 Oct  3 18:52 ..
drwxr-xr-x 2 user user 4096 Oct  3 18:52 0
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ cd 0/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 0/
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 0$ 
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 0$ ls -al
total 12
drwxr-xr-x 2 user user 4096 Oct  3 18:52 .
drwxr-xr-x 3 user user 4096 Oct  3 18:52 ..
-rw-r--r-- 1 user user 1888 Oct  3 18:52 index.zip
user@kylin-server:~/apache-druid-0.16.0-incubating/var/druid/segments/case4/2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z$ 2019-10-03T13:22:19.124Z$ 0$ 
```

Transformations

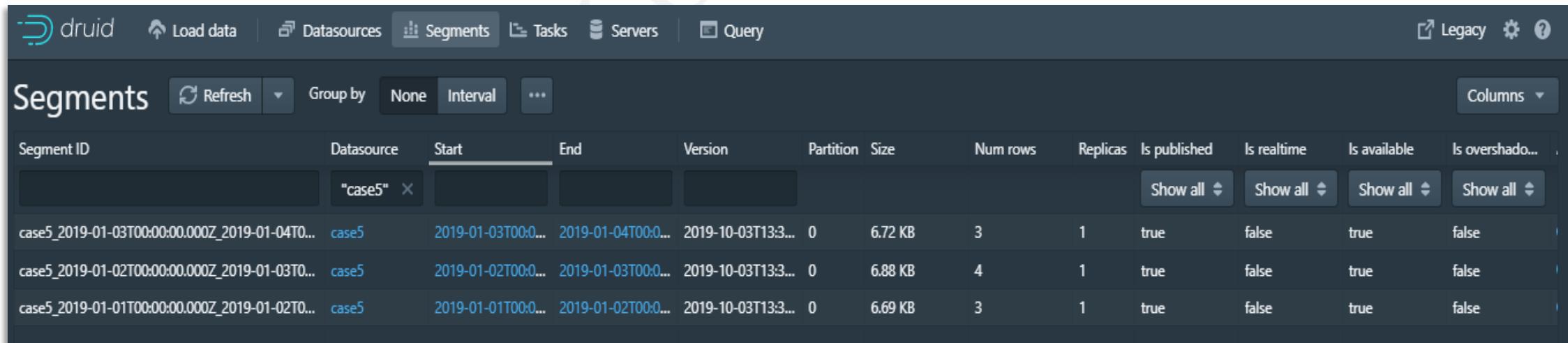
Native Ingestion – Transformations – case5



Native Ingestion – Transformations – case5



Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
case1	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	12.61 KB	12.61 KB	None	4.20 KB	10	
case2	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	16.27 KB	16.27 KB	None	5.42 KB	10	
case3	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	17.08 KB	17.08 KB	None	5.69 KB	10	
case4	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	17.09 KB	17.09 KB	None	5.70 KB	10	
case5	Fully available (3 segments)	No segments to load/drop	Cluster default: loadForever	20.29 KB	20.29 KB	None	6.76 KB	10	



Segment ID	Datasource	Start	End	Version	Partition	Size	Num rows	Replicas	Is published	Is realtime	Is available	Is overshadowed
case5_2019-01-03T00:00:00.000Z_2019-01-04T00:00:00.000Z	case5	2019-01-03T00:00:00.000Z	2019-01-04T00:00:00.000Z	2019-10-03T13:3...	0	6.72 KB	3	1	true	false	true	false
case5_2019-01-02T00:00:00.000Z_2019-01-03T00:00:00.000Z	case5	2019-01-02T00:00:00.000Z	2019-01-03T00:00:00.000Z	2019-10-03T13:3...	0	6.88 KB	4	1	true	false	true	false
case5_2019-01-01T00:00:00.000Z_2019-01-02T00:00:00.000Z	case5	2019-01-01T00:00:00.000Z	2019-01-02T00:00:00.000Z	2019-10-03T13:3...	0	6.69 KB	3	1	true	false	true	false

Native Ingestion – Transformations – case5

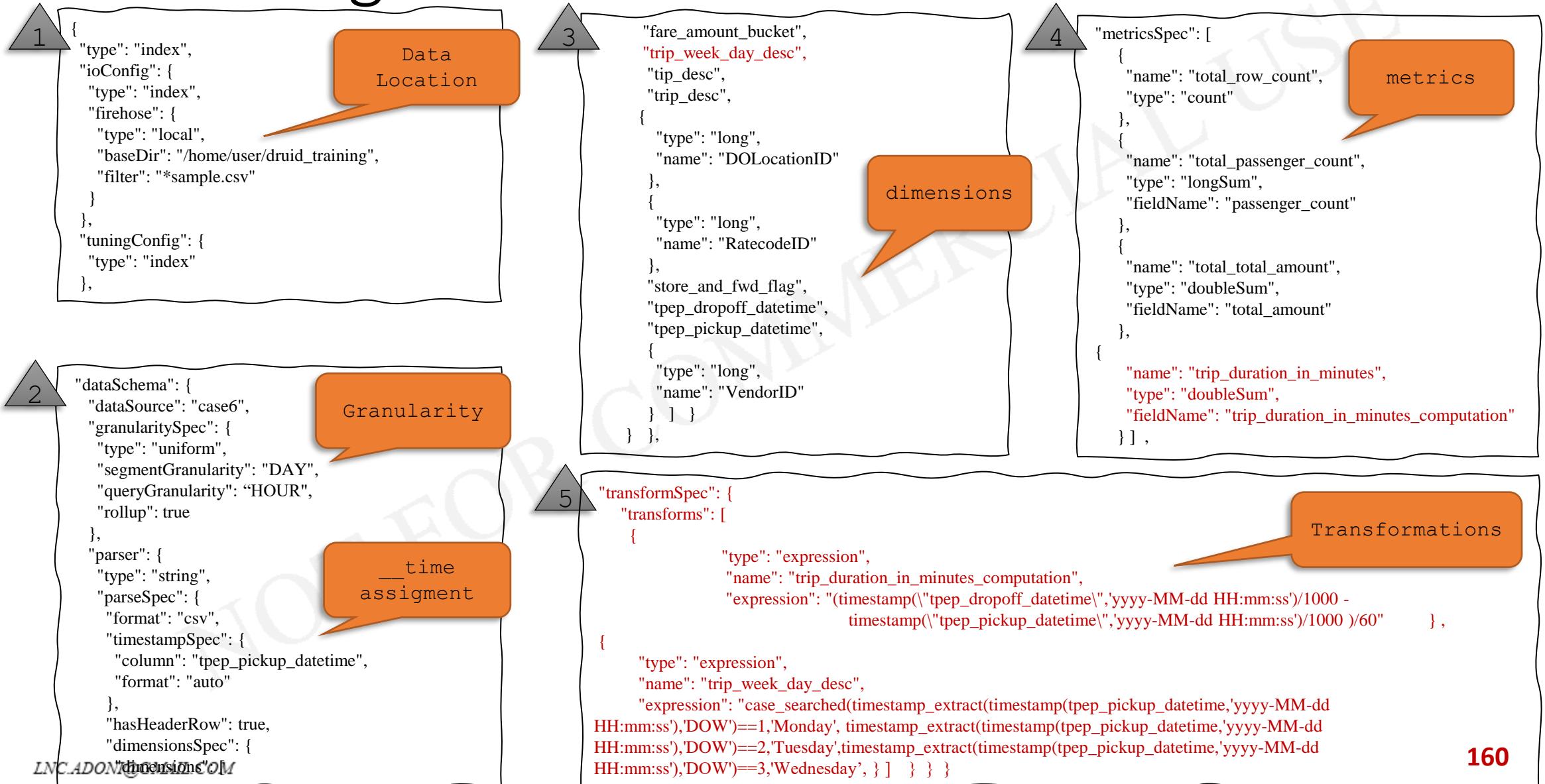
The screenshot shows the Druid UI interface. The top navigation bar includes 'druid', 'Load data', 'Datasources', 'Segments', 'Tasks', 'Servers', and 'Query'. The 'Query' tab is active. On the left, a sidebar lists various columns under the 'case5' dataset, such as '_time', 'DOlocationID', 'fare_amount_bucket', 'payment_type', 'PUlocationID', 'RatecodeID', 'store_and_fwd_flag', 'tip_desc', 'total_congestion_surcharge', 'total_extra', 'total_fare_amount', 'total_improvement_surcharge', 'total_mta_tax', 'total_passenger_count', 'total_row_count', 'total_tip_amount', 'total_tolls_amount', and 'total_total_amount'. A dropdown menu icon is also present. The main area displays a query result table with the following data:

tip_desc	trip_desc	fare_amount_bucket	EXPR\$3	EXPR\$4
No Tips	From 141 to 238	6 to 10	10	1
No Tips	From 142 to 237	6 to 10	7	1
No Tips	From 229 to 234	6 to 10	7.5	1
No Tips	From 244 to 42	more than 10	17.5	1
Party Time	From 137 to 229	6 to 10	6	1
Party Time	From 138 to 148	more than 10	25	1
Party Time	From 161 to 263	more than 10	10.5	1
Party Time	From 237 to 163	6 to 10	7	1
Party Time	From 238 to 74	6 to 10	8.5	1
Party Time	From 263 to 141	1 to 5	4.5	1

Below the table, there are navigation buttons for 'Previous', 'Page 1 of 1', '20 rows', and 'Next'.

More Transformations

Native Ingestion – Transformations – case6



Native Ingestion – Transformations – case6

The screenshot shows the Druid UI interface. On the left, there's a sidebar with a tree view of the schema. The root node 'druid' has a single child node 'case6'. Under 'case6', there are several columns listed: '_time', 'DOLocationID', 'payment_type', 'PULocationID', 'RatecodeID', 'store_and_fwd_flag', 'total_congestion_surcharge', 'total_extra', 'total_fare_amount', 'total_improvement_surcharge', 'total_mta_tax', 'total_passenger_count', 'total_row_count', 'total_tip_amount', 'total_tolls_amount', and 'total_total_amount'. The main panel displays a query result for the SQL statement 'select * from case6'. The results are presented in a table with 10 rows. The columns are: uni, total_tip_amour, total_tolls_amor, total_total_amo, total_trip_distan, tpep_dropoff_datetime, tpep_pickup_datetime, trip_duration_in_minutes, and trip_week_day_desc. The data includes various trip details such as pickup and dropoff times, total fares, and trip durations.

uni	total_tip_amour	total_tolls_amor	total_total_amo	total_trip_distan	tpep_dropoff_datetime	tpep_pickup_datetime	trip_duration_in_minutes	trip_week_day_desc
2.32	0	11.62	1.82		2019-01-01 19:16:42	2019-01-01 19:09:10	7	Tuesday
2.34	0	10.14	1.21		2019-01-01 19:31:01	2019-01-01 19:24:31	6	Tuesday
0	0	10.8	2.38		2019-01-01 19:49:13	2019-01-01 19:39:05	10	Tuesday
1.15	0	6.95	0.8		2019-01-02 20:27:53	2019-01-02 20:24:15	3	Wednesday
5	0	31.3	8.74		2019-01-02 20:43:49	2019-01-02 20:29:20	14	Wednesday
0	0	8.8	1.31		2019-01-02 20:42:10	2019-01-02 20:33:28	8	Wednesday
1.5	0	13.3	2.8		2019-01-02 20:59:04	2019-01-02 20:47:39	11	Wednesday
0	0	18.3	2.93		2019-01-03 15:19:44	2019-01-03 14:55:21	24	Thursday

Query – Details by Pickup, Drop location

Query

```
1 select PULocationID, DOLocationID,  
2 sum(total_fare_amount),  
3 sum(total_trip_distance), sum(trip_duration_in_minutes)  
4 from case6  
5 group by 1, 2  
6 order by 3 desc|
```

Run ... Auto run Smart query limit 10 results in 0.17s ⏲

PULocationID	DOLocationID	EXPR\$2	EXPR\$3	EXPR\$4
138	148	25	8.74	14
244	42	17.5	2.93	24
161	263	10.5	2.8	11
141	238	10	2.38	10
238	74	8.5	1.82	7
229	234	7.5	1.31	8
237	163	7	1.21	6
142	237	7	1.1	8
137	229	6	1.1	5
262	141	4.5	0.0	0

Previous Page 1 of 1 20 rows ▾ Next

Lookup

Defining Lookup - Data

	A	B	C	D
1	LocationID	Borough	Zone	service_zone
2	1	EWR	Newark Airport	EWR
3	2	Queens	Jamaica Bay	Boro Zone
4	3	Bronx	Allerton/Pelham Gardens	Boro Zone
5	4	Manhattan	Alphabet City	Yellow Zone
6	5	Staten Island	Arden Heights	Boro Zone
7	6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
8	7	Queens	Astoria	Boro Zone
9	8	Queens	Astoria Park	Boro Zone
10	9	Queens	Auburndale	Boro Zone
11	10	Queens	Baisley Park	Boro Zone
12	11	Brooklyn	Bath Beach	Boro Zone
13	12	Manhattan	Battery Park	Yellow Zone
14	13	Manhattan	Battery Park City	Yellow Zone
15	14	Brooklyn	Bay Ridge	Boro Zone
16	15	Queens	Bay Terrace/Fort Totten	Boro Zone
17	16	Queens	Bayside	Boro Zone
18	17	Brooklyn	Bedford	Boro Zone
19	18	Bronx	Bedford Park	Boro Zone
20	19	Queens	Bellerose	Boro Zone

	A	B	
1	LocationID	Zone	
2		1 Newark Airport	
3		2 Jamaica Bay	
4		3 Allerton/Pelham Gardens	
5		4 Alphabet City	
6		5 Arden Heights	
7		6 Arrochar/Fort Wadsworth	
8		7 Astoria	
9		8 Astoria Park	
10		9 Auburndale	
11		10 Baisley Park	
12		11 Bath Beach	
13		12 Battery Park	
14		13 Battery Park City	
15		14 Bay Ridge	
16		15 Bay Terrace/Fort Totten	
17		16 Bayside	
18		17 Bedford	
19		18 Bedford Park	
20		19 Bellerose	
21		20 Belmont	
			A
			B
			1 LocationID
			Borough
			2 EWR
			3 Queens
			4 Bronx
			5 Manhattan
			6 Staten Island
			7 Staten Island
			8 Queens
			9 Queens
			10 Queens
			11 Brooklyn
			12 Manhattan
			13 Manhattan
			14 Brooklyn
			15 Queens
			16 Queens
			17 Brooklyn
			18 Bronx
			19 Queens
			20 Bronx

Defining Lookup - Data

```
{  
  "type": "cachedNamespace",  
  "extractionNamespace": {  
    "type": "uri",  
    "uri":  
      "file:/home/user/druid_training/location_zone_lookup.csv",  
    "namespaceParseSpec": {  
      "format": "csv",  
      "columns": [  
        "LocationID",  
        "Zone"  
      ],  
      "keyColumn": "LocationID",  
      "valueColumn": "Zone"  
    },  
    "pollPeriod": "PT30S"  
  },  
  "firstCacheTimeout": 0  
}
```

```
{  
  "type": "cachedNamespace",  
  "extractionNamespace": {  
    "type": "uri",  
    "uri":  
      "file:/home/user/druid_training/location_borough_lookup.csv",  
    "namespaceParseSpec": {  
      "format": "csv",  
      "columns": [  
        "LocationID",  
        "Borough"  
      ],  
      "keyColumn": "LocationID",  
      "valueColumn": "Borough"  
    },  
    "pollPeriod": "PT30S"  
  },  
  "firstCacheTimeout": 0  
}
```

Lookup Limitations

- Lookup mapping data should be string and string. i.e key and value should both be strings.
- While invoking Lookup, the key dimension should be string. Mapped value returned by Lookup is also string.
- If the key dimension is non-string, then it should be cast to string.

Lookup Usage – Location to Zone

```
1 select LOOKUP(cast(PULocationID as varchar), 'location_to_zone') pickup_zone,
2 LOOKUP(cast(DOLocationID as varchar), 'location_to_zone') drop_zone,
3 sum(total_fare_amount) total_amount,
4 sum(total_trip_distance) total_distance, sum(trip_duration_in_minutes) total_trip_minutes
5 from case6
6 group by 1, 2
7
```

Run ... Auto run Smart query limit 10 results in 0.18s

pickup_zone	drop_zone	total_amount	total_distance	total_trip_minutes
LaGuardia Airport	Lower East Side	25	8.74	14
Washington Heights South	Central Harlem North	17.5	2.93	24
Midtown Center	Yorkville West	10.5	2.8	11
Lenox Hill West	Upper West Side North	10	2.38	10
Upper West Side North	East Harlem North	8.5	1.82	7
Sutton Place/Turtle Bay North	Union Sq	7.5	1.31	8
Upper East Side South	Midtown North	7	1.21	6
Lincoln Square East	Upper East Side South	7	1.1	8
Kips Bay	Sutton Place/Turtle Bay North	6	1.1	5
Yorkville West	Lenox Hill West	4.5	0.0	2

Previous Page 1 of 1 20 rows Next

Lookup Usage – Location to Borough

```
1 select LOOKUP(cast(PULocationID as varchar),'location_to_borough') pickup_zone,
2 LOOKUP(cast(DOLocationID as varchar),'location_to_borough') drop_zone,
3 sum(total_fare_amount) total_amount,
4 sum(total_trip_distance) total_distance, sum(trip_duration_in_minutes) total_trip_minutes
5 from case6
6 group by 1, 2
7 order by 3 desc|
```

Run

...

Auto run

Smart query limit

2 results in 0.14s 

pickup_zone	drop_zone	total_amount	total_distance	total_trip_minutes
Manhattan	Manhattan	78.5	15.450000000000001	82
Queens	Manhattan	25	8.74	14

Lookup Usage – Location to Borough

```
1 select LOOKUP(cast(PULocationID as varchar),'location_to_borough') pickup_zone,  
2 LOOKUP(cast(DOLocationID as varchar),'location_to_borough') drop_zone,  
3 sum(total_fare_amount) total_amount,  
4 sum(total_trip_distance) total_distance, sum(trip_duration_in_minutes) total_trip_minutes  
5 from case6  
6 group by 1, 2  
7 order by 3 desc|
```

Run

...

Auto run

Smart query limit

2 results in 0.14s 

pickup_zone	drop_zone	total_amount	total_distance	total_trip_minutes
Manhattan	Manhattan	78.5	15.450000000000001	82
Queens	Manhattan	25	8.74	14

Lookup Usage – Filter/Where

```
1 select LOOKUP(cast(PULocationID as varchar), 'location_to_zone') pickup_zone,
2 LOOKUP(cast(DOLocationID as varchar), 'location_to_zone') drop_zone, *
3 from case6
4 where (upper(LOOKUP(cast(PULocationID as varchar), 'location_to_zone')) like '%BAY%' ) or
5 (LOOKUP(cast(PULocationID as varchar), 'location_to_zone') like '%Hill%' )
```

Run



Auto run



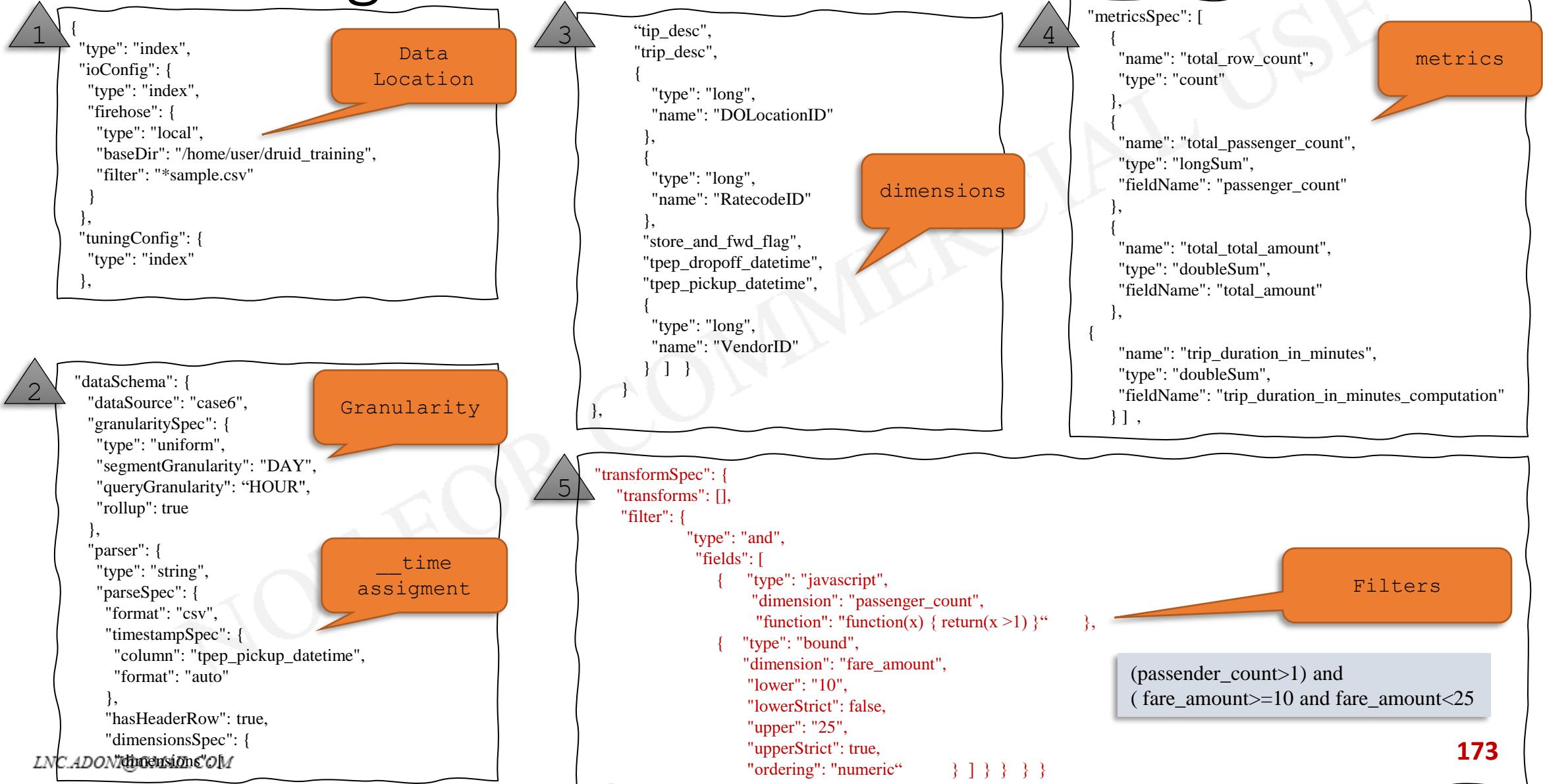
Smart query limit

3 results in 0.17s

pickup_zone	drop_zone	DOLocationID	PULocationID	RatecodeID	VendorID	_time	payment_type	store_and_fwd_	total_cng
Lenox Hill West	Upper West Side	238	141	1	2	2019-01-01T19:00:00	Credit	N	0
Sutton Place/Turtle Bay North	Union Sq	234	229	1	2	2019-01-02T20:00:00	Credit	N	0
Kips Bay	Sutton Place/Turtle Bay North	229	137	1	1	2019-01-03T15:00:00	Cash	N	0

Filter

Native Ingestion – Filter – case7



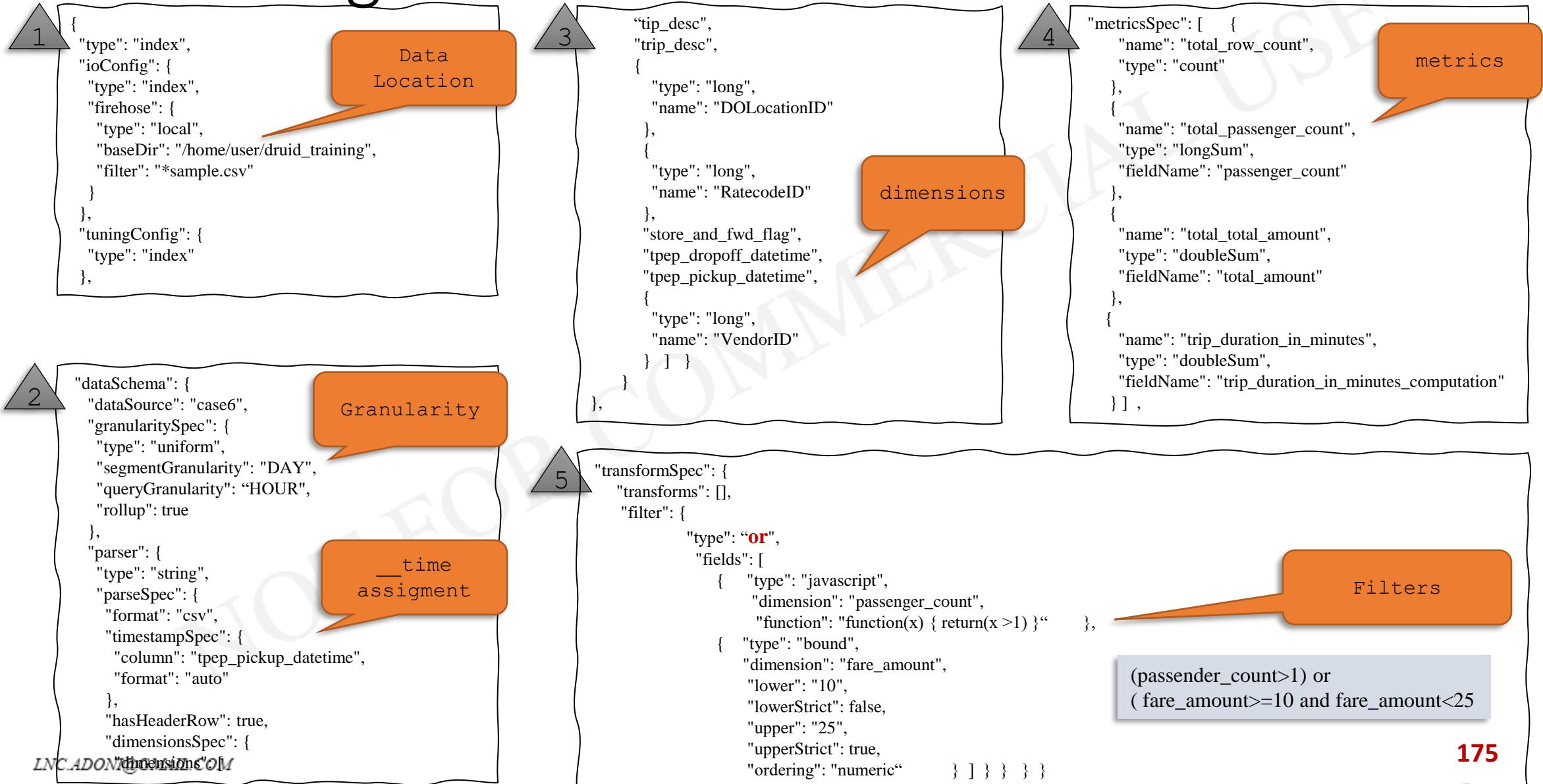
Native Ingestion – Filter– case7

```
VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fwd_flag,PULocationID,DOLocationID,↓  
payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improvement_surcharge,total_amount,congestion_surcharge↓  
↓  
2,2019-01-01 19:09:10,2019-01-01 19:16:42,5,1.82,1,N,238,74,1,8.5,0,0.5,2.32,0,0.3,11.62,↓  
2,2019-01-01 19:24:31,2019-01-01 19:31:01,1,1.21,1,N,237,163,1,7,0,0.5,2.34,0,0.3,10.14,↓  
2,2019-01-01 19:39:05,2019-01-01 19:49:13,5,2.38,1,N,141,238,2,10,0,0.5,0,0,0.3,10.8,↓  
1,2019-01-02 20:24:15,2019-01-02 20:27:53,1,1.80,1,N,263,141,1,4.5,0,0.5,0.5,1.15,0,0.3,6.95,↓  
1,2019-01-02 20:47:39,2019-01-02 20:59:04,1,2.80,1,N,161,263,1,10.5,0.5,0.5,1.5,0,0.3,13.3,↓  
4,2019-01-02 20:29:20,2019-01-02 20:43:49,1,8.74,1,N,138,148,1,25,0,5,0.5,5,0,0.3,31.3,↓  
2,2019-01-02 20:33:28,2019-01-02 20:42:10,1,1.31,1,N,229,234,2,7.5,0.5,0.5,0,0,0.3,8.8,↓  
1,2019-01-03 15:49:53,2019-01-03 15:58:41,1,1.10,1,N,142,237,1,7,0,0.5,0,0,0.3,7.8,↓  
2,2019-01-03 14:55:21,2019-01-03 15:19:44,3,2.93,1,N,244,42,1,17.5,0,0.5,0,0,0.3,18.3,↓  
1,2019-01-03 15:43:44,2019-01-03 15:49:08,1,1.10,1,N,137,229,1,6,0,0.5,1.35,0,0.3,8.15,←
```

Though fare_amount
is >10, passenger
count is 1.

_time	DOLocationID	extra	fare_amount (filtered)	improvement_	mta_tax	passenger_cou (filtered)	payment_type	PULocationID	RatecodeID	store_a
2019-01-01T...	238	0	10	0.3	0.5	5	2	141	1	N
2019-01-03T...	42	0	17.5	0.3	0.5	3	1	244	1	N

Native Ingestion – Filter – case8



Native Ingestion – Filter– case8

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge	
2	2019-01-01 19:09:10	2019-01-01 19:16:42	5	1.82	1	N	238	74	1	8.5	0	0.5	2.32	0	0.3	11.62	,	
2	2019-01-01 19:24:31	2019-01-01 19:31:01	1	1.21	1	N	237	163	1	7	0	0.5	2.34	0	0.3	10.14	,	
2	2019-01-01 19:39:05	2019-01-01 19:49:13	5	2.38	1	N	141	238	1	10	0	0.5	0	0	0.3	10.8	,	
1	2019-01-02 20:24:15	2019-01-02 20:27:53	1	1.80	1	N	263	141	1	4.5	0	0.5	1.15	0	0.3	6.95	,	
1	2019-01-02 20:47:39	2019-01-02 20:59:04	1	2.80	1	N	161	263	1	10.5	0	0.5	0.5	1.5	0	0.3	13.3	,
4	2019-01-02 20:29:20	2019-01-02 20:43:49	1	8.74	1	N	138	148	1	25	0	0.5	5	0	0.3	31.3	,	
2	2019-01-02 20:33:28	2019-01-02 20:42:10	1	1.31	1	N	229	234	1	7.5	0	0.5	0	0	0.3	8.8	,	
1	2019-01-03 15:49:53	2019-01-03 15:58:41	1	1.10	1	N	142	237	2	7	0	0.5	0	0	0.3	7.8	,	
2	2019-01-03 14:55:21	2019-01-03 15:19:44	3	2.93	1	N	244	42	1	17.5	0	0.5	0	0	0.3	18.3	,	
1	2019-01-03 15:43:44	2019-01-03 15:49:08	1	1.10	1	N	137	229	1	6	0	0.5	1.35	0	0.3	8.15	,	

(passenger_count>1) or
(fare_amount>=10 and fare_amount<25)

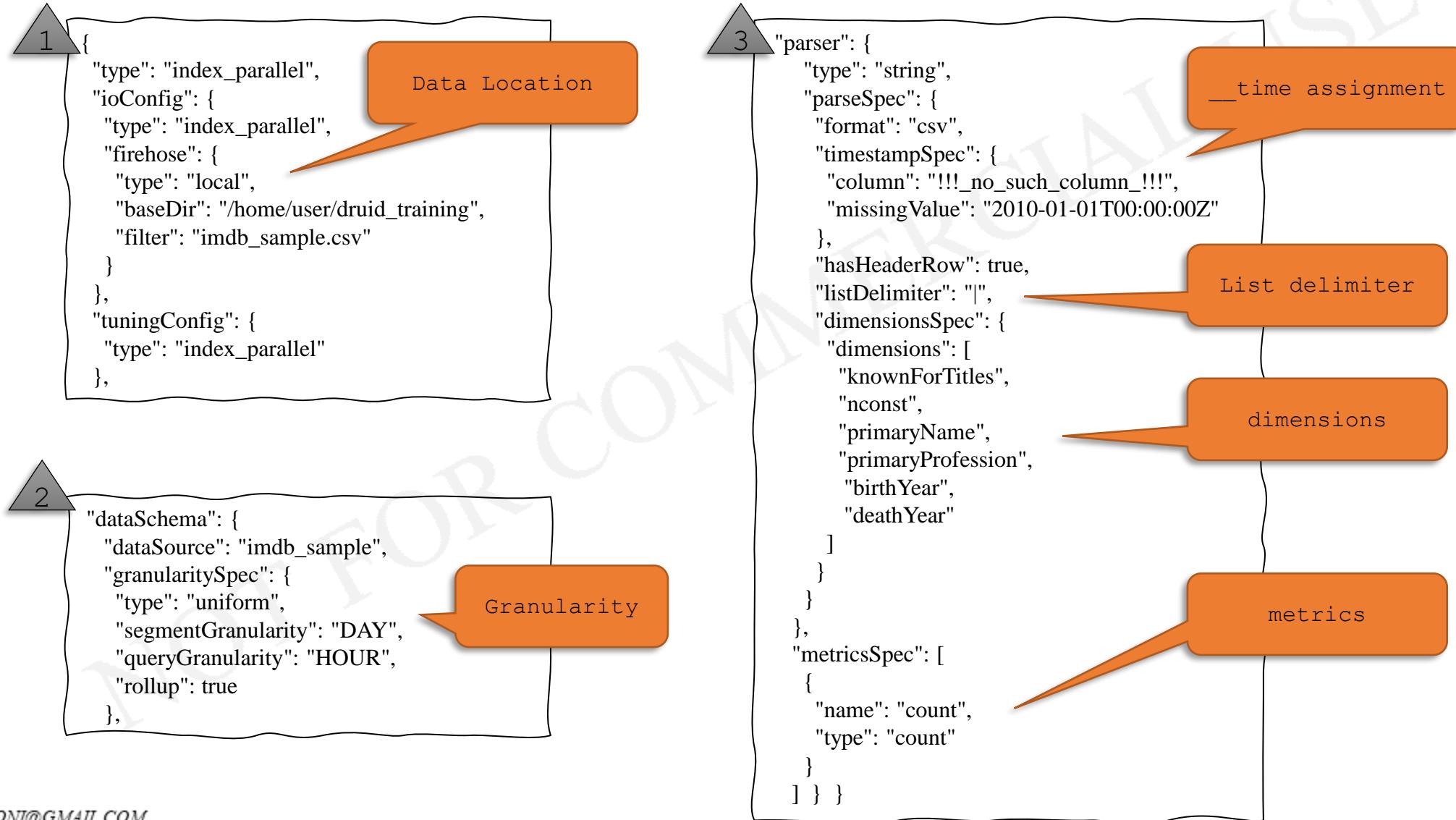
_time	DOLocationID	extra	fare_amount	improvement_	mta_tax	passenger_cou	payment_type	PULocationID	RatecodeID	store_a
2019-01-01T...	74	0	8.5	0.3	0.5	5	1	238	1	N
2019-01-01T...	238	0	10	0.3	0.5	5	2	141	1	N
2019-01-02T...	263	0.5	10.5	0.3	0.5	1	1	161	1	N
2019-01-03T...	42	0	17.5	0.3	0.5	3	1	244	1	N

Multi-Value Dimension

Multi-Value Dimension – Sample Data

	A	B	C	D	E	F	G
1	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles	
2	nm0000005	Ingmar Bergman	1918	2007	writer director actor	tt0069467 tt0083922 tt0050976 tt0050986	
3	nm0000006	Ingrid Bergman	1915	1982	actress soundtrack producer	tt0038787 tt0077711 tt0038109 tt0036855	
4	nm0000007	Humphrey Bogart	1899	1957	actor soundtrack producer	tt0043265 tt0037382 tt0034583 tt0033870	
5	nm0000008	Marlon Brando	1924	2004	actor soundtrack director	tt0070849 tt0068646 tt0078788 tt0044081	
6	nm0000009	Richard Burton	1925	1984	actor producer soundtrack	tt0061184 tt0057877 tt0087803 tt0059749	
7	nm0000010	James Cagney	1899	1986	actor soundtrack director	tt0035575 tt0055256 tt0029870 tt0031867	
8	nm0000011	Gary Cooper	1901	1961	actor soundtrack producer	tt0044706 tt0035896 tt0034167 tt0027996	
9	nm0000012	Bette Davis	1908	1989	actress soundtrack make_up_department	tt0031210 tt0035140 tt0042192 tt0056687	
10	nm0000013	Doris Day	1922	2019	soundtrack actress producer	tt0048317 tt0053172 tt0055100 tt0060463	
11	nm0000020	Henry Fonda	1905	1982	actor producer soundtrack	tt0032551 tt0064116 tt0082846 tt0050083	
12	nm0000033	Alfred Hitchcock	1899	1980	actor director producer	tt0052357 tt0053125 tt0054215 tt0056869	
13	nm0000559	Leonard Nimoy	1931	2015	actor director producer	tt0102975 tt1399103 tt0092007 tt0796366	
14	nm0000149	Jodie Foster	1962	-1	actress director producer	tt0476964 tt0118884 tt0094608 tt0102926	
15							
16							
17							

Multi-Value Dimension – case10



Multi-Value Dimension – Query

```
1 select * from imdb_sample
```

Run ... Auto run Smart query limit 13 results in 0.13s

_time	count	knownForTitles	nconst	primaryName	primaryProfession	sum_birthYear	sum_deathYear
2010-01-01T00:00:00Z	1	["tt0027996","tt0034167","tt0035896"]	nm0000011	Gary Cooper	["actor","producer","soundtrack"]	1901	1961
2010-01-01T00:00:00Z	1	["tt0029870","tt0031867","tt0035575"]	nm0000010	James Cagney	["actor","director","soundtrack"]	1899	1986
2010-01-01T00:00:00Z	1	["tt0031210","tt0035140","tt0042192"]	nm0000012	Bette Davis	["actress","make_up_department"]	1908	1989
2010-01-01T00:00:00Z	1	["tt0032551","tt0050083","tt0064116"]	nm0000020	Henry Fonda	["actor","producer","soundtrack"]	1905	1982
2010-01-01T00:00:00Z	1	["tt0033870","tt0034583","tt0037382"]	nm0000007	Humphrey Bogart	["actor","producer","soundtrack"]	1899	1957
2010-01-01T00:00:00Z	1	["tt0036855","tt0038109","tt0038787"]	nm0000006	Ingrid Bergman	["actress","producer","soundtrack"]	1915	1982
2010-01-01T00:00:00Z	1	["tt0044081","tt0068646","tt0070849"]	nm0000008	Marlon Brando	["actor","director","soundtrack"]	1924	2004
2010-01-01T00:00:00Z	1	["tt0048317","tt0053172","tt0055100"]	nm0000013	Doris Day	["actress","producer","soundtrack"]	1922	2019
2010-01-01T00:00:00Z	1	["tt0056076","tt0056081","tt0056087"]	nm0000005	1918	2007

Multi-Value Dimension – Query

```
1 select primaryName, count(distinct primaryProfession) profession_count, count(distinct knownForTitles) titles_count
2 from imdb_sample
3 where primaryProfession = 'producer'
4 group by 1
```

Run ... Auto run Smart query limit 9 results in 0.17s ⏲

primaryName	profession_count	titles_count
Alfred Hitchcock	3	4
Doris Day	3	4
Gary Cooper	3	4
Henry Fonda	3	4
Humphrey Bogart	3	4
Ingrid Bergman	3	4
Jodie Foster	3	4
Leonard Nimoy	3	4
... 1 more

Previous Page 1 of 1 20 rows Next

Multi-Value Dimension – Query

```
1 select primaryName, primaryProfession  
2 from imdb_sample_multivaluedimension  
3 where  
4 |mv_to_string(primaryProfession, ',') like '%producer%' and  
5 mv_to_string(primaryProfession, ',') like '%actor%'
```

Run ... Auto run Smart query limit

primaryName	primaryProfession
Gary Cooper	["actor", "producer", "soundtrack"]
Henry Fonda	["actor", "producer", "soundtrack"]
Humphrey Bogart	["actor", "producer", "soundtrack"]
Alfred Hitchcock	["actor", "director", "producer"]
Richard Burton	["actor", "producer", "soundtrack"]
Leonard Nimoy	["actor", "director", "producer"]

```
1 select primaryName, primaryProfession  
2 from imdb_sample_multivaluedimension  
3 where  
4 ( NOT ( mv_to_string(primaryProfession, ',') like '%producer%' ) ) and  
5 mv_to_string(primaryProfession, ',') like '%actor%'
```

Run ... Auto run Smart query limit

primaryName	primaryProfession
James Cagney	["actor", "director", "soundtrack"]
Marlon Brando	["actor", "director", "soundtrack"]
Ingmar Bergman	["actor", "director", "writer"]

Multi-Value Dimension – Query

```
1 select primaryName, knownForTitles, count(distinct primaryProfession), count(distinct knownForTitles)
2 from imdb_sample
3 where primaryProfession = 'actor'
4 group by 1,2
```

Run



Auto run



Smart query limit

36 results in 0.19s

primaryName	knownForTitles	EXPR\$2	EXPR\$3
Alfred Hitchcock	tt0052357	3	4
Alfred Hitchcock	tt0053125	3	4
Alfred Hitchcock	tt0054215	3	4
Alfred Hitchcock	tt0056869	3	4
Gary Cooper	tt0027996	3	4
Gary Cooper	tt0034167	3	4
Gary Cooper	tt0035896	3	4
Gary Cooper	tt0044706	3	4
...

Previous

Page 1 of 2

20 rows

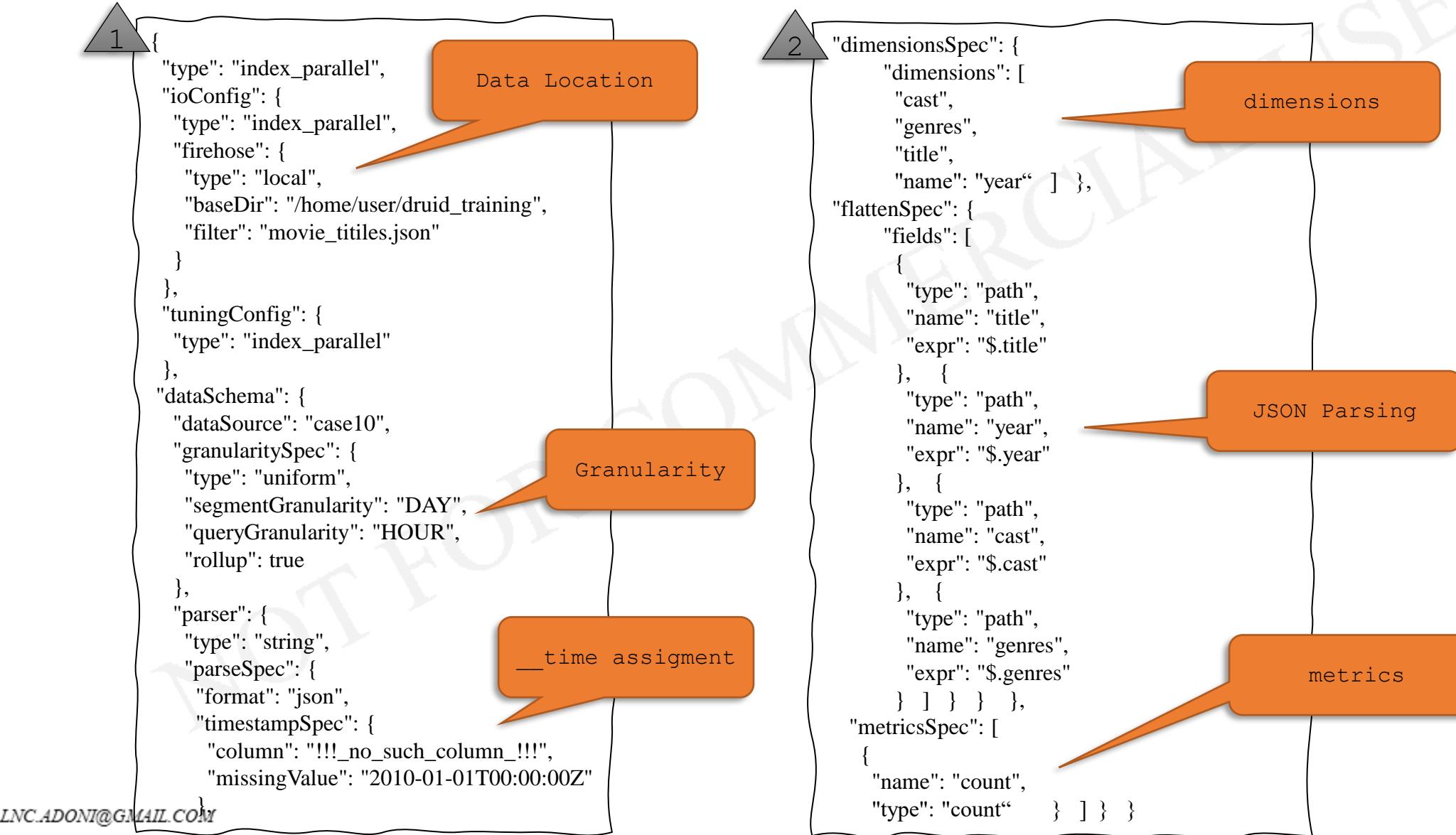
Next

Ingestion – JSON (Simple)

JSON Ingestion – Simple Data

```
1  {"title": "After Dark in Central Park", "year": 1900, "cast": [], "genres": []},
2  {"title": "Boarding School Girls' Pajama Parade", "year": 1900, "cast": [], "genres": []},
3  {"title": "Buffalo Bill's Wild West Parade", "year": 1900, "cast": [], "genres": []},
4  {"title": "Caught", "year": 1900, "cast": [], "genres": []},
5  {"title": "Clowns Spinning Hats", "year": 1900, "cast": [], "genres": []},
6  {"title": "Capture of Boer Battery by British", "year": 1900, "cast": [], "genres": ["Short", "Documentary"]},
7  {"title": "The Enchanted Drawing", "year": 1900, "cast": [], "genres": []},
8  {"title": "Feeding Sea Lions", "year": 1900, "cast": ["Paul Boyton"], "genres": []},
9  {"title": "How to Make a Fat Wife Out of Two Lean Ones", "year": 1900, "cast": [], "genres": ["Comedy"]},
10 {"title": "New Life Rescue", "year": 1900, "cast": [], "genres": []},
11 {"title": "New Morning Bath", "year": 1900, "cast": [], "genres": []},
12 {"title": "Searching Ruins on Broadway, Galveston, for Dead Bodies", "year": 1900, "cast": [], "genres": []},
13 {"title": "The Tribulations of an Amateur Photographer", "year": 1900, "cast": [], "genres": []},
14 {"title": "Trouble in Hogan's Alley", "year": 1900, "cast": [], "genres": ["Comedy"]},
```

Simple JSON Ingestion – case10



JSON Ingestion – Simple Data

1 `select * from case10 where genres is not null and "cast" is not null and genres='Comedy'`

Run ... Auto run Smart query limit 3,965 results in 0.79s

_time	cast	count	genres	title	year
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Fred MacMurray"]	1	Comedy	Romantic Honeymoon	1943
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Fred MacMurray"]	1	Comedy	No Time for Love	1943
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Fred MacMurray"]	1	Comedy	Practically Yours	1944
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Fred MacMurray"]	1	Comedy	The Egg and I	1947
2010-01-01T00:00:00.000Z	["Claudette Colbert", "George Brent"]	1	Comedy	The Phantom President	1932
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Herbert Marshall"]	1	Comedy	Zaza	1939
2010-01-01T00:00:00.000Z	["Claudette Colbert", "John Wayne"]	1	Comedy	Without Reservations	1946
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Maurice Chevalier"]	1	["Comedy", "Drama"]	The Big Pond	1930
2010-01-01T00:00:00.000Z	["Claudette Colbert", "Richard Barthelmess"]	1	Comedy	Three-Cornered Moon	1933

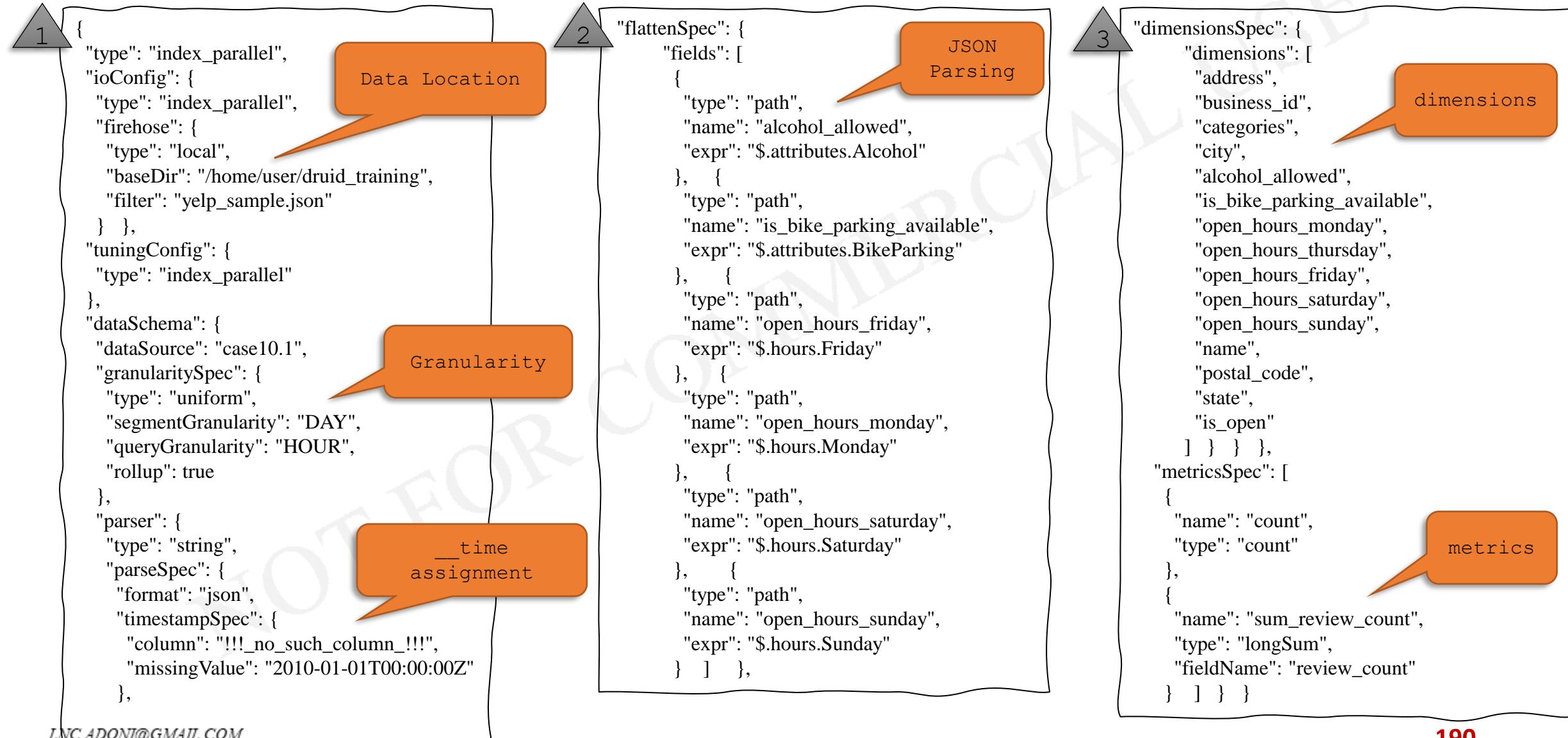
Previous Page 10 of 40 100 rows Next

Ingestion – JSON (Nested)

JSON Ingestion – Sample Nested Data (Yelp data)

```
{  
    "business_id": "QXAEGFB4oINsVuTFxEYKFQ",  
    "name": "Emerald Chinese Restaurant",  
    "address": "30 Eglinton Avenue W",  
    "city": "Mississauga",  
    "state": "ON",  
    "postal_code": "L5R 3E7",  
    "latitude": 43.6054989743,  
    "longitude": -79.652288909,  
    "stars": 2.5,  
    "review_count": 128,  
    "is_open": 1,  
    "attributes": {  
        "RestaurantsReservations": "True",  
        "GoodForMeal": "{ 'dessert': False, 'lateral': False, 'lunch': True, 'dinner': True, 'brunch': False, 'breakfast': False }",  
        "BusinessParking": "{ 'garage': False, 'street': False, 'validated': False, 'lot': True, 'valet': False }",  
        "Caters": "True",  
        "NoiseLevel": "u'loud'",  
        "RestaurantsTableService": "True",  
        "RestaurantsTakeOut": "True",  
        "RestaurantsPriceRange2": "2",  
        "OutdoorSeating": "False",  
        "BikeParking": "False",  
        "Ambience": "{ 'romantic': False, 'intimate': False, 'classy': False, 'hipster': False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': True }",  
        "HasTV": "False",  
        "WiFi": "u'no'",  
        "GoodForKids": "True",  
        "Alcohol": "u'full_bar'",  
        "RestaurantsAttire": "u'casual'",  
        "RestaurantsGoodForGroups": "True",  
        "RestaurantsDelivery": "False"  
    },  
    "categories": ["Specialty Food", "Restaurants", "Dim Sum", "Imported Food", "Food", "Chinese", "Ethnic Food", "Seafood"],  
    "hours": {"Monday": "9:0-0:0", "Tuesday": "9:0-0:0", "Wednesday": "9:0-0:0", "Thursday": "9:0-0:0", "Friday": "9:0-1:0", "Saturday": "9:0-1:0", "Sunday": "9:0-0:0"}  
}
```

Nested JSON Ingestion – case10.1



JSON Ingestion – Nested Data

1 `select * from "case10.1"`

Run Auto run Smart query limit 8 results in 0.10s

_time	address	alcohol_allowed	business_id	categories	city	count	is_b
2010-01-01T00:00:00.000Z	10110 Johnston Rd, S	u'beer_and_wine'	gnKjwL_1w79qoiV3IC_xQQ	["Japanese","Restaurants","Sushi Bars"]	Charlotte	1	True
2010-01-01T00:00:00.000Z	15655 W Roosevelt S		xvX2CttrVhyG2z1dFg_0xw	["Financial Services","Insurance"]	Goodyear	1	
2010-01-01T00:00:00.000Z	20 Douglas Woods D		5JucpCfHZltjh5r1jabjDg	["Beauty & Spas","Hair Salons"]	Calgary	1	
2010-01-01T00:00:00.000Z	30 Eglinton Avenue N	u'full_bar'	QXAEGFB4oINsVuTFxEYKFQ	["Chinese","Dim Sum","Ethnic Food","Fast Food"]	Mississauga	1	False
2010-01-01T00:00:00.000Z	4209 Stuart Andrew E		HhxOkGAM07SRYtlQ4wMF	["Home & Garden","Home Services","Hotels"]	Charlotte	1	
2010-01-01T00:00:00.000Z	4545 E Tropicana Rd		gbQN7vr_caG_A1ugSmGhW	["Barbers","Beauty & Spas","Cosmetics"]	Las Vegas	1	False
2010-01-01T00:00:00.000Z	5940 W Union Hills Dr		Y6iyemLX_oylRpnrr38vgMA	["Beauty & Spas","Day Spas","Nail Salons"]	Glendale	1	False
2010-01-01T00:00:00.000Z	Credit Valley Town Pl		68dUKd8_8liJ7in4aWOSEA	["Couriers & Delivery Services","Local Businesses"]	Mississauga	1	

◀ ▶ Previous Page 1 of 1 20 rows ▲ Next

JSON Ingestion – Nested Data

```
1 select city, state, open_hours_friday, count(business_id)
2 from "case10.1" where upper(categories)='RESTAURANTS'
3 group by 1,2,3
```

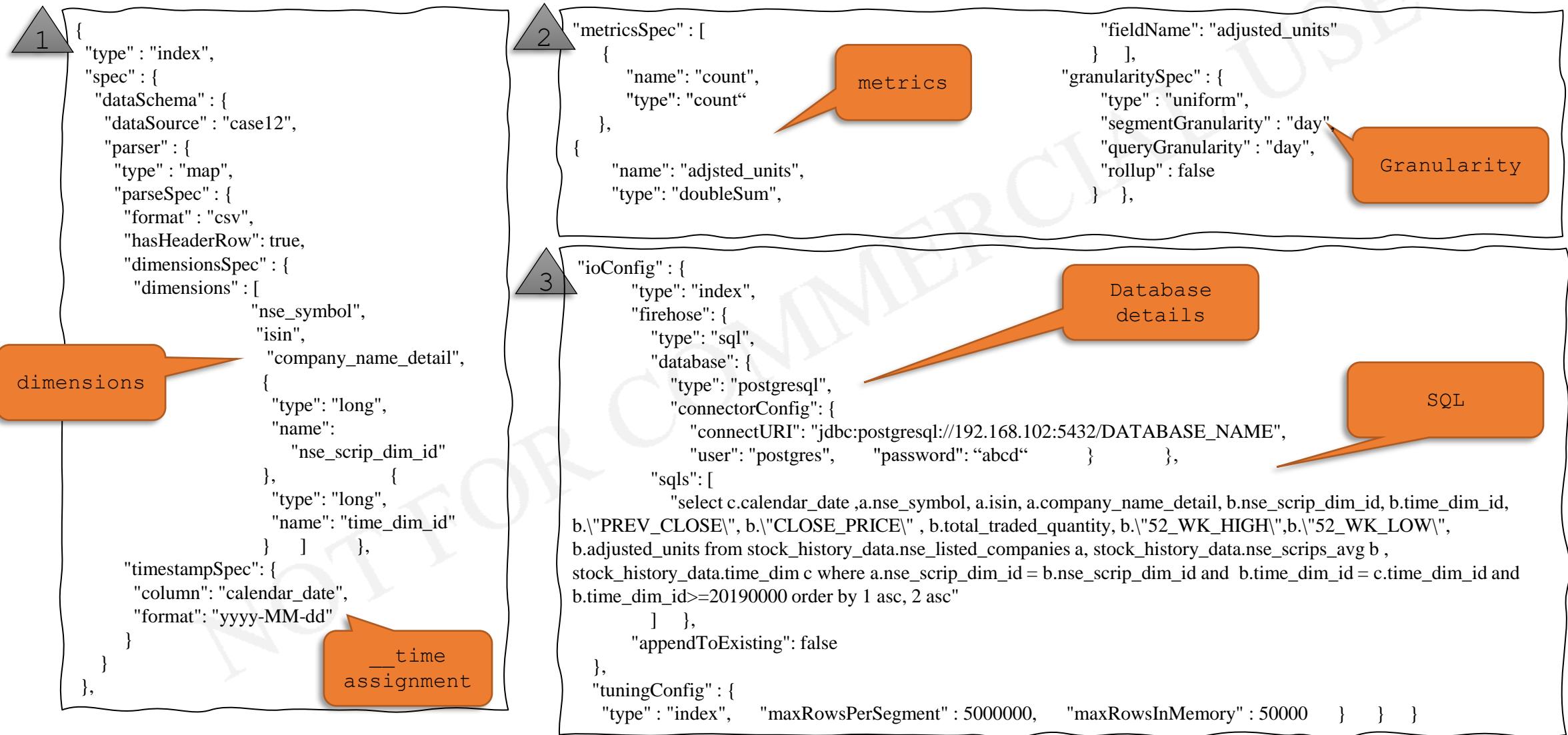
Run ... Auto run Smart query limit 2 results in 0.16s ⏪

city	state	open_hours_friday	EXPR\$3
Charlotte	NC	17:30-22:0	1
Mississauga	ON	9:0-1:0	1

Previous Page 1 of 1 20 rows ⏪ Next

Ingestion – Database

Ingestion from Database – case12



Ingestion from Database

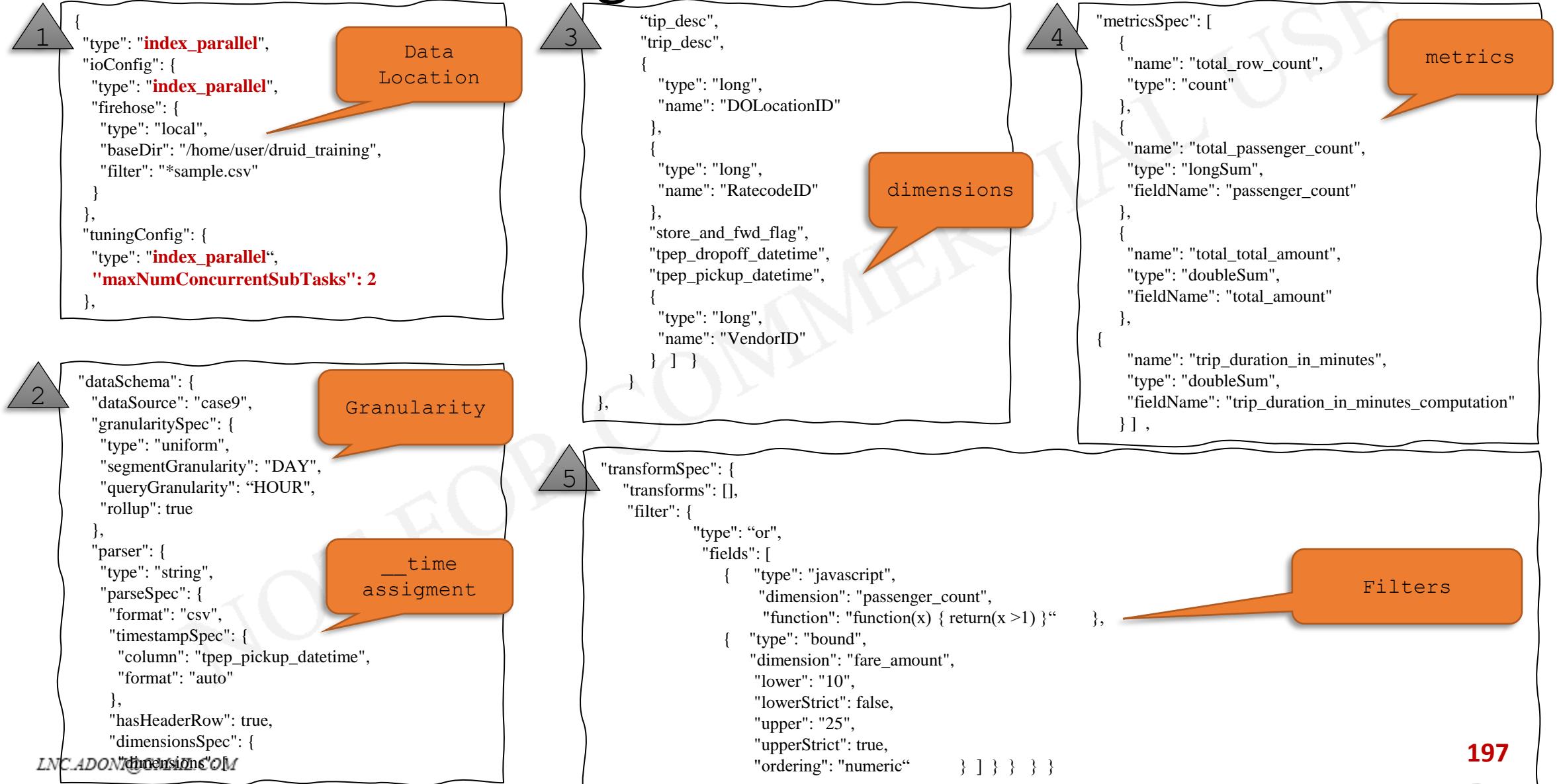
```
1 select TIME_EXTRACT(_time,'year'), TIME_EXTRACT(_time,'month'),  
2 company_name_detail, min(today_close_price), max(today_close_price) from case12  
3 group by 1 , 2, 3 order by 1 asc, 2 asc, 3 asc;
```

Run ... Auto run Smart query limit 99+ results in 1.47s ▾

XPR\$0	EXPR\$1	company_name_detail	EXPR\$3	EXPR\$4
2019	1	20 Microns Limited	36.9	44.25
2019	1	21st Century Management Services	21.55	23.8
2019	1	3M India Limited	19985.75	20984.5
2019	1	3P Land Holdings Limited	10.5	12.25
2019	1	3i Infotech Limited	3.3	3.9
2019	1	5Paisa Capital Limited	241.35	313.35
2019	1	63 Moons Technologies Limited	82.1	99.6
2019	1	8K Miles Software Services Limited	114.3	159.4
2019	1	A2Z INFRATECH ENGINEERING LIMITED	0.45	12.05

Batch Ingestion – Native Parallel Indexing

Native Parallel Ingestion – case9



Native Ingestion : Index vs Index_parallel

Index	Index_Parallel
Each task (ingestion job) is single threaded	Each task can run in parallel. Depending on (maxNumConcurrentSubTasks>1)
It can append or overwrite segments (appendToExisting)	It can append or overwrite segments (appendToExisting)
No dependencies on external components	No dependencies on external components
Input: local, http, RDBMS, existing segment	Input: local, http, RDBMS, existing segment
File Format: csv, tsv, json	File Format: csv, tsv, json

From 0.16 onwards, when a task us submitted in index_parallel mode, based on maxNumConcurrentSubTassks; a number of threads are launched in parallel which run within a single JVM parent task.

Batch Ingestion – Hadoop based Indexing

Hadoop Based Ingestion

1

```
{  
  "type": "index_hadoop",  
  "ioConfig": {  
    "type": "hadoop",  
    "inputSpec": {  
      "type": "static",  
      "paths": "/home/user/druid_training/*sample.csv"  
    }  
  },  
  "tuningConfig": {  
    "type": "hadoop",  
    "partitionsSpec": {  
      "type": "hashed",  
      "targetPartitionSize": 5000000  
    },  
    "forceExtendableShardSpecs": true,  
    "jobProperties": {  
      "mapreduce.job.classloader": "true",  
      "io.compression.codecs":  
        "org.apache.hadoop.io.compress.GzipCodec,org.apache.hadoop.io.compress.DefaultCodec,org.apache.hadoop.io.compress.BZip2Codec,org.apache.hadoop.io.compress.SnappyCodec",  
      "mapreduce.map.java.opts": "-Duser.timezone=UTC -Dfile.encoding=UTF-8",  
      "mapreduce.job.user.classpath.first": "true",  
      "mapreduce.reduce.java.opts": "-Duser.timezone=UTC -Dfile.encoding=UTF-8",  
      "hdp.version": "3.1.0.0-78",  
      "mapreduce.job.classloader.system.classes": "-javax.validation,java,javax,org.apache.commons.logging,,org.apache.log4j,,org.apache.hadoop."  
    }  
  },  
}
```

Data Location

Hadoop Configurations

2

```
{  
  "dataSchema": {  
    "dataSource": "case12",  
    "granularitySpec": {  
      "type": "uniform",  
      "segmentGranularity": "DAY",  
      "queryGranularity": "HOUR",  
      "rollup": true  
    },  
    "parser": {  
      "type": "hadoopString",  
      "parseSpec": {  
        "format": "csv",  
        "timestampSpec": {  
          "column": "tpep_pickup_datetime",  
          "format": "auto"  
        },  
        "hasHeaderRow": true,  
        "dimensionsSpec": {  
          "dimensions": [  
            "tpep_dropoff_datetime",  
            "tpep_pickup_datetime",  
            {  
              "type": "long",  
              "name": "VendorID"  
            }  
          ]  
        },  
        "metricsSpec": [  
          {  
            "name": "trip_duration_in_minutes",  
            "type": "doubleSum",  
            "fieldName": "trip_duration_in_minutes_computation"  
          }  
        ]  
      },  
      "hadoopDependencyCoordinates": ["org.apache.hadoop:hadoop-client:2.8.3"]  
    }  
  }  
}
```

_time assignment

dimensions

metrics

Copy **core-site.xml**, **hdfs-site.xml**, **yarn-site.xml**, **mapred-site.xml** files to <druid home>/conf/druid/cluster/_common

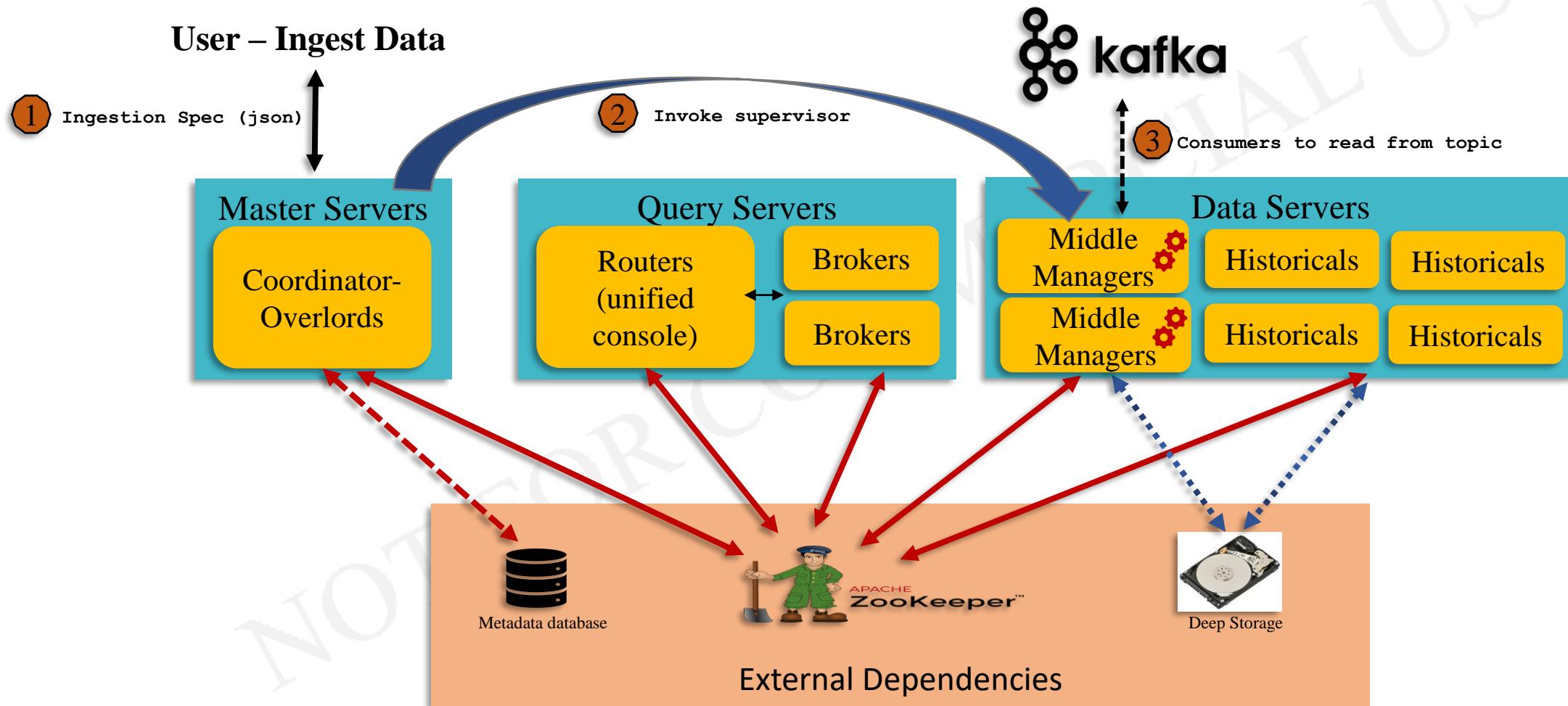
LIVEALONE@GMAIL.COM

Native Ingestion : Index vs Index_parallel

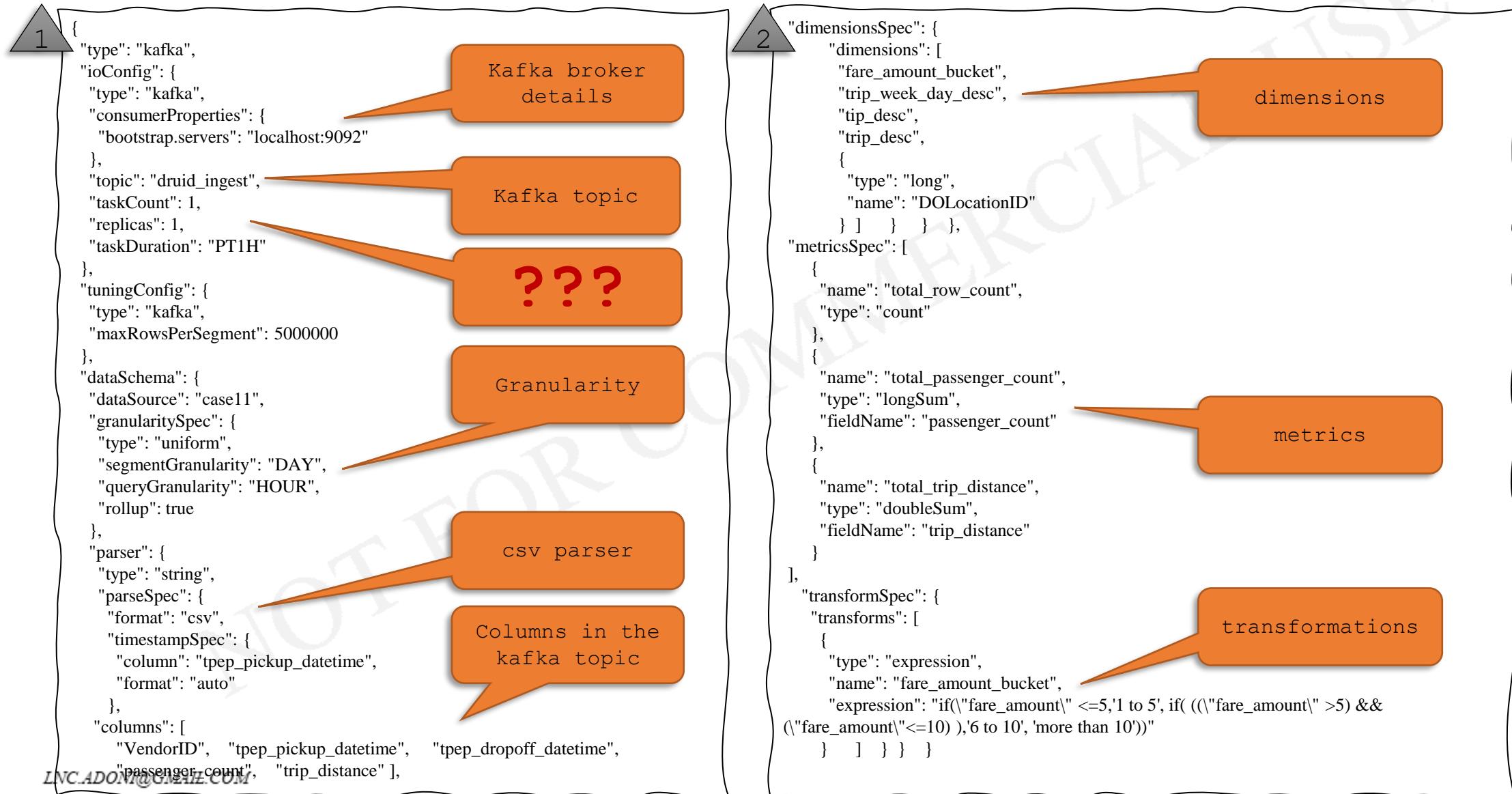
Index	Index_Parallel	Hadoop Based
Each task (ingestion job) is single threaded (index)	Each task can run in parallel. Depending on (maxNumConcurrentSubTasks>1) (index_parallel)	Each task can run in parallel. (index.hadoop)
It can append or overwrite segments (appendToExisting)	It can append or overwrite segments (appendToExisting)	It can only overwrite segments.
No dependencies on external components	No dependencies on external components	Depends on Hadoop cluster (Druid submits MapReduce job)
Input: local, http, RDBMS, existing segment	Input: local, http, RDBMS, existing segment	Input: All native data sources and also Hadoop based files system (HDFS)
File Format: csv, tsv, json	File Format: csv, tsv, json	File Format: csv, tsv, json, Parquet, ORC, Avro...

Realtime Ingestion – Kafka

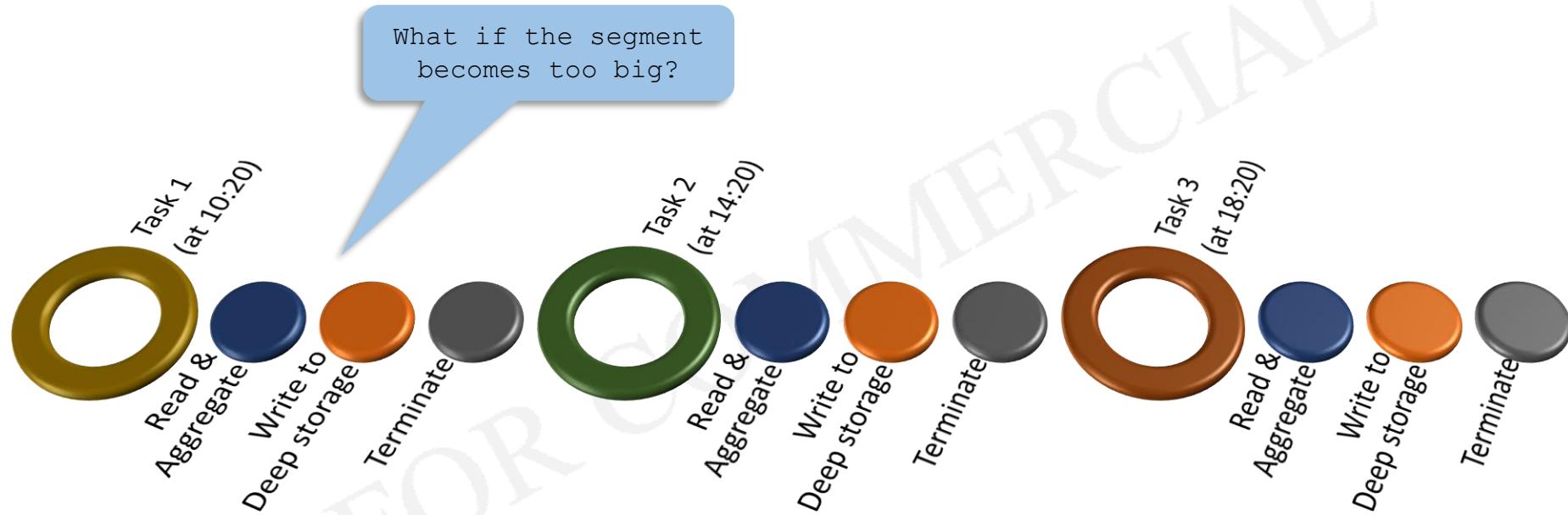
Realtime Ingestion - Kafka



Realtime Ingestion - Kafka

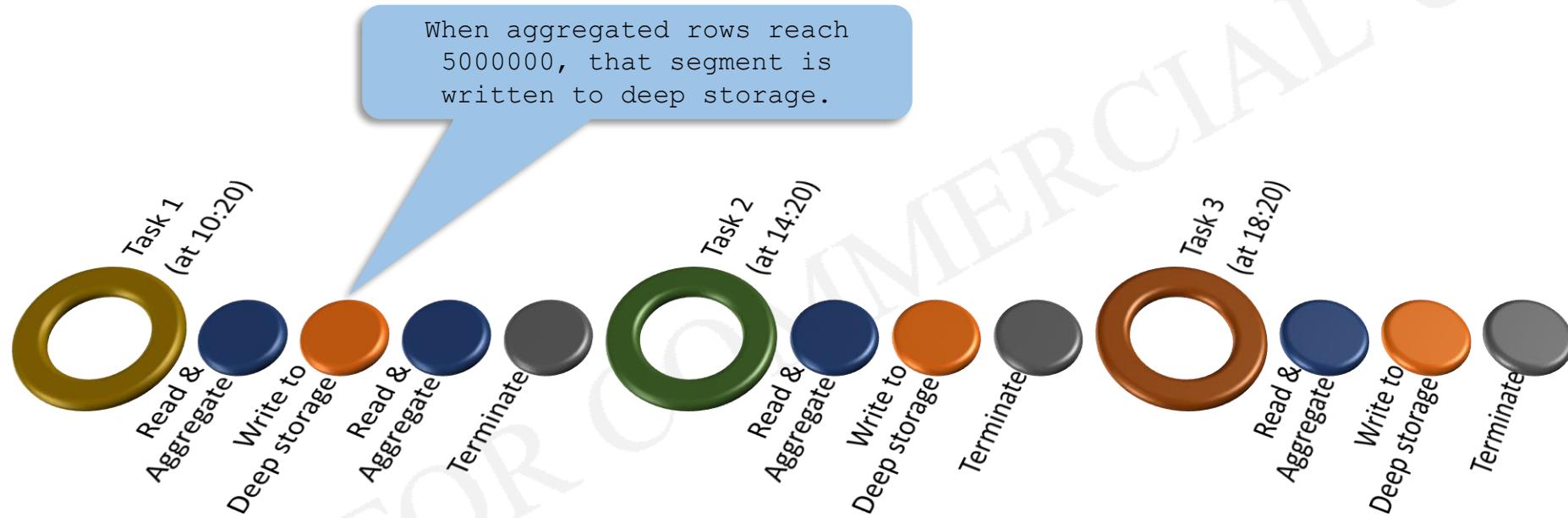


Realtime Ingestion – Segments Handoff



taskDuration: PT4H

Realtime Ingestion – Segments Handoff



taskDuration: PT4H

maxRowsPerSegment: 5000000

Realtime Ingestion – Kafka Configuration

- replicas
 - Defines on how many replicas to run across middlemanagers. If set to 1, then only 1 middlemanager will run the supervisor. It is recommended to run on multiple middlemanagers (replicas > 1) for resilience.
- taskCount
 - How many tasks to run in each middlemanager. Each task would be responsible for a kafka partition.
- taskDuration
 - Duration for each task. If set to 1hr, a task runs for only 1 hour. Then a new task is initiated.
 - Each task writes to deepstorage (before exiting); till then the middlemanager keeps the data, read from kafka, in memory.
 - If the duration is set to 24hrs, then 24hrs data would be in memory. Only after 24hrs, the data/segment is written to deep storage.

Realtime Ingestion – Segments Handoff

The screenshot shows a database interface with three separate query panes and their corresponding results.

Query 1:

```
1 select __time, count(*) from case11 group by __time order by __time asc|
```

Query 2:

```
1 select __time, count(*) from case11 group by __time order by __time asc|
```

Query 3:

```
1 select __time, count(*) from case11 group by __time order by __time asc|
```

Run Buttons and Auto Settings:

- Top-left pane: Run button, three-dot menu, Auto toggle (disabled).
- Middle-left pane: Run button, three-dot menu, Auto toggle (disabled).
- Bottom-right pane: Run button, three-dot menu, Auto run toggle (enabled), Smart query limit toggle (enabled).

Result Data:

__time	count(*)
2019-09-09T19:00:00.000Z	3
2019-10-09T19:00:00.000Z	1
2019-10-08T15:00:00.000Z	1
2019-10-09T19:00:00.000Z	1

Footnote:

LNC.ADONI@GMAIL.COM

Ingest from Existing Segments

Segment to Segment Ingestion

Cube: case9

Dimensions: 11

```
fare_amount_bucket,  
trip_week_day_desc,  
tip_desc,  
trip_desc,  
DOLocationID,  
payment_type,  
PULocationID,  
RatecodeID,  
store_and_fwd_flag,  
tpep_dropoff_datetime,  
VendorID
```

Metrics: 12

```
total_row_count,  
total_passenger_count,  
total_trip_distance,  
total_fare_amount,  
total_extra,  
total_mta_tax,  
total_tip_amount,  
total_tolls_amount,  
total_improvement_surcharge,  
total_total_amount,  
total_congestion_surcharge,  
trip_duration_in_minutes
```

Cube: case20

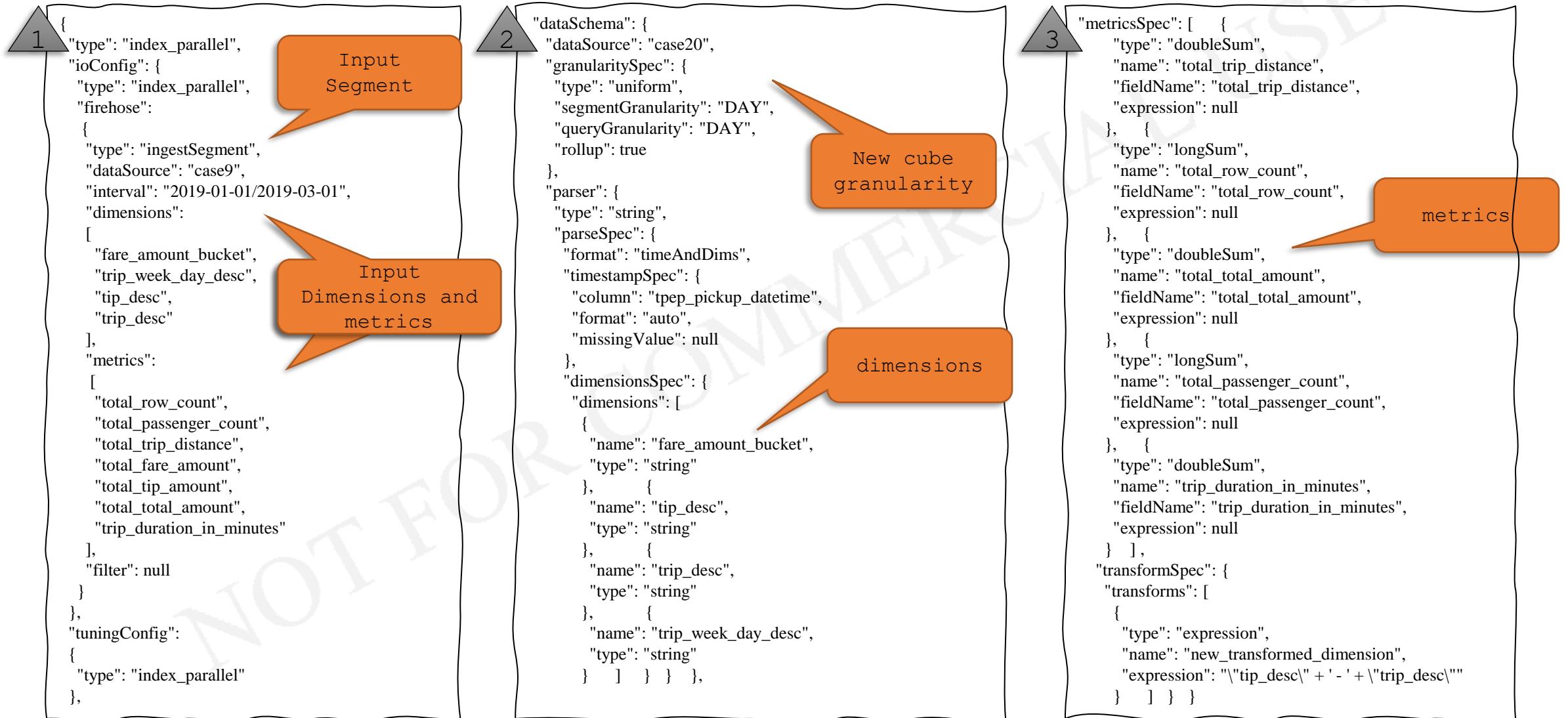
Dimensions: 4

```
fare_amount_bucket,  
trip_week_day_desc,  
tip_desc,  
trip_desc
```

Metrics: 5

```
total_row_count,  
total_trip_distance,  
total_total_amount,  
total_passenger_count,  
trip_duration_in_minutes
```

Ingest from Existing Segment



Cubes comparison

The screenshot shows the Druid UI with the 'Datasources' tab selected. The page title is 'Datasources'. There are two data rows:

Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
"case20"	Fully available (59 segments)	No segments to load/dr...	Cluster default: loadFore...	48.26 MB	48.26 MB	None	817.99 KB	1,212,028	🔍 🔧

The screenshot shows the Druid UI with the 'Datasources' tab selected. The page title is 'Datasources'. There are two data rows:

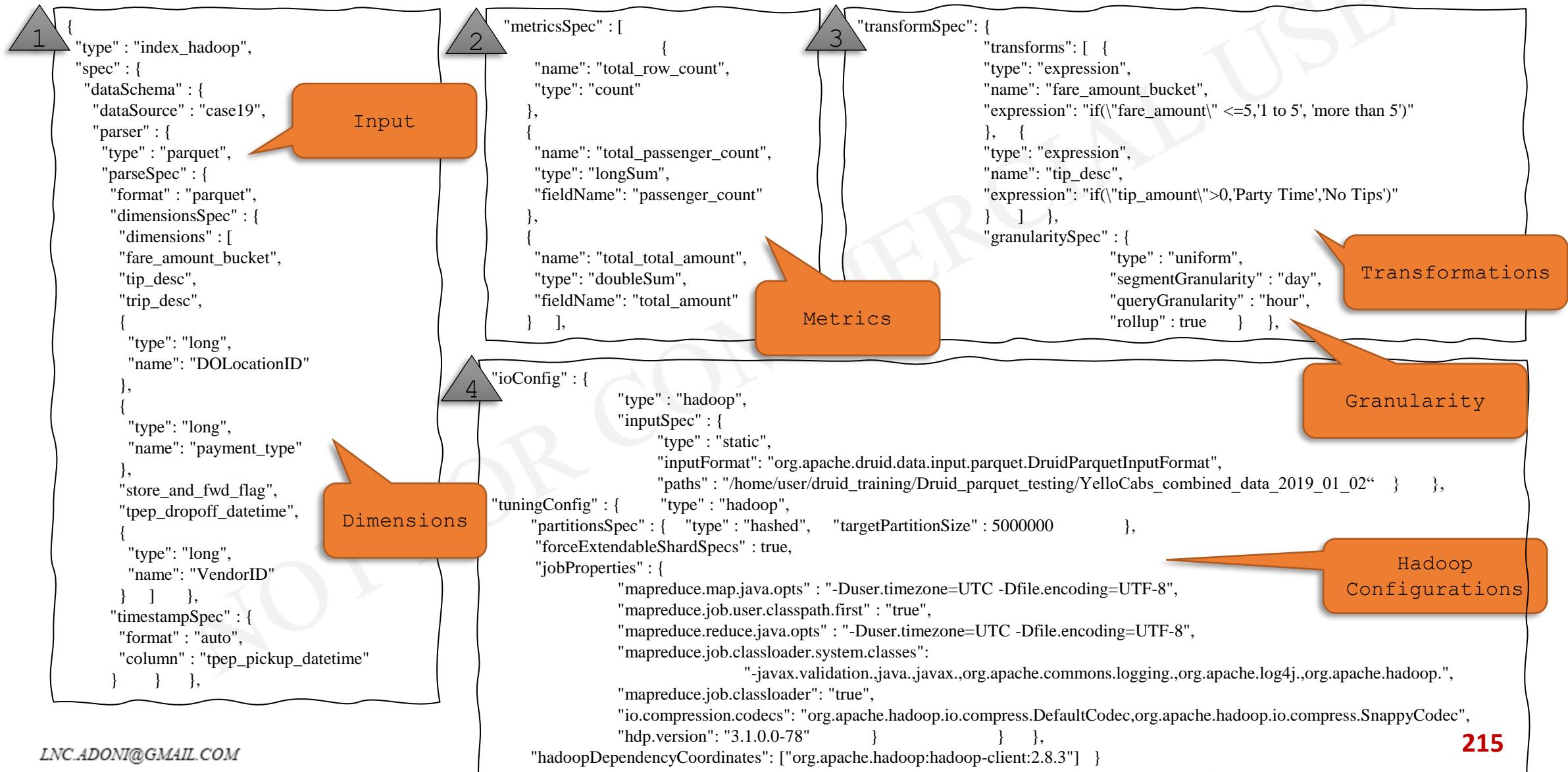
Datasource	Availability	Segment load/drop	Retention	Replicated size	Size	Compaction	Avg. segment...	Num rows	Actions
"case9"	Fully available (141 segments)	No segments to load/dr...	Cluster default: loadFore...	505.60 MB	505.60 MB	None	3.59 MB	14,680,221	🔍 🔧

Ingest Parquet data on
Non-Hadoop cluster

Ingest Parquet on Non-Hadoop Cluster

- Requirement
 - No Hadoop Cluster
 - But data is in parquet format.
- Solution
 - Remove *core-site.xml*, *hdfs-site.xml*, *yarn-site.xml*, *mapred-site.xml* from *<druid home>/conf/druid/cluster/_common*
 - Use Hadoop ingestion spec

Ingest Parquet on Non-Hadoop Cluster – case19



Order of execution of ingestionSpec

- **flattenSpec**
 - Used for json data parsing.
- **timestampSpec**
 - Mapping of timestamp data to **time**.
- **transformSpec**
 - Transformations on input columns.
 - Transformations are done per row.
 - Transformations can be used for both dimensions and metrics.
- **dimensionSpec**
 - Listing of dimensions and data type of dimensions.
- **metricsSpec**
 - Listing of metrics and aggregation used to computed metrics (only when `rollup=true`).

Drop Segments

Drop Segments

```
1 select TIME_EXTRACT(__time,'YEAR'),TIME_EXTRACT(__time,'month'),TIME_EXTRACT(__time,'day'), count(*)  
2 from case1 group by 1,2,3|
```

Run



Auto run



Smart query limit

3 results in 0.12s

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019	1	1	3
2019	1	2	4
2019	1	3	3

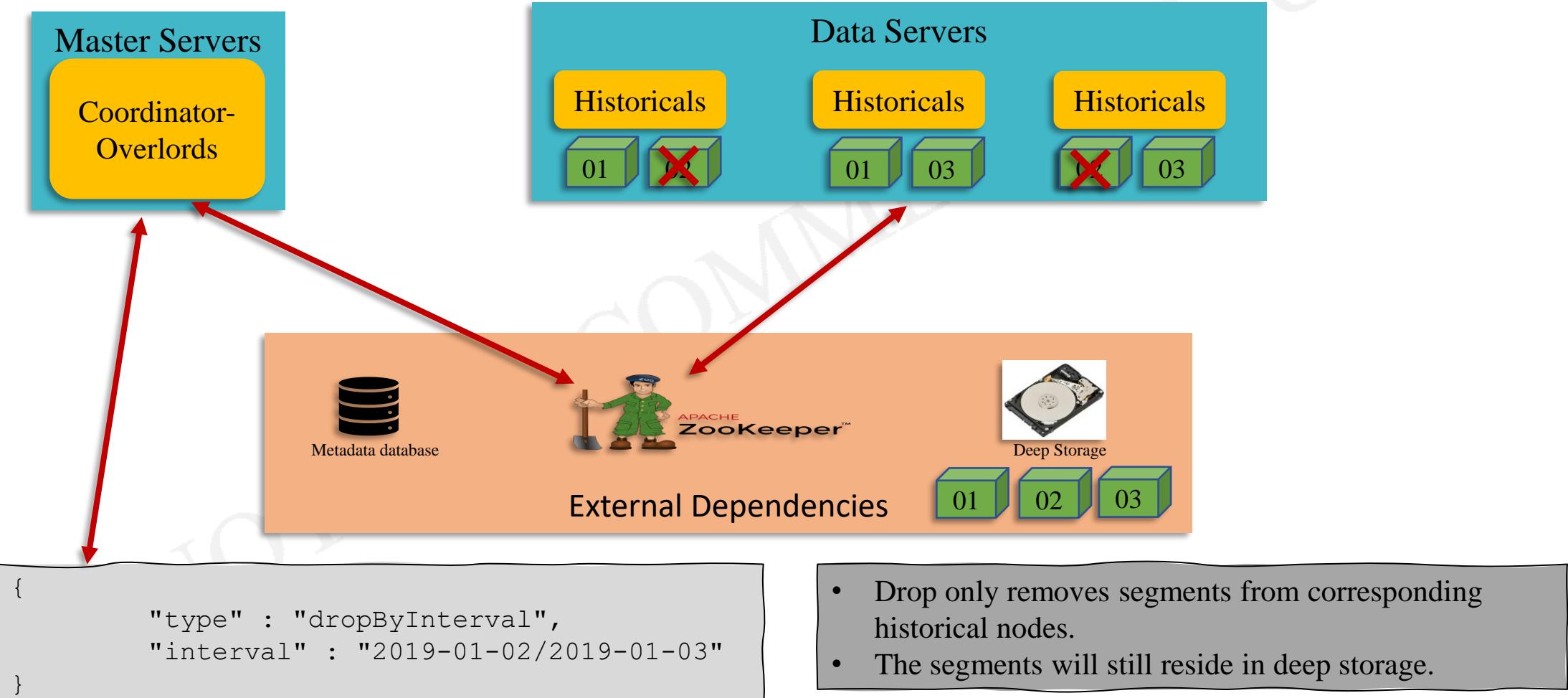
Previous

Page 1 of 1

20 rows

Next

Drop Segments



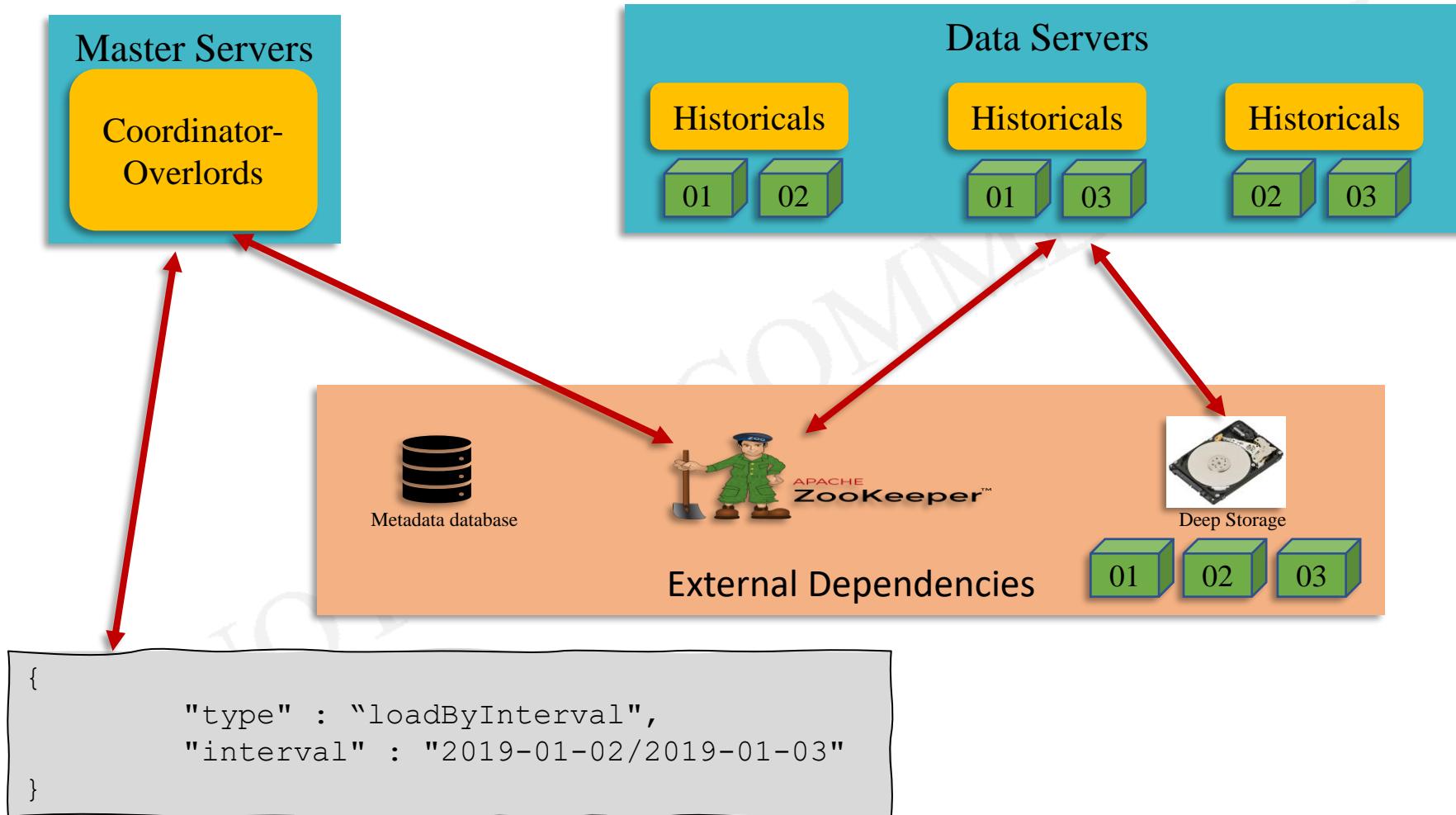
Drop Segments – After Drop

```
1 select TIME_EXTRACT(__time,'YEAR'),TIME_EXTRACT(__time,'month'),TIME_EXTRACT(__time,'day'), count(*)  
2 from case1 group by 1,2,3
```

The screenshot shows a data analysis interface with a dark theme. At the top, there is a code editor window containing a SQL query. Below the code editor are several controls: a 'Run' button, a three-dot menu button, an 'Auto run' toggle switch, and a 'Smart query limit' toggle switch. To the right of these controls, it says '2 results in 0.23s' with a download icon. The main area displays a table with four columns labeled EXPR\$0, EXPR\$1, EXPR\$2, and EXPR\$3. The table has two rows of data. The first row shows '2019' in EXPR\$0, '1' in EXPR\$1, '1' in EXPR\$2, and '3' in EXPR\$3. The second row also shows '2019' in EXPR\$0, '1' in EXPR\$1, '3' in EXPR\$2, and '3' in EXPR\$3. At the bottom of the interface, there are navigation buttons for 'Previous', 'Page 1 of 1', '20 rows' (with a dropdown arrow), and 'Next'.

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019	1	1	3
2019	1	3	3

Reload Dropped Segments



Reload Dropped Segments

```
1 select TIME_EXTRACT(__time,'YEAR'),TIME_EXTRACT(__time,'month'),TIME_EXTRACT(__time,'day'), count(*)  
2 from case1 group by 1,2,3
```

Run



Auto run



Smart query limit

3 results in 0.12s

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019	1	1	3
2019	1	2	4
2019	1	3	3

Previous

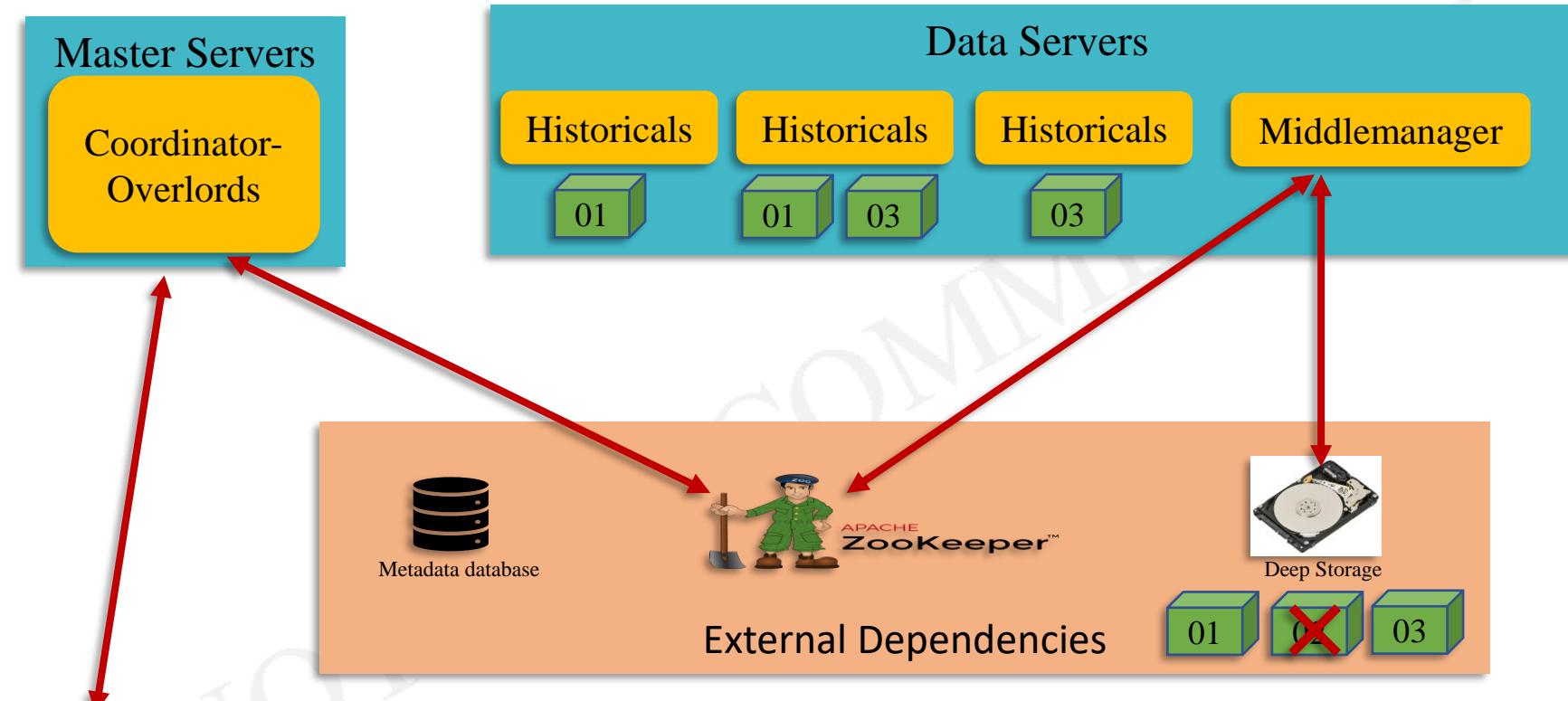
Page 1 of 1

20 rows

Next

Kill Segments – Permanently Delete Segments from Deep Storage

Drop Segments



```
{  
  "type": "kill",  
  "dataSource": "case1",  
  "interval" : "2019-01-02/2019-01-05"  
}
```

- Though interval is mentioned from 02 to 05, as only 02 segment is “dropped”, only 02 segment will be removed permanently from deep storage.
- As 03 segment will still be in use.

Ingestion Task, Query Submit

Ingestion

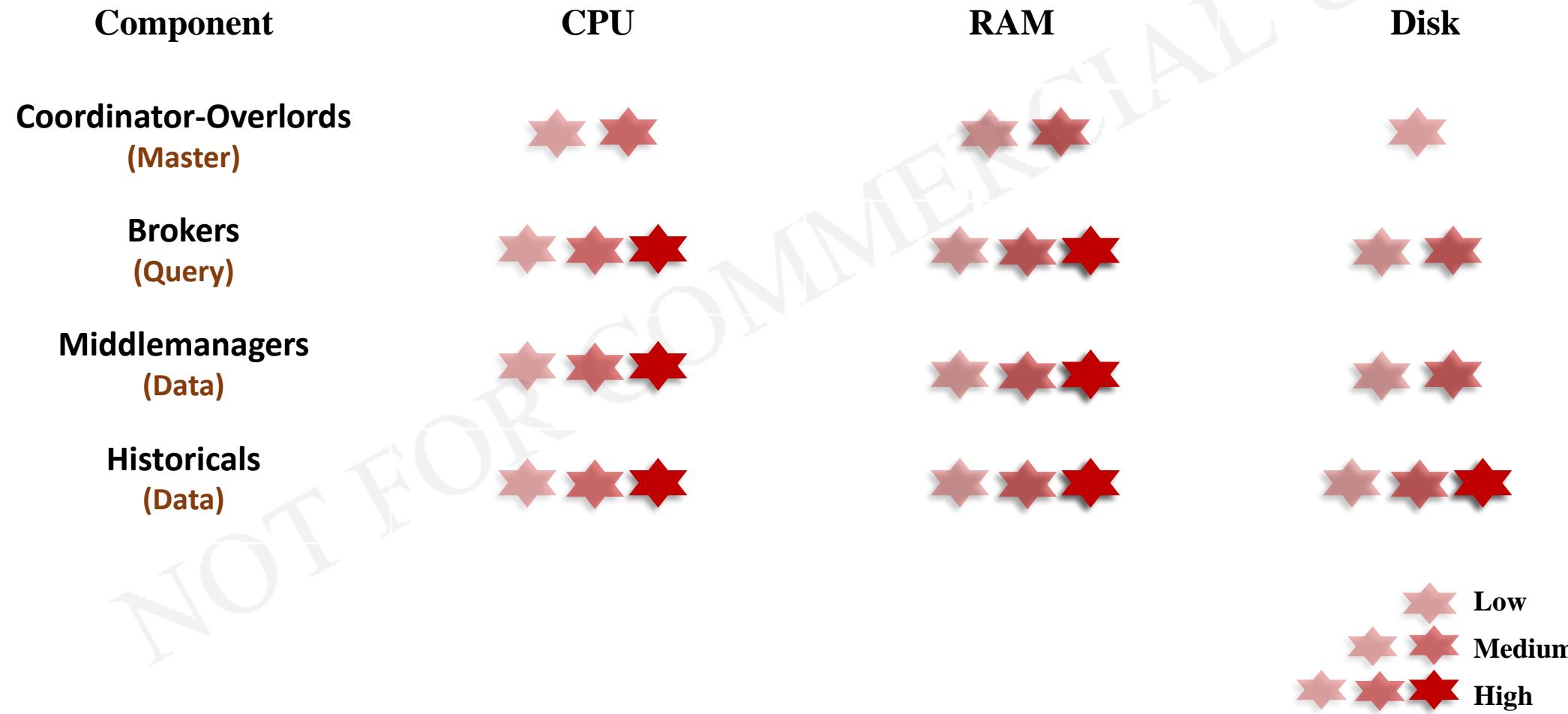
```
<druid home>/bin/post-index-task --file ingestion_spec.json --url http://localhost:8081
```

```
curl -X 'POST' -H 'Content-Type:application/json' -d @ingestion_spec.json  
http://localhost:8081/druid/indexer/v1/task
```

Query

```
curl -X POST 'http://localhost:8888/druid/v2/?pretty' -H 'Content-Type:application/json'  
-H 'Accept:application/json' -d @query.json
```

Resource Utilization by Individual Component



Apache Druid – Advance concepts

- **Why Probabilistic Algorithms**

- Introduction to HLL, DataSketch
- DataSketch – HLL, Theta
- DataSketch usage at Data Ingestion
- DataSketch usage at Query
 - In SQL & limitations.
 - Native (JSON) querying.
 - Usage of Post Aggregations on DataSketch

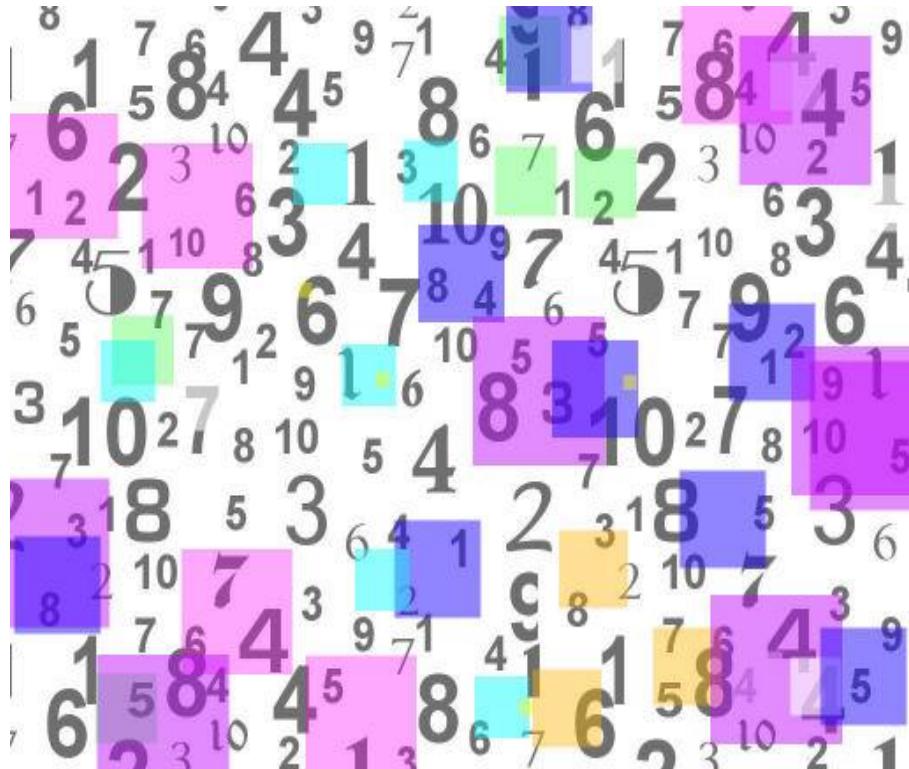
- **Demo & Live Presentation**

- Querying
 - JSON – Simple, Post aggregations, JavaScript
 - Querying on Multi Value Columns
 - Querying using Lookup – one-to-one, many-to-one, one-to-many, many-to-many
 - SQL in JSON Native querying
 - Querying using SQL Editors (DBeaver, Squirrel...)

Data is Ingested. What next?

Of course....Query/Extract

What involves Querying data



Query

Select

- Column Projection
- Dynamic Column Projection
- Aggregation

Filter

Where

Grouping

- Column Projection
- Dynamic Column Projection

Having

Aggregation Filter

```
select product_category, concat(state,'-',city) state_city,  
sum(unit_price*quantity) total_sales , count(trans_id), count(distinct  
customer_id) unique_customers from sales_cube where  
time_extract(_time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

Complexity – Filter

Query

Select

- Column Projection
- Dynamic Column Projection
- Aggregation

Filter

Where

Grouping

- Column Projection
- Dynamic Column Projection

Having

- Aggregation Filter

Solution: Inverse Indexes

```
select product_category, concat(state,'-',city) state_city,  
sum(unit_price*quantity) total_sales , count(trans_id), count(distinct  
customer_id) unique_customers from sales_cube where  
time extract( time,'year')=2019 and age > 30 group by  
product_category, state_city having unique_customers >100
```

Complexity – Column Projection

Query

Select

Column Projection

Dynamic Column Projection

Aggregation

Filter

Where

Grouping

Column Projection

Dynamic Column Projection

Having

Aggregation Filter

Solution: Dictionary
Encoding, Column Storage

```
select product_category, concat(state,'-',city) state_city,  
sum(unit_price*quantity) total_sales , count(trans_id), count(distinct  
customer_id) unique_customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

Complexity – Aggregation

Query

Select

- Column Projection
- Dynamic Column Projection
- Aggregation**

Filter

Where

Grouping

- Column Projection
- Dynamic Column Projection

Having

Aggregation Filter

Variants of Aggregation

```
select product_category, concat(state,'-',city) state_city,  
sum(unit price*quantity) total sales, count(trans id), count(distinct  
customer id) unique customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

Complexity – Aggregation Types

Query

Select

Column Projection
Dynamic Column Projection

Aggregation

Filter

Where

Grouping

Column Projection
Dynamic Column Projection

Having

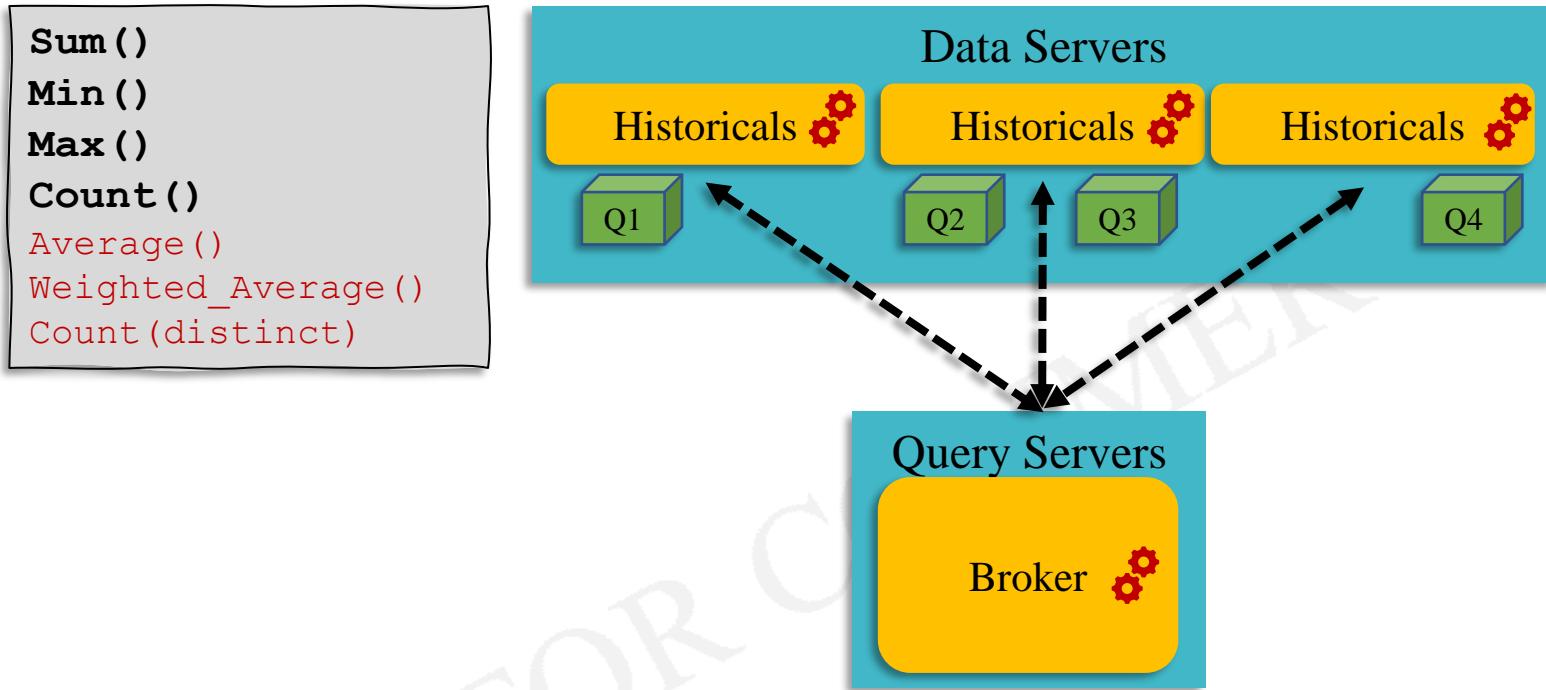
Aggregation Filter

Sum()
Min()
Max()
Count()
Average()
Weighted_Average()
Count(distinct)

Solution: ???

```
select product_category, concat(state,'-',city) state_city,  
sum(unit price*quantity) total sales, count(trans id), count(distinct  
customer id) unique customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

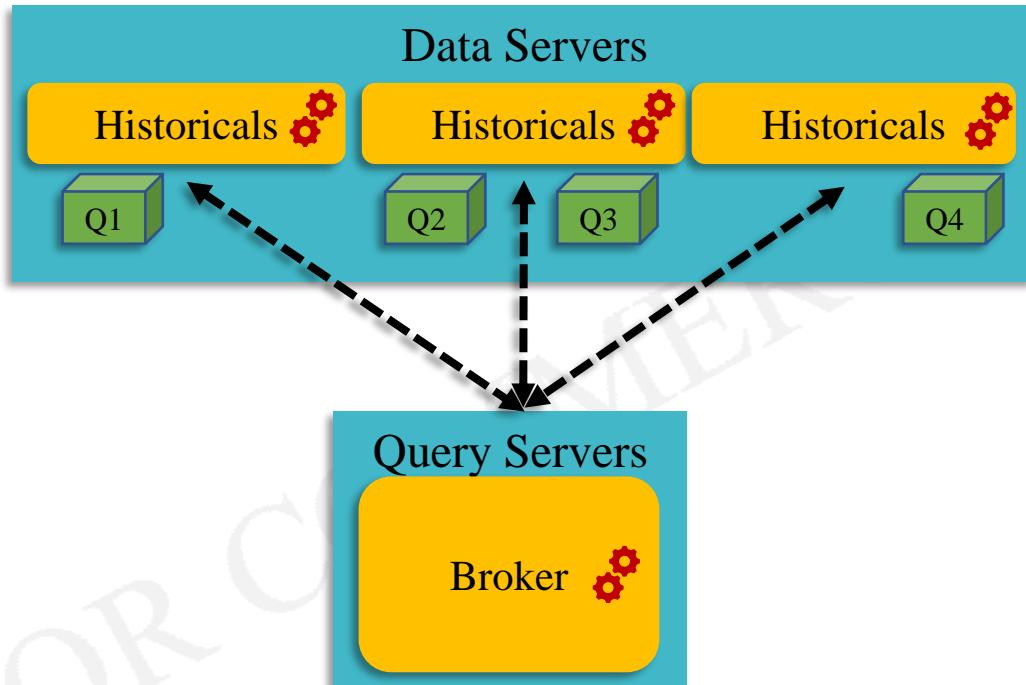
Aggregation - Additive



```
select product_category, concat(state,'-',city) state_city,  
sum(unit price*quantity) total sales, count(trans id), count(distinct  
customer id) unique customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

Aggregation – Average, Weighted Average

Sum()
Min()
Max()
Count()
Average()
Weighted_Average()
Count(distinct)



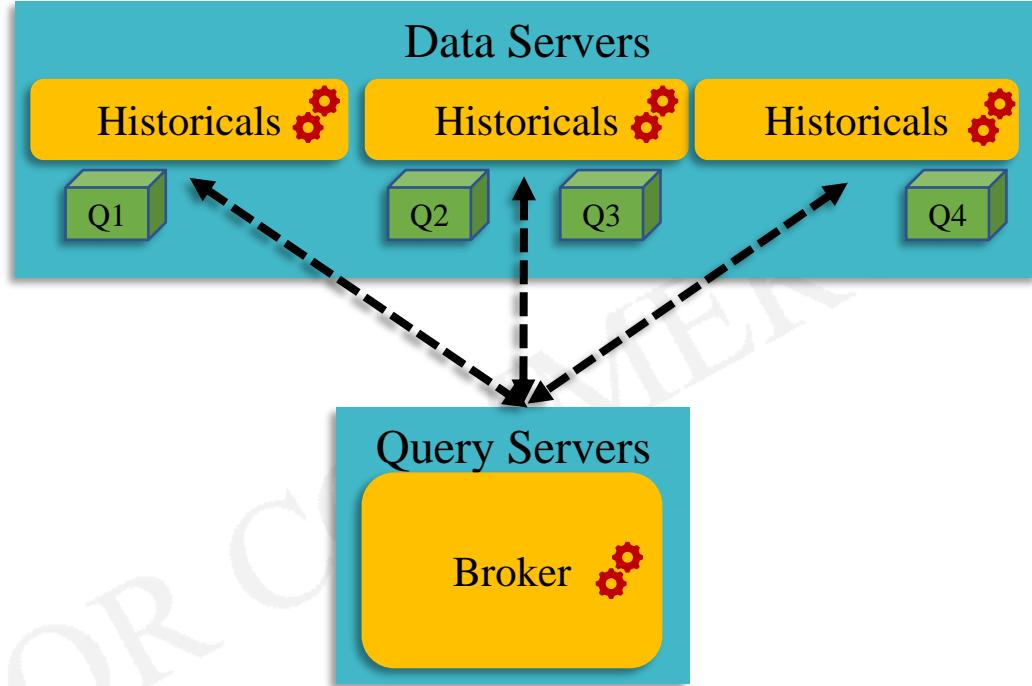
- Compute sum of numerator, sum of denominator independently in each historical node.
- Pass both aggregated values to broker.
- Broker to consolidate and further aggregate numerator and denominator.

```
select product_category, concat(state,'-',city) state_city,  
sum(unit price*quantity) total sales, count(trans id), count(distinct  
customer id) unique customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

What about `count(distinct)`

Aggregation – Count(distinct)

```
Sum()  
Min()  
Max()  
Count()  
Average()  
Weighted_Average()  
Count(distinct)
```



- Compute unique values in each historical nodes and prepare a list.
- Pass the unique list to broker
- Broker collects all unique lists from historicals, again compute unique list.
- Compute count on the final unique list.

```
select product_category, concat(state,'-',city) state_city,  
sum(unit price*quantity) total sales, count(trans id), count(distinct  
customer id) unique customers from sales_cube where  
time_extract(__time,'year')=2019 and age > 30 group by product_category,  
state_city having unique_customers >100
```

Druid – count(distinct) - Exact

- Computes exact count distinct.
- Performance intensive (CPU & RAM)
- Only one count(distinct) is allowed per query.
- Not recommended, instead used DataSketch.

```
1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4 from case21
5 group by 1
```

Run ... Auto run Smart query limit 7 results in 159.34s

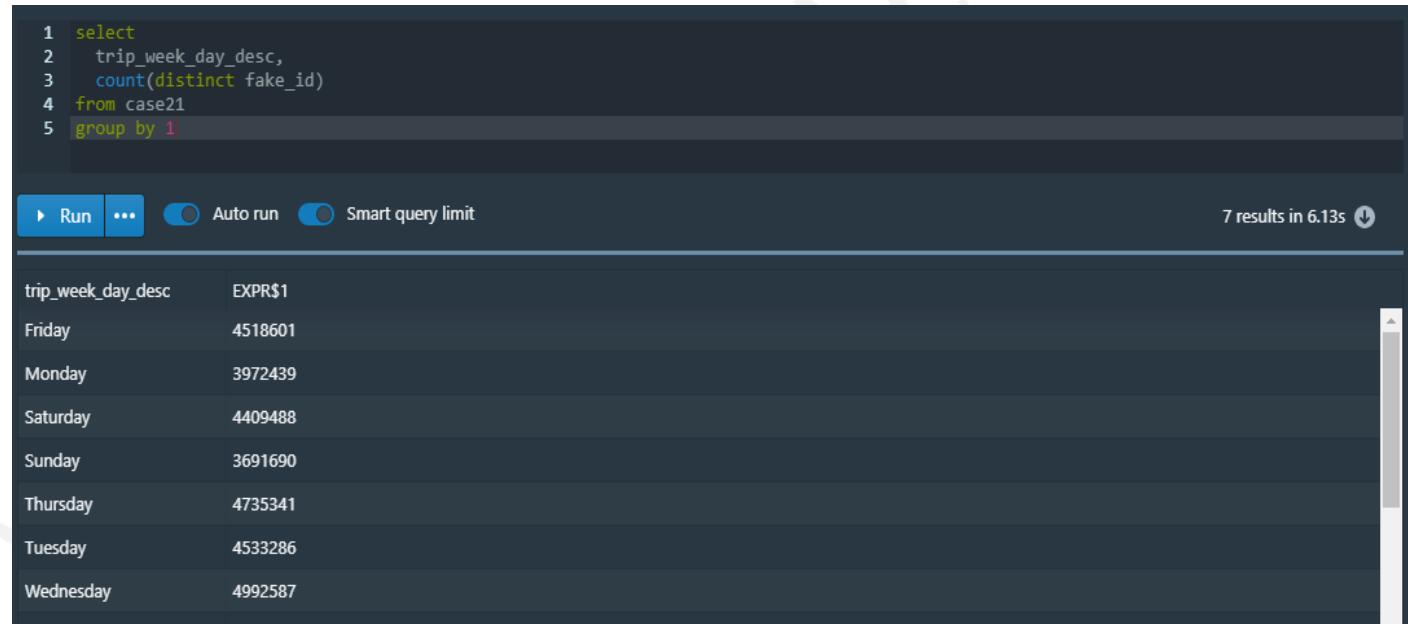
trip_week_day_desc	EXPR\$1
Friday	4619823
Monday	4047415
Saturday	4390413
Sunday	3727397
Thursday	4672737
Tuesday	4637626
Wednesday	4895761

conf/druid/single-server/micro-quickstart/broker/runtime.properties
druid.sql.planner.useApproximateCountDistinct=false (Default is true)

- count (distinct A), count (B) – allowed
- count (distinct A), count (distinct B) – not allowed
- count (distinct A), approx_count_distinct (B) – no allowed

Druid – count(distinct) - Approximate

- Uses HyperLogLog to compute approximate count distinct.
- No parameters/levers to improve the accuracy.
- **Not recommended,** instead used DataSketch.



A screenshot of a Druid SQL interface. At the top, there is a code editor with the following SQL query:

```
1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4 from case21
5 group by 1
```

Below the code editor are three buttons: "Run", "...", "Auto run", and "Smart query limit". To the right of these buttons, it says "7 results in 6.13s". The main area shows a table with the following data:

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

conf/druid/single-server/micro-quickstart/broker/runtime.properties
druid.sql.planner.useApproximateCountDistinct=true (Default is true)

Druid – APPROX_COUNT_DISTINCT()

- Uses HyperLogLog to compute approximate count distinct.
- Always computes approximate (irrespective of `useApproximateCountDistinct`)
- Not recommended, instead used DataSketch.

The screenshot shows a Druid query interface with the following code:

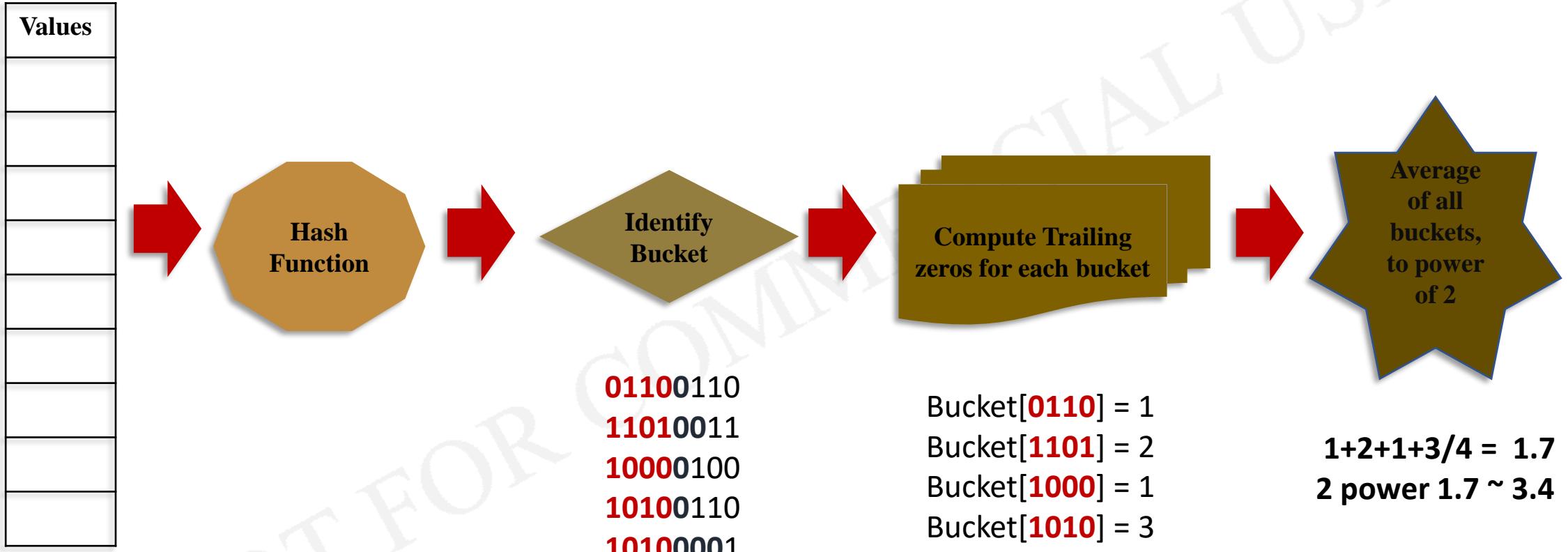
```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT(fake_id)
4 from case21
5 group by 1
```

Below the code, there are buttons for "Run" and "Smart query limit". The results section displays the following data:

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

At the bottom right, it says "7 results in 5.21s".

HyperLogLog



- HyperLogLog supports only union of HyperLogLogs'.
- Does not support intersect operations.
- Implementation of HLL may vary based on application.

Druid: count(distinct) Performance & Accuracy

Count(Distinct)

```
1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4 from case21
5 group by 1
```

Run ... Auto run

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

6.13s

druid.sql.planner.useApproximateCountDistinct=true

- Default behavior, i.e uses HLL approximation algorithm.
- Always returns approximate count.

Count(Distinct)

```
1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4 from case21
5 group by 1
```

Run ... Auto run

trip_week_day_desc	EXPR\$1
Friday	4619823
Monday	4047415
Saturday	4390413
Sunday	3727397
Thursday	4672737
Tuesday	4637626
Wednesday	4895761

159.34s

druid.sql.planner.useApproximateCountDistinct=false

- Computes exact count distinct, performance intensive.
- Not recommended

Approx_Count_Distinct

```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT(fake_id)
4 from case21
5 group by 1
```

Run ... Auto run Smart que

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

5.21s

- Always uses HLL, and computes approximate count distinct.
- No way to get exact count.

DataSketch

DataSketch

- Developed and open sourced by Yahoo!
- Two variants of DataSketchs
 - HLL
 - Thetha
- DataSketch – Theta (`APPROX_COUNT_DISTINCT_DS_THETA(expr, size)`)
 - Levers are provided to improve accuracy, but accuracy is not guaranteed.
 - “size” parameter must be power of 2. Default is 16384. Higher the size, better accuracy and more space requirement for sketch.
 - Supports **Union**, **Intersect** and **NOT** on these datasketches.
- DataSketch – HLL (`APPROX_COUNT_DISTINCT_DS_HLL(expr, logK, HLLType)`)
 - Levers are provided to improve accuracy, but accuracy is not guaranteed.
 - logK represents number of buckets. Must be from 4 to 21. Default is 12.
 - HLLType represents type of HLL Sketch. Must be “HLL_4”, “HLL_6”, “HLL_8”. Default is “HLL_4”.

DS – PPROX_COUNT_DISTINCT_DS_THETA

- Always computes approximate (irrespective of `useApproximateCountDistinct`)
- Increase “size” parameter to increase accuracy.
- Not parameter to instruct to the algorithm to return exact count distinct.
- Possible to intersect DS-Theta (only using Native Json Query, not SQL as of 0.16 version).
- No formula/approach to identify ideal “size” parameter. It differs from case to case.

The screenshot shows a data analysis interface with two separate query sections. Each section includes a code editor at the top, a toolbar below it, and a results table below that.

Top Section:

```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_THETA(fake_id )
4   from case21
5   group by 1
```

Toolbar: Run, Auto run, Smart query limit. Results: 7 results in 1.69s.

trip_week_day_desc	EXPR\$1
Friday	4661543
Monday	4023947
Saturday	4418994
Sunday	3742394
Thursday	4682777
Tuesday	4663698
Wednesday	5036394

Bottom Section:

```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_THETA(fake_id ,32768)
4   from case21
5   group by 1
```

Toolbar: Run, Auto run, Smart query limit. Results: 7 results in 2.49s.

trip_week_day_desc	EXPR\$1
Friday	4630111
Monday	4075038
Saturday	4422333
Sunday	3756141
Thursday	4676486
Tuesday	4654580
Wednesday	4946453

DS – PPROX_COUNT_DISTINCT_DS_HLL

- Always computes approximate (irrespective of `useApproximateCountDistinct`)
- Increase buckets and HLL Type to increase accuracy.
- **Not parameter to instruct the algorithm to return exact count distinct.**
- Possible to do only union of DS-HLL (only using Native Json Query, not SQL as of 0.16 version).
- No formula/approach to identify ideal values for parameters to increase accuracy.

The screenshot shows a database interface with two queries and their results. The top query is:

```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_HLL(fake_id)
4   from case21
5   group by 1
```

The bottom query is:

```
1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_HLL(fake_id ,21, 'HLL_8')
4   from case21
5   group by 1
```

The results for the first query are:

trip_week_day_desc	EXPR\$1
Friday	4575155
Monday	3946928
Saturday	4502544
Sunday	3663443
Thursday	4589364
Tuesday	4691523
Wednesday	4998154

The results for the second query are:

trip_week_day_desc	EXPR\$1
Friday	4622139
Monday	4045711
Saturday	4391812
Sunday	3727322
Thursday	4668196
Tuesday	4637773
Wednesday	4896635

Druid: count(distinct) Performance & Accuracy

Count(Distinct)

```

1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4   from case21
5   group by 1

```

Run ... Auto run

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

6.13s

druid.sql.planner.useApproximateCountDistinct=true

- Default behavior, i.e uses HLL approximation algorithm.
- Always returns approximate count.

Count(Distinct)

```

1 select
2   trip_week_day_desc,
3   count(distinct fake_id)
4   from case21
5   group by 1

```

Run ... Auto run

trip_week_day_desc	EXPR\$1
Friday	4619823
Monday	4047415
Saturday	4390413
Sunday	3727397
Thursday	4672737
Tuesday	4637626
Wednesday	4895761

159.34s

druid.sql.planner.useApproximateCountDistinct=false

- Computes exact count distinct, performance intensive.
- Not recommended

Approx_Count_Distinct

```

1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT(fake_id)
4   from case21
5   group by 1

```

Run ... Auto run Smart query limit

trip_week_day_desc	EXPR\$1
Friday	4518601
Monday	3972439
Saturday	4409488
Sunday	3691690
Thursday	4735341
Tuesday	4533286
Wednesday	4992587

5.21s

- Always uses HLL, and computes approximate count distinct.
- No way to get exact count.

DS Theta

```

1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_THETA(fake_id)
4   from case21
5   group by 1

```

Run ... Auto run Smart query limit

trip_week_day_desc	EXPR\$1
Friday	4661543
Monday	4023947
Saturday	4418994
Sunday	3742394
Thursday	4682777
Tuesday	4663698
Wednesday	5036394

4.37s

- Uses DataSketch Theta algorithm to compute distinct count.
- Accuracy can be improved with parameters.

DS HLL

```

1 select
2   trip_week_day_desc,
3   APPROX_COUNT_DISTINCT_DS_HLL(fake_id)
4   from case21
5   group by 1

```

Run ... Auto run Smart query limit

trip_week_day_desc	EXPR\$1
Friday	4575155
Monday	3946928
Saturday	4502544
Sunday	3663443
Thursday	4589364
Tuesday	4691523
Wednesday	4998154

4.37s

- Uses DataSketch HLL algorithm to compute distinct count.
- Accuracy can be improved with parameters.

DataSketch During Ingestion

Why Sketch during ingestion

id	trans_time	pickup	drop	distance	amount
1	2019-01-01 00:10:16	Bronx	Brooklyn	2	24.95
2	2019-01-01 00:27:11	Queens	Manhattan	3	18.20
3	2019-01-01 00:39:09	Bronx	Brooklyn	2	28.50
4	2019-01-01 00:50:10	Queens	Bronx	4	52.00
5	2019-01-01 01:01:56	Manhattan	EWR	6	45.00
6	2019-01-01 01:10:09	Bronx	Brooklyn	2	26.00
7	2019-01-01 02:10:25	Queens	Manhattan	3	19.00
8	2019-01-02 14:20:11	Queens	Bronx	4	49.50

time **Dimensions** **Metrics**



segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount
1	2019-01-01	Bronx	Brooklyn	2	24.95
2	2019-01-01	Queens	Manhattan	3	18.20
3	2019-01-01	Bronx	Brooklyn	2	28.50
4	2019-01-01	Queens	Bronx	4	52.00
5	2019-01-01	Manhattan	EWR	6	45.00
6	2019-01-01	Bronx	Brooklyn	2	26.00
7	2019-01-01	Queens	Manhattan	3	19.00

id	trans_time	pickup	drop	distance	amount	count
<1,3,6>	2019-01-01	Bronx	Brooklyn	6	79.45	3
<2,7>	2019-01-01	Queens	Manhattan	6	37.20	2
<4>	2019-01-01	Queens	Bronx	4	52.00	1
<5>	2019-01-01	Manhattan	EWR	6	45.00	1

rollup : true
segmentGranularity: "day"
queryGranularity: "day"

When to use Sketch during ingestion

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
<1, 3, 6>	2019-01-01	Bronx	Brooklyn	6	79.45	3
<2, 7>	2019-01-01	Queens	Manhattan	6	37.20	2
<4>	2019-01-01	Queens	Bronx	4	52.00	1
<5>	2019-01-01	Manhattan	EWR	6	45.00	1

- Dimensions' role is only to uniquely count the instances of data (numeric or string).
- No use case to use such Dimensions' in filter (where clause).
- Multiple options to store such dimensions
 - HyperUnique
 - Cardinality
 - Datasketch-HLL
 - DataSketh-Thetha

These are stored as metrics and not dimensions.

How to use Sketch during ingestion

segment-2019-01-01-00h_to_2019-01-02-00h

id	trans_time	pickup	drop	distance	amount	count
<1, 3, 6>	2019-01-01	Bronx	Brooklyn	6	79.45	3
<2, 7>	2019-01-01	Queens	Manhattan	6	37.20	2
<4>	2019-01-01	Queens	Bronx	4	52.00	1
<5>	2019-01-01	Manhattan	EWR	6	45.00	1

```
"metricsSpec" : [
  {
    "name": "total_row_count",
    "type": "count"
  },
  {
    "name": "total_passenger_count",
    "type": "longSum",
    "fieldName": "passenger_count"
  },
  {
    "name" : "fake_id_hyperunique",
    "type" : "hyperUnique",
    "fieldName" : "fake_id"
  }
],
```

```
"metricsSpec" : [
  {
    "name": "total_row_count",
    "type": "count"
  },
  {
    "name": "total_passenger_count",
    "type": "longSum",
    "fieldName": "passenger_count"
  },
  {
    "name" : "fake_id_cardinality",
    "type" : "cardinality",
    "fieldName" : "fake_id"
  }
],
```

```
"metricsSpec" : [
  {
    "name": "total_row_count",
    "type": "count"
  },
  {
    "name": "total_passenger_count",
    "type": "longSum",
    "fieldName": "passenger_count"
  },
  {
    "name" : "fake_id_thetaSketch",
    "type" : "thetaSketch",
    "fieldName" : "fake_id",
    "size" : 128
  }
],
```

```
"metricsSpec" : [
  {
    "name": "total_row_count",
    "type": "count"
  },
  {
    "name": "total_passenger_count",
    "type": "longSum",
    "fieldName": "passenger_count"
  },
  {
    "name" : "fake_id_HLLSketch",
    "type" : "HLLSketchBuild",
    "fieldName" : "fake_id",
    "lgK" : 8,
    "tgtHllType" : "HLL_6"
  }
],
```

How does Sketch data look

```
1 select fake_id, fake_id_hyperunique, fake_id_thetaSketch, fake_id_HLLSketch from case21
```

Run ... Auto run Smart query limit 99+ results in 0.25s

fake_id	fake_id_hyperunique	fake_id_thetaSketch	fake_id_HLLSketch
978695525000089100	"AQAAAQAAAACiMA=="	"AQMDAAAazJO3du09c896Ww=="	"AgEHCAMIAQRu7doH"
981126447000193300	"AQAAAQAAAAP8Ag=="	"AQMDAAAazJOQkycf0XDspw=="	"AgEHCAMIAQQgJ08G"
1041403015000068000	"AQAAAQAAAALcEA=="	"AQMDAAAazJOYN3NthGYAWQ=="	"AgEHCAMIAQQxb+YG"
1041377159000238100	"AQAAAQAAAFAZIA=="	"AQMDAAAazJPG3h+qdNw9Eg=="	"AgEHCAMIAQSMvT8E"
1041461529000162200	"AQAAAQAAAABEAw=="	"AQMDAAAazJPLQ8udqUHGaw=="	"AgEHCAMIAQSXh5YX"
1041433151000264200	"AQAAAQAAAAOHUA=="	"AQMDAAAazJNC2n7Q5W+HdA=="	"AgEHCAMIAQSftP0M"
1041379083000229100	"AQAAAQAAAACuiA=="	"AQMDAAAazJOkY8nzxa+SLw=="	"AgEHCAMIAQRlx5IH"
1041379512000193300	"AQAAAQAAAAN+EA=="	"AQMDAAAazJPdGCL6ws2KCw=="	"AgEHCAMIAQS7MUQM"

Previous Page 1 of 5 20 rows Next

Comparison of DataSketch results

```
1 select TIME_EXTRACT(__time,'day'),
2   count(distinct fake_id),
3   count(distinct fake_id_hyperunique),
4   APPROX_COUNT_DISTINCT(fake_id),
5   APPROX_COUNT_DISTINCT(fake_id_hyperunique),
6   APPROX_COUNT_DISTINCT_DS_THETA(fake_id_thetaSketch),
7   APPROX_COUNT_DISTINCT_DS_HLL(fake_id_HLLSketch)
8 from case21 group by 1 order by 1 asc
```

Run



Auto run



Smart query limit

31 results in 14.91s

EXPR\$0	EXPR\$1
1	1016932
2	1021893
3	999752
4	835348
5	1009756
6	1037934
7	1023590
8	1086989

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4	EXPR\$5	EXPR\$6
1	1013141	965745	1013141	965745	1010039	988120
2	1031683	996508	1031683	996508	1011652	1016873
3	1053284	983415	1053284	983415	990121	1022229
4	831658	836172	831658	836172	837011	826570
5	1006639	997006	1006639	997006	1002595	985223
6	1065251	1037856	1065251	1037856	1045544	1088789
7	1012017	1046044	1012017	1046044	1011312	1007141
8	1058116	1064155	1058116	1064155	1088855	1092268

Previous

Page 1 of 2

20 rows

Next

Comparison of DataSketch results

```
1 select TIME_EXTRACT(__time,'day'),
2   count(distinct fake_id),
3   count(distinct fake_id_hyperunique),
4   APPROX_COUNT_DISTINCT(fake_id),
5   APPROX_COUNT_DISTINCT(fake_id_hyperunique),
6   APPROX_COUNT_DISTINCT_DS_THETA(fake_id_thetaSketch, 65536),
7   APPROX_COUNT_DISTINCT_DS_HLL(fake_id_HLLSketch,21,'HLL_8')
8 from case21 group by 1 order by 1 asc
```

Run



Auto run



Smart query limit

31 results in 40.02s

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4	EXPR\$5	EXPR\$6
1	1013141	965745	1013141	965745	1014396	1016908
2	1031683	996508	1031683	996508	1016528	1022333
3	1053284	983415	1053284	983415	993897	1000030
4	831658	836172	831658	836172	835067	835849
5	1006639	997006	1006639	997006	1008093	1008450
6	1065251	1037856	1065251	1037856	1046920	1037056
7	1012017	1046044	1012017	1046044	1019177	1023875
8	1058116	1064155	1058116	1064155	1086331	1087001

Previous

Page

1

of 2

20 rows

Next

Query - SQL

Query - SQL

```
1 select
2 TIME_EXTRACT(__time,'month'),
3 TIME_EXTRACT(__time,'day'),
4 count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)
5 from case21 group by 1,2 order by 1 asc
```

Run ... Auto run Smart query limit 99+ results in 6.70s ▾

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
1	1	407163	1465240.2799999793	5710088.87
1	2	431604	1355121.4600000079	5606537.649999994
1	3	488822	1404973.940000007	6067333.629999987
1	4	352544	975226.270000011	4307949.369999997
1	5	467959	1316379.600000001	5896214.789999992
1	6	473771	1415412.4200000037	5955269.1699999925
1	7	435518	1361648.1400000243	5752626.539999994
1	8	469340	1417104.000000006	6282111.409999987

Previous Page 1 of 5 20 rows ▾ Next

Query – SQL, Condition

```
1 select
2 TIME_EXTRACT(__time,'month'),
3 TIME_EXTRACT(__time,'day'),
4 count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)
5 from case21
6 where total_passenger_count>2
7 group by 1,2 order by 1 asc
```

▶ Run

...

Auto run

Smart query limit

99+ results in 1.81s ⏵

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
1	1	68496	244689.46	944516.1
1	2	62684	204106.1600000002	822407.53
1	3	64187	200467.95000000022	847611.5
1	4	48601	141916.06000000006	607880.77
1	5	73534	196685.98000000004	867905.3500000001
1	6	70486	213393.8	891487.73
1	7	63067	188876.63999999885	776157.1900000001
1	8	59869	184849.3699999999	796774.49

Previous Page 1 of 5 20 rows ▲ Next

Query – SQL, Condition, Lookup

```
1 select
2 TIME_EXTRACT(__time,'month'),
3 TIME_EXTRACT(__time,'day'),
4 count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)
5 from case21
6 where total_passenger_count>2 and LOOKUP(cast(PULocationID as varchar),'location_to_borough') in ('Queens','Bronx','Brooklyn')
7 group by 1,2 order by 1 asc
```

▶ Run

...

Auto run

Smart query limit

64 results in 0.38s ⏪

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
1	1	7468	72546.71999999993	222143.33000000002
1	2	6409	70356.7099999999	216052.14
1	3	5721	61426.790000000095	192859.73
1	4	3450	37477.52000000001	118368.55
1	5	4635	47028.039999999964	146878.16
1	6	5794	62380.35000000005	192977.88
1	7	5526	58624.87000000005	181021.3
1	8	5549	57579.32999999997	181497.15999999997

Previous

Page 1 of 4

20 rows ⏪

Next

Query – SQL, Condition, Sub-Query

```
1 select
2 TIME_EXTRACT(__time, 'month'),
3 TIME_EXTRACT(__time, 'day'),
4 count(distinct fake_id), sum(total_trip_distance), sum(total_fare_amount)
5 from case21
6 where total_passenger_count > 2 and
7 LOOKUP(cast(PULocationID as varchar), 'location_to_borough') in
8 (
9     select distinct LOOKUP(cast(PULocationID as varchar), 'location_to_borough') from case21
10    where LOOKUP(cast(PULocationID as varchar), 'location_to_borough') in ('Queens', 'Bronx', 'Brooklyn')
11 )
12 group by 1,2 order by 1 asc
```

Run ... Auto run Smart query limit 64 results in 2.64s ⏪

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
1	1	7468	72546.71999999993	222143.33000000002
1	2	6409	70356.7099999999	216052.14
1	3	5721	61426.790000000095	192859.73
1	4	3450	37477.52000000001	118368.55
1	5	4635	47028.039999999964	146878.16
1	6	5794	62380.35000000005	192977.88

Previous Page 1 of 4 20 rows ⏪ Next

Query – SQL, Condition (2 different cubes)

```
1 select
2 TIME_EXTRACT(__time,'month'),
3 TIME_EXTRACT(__time,'day'),
4 count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)
5 from case21
6 where total_passenger_count>2 and
7 LOOKUP(cast(PULocationID as varchar),'location_to_borough') in
8 (
9 select distinct LOOKUP(cast(PULocationID as varchar),'location_to_borough') from case19
10 where LOOKUP(cast(PULocationID as varchar),'location_to_borough') in ('Queens','Bronx','Brooklyn')
11 )
12 group by 1,2 order by 1 asc
```

Run ... Auto run Smart query limit 64 results in 1.71s ⏪

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4
1	1	7468	72546.71999999993	222143.33000000002
1	2	6409	70356.7099999999	216052.14
1	3	5721	61426.790000000095	192859.73
1	4	3450	37477.52000000001	118368.55
1	5	4635	47028.039999999964	146878.16
1	6	5794	62380.35000000005	192977.88

Previous Page 1 of 4 20 rows ⏪ Next

Query – SQL over JSON

Query – SQL over JSON (output: object)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), TIME_EXTRACT(__time,'day'), count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)  
from case21 where total_passenger_count>2 group by 1,2 order by 1 asc",  
  "resultFormat" : "object"  
}
```

```
C:\Users\Lenovo\Desktop\Druid Training\Native Queries>curl -XPOST -H"Content-Type:application/json" http://192.168.0.101:  
8082/druid/v2/sql/ -d @1_sql_over_json_result_object.json
```

```
[{  
  "EXPR$0": 1,  
  "EXPR$1": 1,  
  "EXPR$2": 68496,  
  "EXPR$3": 244689.46,  
  "EXPR$4": 944516.1  
}, {  
  "EXPR$0": 1,  
  "EXPR$1": 2,  
  "EXPR$2": 62684,  
  "EXPR$3": 204106.1600000002,  
  "EXPR$4": 822407.53  
}, {  
  "EXPR$0": 1,  
  "EXPR$1": 3,  
  "EXPR$2": 64187,  
  "EXPR$3": 200467.95000000022,  
  "EXPR$4": 847611.5  
}, {  
  "EXPR$0": 1,  
  "EXPR$1": 4,  
  "EXPR$2": 48601,  
  "EXPR$3": 141916.0600000006,  
  "EXPR$4": 607880.77  
}]
```

Query – SQL over JSON (output: array)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), TIME_EXTRACT(__time,'day'), count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)  
from case21 where total_passenger_count>2 group by 1,2 order by 1 asc",  
  "resultFormat" : "array"  
}
```

```
C:\Users\Lenovo\Desktop\Druid Training\Native Queries>curl -XPOST -H"Content-Type:application/json" http://192.168.0.101:  
8082/druid/v2/sql/ -d @2_sql_over_json_result_array.json
```

```
[  
  [1, 1, 68496, 244689.46, 944516.1],  
  [1, 2, 62684, 204106.1600000002, 822407.53],  
  [1, 3, 64187, 200467.95000000022, 847611.5],  
  [1, 4, 48601, 141916.0600000006, 607880.77],  
  [1, 5, 73534, 196685.9800000004, 867905.3500000001],  
  [1, 6, 70486, 213393.8, 891487.73],  
  [1, 7, 63067, 188876.6399999985, 776157.1900000001],  
  [1, 8, 59869, 184849.3699999999, 796774.49],  
  [1, 9, 63881, 188585.68000000052, 831292.66],  
  [1, 10, 65792, 197814.56000000032, 876801.4700000001],  
  [1, 11, 74969, 213705.33000000118, 949688.7599999999],  
  [1, 12, 76585, 215586.8700000008, 964312.6299999999]  
]
```

JSON Array of JSON arrays.

Query – SQL over JSON (output: objectLines)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), TIME_EXTRACT(__time,'day'), count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)  
from case21 where total_passenger_count>2 group by 1,2 order by 1 asc",  
  "resultFormat" : "objectLines"  
}
```

```
C:\Users\Lenovo\Desktop\Druid Training\Native Queries>curl -XPOST -H"Content-Type:application/json" http://192.168.0.101:  
8082/druid/v2/sql/ -d @3_sql_over_json_result_objectLines.json
```

```
{"EXPR$0":1,"EXPR$1":1,"EXPR$2":68496,"EXPR$3":244689.46,"EXPR$4":944516.1}  
{"EXPR$0":1,"EXPR$1":2,"EXPR$2":62684,"EXPR$3":204106.1600000002,"EXPR$4":822407.53}  
{"EXPR$0":1,"EXPR$1":3,"EXPR$2":64187,"EXPR$3":200467.95000000022,"EXPR$4":847611.5}  
{"EXPR$0":1,"EXPR$1":4,"EXPR$2":48601,"EXPR$3":141916.06000000006,"EXPR$4":607880.77}  
 {"EXPR$0":1,"EXPR$1":5,"EXPR$2":73534,"EXPR$3":196685.98000000004,"EXPR$4":867905.3500000001}  
 {"EXPR$0":1,"EXPR$1":6,"EXPR$2":70486,"EXPR$3":213393.8,"EXPR$4":891487.73}  
 {"EXPR$0":1,"EXPR$1":7,"EXPR$2":63067,"EXPR$3":188876.63999999885,"EXPR$4":776157.1900000001}
```

JSON Objects
separated by line,
instead of array

Query – SQL over JSON (output: arrayLines)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), TIME_EXTRACT(__time,'day'), count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)  
from case21 where total_passenger_count>2 group by 1,2 order by 1 asc",  
  "resultFormat" : "arrayLines"  
}
```

```
C:\Users\Lenovo\Desktop\Druid Training\Native Queries>curl -XPOST -H"Content-Type:application/json" http://192.168.0.101  
:8082/druid/v2/sql/ -d @4_sql_over_json_result_arrayLines.json
```

```
[1, 1, 68496, 244689.46, 944516.1],  
[1, 2, 62684, 204106.1600000002, 822407.53],  
[1, 3, 64187, 200467.9500000022, 847611.5],  
[1, 4, 48601, 141916.0600000006, 607880.77],  
[1, 5, 73534, 196685.9800000004, 867905.3500000001],  
[1, 6, 70486, 213393.8, 891487.73],
```

```
[1, 7, 63067, 188876.6399999985, 776157.1900000001],  
[1, 8, 59869, 184849.3699999999, 796774.49],  
[1, 9, 63881, 188585.6800000052, 831292.66],  
[1, 10, 65792, 197814.5600000032, 876801.4700000001],  
[1, 11, 74969, 213705.33000000118, 949688.7599999999],  
[1, 12, 76585, 215586.8700000008, 964312.6299999999]
```

Arrays, but not wrapped as JSON object.

Query – SQL over JSON (output: csv)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), TIME_EXTRACT(__time,'day'), count(distinct fake_id),sum(total_trip_distance), sum(total_fare_amount)  
from case21 where total_passenger_count>2 group by 1,2 order by 1 asc",  
  "resultFormat" : "CSV",  
  "header" : true  
}
```

```
C:\Users\Lenovo\Desktop\Druid Training\Native Queries>curl -XPOST -H"Content-Type:application/json" http://192.168.0.101:  
8082/druid/v2/sql/ -d @5_sql_over_json_result_csv.json
```

EXPR\$0,EXPR\$1,EXPR\$2,EXPR\$3,EXPR\$4	1,9,63881,188585.68000000052,831292.66
1,1,68496,244689.46,944516.1	1,10,65792,197814.56000000032,876801.4700000001
1,2,62684,204106.1600000002,822407.53	1,11,74969,213705.33000000118,949688.7599999999
1,3,64187,200467.95000000022,847611.5	1,12,76585,215586.8700000008,964312.6299999999
1,4,48601,141916.0600000006,607880.77	1,13,79037,219094.51000000085,933380.92
1,5,73534,196685.98000000004,867905.3500000001	1,14,73373,201356.06000000096,861388.7499999998
1,6,70486,213393.8,891487.73	1,15,68177,191964.1100000003,804548.3200000001
1,7,63067,188876.63999999885,776157.1900000001	1,16,70881,201920.38999999943,880180.26
1,8,59869,184849.3699999999,796774.49	1,17,70034,200629.74999999962,889464.8799999999

Query – SQL over JSON (parameter override)

```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), count(distinct fake_id) from case21 where  
              total_passenger_count>2 group by 1 order by 1 asc",  
  "resultFormat" : "csv",  
  "header" : true,  
  "context" : {  
    "useApproximateCountDistinct" : true  
  }  
}
```

EXPR\$0,EXPR\$1
1,2136178
2,2019216
3,83
4,22
5,34
6,19
7,1
8,2
11,9
12,166



```
{  
  "query" : "select TIME_EXTRACT(__time,'month'), count(distinct fake_id) from case21 where  
              total_passenger_count>2 group by 1 order by 1 asc",  
  "resultFormat" : "csv",  
  "header" : true,  
  "context" : {  
    "useApproximateCountDistinct" : false  
  }  
}
```

EXPR\$0,EXPR\$1
1,2151928
2,2013983
3,82
4,22
5,34
6,19
7,1
8,2
11,9
12,169



Query – Native JSON

Query: Native JSON

- Druid supports following queries over Native JSON
 - **Timeseries**
 - Fastest in terms of performance over all other queries.
 - Does not support groupby any dimension. (in SQL terminology, supports groupby only `_time`).
 - In SQL terminology only `_time` dimension in “group by clause”.
 - **TopN**
 - In SQL terminology `_time` and any other dimension in “group by clause”
 - **GroupBy**
 - In SQL terminology **any number of** dimensions in “group by clause”

Query – Native JSON Timeseries

Query: Native JSON - Timeseries

```
{  
  "queryType": "timeseries",  
  "dataSource": "case21",  
  "granularity": "day",  
  "descending": "true",  
  "aggregations": [  
    { "type": "longSum", "name": "total_passenger_count", "fieldName": "total_passenger_count" },  
    { "type": "doubleSum", "name": "total_trip_distance", "fieldName": "total_trip_distance" },  
    { "type": "doubleSum", "name": "total_fare_amount", "fieldName": "total_fare_amount" }  
  ],  
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]  
}
```

```
[  
  {  
    "timestamp" : "2019-09-29T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 16.5,  
      "total_passenger_count" : 1,  
      "total_trip_distance" : 4.86  
    }  
  }, {  
    "timestamp" : "2019-09-03T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 4.5,  
      "total_passenger_count" : 1,  
      "total_trip_distance" : 0.64  
    }  
  }, {  
    "timestamp" : "2019-08-17T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 90.5,  
      "total_passenger_count" : 2,  
      "total_trip_distance" : 30.02  
    }  
  }, {  
    "timestamp" : "2019-08-13T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 21.5,  
      "total_passenger_count" : 2,  
      "total_trip_distance" : 5.37  
    }  
  }, {  
    "timestamp" : "2019-08-12T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 5.5,  
      "total_passenger_count" : 2,  
      "total_trip_distance" : 0.93  
    }  
  }, {  
    "timestamp" : "2019-08-03T00:00:00.000Z",  
    "result" : {  
      "total_fare_amount" : 16,  
      "total_passenger_count" : 12,  
      "total_trip_distance" : 2.2  
    }  
  }]
```

```
select  
  DATE_TRUNC('day', __time),  
  sum(total_fare_amount),sum(total_passenger_count),sum(total_trip_distance)  
from case21  
where __time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000'  
group by 1 order by 1 desc
```

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019-09-29T00:00:00.000Z	16.5	1	4.86
2019-09-03T00:00:00.000Z	4.5	1	0.64
2019-08-17T00:00:00.000Z	90.5	2	30.02
2019-08-13T00:00:00.000Z	21.5	2	5.37
2019-08-12T00:00:00.000Z	5.5	2	0.93
2019-08-03T00:00:00.000Z	16	12	2.2

Query: Native JSON - Timeseries

```
{  
  "queryType": "timeseries",  
  "dataSource": "case21",  
  "granularity": "month",  
  "descending": "true",  
  "aggregations": [  
    { "type": "longSum", "name": "total_passenger_count", "fieldName": "total_passenger_count" },  
    { "type": "doubleSum", "name": "total_trip_distance", "fieldName": "total_trip_distance" },  
    { "type": "doubleSum", "name": "total_fare_amount", "fieldName": "total_fare_amount" }  
  ],  
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]  
}
```

```
[ {  
  "timestamp" : "2019-09-01T00:00:00.000Z",  
  "result" : {  
    "total_passenger_count" : 2,  
    "total_fare_amount" : 21.0,  
    "total_trip_distance" : 5.5  
  }  
, {  
  "timestamp" : "2019-08-01T00:00:00.000Z",  
  "result" : {  
    "total_passenger_count" : 18,  
    "total_fare_amount" : 133.5,  
    "total_trip_distance" : 38.519999999999996  
  }  
, {  
  "timestamp" : "2019-07-01T00:00:00.000Z",  
  "result" : {  
    "total_passenger_count" : 24,  
    "total_fare_amount" : 205.5,  
    "total_trip_distance" : 50.77  
  }  
}
```

```
select  
  DATE_TRUNC('month', __time),  
  sum(total_fare_amount),sum(total_passenger_count),sum(total_trip_distance)  
from case21  
where __time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000'  
group by 1 order by 1 desc
```

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019-09-01T00:00:00.000Z	21	2	5.5
2019-08-01T00:00:00.000Z	133.5	18	38.519999999999996
2019-07-01T00:00:00.000Z	205.5	24	50.77
2019-06-01T00:00:00.000Z	321.68	125	74.08
2019-05-01T00:00:00.000Z	419	208	95.05
2019-04-01T00:00:00.000Z	341	142	65.33999999999999

Query: Native JSON – Timeseries - Filter

```
{  
  "queryType": "timeseries",  
  "dataSource": "case21",  
  "granularity": "month",  
  "descending": "true",  
  "filter": {  
    "type": "javascript",  
    "dimension": "total_passenger_count",  
    "function": "function(value) { return (value>2); }"  
  },  
  "aggregations": [  
    { "type": "longSum", "name": "total_passenger_count", "fieldName": "total_passenger_count" },  
    { "type": "doubleSum", "name": "total_trip_distance", "fieldName": "total_trip_distance" },  
    { "type": "doubleSum", "name": "total_fare_amount", "fieldName": "total_fare_amount" }  
  ],  
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]  
}
```

```
[ {  
  "timestamp": "2019-09-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 0,  
    "total_fare_amount": 0.0,  
    "total_trip_distance": 0.0  
  }  
, {  
  "timestamp": "2019-08-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 12,  
    "total_fare_amount": 16.0,  
    "total_trip_distance": 2.2  
  }  
, {  
  "timestamp": "2019-07-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 6,  
    "total_fare_amount": 17.5,  
    "total_trip_distance": 4.1  
  }  
, {  
  "timestamp": "2019-06-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 114,  
    "total_fare_amount": 157.5,  
    "total_trip_distance": 28.62  
  }  
, {  
  "timestamp": "2019-05-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 204,  
    "total_fare_amount": 319,  
    "total_trip_distance": 64.1799  
  }  
, {  
  "timestamp": "2019-04-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 126,  
    "total_fare_amount": 189.5,  
    "total_trip_distance": 35.44  
  }  
, {  
  "timestamp": "2019-03-01T00:00:00.000Z",  
  "result": {  
    "total_passenger_count": 402,  
    "total_fare_amount": 696.5,  
    "total_trip_distance": 154.6  
  }  
}
```

```
select  
  DATE_TRUNC('month', __time),  
  sum(total_fare_amount),sum(total_passenger_count),sum(total_trip_distance)  
from case21  
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000') and  
total_passenger_count>2  
group by 1 order by 1 desc
```

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3
2019-08-01T00:00:00.000Z	16	12	2.2
2019-07-01T00:00:00.000Z	17.5	6	4.1
2019-06-01T00:00:00.000Z	157.5	114	28.62
2019-05-01T00:00:00.000Z	319	204	64.1799
2019-04-01T00:00:00.000Z	189.5	126	35.44
2019-03-01T00:00:00.000Z	696.5	402	154.6

Query: Native JSON – Timeseries - Filter

```
{  
  "queryType": "timeseries",  
  "dataSource": "case21",  
  "granularity": "month",  
  "descending": "true",  
  "filter": {  
    "type" : "javascript",  
    "dimension" : "total_passenger_count",  
    "function" : "function(value) { return (value>2); }"  
  },  
  "aggregations": [  
    { "type": "longSum", "name": "total_passenger_count", "fieldName": "total_passenger_count" },  
    { "type": "doubleSum", "name": "total_trip_distance", "fieldName": "total_trip_distance" },  
    { "type": "doubleSum", "name": "total_fare_amount", "fieldName": "total_fare_amount" }  
  ],  
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ],  
  "context": {  
    "skipEmptyBuckets": "true",  
    "grandTotal": true  
  }  
}
```

```
select  
  DATE_TRUNC('month', __time),  
  sum(total_fare_amount),sum(total_passenger_count),sum(total_trip_distance)  
from case21  
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000') and  
  total_passenger_count>2  
group by 1 order by 1 desc
```

EXPR\$0	2019-08-01T00:00:00.000Z	319	204	64.1799
	2019-07-01T00:00:00.000Z	189.5	126	35.44
	2019-06-01T00:00:00.000Z	696.5	402	154.6
	2019-05-01T00:00:00.000Z			
	2019-04-01T00:00:00.000Z			
	2019-03-01T00:00:00.000Z			

Compute new “total fare amount”
if passenger_count>2, increase amount by
10%, else use fare amount as is.

Query: Native JSON: Timeseries - Aggregation

```
{
  "queryType": "timeseries",
  "dataSource": "case21",
  "granularity": "month",
  "descending": "true",
  "aggregations": [
    { "type": "longSum", "name": "agg_passenger_count", "fieldNames": ["total_passenger_count"] },
    { "type": "doubleSum", "name": "agg_trip_distance", "fieldNames": ["total_trip_distance"] },
    { "type": "doubleSum", "name": "agg_fare_amount", "fieldNames": ["total_fare_amount"] },
    {
      "type": "javascript",
      "name": "agg_new_fare_amount",
      "fieldNames": ["total_passenger_count", "total_fare_amount"],
      "fnAggregate": "function(current, a, b) { if(a>2) return (current + (b*1.1)); else return (current + b); }",
      "fnCombine": "function(partialA, partialB) { return partialA + partialB; }",
      "fnReset": "function() { return 0; }"
    }
  ],
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]
}
```

Limitation:
JavaScript can return
only 1 float value.

```
select
  DATE_TRUNC('month', __time),
  sum(total_fare_amount),sum(total_passenger_count),
  sum(total_trip_distance),max(total_passenger_count),
  sum( (case when total_passenger_count >2 then 1.1 else 1 end) * total_fare_amount )
from case21
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000')
group by 1 order by 1 desc
INCADON@GMAIL.COM
```

```
[
  {
    "timestamp": "2019-09-01T00:00:00.000Z",
    "result": {
      "agg_new_fare_amount": 21.0,
      "agg_passenger_count": 2,
      "agg_trip_distance": 5.5,
      "agg_fare_amount": 21.0
    }
  },
  {
    "timestamp": "2019-08-01T00:00:00.000Z",
    "result": {
      "agg_new_fare_amount": 135.1,
      "agg_passenger_count": 18,
      "agg_trip_distance": 38.51999999999996,
      "agg_fare_amount": 133.5
    }
  },
  {
    "timestamp": "2019-07-01T00:00:00.000Z",
    "result": {
      "agg_new_fare_amount": 207.25,
      "agg_passenger_count": 24,
      "agg_trip_distance": 50.77,
      "agg_fare_amount": 205.5
    }
  }
]
```

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	EXPR\$4	EXPR\$5
2019-09-01T00:00:00.000Z	21	2	5.5	1	21
2019-08-01T00:00:00.000Z	133.5	18	38.51999999999996	6	135.1
2019-07-01T00:00:00.000Z	205.5	24	50.77	6	207.25
2019-06-01T00:00:00.000Z	321.68	125	74.08	6	337.4300
2019-05-01T00:00:00.000Z	419	208	95.05	6	450.9000
2019-04-01T00:00:00.000Z	341	142	65.33999999999999	6	359.95

Compute: Average fare amount per passenger

Query: Native JSON: Timeseries – Post Aggregation

```
{
  "queryType": "timeseries",
  "dataSource": "case21",
  "granularity": "month",
  "descending": "true",
  "aggregations": [
    { "type": "longSum", "name": "agg_passenger_count", "fieldName": "total_passenger_count" },
    { "type": "doubleSum", "name": "agg_trip_distance", "fieldName": "total_trip_distance" },
    { "type": "doubleSum", "name": "agg_fare_amount", "fieldName": "total_fare_amount" },
    {
      "type": "javascript",
      "name": "avg_amount_per_person2",
      "fieldNames": ["total_passenger_count", "total_fare_amount"],
      "fnAggregate": "function(current, a, b) { return (current + (b/a)); }",
      "fnCombine": "function(partialA, partialB) { return partialA + partialB; }",
      "fnReset": "function() { return 0; }"
    }
  ],
  "postAggregations": [
    { "type": "arithmetic",
      "name": "avg_amount_per_person1",
      "fn": "/",
      "fields": [
        { "type": "fieldAccess", "name": "agg_fare_amount2", "fieldName": "agg_fare_amount" },
        { "type": "fieldAccess", "name": "agg_passenger_count2", "fieldName": "agg_passenger_count" }
      ]
    },
    { "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ] }
  ]
}
```

```
select
  DATE_TRUNC('month', __time),
  sum(total_fare_amount), sum(total_trip_distance),
  sum(total_passenger_count),
  sum(total_fare_amount)/sum(total_passenger_count) avg_amount_per_person1,
  sum(total_fare_amount/total_passenger_count) avg_amount_per_person2
from case21
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000') group by 1 order by 1
```



[{
 "timestamp" : "2019-09-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 2,
 "agg_trip_distance" : 5.5,
 "agg_fare_amount" : 21.0,
 "avg_amount_per_person1" : 10.5,
 "avg_amount_per_person2" : 21.0
 }
}, {
 "timestamp" : "2019-08-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 18,
 "agg_trip_distance" : 38.519999999999996,
 "agg_fare_amount" : 133.5,
 "avg_amount_per_person1" : 7.416666666666667,
 "avg_amount_per_person2" : 106.66666666666667
 }
}, {
 "timestamp" : "2019-07-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 24,
 "agg_trip_distance" : 50.77
 }
}, {
 "timestamp" : "2019-06-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 125,
 "agg_trip_distance" : 321.68
 }
}, {
 "timestamp" : "2019-05-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 208,
 "agg_trip_distance" : 74.08
 }
}, {
 "timestamp" : "2019-04-01T00:00:00.000Z",
 "result" : {
 "agg_passenger_count" : 142,
 "agg_trip_distance" : 65.33999999999999
 }
}]

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	avg_amount_per_person1	avg_amount_per_person2
2019-09-01T00:00:00.000Z	21	5.5	2	10.5	21
2019-08-01T00:00:00.000Z	133.5	38.519999999999996	18	7.416666666666667	106.66666666666667
2019-07-01T00:00:00.000Z	205.5	50.77	24	8.5625	179.66666666666666
2019-06-01T00:00:00.000Z	321.68	74.08	125	2.57344	152.68
2019-05-01T00:00:00.000Z	419	95.05	208	2.014423	136.91666666666666
2019-04-01T00:00:00.000Z	341	65.33999999999999	142	2.401408	180.33333333333334

Compute: Average miles travelled per passenger

Query: Native JSON: Timeseries – Post Aggregation

```
{
  "queryType": "timeseries",
  "dataSource": "case21",
  "granularity": "month",
  "descending": "true",
  "aggregations": [
    { "type": "longSum", "name": "agg_passenger_count", "fieldName": "total_passenger_count" },
    { "type": "doubleSum", "name": "agg_trip_distance", "fieldName": "total_trip_distance" },
    { "type": "doubleSum", "name": "agg_fare_amount", "fieldName": "total_fare_amount" },
    {
      "type": "javascript",
      "name": "avg_distanace_per_person2",
      "fieldNames": ["total_passenger_count", "total_trip_distance"],
      "fnAggregate": "function(current, a, b) { return (current + (b/a)); }",
      "fnCombine": "function(partialA, partialB) { return partialA + partialB; }",
      "fnReset": "function() { return 0; }"
    }
  ],
  "postAggregations": [
    { "type": "arithmetic",
      "name": "avg_distanace_per_person1",
      "fn": "/",
      "fields": [
        { "type": "fieldAccess", "name": "agg_trip_distance2", "fieldName": "agg_trip_distance" },
        { "type": "fieldAccess", "name": "agg_passenger_count2", "fieldName": "agg_passenger_count" }
      ]
    }
  ],
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]
}
```

```
select
  DATE_TRUNC('month', __time),
  sum(total_fare_amount), sum(total_trip_distance),
  sum(total_passenger_count),
  sum(total_trip_distance)/sum(total_passenger_count) avg_distanace_per_person1,
  sum(total_trip_distance/total_passenger_count) avg_distanace_per_person2
from case21
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000')
group by 1 order by 1 desc
```

```
[
  {
    "timestamp": "2019-09-01T00:00:00.000Z",
    "result": {
      "avg_distanace_per_person2": 5.5,
      "agg_passenger_count": 2,
      "agg_trip_distance": 5.5,
      "agg_fare_amount": 21.0,
      "avg_distanace_per_person1": 2.75
    }
  },
  {
    "timestamp": "2019-08-01T00:00:00.000Z",
    "result": {
      "avg_distanace_per_person2": 33.536666666666667,
      "agg_passenger_count": 18,
      "agg_trip_distance": 38.519999999999996,
      "agg_fare_amount": 133.5,
      "avg_distanace_per_person1": 2.1399999999999997
    }
  },
  {
    "timestamp": "2019-07-01T00:00:00.000Z",
    "result": {
      "avg_distanace_per_person2": 45.66333333333334,
      "agg_passenger_count": 24
    }
  }
]
```

EXPR\$0	EXPR\$1	EXPR\$2	EXPR\$3	avg_dist	avg_distanace_per_person2
2019-09-01T0	21	5.5	2	2.75	5.5
2019-08-01T0	133.5	38.5199	18	2.139999	33.536666666666667
2019-07-01T0	205.5	50.77	24	2.115416	45.66333333333334
2019-06-01T0	321.68	74.08	125	0.59264	37.31
2019-05-01T0	419	95.05	208	0.456971	35.946666666666666
2019-04-01T0	341	65.3399	142	0.460140	35.361666666666665

67

281

Query – Native JSON

TopN

Query: Native JSON - TopN

```
{
  "queryType": "topN",
  "dataSource": "case21",
  "dimension": "trip_desc",
  "threshold": 3,
  "metric": "agg_fare_amount",
  "granularity": "month",
  "aggregations": [
    {
      "type": "longSum",
      "name": "agg_passenger_count",
      "fieldName": "total_passenger_count"
    },
    {
      "type": "doubleSum",
      "name": "agg_trip_distance",
      "fieldName": "total_trip_distance"
    },
    {
      "type": "doubleSum",
      "name": "agg_fare_amount",
      "fieldName": "total_fare_amount"
    }
  ],
  "intervals": [ "2019-01-01T00:00:00.000/2020-01-01T00:00:00.000" ]
}
```

```
select
DATE_TRUNC('month', __time), trip_desc,
sum(total_fare_amount), sum(total_trip_distance),
sum(total_passenger_count)
from case21
where (__time >= '2019-01-01T00:00:00.000' and __time < '2020-01-01T00:00:00.000')
group by 1,2 order by 1 asc, 3 desc
```

```
[
  {
    "timestamp": "2019-01-01T00:00:00.000Z",
    "result": [
      {
        "agg_passenger_count": 191136,
        "agg_trip_distance": 344965.19999999995,
        "agg_fare_amount": 2000056.86,
        "trip_desc": "From 264 to 264"
      },
      {
        "agg_passenger_count": 194,
        "agg_trip_distance": 342.8300000000004,
        "agg_fare_amount": 624990.36,
        "trip_desc": "From 237 to 90"
      },
      {
        "agg_passenger_count": 7936,
        "agg_trip_distance": 102580.84000000001,
        "agg_fare_amount": 387636.9800000004,
        "trip_desc": "From 132 to 265"
      }
    ]
  },
  {
    "timestamp": "2019-02-01T00:00:00.000Z",
    "result": [
      {
        "agg_passenger_count": 76278,
        "agg_trip_distance": 137756.83999999994,
        "agg_fare_amount": 750606.12,
        "trip_desc": "From 264 to 264"
      }
    ]
  }
]
```

EXPR\$0	trip_desc	EXPR\$2	EXPR\$3	EXPR\$4
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 264 to 264	2000056.86	344965.2	191136
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 237 to 90	643786.86	4081.57	2198
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 132 to 265	387636.979999	102580.84	7936
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 132 to 230	378538.590000	132381.21	12632
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 138 to 230	356487.31	109451.36	17453
2019-01-01T00:00:00.000/2020-01-01T00:00:00.000	From 24 to 264	356011.98	42.62000000	35

Query – Native JSON GroupBy

Query: Native JSON - GroupBy

```
{
  "queryType": "groupBy",
  "dataSource": "case21",
  "granularity": "day",
  "dimensions": ["trip_desc", "tip_desc"],
  "limitSpec": { "type": "default", "limit": 5, "columns": [
      {
        "dimension": "trip_desc", "direction": "ascending", "dimensionOrder": "lexicographic"
      }, {
        "dimension": "agg_fare_amount", "direction": "descending", "dimensionOrder": "numeric"
      } ] },
  "aggregations": [ {
    "type": "longSum", "name": "agg_passenger_count", "fieldName": "total_passenger_count"
  }, {
    "type": "doubleSum", "name": "agg_trip_distance", "fieldName": "total_trip_distance"
  }, {
    "type": "doubleSum", "name": "agg_fare_amount", "fieldName": "total_fare_amount" } ],
  "intervals": [ "2019-01-01T00:00:00.000/2019-01-03T00:00:00.000" ],
  "having": {
    "type": "lessThan", "aggregation": "agg_passenger_count", "value": 50 } }
```

```
select
DATE_TRUNC('day', __time), trip_desc, tip_desc,
sum(total_fare_amount), sum(total_trip_distance),
sum(total_passenger_count)
from case21 where (__time >= '2019-01-01T00:00:00.000' and
__time < '2020-01-03T00:00:00.000') group by 1,2,3
having sum(total_passenger_count) < 50 order by 1 asc, 2 asc, 3 desc limit 5;
```

```
[ {
  "version": "v1",
  "timestamp": "2019-01-01T00:00:00.000Z",
  "event": {
    "agg_passenger_count": 15,
    "agg_trip_distance": 19.3,
    "trip_desc": "From 1 to 1",
    "tip_desc": "Party Time",
    "agg_fare_amount": 724.5
  }
}, {
  "version": "v1",
  "timestamp": "2019-01-01T00:00:00.000Z",
  "event": {
    "agg_passenger_count": 24,
    "agg_trip_distance": 67.82,
    "trip_desc": "From 1 to 1",
    "tip_desc": "No Tips",
    "agg_fare_amount": 712.5
  }
}, {
  "version": "v1",
  "timestamp": "2019-01-01T00:00:00.000Z",
  "event": {
    "agg_passenger_count": 3,
    "agg_trip_distance": 41.28,
    "trip_desc": "From 1 to 132",
    "tip_desc": "No Tips",
    "agg_fare_amount": 106.0
  }
}]
```

EXPR\$0	trip_desc				
2019-01-01T00:00:00.000/2019-01-03T00:00:00.000	From 1 to 1	Party Time	724.5	19.3	15
2019-01-01T00:00:00.000/2019-01-03T00:00:00.000	From 1 to 1	No Tips	712.5	67.82	24
2019-01-01T00:00:00.000/2019-01-03T00:00:00.000	From 1 to 132	No Tips	106.0	41.28	3
2019-01-01T00:00:00.000/2019-01-03T00:00:00.000	From 10 to 10	Party Time	14	2.67	2
2019-01-01T00:00:00.000/2019-01-03T00:00:00.000	From 10 to 128	No Tips	52.05	19.47	1

Query – Native JSON

DataSketch - Theta

DataSketch - Theta

```
{  
  "queryType": "groupBy",  
  "dataSource": "case21",  
  "granularity": "day",  
  "dimensions": ["trip_desc", "tip_desc"],  
  "limitSpec": { "type": "default", "limit": 5, "columns": [  
    { "dimension": "trip_desc", "direction": "ascending", "dimensionOrder":  
      "lexicographic" }, {  
      "dimension": "unique_count_thetha", "direction": "descending", "dimensionOrder": "numeric"  
    } ] },  
  "aggregations": [  
    { "type": "longSum", "name": "agg_passenger_count", "fieldName": "total_passenger_count" },  
    { "type": "doubleSum", "name": "agg_fare_amount", "fieldName": "total_fare_amount" },  
    { "type": "thetaSketch", "name": "unique_count_thetha", "fieldName":  
      "fake_id_thetaSketch", "size": 16384 } ],  
  "intervals": [ "2019-01-01T00:00:00.000/2019-01-03T00:00:00.000" ],  
  "having": { "type": "lessThan", "aggregation": "agg_passenger_count", "value": 50 } }
```

```
select  
DATE_TRUNC('day', __time), trip_desc, tip_desc,  
sum(total_fare_amount), sum(total_passenger_count),  
APPROX_COUNT_DISTINCT_DS_THETA(fake_id_thetaSketch)  
from case21  
where (__time >= '2019-01-01T00:00:00.000' and  
__time < '2020-01-03T00:00:00.000')  
group by 1,2,3 having sum(total_passenger_count) < 50  
order by 1 asc, 2 asc, 5 desc limit 5;
```

```
[ {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "agg_passenger_count": 24,  
    "trip_desc": "From 1 to 1",  
    "unique_count_thetha": 11.0,  
    "tip_desc": "No Tips",  
    "agg_fare_amount": 712.5  
  } }, {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "agg_passenger_count": 15,  
    "trip_desc": "From 1 to 1",  
    "unique_count_thetha": 9.0,  
    "tip_desc": "Party Time",  
    "agg_fare_amount": 724.5  
  } }, {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "agg_passenger_count": 3,  
    "trip_desc": "From 1 to 132",  
    "unique_count_thetha": 1.0,  
    "tip_desc": "No Tips",  
    "agg_fare_amount": 106.0  
  } } ]
```

EXPR\$0	trip_desc	tip_desc	EXPR\$3	EXPR\$4	EXPR\$5
2019-01-01T00:00:00.000Z	From 1 to 1	No Tips	712.5	24	11
2019-01-01T00:00:00.000Z	From 1 to 1	Party Time	724.5	15	9
2019-01-01T00:00:00.000Z	From 1 to 132	No Tips	106.0	3	1
2019-01-01T00:00:00.000Z	From 10 to 10	Party Time	14.0	2	2
2019-01-01T00:00:00.000Z	From 10 to 128	No Tips	52.05	1	1

Query – Native JSON DataSketch Theta- Intersect

Usecase – DataSketch Theta: Intersect

trans_time	pickup	drop
2019-01-01	Bronx	Brooklyn
2019-01-01	Queens	Manhattan
2019-01-01	Queens	Bronx
2019-02-01	Manhattan	EWR
2019-02-01	Queens	Bronx

- How many trip combinations (pickup and drop), happened on 01st Jan-19 and also same combination on 01st Feb-19, group by Pickup location.

trans_time	pickup	drop
2019-01-01	Bronx	<Brooklyn>
2019-01-01	Queens	<Manhattan, Bronx>

trans_time	pickup	drop
2019-02-01	Manhattan	<EWR>
2019-02-01	Queens	<Bronx>

- Bronx – 0
- Queens – 1 (Queens to Bronx).

DataSketch Theta: Intersect

```
{  
  "queryType": "groupBy",  
  "dataSource": "case21",  
  "granularity": "ALL",  
  "dimensions": ["PULocationID"],  
  "aggregations": [  
    {  
      "type" : "filtered",  
      "filter" : {  
        "type" : "interval",  
        "dimension" : "__time",  
        "intervals" : ["2019-01-01T00:00:00.000Z/2019-01-02T00:00:00.000Z"]  
      },  
      "aggregator" : {  
        "type": "thetaSketch", "name": "unique_trips_jan_day1", "fieldName": "trip_desc"  
      }  }, {  
      "type" : "filtered",  
      "filter" : {  
        "type" : "interval",  
        "dimension" : "__time",  
        "intervals" : ["2019-02-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]  },  
      "aggregator" : {  
        "type": "thetaSketch",  
        "name": "unique_trips_feb_day1",  
        "fieldName": "trip_desc"  
      }  }],  
  "LNC.ADONI@GMAIL.COM
```

```
  "postAggregations": [  
    {  
      "type": "thetaSketchEstimate",  
      "name": "final_unique_trips",  
      "field": "  
      {  
        "type": "thetaSketchSetOp",  
        "name": "final_unique_trips_sketch",  
        "func": "INTERSECT",  
        "fields": [  
          {  
            "type": "fieldAccess",  
            "fieldName": "unique_trips_jan_day1"  
          },  
          {  
            "type": "fieldAccess",  
            "fieldName": "unique_trips_feb_day1"  
          }  
        ]  
      }  
    },  
    {"intervals":  
      ["2019-01-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]  
    }  
  ]
```



```
[  
  {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
      "final_unique_trips" : 1.0,  
      "unique_trips_feb_day1" : 2.0,  
      "PULocationID" : "1",  
      "unique_trips_jan_day1" : 2.0  
    }  
  }, {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
      "final_unique_trips" : 25.0,  
      "unique_trips_feb_day1" : 51.0,  
      "PULocationID" : "10",  
      "unique_trips_jan_day1" : 47.0  
    }  
  }, {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
      "final_unique_trips" : 129.0,  
      "unique_trips_feb_day1" : 150.0,  
      "PULocationID" : "100",  
      "unique_trips_jan_day1" : 159.0  
    }  
  }]
```

Query – Native JSON DataSketch Theta - NOT

Usecase – DataSketch Theta: Not

trans_time	pickup	drop
2019-01-01	Bronx	Brooklyn
2019-01-01	Queens	Manhattan
2019-01-01	Queens	Bronx
2019-02-01	Manhattan	EWR
2019-02-01	Queens	Bronx

- How many trip combinations (pickup and drop), happened on 01st Jan-19 which did not happen (the combination) on 01st Feb-19, group by Pickup location.

trans_time	pickup	drop
2019-01-01	Bronx	<Brooklyn>
2019-01-01	Queens	<Manhattan, Bronx>

trans_time	pickup	drop
2019-02-01	Manhattan	<EWR>
2019-02-01	Queens	<Bronx>

- Bronx – 1 (Bronx to Brooklyn)
- Queens – 1 (Queens to Manhattan).

DataSketch Theta: Intersect

```
{  
  "queryType": "groupBy",  
  "dataSource": "case21",  
  "granularity": "ALL",  
  "dimensions": ["PULocationID"],  
  "aggregations": [ {  
      "type" : "filtered",  
      "filter" : {  
          "type" : "interval",  
          "dimension" : "__time",  
          "intervals" :  
            ["2019-01-01T00:00:00.000Z/2019-01-02T00:00:00.000Z"]      },  
      "aggregator" : {  
          "type": "thetaSketch", "name": "unique_trips_jan_day1", "fieldName":  
          "trip_desc"  
        }  }, {  
      "type" : "filtered",  
      "filter" : {  
          "type" : "interval",  
          "dimension" : "__time",  
          "intervals" :  
            ["2019-02-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]      },  
      "aggregator" : {  
          "type": "thetaSketch",  
          "name": "unique_trips_feb_day1",  
          "fieldName": "trip_desc"  
        }  },  
  
  "postAggregations": [  {  
      "type": "thetaSketchEstimate",  
      "name": "final_unique_trips_inJan_notInFeb",  
      "field":  
        {  
          "type": "thetaSketchSetOp",  
          "name": "final_unique_trips_sketch",  
          "func": "NOT",  
          "fields": [    {  
              "type": "fieldAccess",  
              "fieldName": "unique_trips_jan_day1"  
            },    {  
              "type": "fieldAccess",  
              "fieldName": "unique_trips_feb_day1"  
            }  ]  },  
        {  
          "type": "thetaSketchEstimate",  
          "name": "final_unique_trips",  
          "field":  
            {  
              "type": "thetaSketchSetOp",  
              "name": "final_unique_trips_sketch",  
              "func": "INTERSECT",  
              "fields": [    {  
                  "type": "fieldAccess",  
                  "fieldName": "unique_trips_jan_day1"  
                },    {  
                  "type": "fieldAccess",  
                  "fieldName": "unique_trips_feb_day1"  
                }  ]  }  ],  
      "intervals":  
        ["2019-01-01T00:00:00.000/2019-02-02T00:00:00.000"]  } ]}
```

```
"name": "final_unique_trips_inJan_notInFeb",  
"field":  
{  
  "type": "thetaSketchSetOp",  
  "name": "final_unique_trips_sketch",  
  "func": "NOT",  
  "fields": [    {  
      "type": "fieldAccess",  
      "fieldName": "unique_trips_jan_day1"  
    },    {  
      "type": "fieldAccess",  
      "fieldName": "unique_trips_feb_day1"  
    }  ]  },  
  {  
    "type": "thetaSketchEstimate",  
    "name": "final_unique_trips",  
    "field":  
      {  
        "type": "thetaSketchSetOp",  
        "name": "final_unique_trips_sketch",  
        "func": "INTERSECT",  
        "fields": [          {  
            "type": "fieldAccess",  
            "fieldName": "unique_trips_jan_day1"  
          },          {  
            "type": "fieldAccess",  
            "fieldName": "unique_trips_feb_day1"  
          }        ]      }    },  
    "intervals":  
      ["2019-01-01T00:00:00.000/2019-02-02T00:00:00.000"]  }
```

```
[ {  
  "version" : "v1",  
  "timestamp" : "2019-01-01T00:00:00.000Z",  
  "event" : {  
    "final_unique_trips_inJan_notInFeb" : 1.0,  
    "unique_trips_feb_day1" : 2.0,  
    "PULocationID" : "1",  
    "unique_trips_jan_day1" : 2.0,  
    "final_unique_trips_both_Jan_Feb" : 1.0  
  }  
, {  
  "version" : "v1",  
  "timestamp" : "2019-01-01T00:00:00.000Z",  
  "event" : {  
    "final_unique_trips_inJan_notInFeb" : 22.0,  
    "unique_trips_feb_day1" : 51.0,  
    "PULocationID" : "10",  
    "unique_trips_jan_day1" : 47.0,  
    "final_unique_trips_both_Jan_Feb" : 25.0  
  }  
, {  
  "version" : "v1",  
  "timestamp" : "2019-01-01T00:00:00.000Z",  
  "event" : {  
    "final_unique_trips_inJan_notInFeb" : 30.0,  
    "unique_trips_feb_day1" : 150.0,  
    "PULocationID" : "100",  
    "unique_trips_jan_day1" : 159.0,  
    "final_unique_trips_both_Jan_Feb" : 129.0  
  }  
}
```

Query – Native JSON DataSketch Theta - UNION

Usecase – DataSketch Theta: UNION

trans_time	pickup	drop
2019-01-01	Bronx	Brooklyn
2019-01-01	Queens	Manhattan
2019-01-01	Queens	Bronx
2019-02-01	Manhattan	EWR
2019-02-01	Queens	Bronx

- How many trip combinations (pickup and drop), happened on 01st Jan-19 union with the trips(the combination) on 01st Feb-19, group by Pickup location. i.e trip in either 01st Jan-19 or 01st Feb-19.

Queens to Bronx is counted only once.

trans_time	pickup	drop
2019-01-01	Bronx	<Brooklyn>
2019-01-01	Queens	<Manhattan, Bronx>

trans_time	pickup	drop
2019-02-01	Manhattan	<EWR>
2019-02-01	Queens	<Bronx>

- Bronx – 1 (Bronx to Brooklyn)
- Queens – 2 (Queens to Manhattan, Bronx).
- Manhattan – 1 (Manhattan to EWR)

DataSketch Theta: Intersect

```
{  
  "queryType": "groupBy",  
  "dataSource": "case21",  
  "granularity": "ALL",  
  "dimensions": ["PULocationID"],  
  "aggregations": [  
    {  
      "type": "filtered",  
      "filter": {  
        "type": "interval",  
        "dimension": "__time",  
        "intervals": ["2019-01-01T00:00:00.000Z/2019-01-02T00:00:00.000Z"]  
      },  
      "aggregator": {  
        "type": "thetaSketch", "name": "unique_trips_jan_day1",  
        "fieldName": "trip_desc"  
      },  
      {  
        "type": "filtered",  
        "filter": {  
          "type": "interval",  
          "dimension": "__time",  
          "intervals": ["2019-02-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]  
        },  
        "aggregator": {  
          "type": "thetaSketch", "name": "unique_trips_feb_day1", "fieldName":  
            "trip_desc"  
        }  
      },  
      "postAggregations": [  
        {  
          "type": "thetaSketchEstimate",  
          "name": "final_unique_trips_union_Jan_Feb",  
          "field": {  
            "type": "thetaSketchSetOp",  
            "name": "final_unique_trips_sketch",  
            "func": "UNION",  
            "fields": [  
              {  
                "type": "fieldAccess",  
                "fieldName": "unique_trips_jan_day1"  
              },  
              {  
                "type": "fieldAccess",  
                "fieldName": "unique_trips_feb_day1"  
              }  
            ]  
          }  
        },  
        {"intervals": ["2019-01-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]}  
      ]  
    ]  
  ]  
}
```



```
[ {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "unique_trips_feb_day1": 2.0,  
    "final_unique_trips_union_Jan_Feb": 3.0,  
    "PULocationID": "1",  
    "unique_trips_jan_day1": 2.0  
  }  
, {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "unique_trips_feb_day1": 51.0,  
    "final_unique_trips_union_Jan_Feb": 73.0,  
    "PULocationID": "10",  
    "unique_trips_jan_day1": 47.0  
  }  
, {  
  "version": "v1",  
  "timestamp": "2019-01-01T00:00:00.000Z",  
  "event": {  
    "unique_trips_feb_day1": 150.0,  
    "final_unique_trips_union_Jan_Feb": 180.0,  
    "PULocationID": "100",  
    "unique_trips_jan_day1": 159.0  
  }  
}
```

Query – Native JSON

DataSketch - HLL

DataSketch HLL

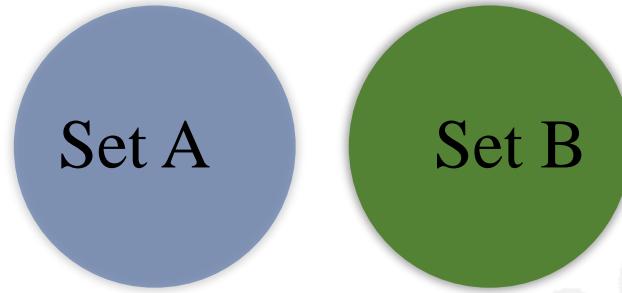
```
{  
  "queryType": "groupBy",  
  "dataSource": "case21",  
  "granularity": "ALL",  
  "dimensions": ["PULocationID"],  
  "aggregations": [ {  
      "type" : "filtered",  
      "filter" : {  
          "type" : "interval",  
          "dimension" : "__time",  
          "intervals" : ["2019-01-01T00:00:00.000Z/2019-01-02T00:00:00.000Z"]  
      },  
      "aggregator" : {  
          "type": "HLLSketchBuild",  
          "name": "unique_trips_jan_day1",  
          "fieldName": "trip_desc", "lgK" : 16, "tgtHllType" : "HLL_8"  
      } }, {  
      "type" : "filtered",  
      "filter" : {  
          "type" : "interval",  
          "dimension" : "__time",  
          "intervals" : ["2019-02-01T00:00:00.000Z/2019-02-02T00:00:00.000Z"]  
      },  
      "aggregator" : {  
          "type": "HLLSketchBuild", "name": "unique_trips_feb_day1", "fieldName": "trip_desc", "lgK" :  
16, "tgtHllType" : "HLL_8  
      } } ],  
  "intervals": ["2019-01-01T00:00:00.000/2019-02-02T00:00:00.000"] }
```



```
[ {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
        "unique_trips_feb_day1" : 2.000000004967054,  
        "PULocationID" : "1",  
        "unique_trips_jan_day1" : 2.000000004967054  
    }  
, {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
        "unique_trips_feb_day1" : 51.000006332994424,  
        "PULocationID" : "10",  
        "unique_trips_jan_day1" : 47.00000536938579  
    }  
, {  
    "version" : "v1",  
    "timestamp" : "2019-01-01T00:00:00.000Z",  
    "event" : {  
        "unique_trips_feb_day1" : 150.00005550684915,  
        "PULocationID" : "100",  
        "unique_trips_jan_day1" : 159.00006239119014  
    }  
, {
```

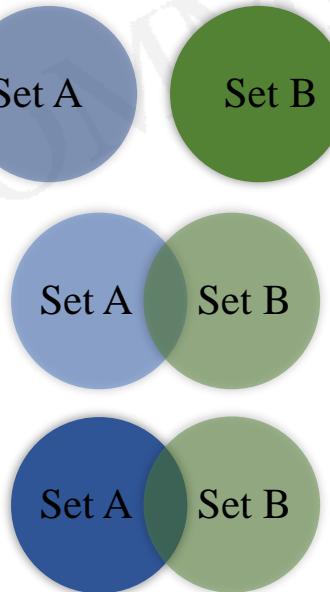
DataSketch: HLL vs Theta

DataSketch: HLL vs Theta



DataSketch: HLL

- Union – Present in A or B, no duplicate counting.



DataSketch: Theta

- Union – Present in A or B, no duplicate counting.
- Intersect – Present in both A and B.
- NOT – Present in one set but not in another set. Ex: present in A but not in B.

Data Extraction Queries – Scan, Select

Data Extraction Query - Scan

- Used to extract **bulk raw** data from Druid
- Only raw data as is extracted with filters. No aggregation in queries/extract.
- The data is returned in a single instance, without pagination.
- The data sent across to client is discarded by Druid (retain only relevant data in memory, send to client and discard). Thus requiring memory for only the data which is being send over the network.
- Efficient over Select query.

Data Extraction Query – Scan (list)

```
{  
  "queryType": "scan",  
  "dataSource": "case19",  
  "resultFormat": "list",  
  "filter": {  
    "type" : "javascript",  
    "dimension" : "total_passenger_count",  
    "function" : "function(value) { return (value>2); }"  
  },}  
  
  "columns": ["__time", "fare_amount_bucket", "tip_desc", "trip_desc",  
  "total_row_count", "total_passenger_count", "total_trip_distance",  
  "total_fare_amount"],  
  "intervals": [ "2019-01-01/2019-10-10" ],  
  "batchSize":10000000,  
  "limit":10,  
  "order": "ascending"  
}
```

```
"segmentId" : "case19_2019-01-18T00:00:00.000Z_2019-01-19T00:00:00.000Z_2019-10-07T14:24:00.300Z",  
"columns" : [ "__time", "fare_amount_bucket", "tip_desc", "trip_desc", "total_row_count", "total_passenger_count", "total_trip_distance", "total_fare_amount" ],  
"events" : [ {  
  "__time" : 1547809200000,  
  "fare_amount_bucket" : "6 to 10",  
  "tip_desc" : "Party Time",  
  "trip_desc" : "From 75 to 237",  
  "total_row_count" : 1,  
  "total_passenger_count" : 3,  
  "total_trip_distance" : 1.2,  
  "total_fare_amount" : 6.5  
} ]  
}, {  
  "segmentId" : "case19_2019-01-18T00:00:00.000Z_2019-01-19T00:00:00.000Z_2019-10-07T14:24:00.300Z",  
  "columns" : [ "__time", "fare_amount_bucket", "tip_desc", "trip_desc", "total_row_count", "total_passenger_count", "total_trip_distance", "total_fare_amount" ],  
  "events" : [ {  
    "__time" : 1547809200000,  
    "fare_amount_bucket" : "6 to 10",  
    "tip_desc" : "Party Time",  
    "trip_desc" : "From 75 to 41",  
    "total_row_count" : 1,  
    "total_passenger_count" : 3,  
    "total_trip_distance" : 1.04,  
    "total_fare_amount" : 6.0  
  } ]
```

- Only **columns** mentioned would be returned. If left blank, all columns in the cube will be returned.
- Druid uses memory for rows specified in **batchSize**.
- Only the number of rows specified in **limit** will be returned.

Data Extraction Query – Scan (compactedList)

```
{  
  "queryType": "scan",  
  "dataSource": "case19",  
  "resultFormat": "compactedList",  
  "filter": {  
    "type" : "javascript",  
    "dimension" : "total_passenger_count",  
    "function" : "function(value) { return (value>2); }"  
  },}  
  "columns":["__time","fare_amount_bucket", "tip_desc", "trip_desc",  
  "total_row_count", "total_passenger_count", "total_trip_distance",  
  "total_fare_amount"],  
  "intervals": [ "2019-01-01/2019-10-10" ],  
  "batchSize":10000000,  
  "limit":10,  
  "order": "ascending"  
}
```

```
}, {  
  "segmentId" : "case19_2019-01-26T00:00:00.000Z_2019-01-27T00:00:00.000Z_2019-10-07T14:24:00.300Z",  
  "columns" : [ "__time", "fare_amount_bucket", "tip_desc", "trip_desc", "total_row_count", "total_passenger_count", "total_trip_distance", "total_fare_amount" ],  
  "events" : [ [ 1548507600000, "more than 10", "Party Time", "From 68 to 211", 1, 6, 2.25, 12.5 ] ]  
}, {  
  "segmentId" : "case19_2019-01-26T00:00:00.000Z_2019-01-27T00:00:00.000Z_2019-10-07T14:24:00.300Z",  
  "columns" : [ "__time", "fare_amount_bucket", "tip_desc", "trip_desc", "total_row_count", "total_passenger_count", "total_trip_distance", "total_fare_amount" ],  
  "events" : [ [ 1548507600000, "more than 10", "Party Time", "From 68 to 230", 1, 6, 2.38, 12.5 ] ]  
}, {  
  "segmentId" : "case19_2019-01-26T00:00:00.000Z_2019-01-27T00:00:00.000Z_2019-10-07T14:24:00.300Z",  
  "columns" : [ "__time", "fare_amount_bucket", "tip_desc", "trip_desc", "total_row_count", "total_passenger_count", "total_trip_distance", "total_fare_amount" ],  
  "events" : [ [ 1548507600000, "more than 10", "Party Time", "From 68 to 231", 1, 4, 2.1, 10.5 ] ]  
}
```

Connecting to Druid using SQL Editor (DBeaver)

Avatica – JDBC Driver for Druid

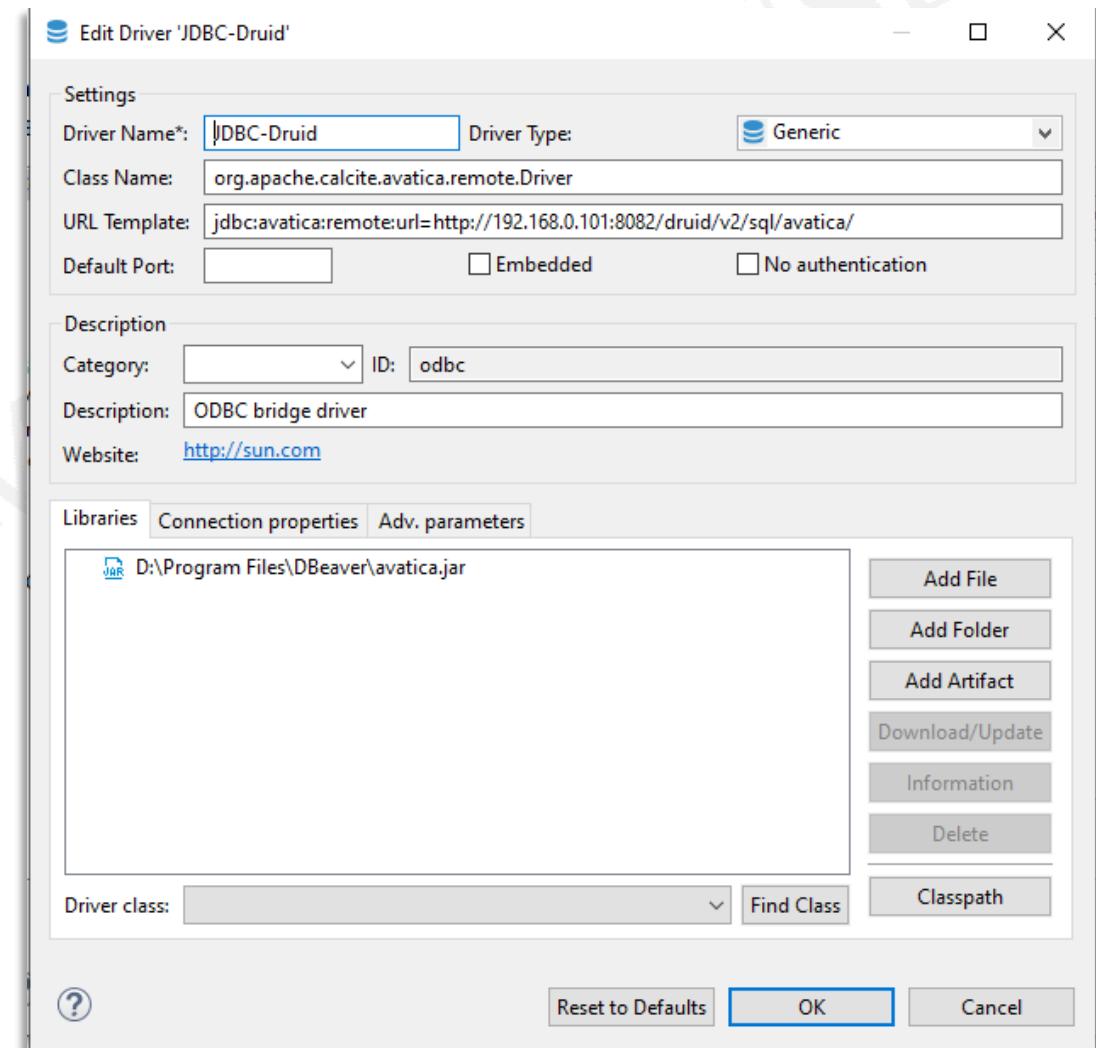
- Download JDBC Driver from
<https://calcite.apache.org/avatica/downloads/avatica.html>
- Create a ODBC or JDBC connection in DBeaver using the downloaded JDBC Driver.

Class Name:

`org.apache.calcite.avatica.remote.Driver`

URL Template:

`jdbc:avatica:remote:url=http://192.168.0.101:8082/druid/v2/sql/avatica/`



DBeaver - Query

JDBC-Druid - New Connection

druid

INFORMATION_SCHEMA

druid

Tables

case1

case10

case10.1

case11

case12

case19

case2

case20

case21

case3

case4

case5

case6

case7

case8

case9

imbd_sample_multivaluedimension

Views

Indexes

Procedures

Data Types

sys

SELECT * FROM "case19" LIMIT 100;

Result

SELECT * FROM "case19" LIMIT 100 | Enter a SQL expression to filter results (use Ctrl+Space)

	DOlocationID	PULocationID	RatecodeID	VendorID	_time	fare_amount_bucket
1	193	193	1	2	2001-02-02 09:00:00	1 to 5
2	193	264	1	2	2002-12-31 18:00:00	1 to 5
3	193	264	1	2	2002-12-31 18:00:00	1 to 5
4	116	132	2	2	2008-12-31 01:00:00	more than 10
5	89	132	1	2	2008-12-31 16:00:00	more than 10
6	107	107	1	2	2008-12-31 17:00:00	1 to 5
7	148	232	1	2	2008-12-31 17:00:00	1 to 5
8	234	234	1	2	2008-12-31 17:00:00	1 to 5
9	236	236	1	2	2008-12-31 17:00:00	1 to 5

Connecting to Druid using ETL Tool(**Talend**)

Avatica – JDBC Driver for Druid

- Download JDBC Driver from
<https://calcite.apache.org/avatica/downloads/avatica.html>
- Create a ODBC or JDBC connection in DBeaver using the downloaded JDBC Driver.

Class Name:

`org.apache.calcite.avatica.remote.Driver`

URL Template:

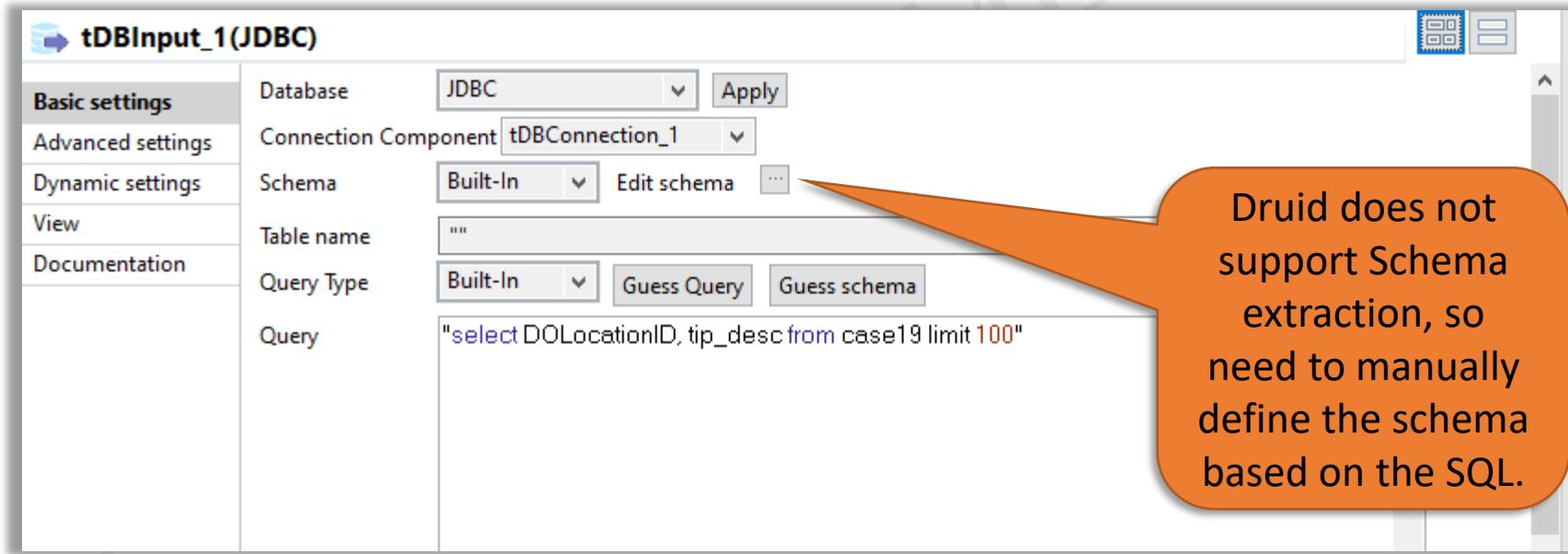
`jdbc:avatica:remote:url=http://192.168.0.1
01:8082/druid/v2/sql/avatica/`



Talend - Query



Use the connection,
to execute Query.



Tips

- Do not create additional dimensions on `_time`
 - No need to create additional dimensions – year, month, quarter...; these are usually created in traditional star or snowflake schema.
- Create as many dimensions as possible, based on business requirement.
 - `Concat(category,'-',sub-category)`
 - `Concat(state,'-',city)`
- Always use `druid.sql.planner.useApproximateCountDistinct=true` (approximate count). Use DataSketch-Theta instead for approximate count.
- In historical node runtime properties file, increase value of `druid.query.groupBy.maxOnDiskStorage` (in bytes). This will enhance the number of rows to process for groupBy queries.
 - This will have performance overhead, as the data would be written to disk for temporary storage and computation.

JavaScript Usage

- Ingestion
 - Filter
- Query
 - Aggregations
 - Post-Aggregations
 - DimensionSpec
 - Ex: `select concat('category', '-' , 'sub-category') new_dim, sum(sales)`
`from cube group by new_dim`

Thank You

References

- History of RDBMS
 - <https://www.slideshare.net/sethuntu/introduction-history-of-dbms>
 - <https://www.slideshare.net/HafizHamza1/history-of-database-62559958>
- Cassandra
 - <https://www.youtube.com/watch?v=B-HTdrTgGNs>
- Druid
 - <https://www.slideshare.net/KashifKhan54/druid-deep-dive>
 - <https://www.youtube.com/watch?v=eCbXoGSyHbg>
 - <https://www.slideshare.net/doriwaldman/druid-88876307>
- Parquet, ORC
 - https://www.youtube.com/watch?v=_0Wpj_gvzg
 - https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet.html
 - <https://community.cloudera.com/t5/Support-Questions/ORC-vs-Parquet-When-to-use-one-over-the-other/td-p/95942>
 - <https://blog.cloudera.com/orcfile-in-hdp-2-better-compression-better-performance/>