Altinity

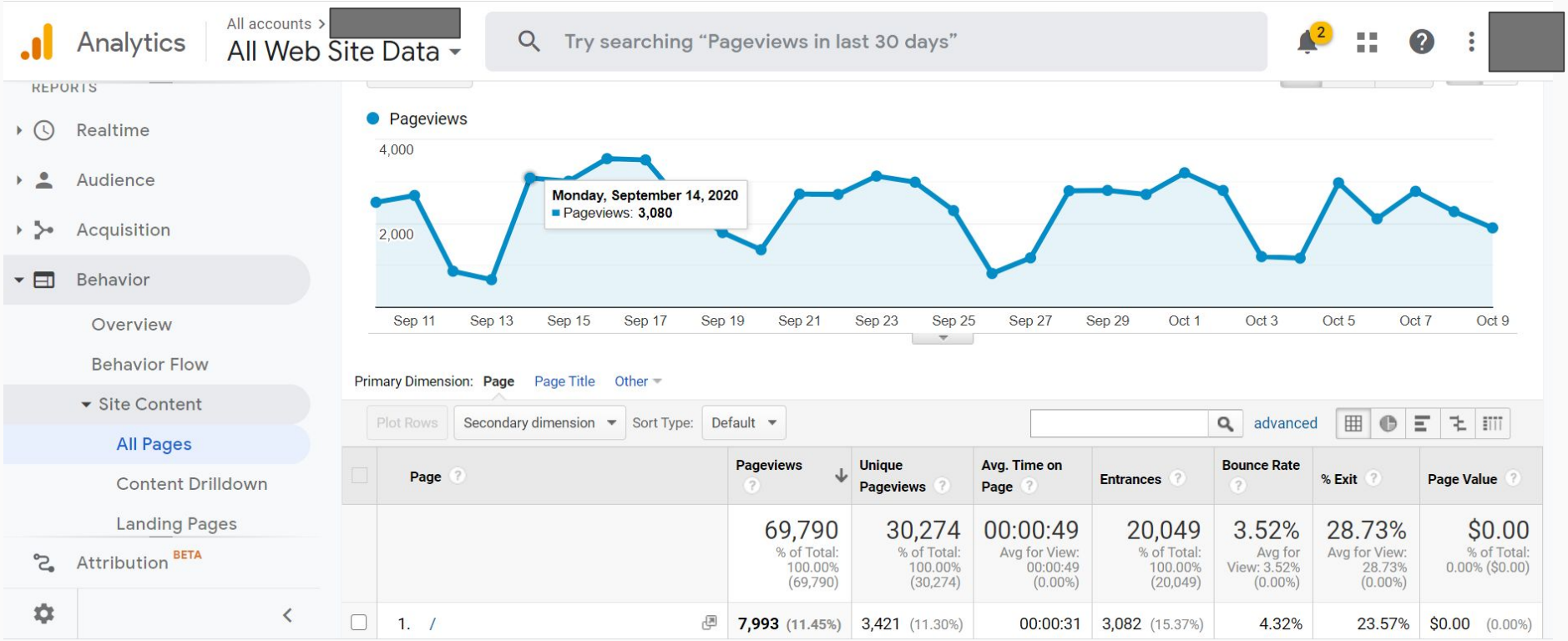
Data Warehouse Basics

Intro to SQL Databases for Analytic Applications

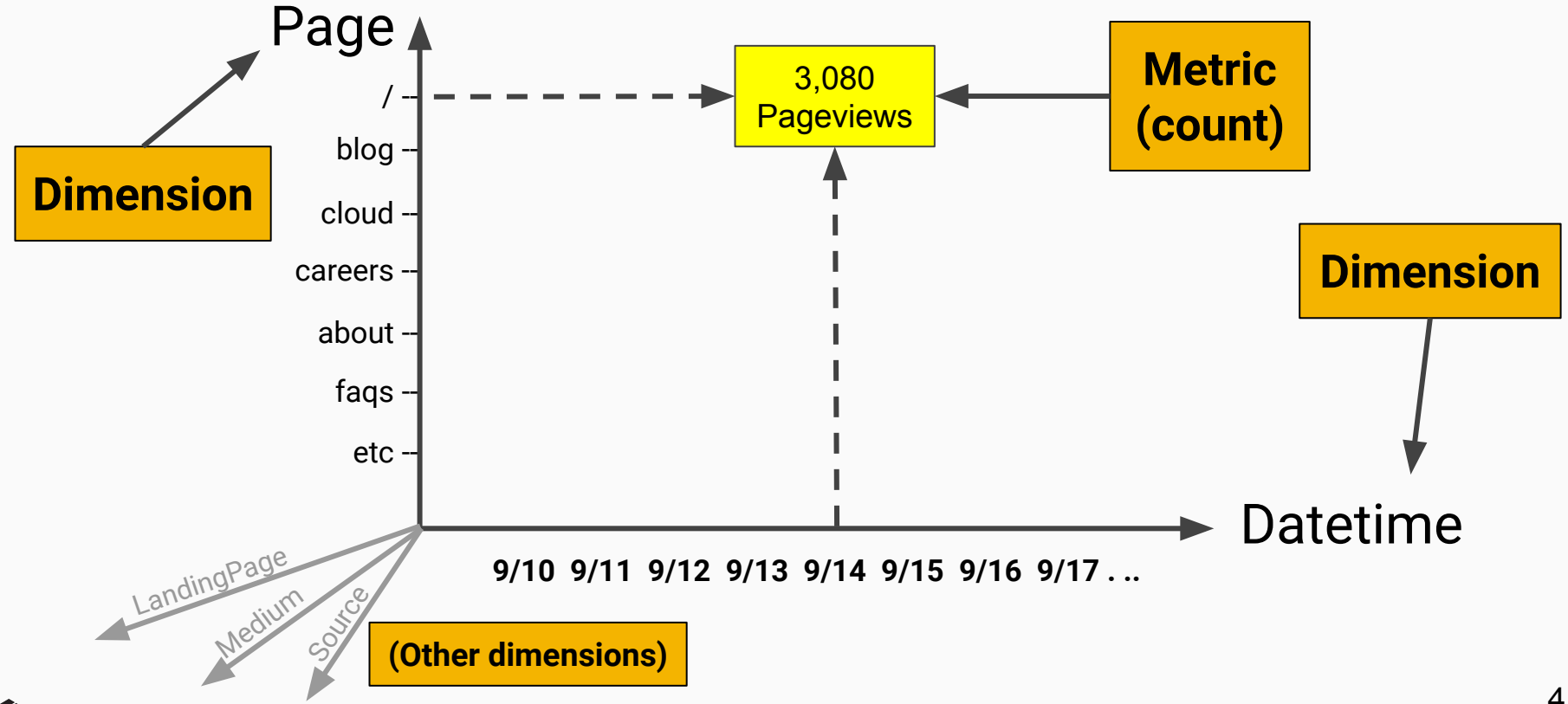
Altinity Engineering Team

What is a data warehouse?

Let’s start with a concrete problem



OLAP: Online Analytical Processing



Technical challenges

Scanning and aggregating large amounts of data

I/O and compute intensive!

Choosing different ad-hoc slices of data

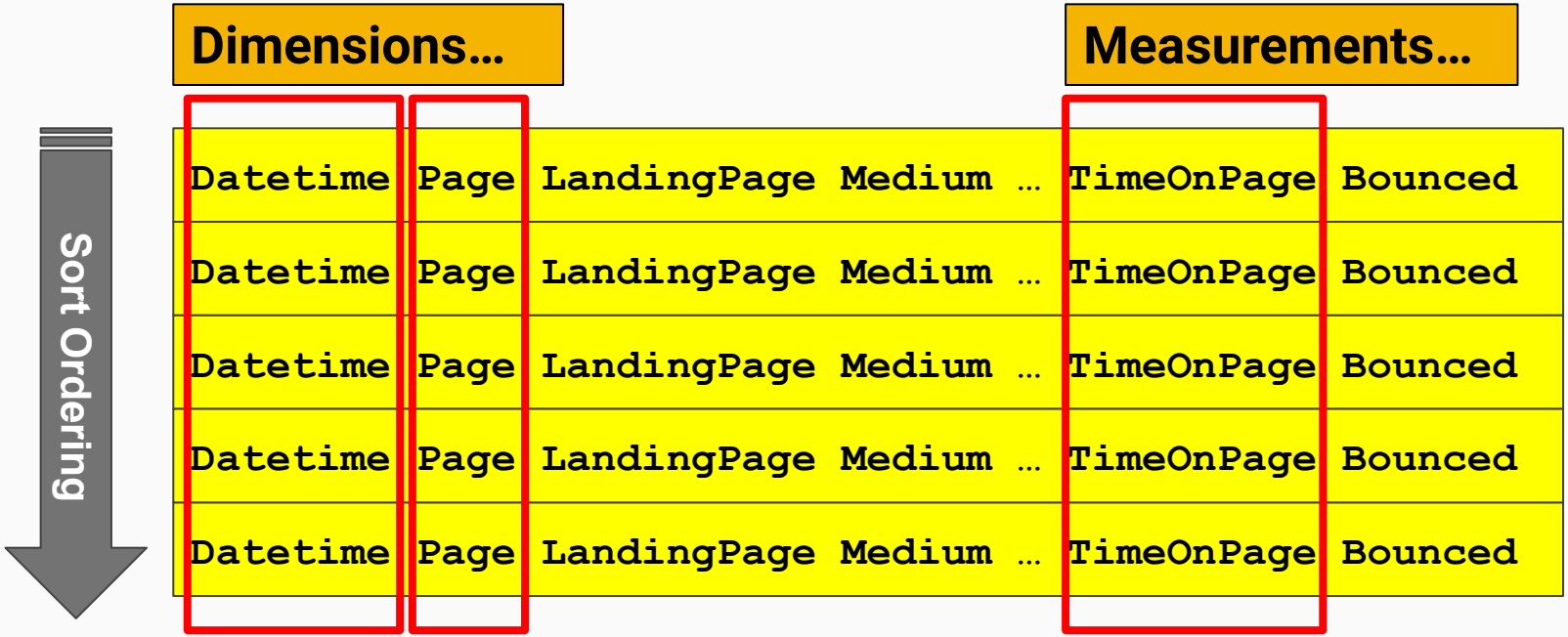
Caches, indexes, pre-aggregation have limited value

Delivering extremely fast responses

Response required in as little as 20ms

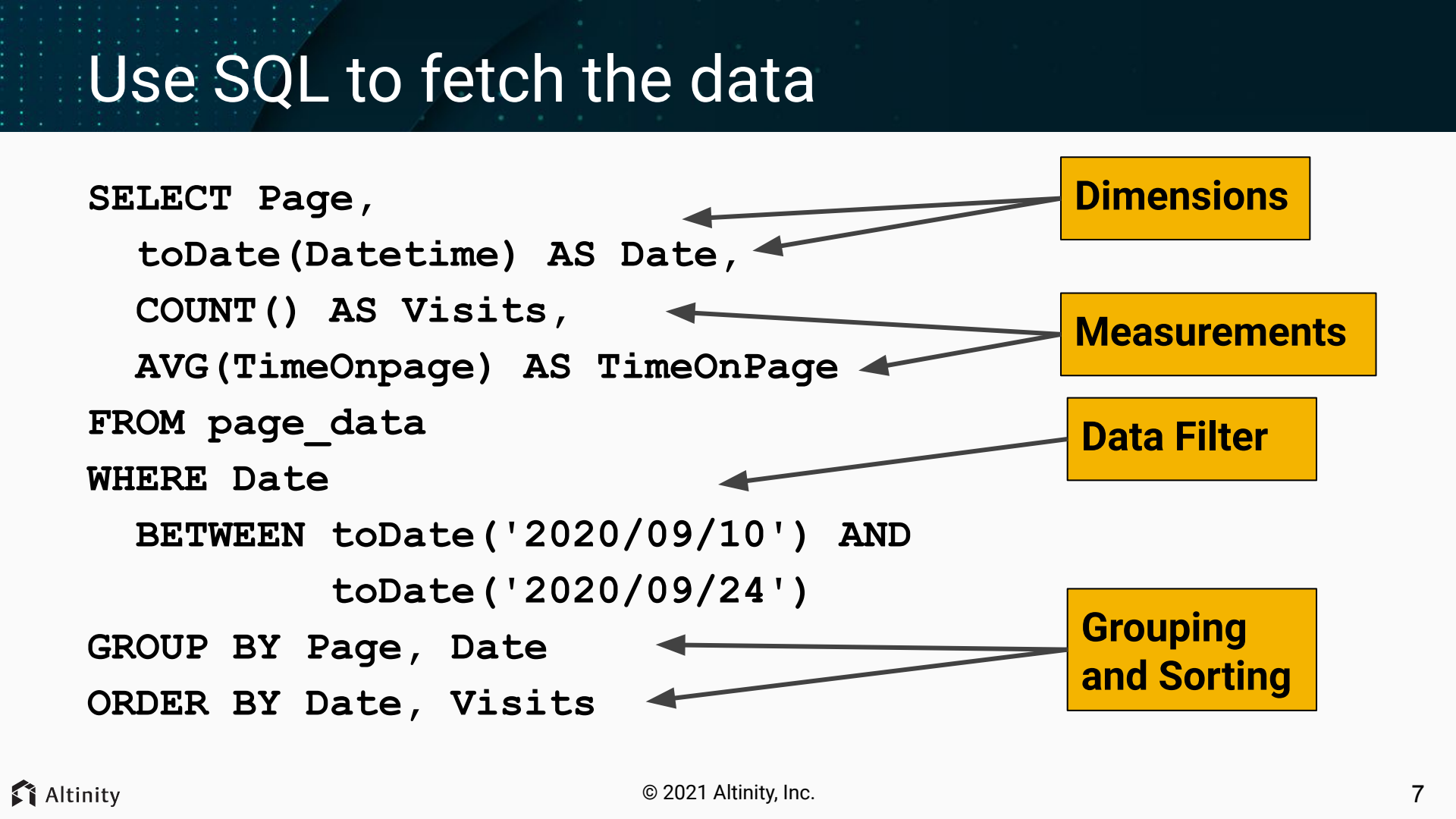
5

Solution: Store table data as sorted columns

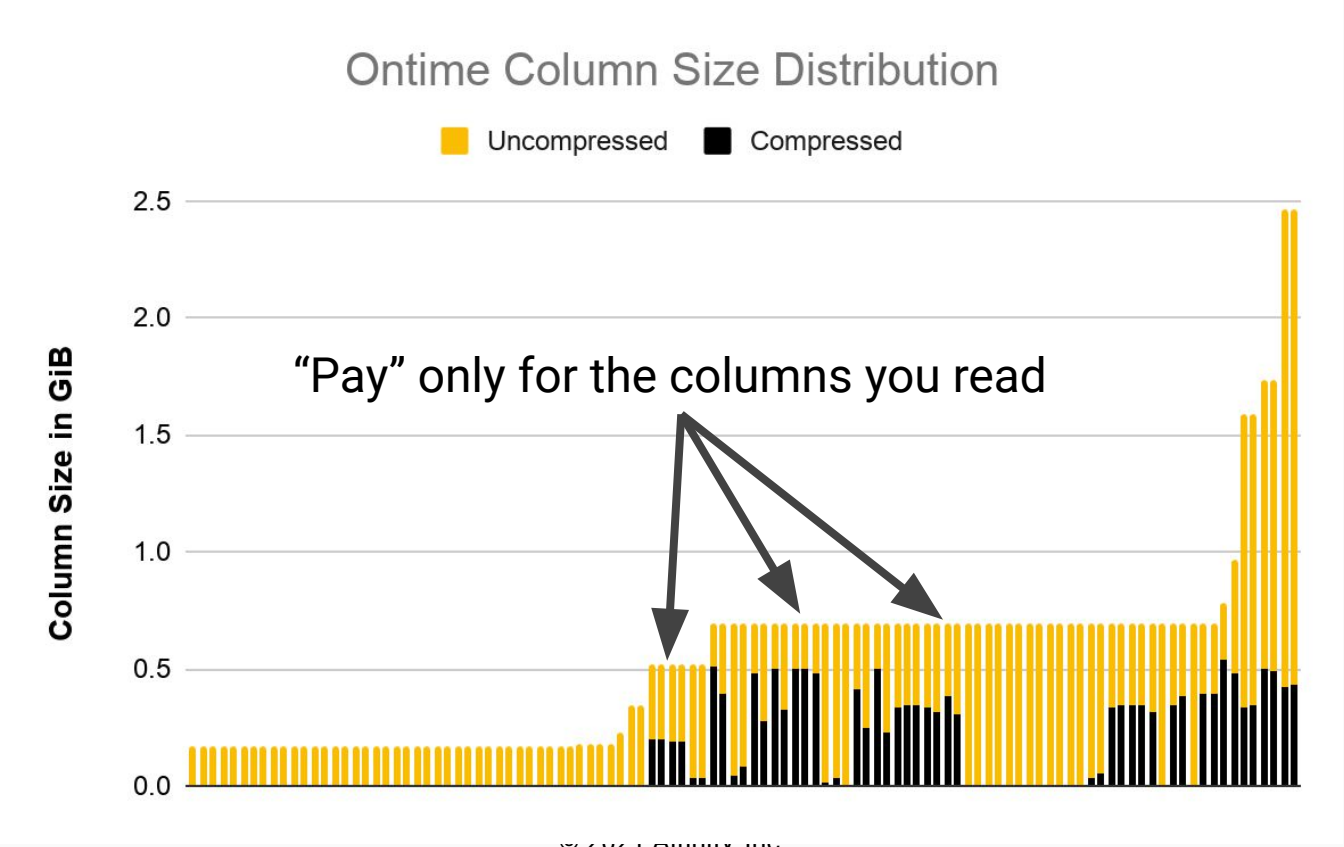


Use SQL to fetch the data

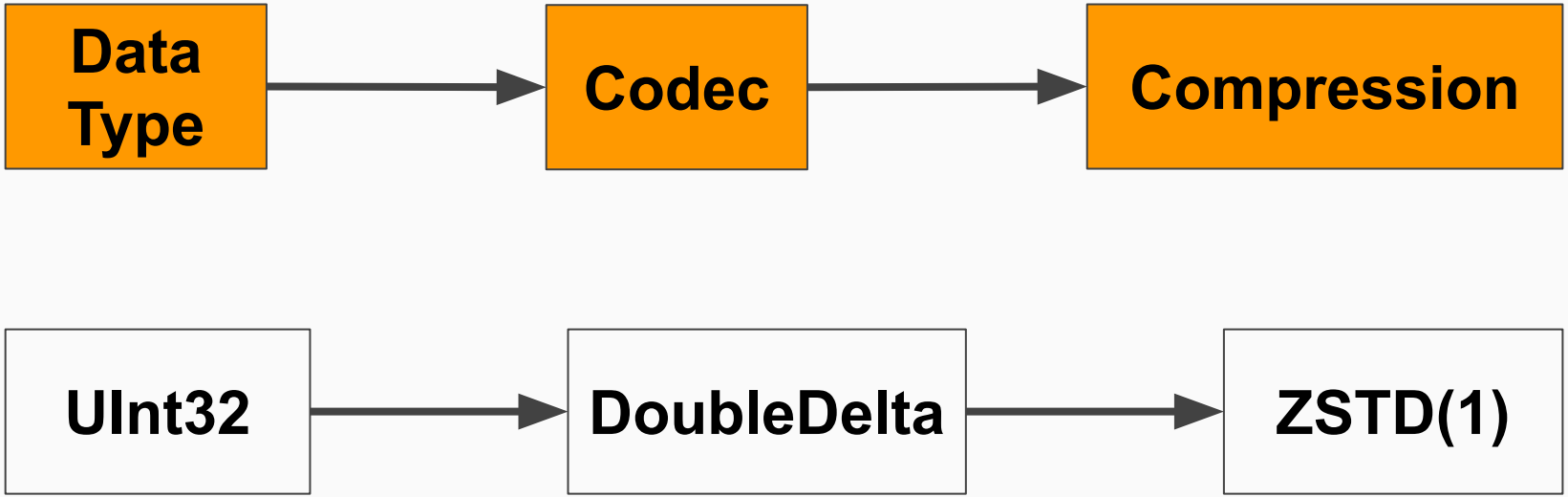
```
SELECT Page,
  toDate(Datetime) AS Date,
  COUNT() AS Visits,
  AVG(TimeOnpage) AS TimeOnPage
FROM page_data
WHERE Date
  BETWEEN toDate('2020/09/10') AND
          toDate('2020/09/24')
GROUP BY Page, Date
ORDER BY Date, Visits
```



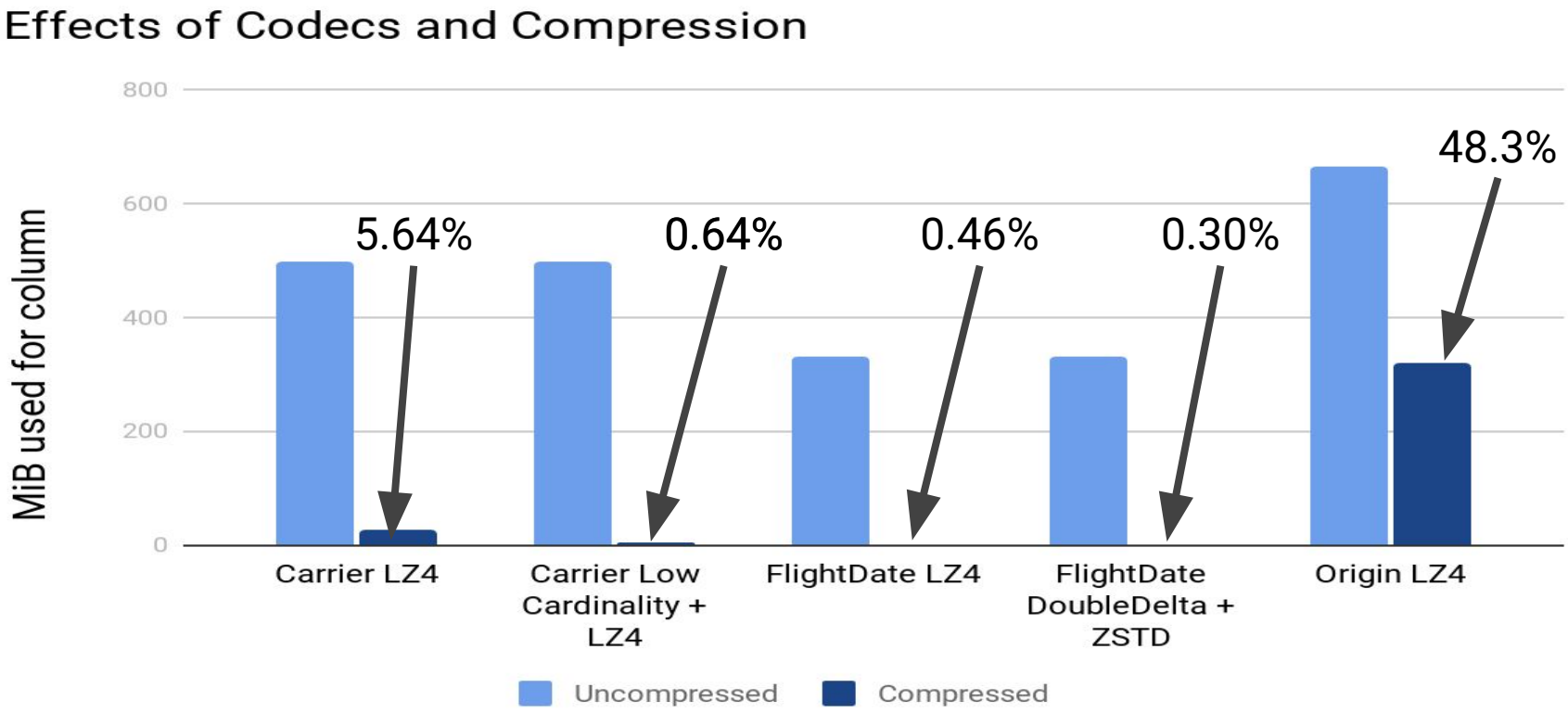
No penalty for columns you don’t use



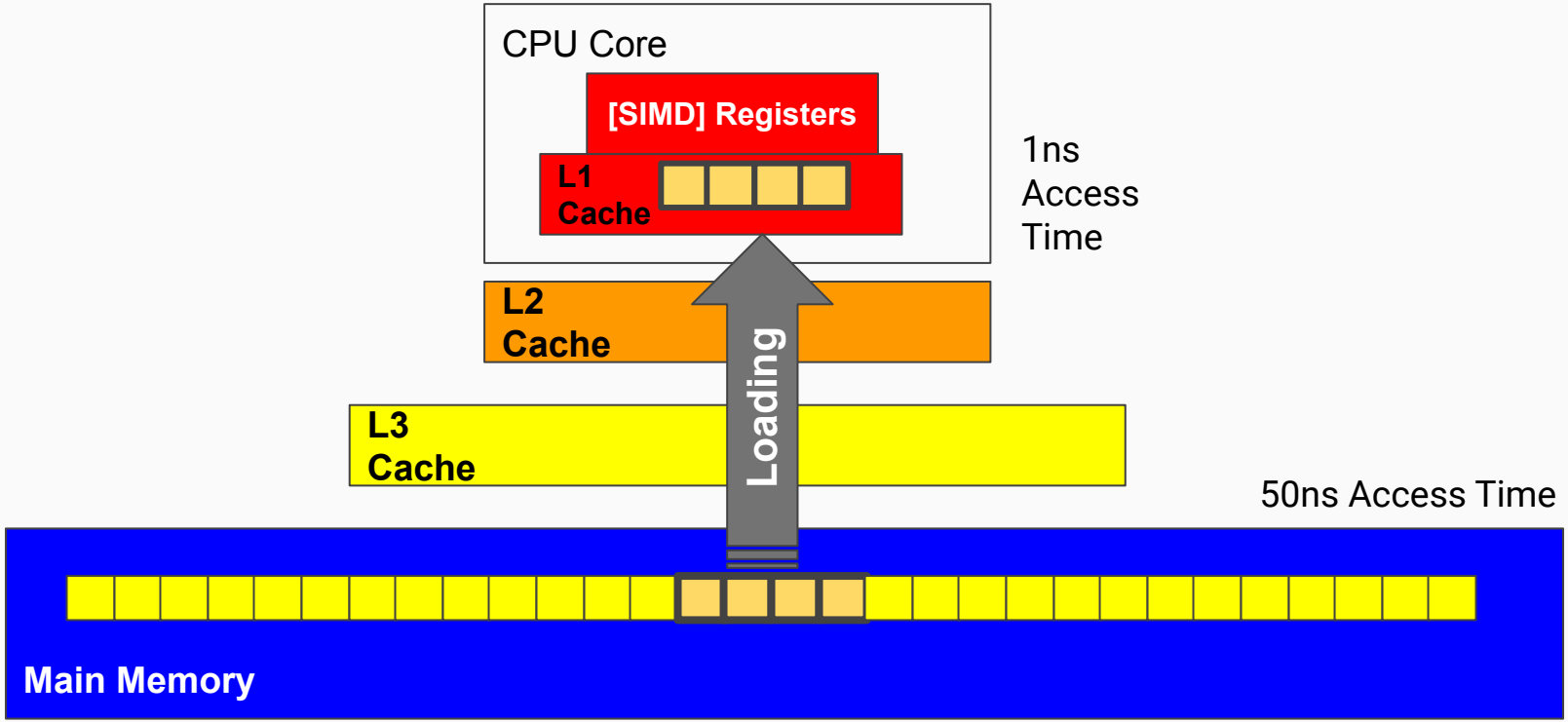
Arrays compress very well



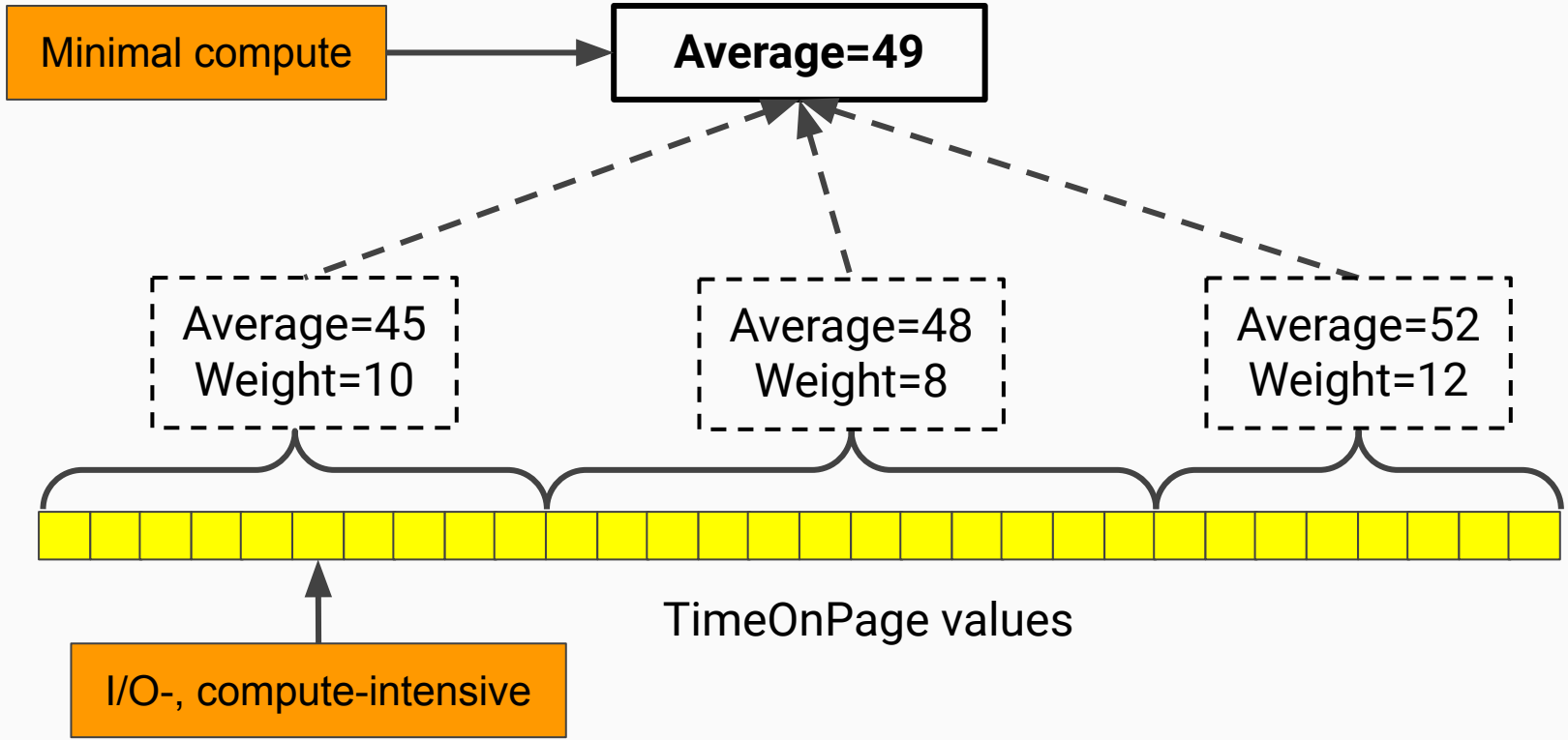
Data reductions from real data



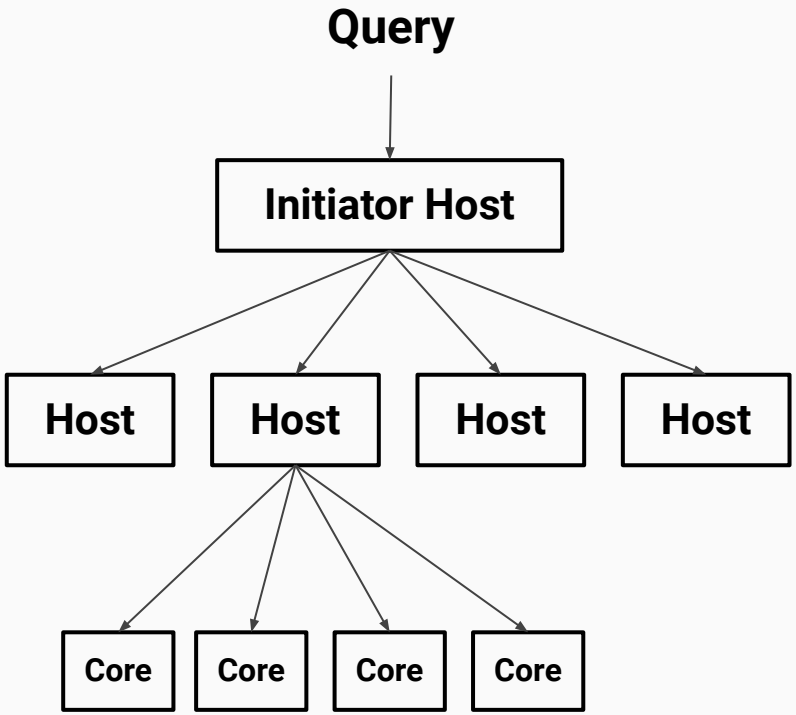
Array layout assists hardware performance

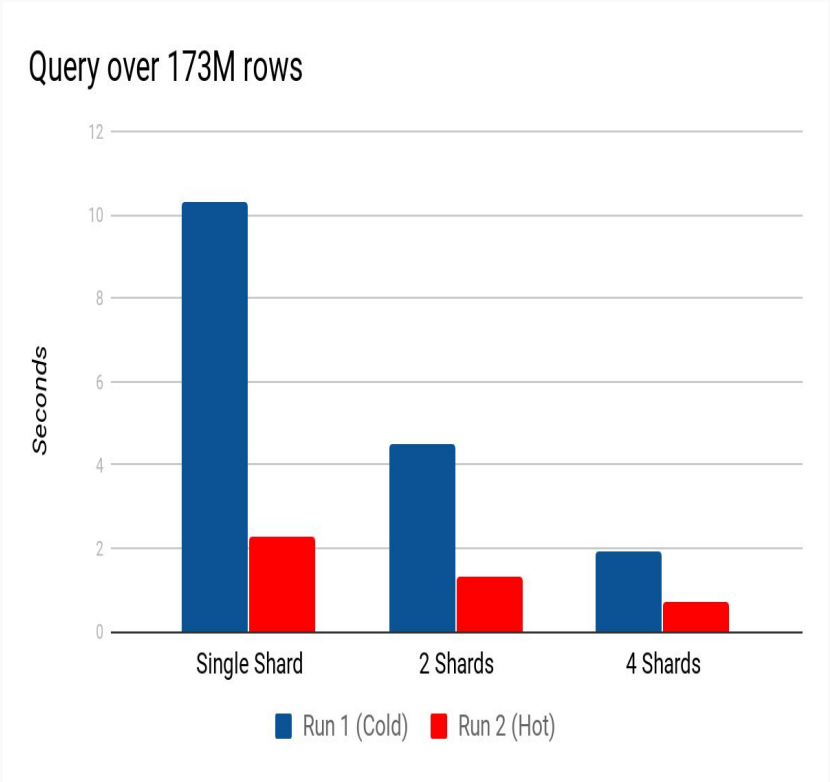


Aggregate computations distribute well

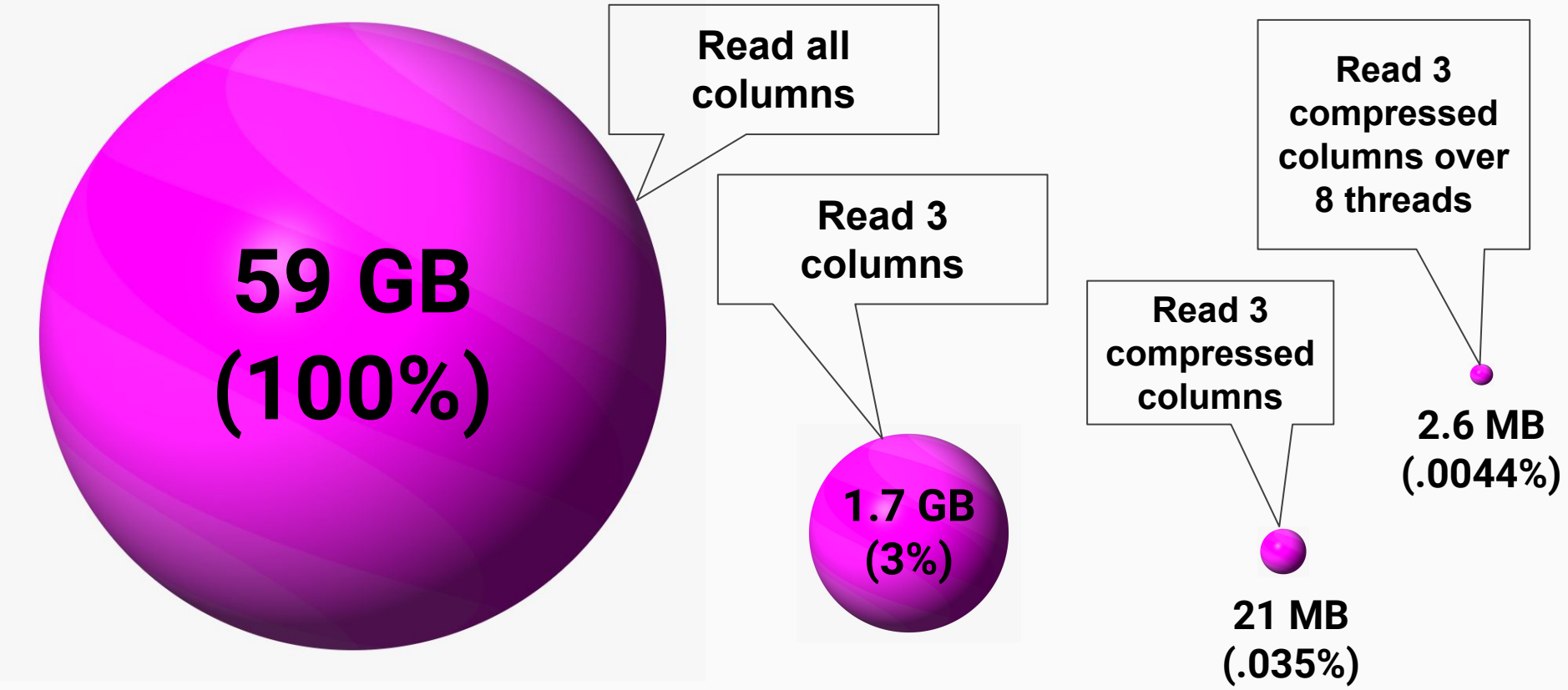


Data warehouses parallelize like crazy





Effect of reduced I/O and parallelization



Real-life queries can be extremely fast

```
SELECT Carrier, toYear(FlightDate) AS Year,
    (sum(Cancelled) / count(*)) * 100. AS cancelled
FROM default.ontime_ref WHERE Year = 2017
GROUP BY Carrier, Year HAVING cancelled > 1.
ORDER BY Carrier ASC

┌─Carrier─┬─Year─┬─────────────cancelled─┐
│ AA      │ 2017 │ 1.3541615533252709 │
│ B6      │ 2017 │   2.775117694723 66 │
│ EV      │ 2017 │ 2.935138908114 1894 │
. . .
8 rows in set. Elapsed: 0.221 sec. Processed 10.67 million
rows, 53.36 MB (48.26 million rows/s., 241.32 MB/s.)
```

There’s no free lunch, of course

Weaknesses	Strengths
(-) Lots of “small” lookups	(+) Very long tables
(-) Lots of updates	(+) Very wide tables
(-) High concurrency	(+) Open ended questions
(-) Consistency critical	(+) Lots of aggregates

Installing and connecting to ClickHouse

Introduction to ClickHouse

Understands SQL

Runs on bare metal to cloud

Shared nothing architecture

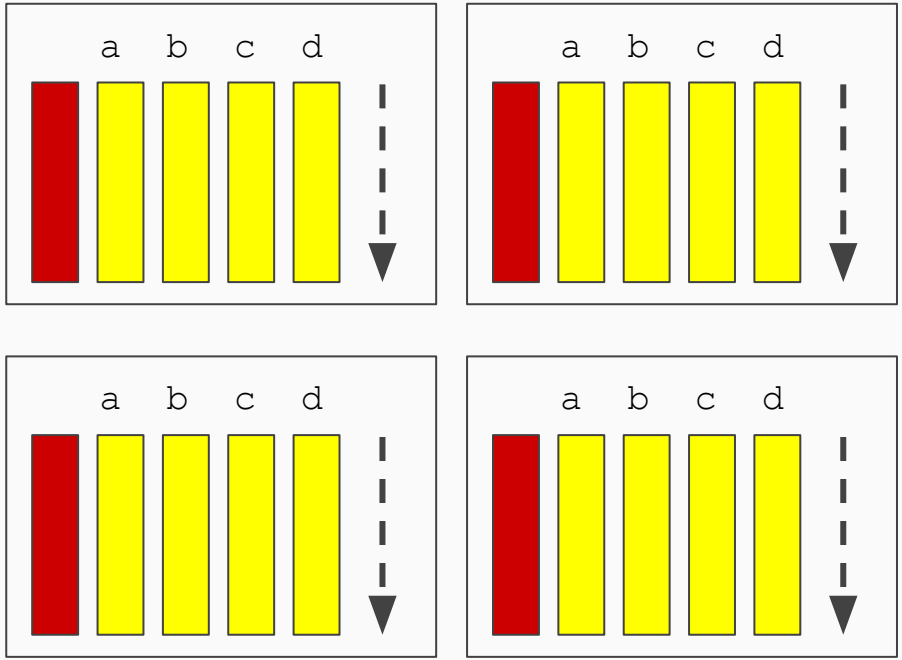
Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)

And it's <u>really</u> fast!



Installing ClickHouse on Linux

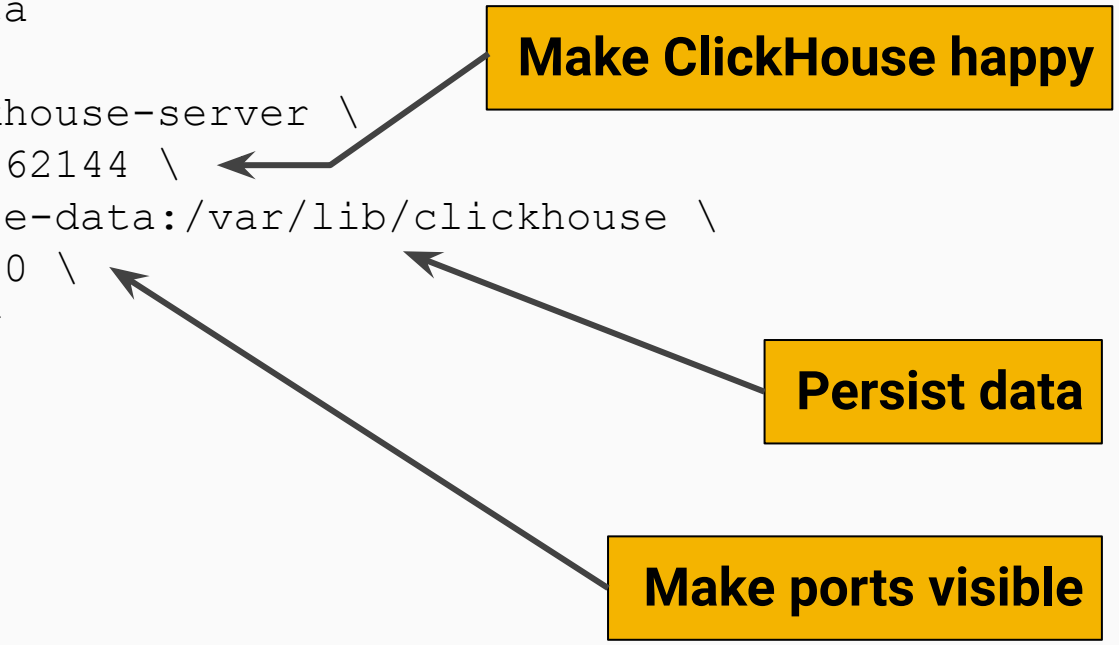
Debian Packages | **RPMs** | **Tarballs**

```
# UBUNTU/DEBIAN INSTALL
sudo apt-get install apt-transport-https ca-certificates dirmngr
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 \
    --recv E0C56BD4
echo "deb https://repo.clickhouse.tech/deb/stable/ main/" | sudo tee \
    /etc/apt/sources.list.d/clickhouse.list
sudo apt-get update

sudo apt install -y clickhouse-server clickhouse-client
sudo systemctl start clickhouse-server
```

Visit https://clickhouse.tech/docs/en/getting-started/install/

Installing ClickHouse Docker images

```
mkdir $HOME/clickhouse-data

docker run -d --name clickhouse-server \
  --ulimit nofile=262144:262144 \
  --volume=$HOME/clickhouse-data:/var/lib/clickhouse \
  -p 8123:8123 -p 9000:9000 \
  yandex/clickhouse-server
```

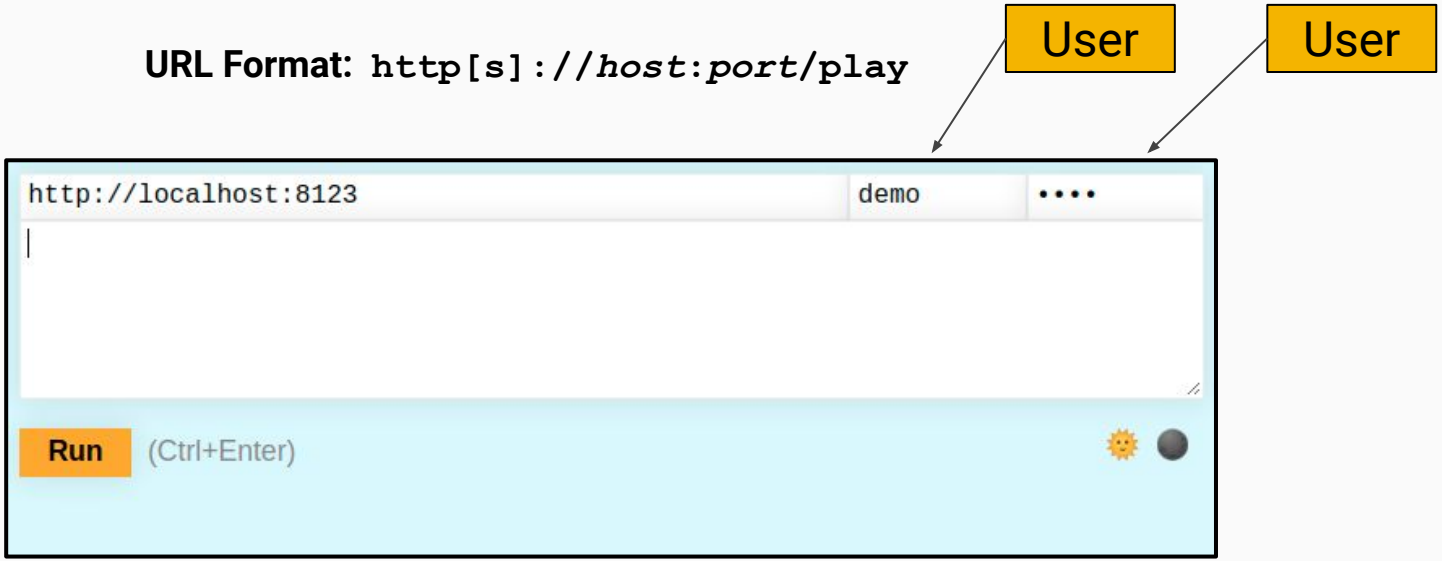


Connecting from command line

```
-- ClickHouse server on port 9440 with TLS encryption.
clickhouse-client \
  --host=github.demo.trial.altinity.cloud \
  --port=9440 --secure \
  --user=demo --password=demo

-- Local ClickHouse server with default user and no encryption.
clickhouse-client \
  --host=localhost \
  --port=9000 --user=default \

-- Same as above.
clickhouse-client
```

Connecting to built-in web UI

URL Format: `http[s]://host:port/play`



Example: https://github.demo.trial.altinity.cloud:8443/play

Example: http://localhost:8123/play

Is there a cloud service for ClickHouse?

Yes!

Several of them, in fact.

Altinity.Cloud offers managed ClickHouse in Amazon

https://altinity.com/cloud-database

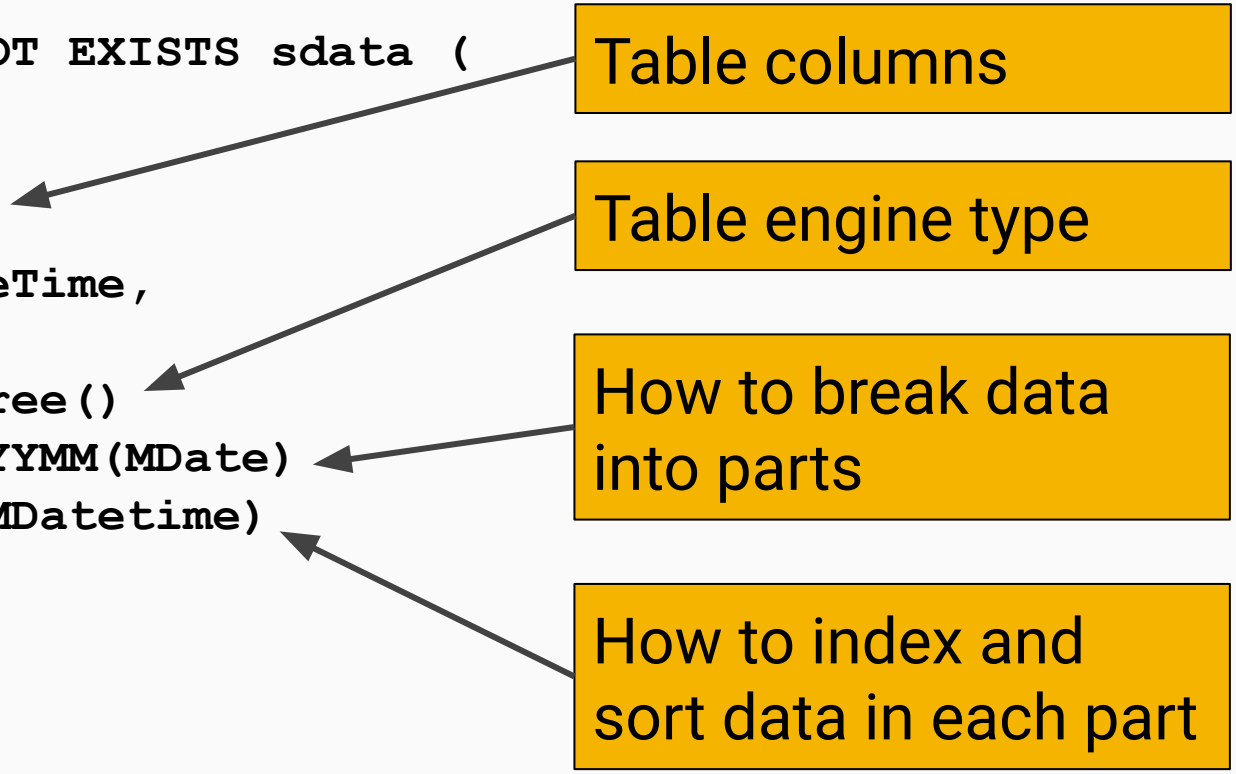
Creating Data Warehouse Tables

Introducing the CREATE TABLE command

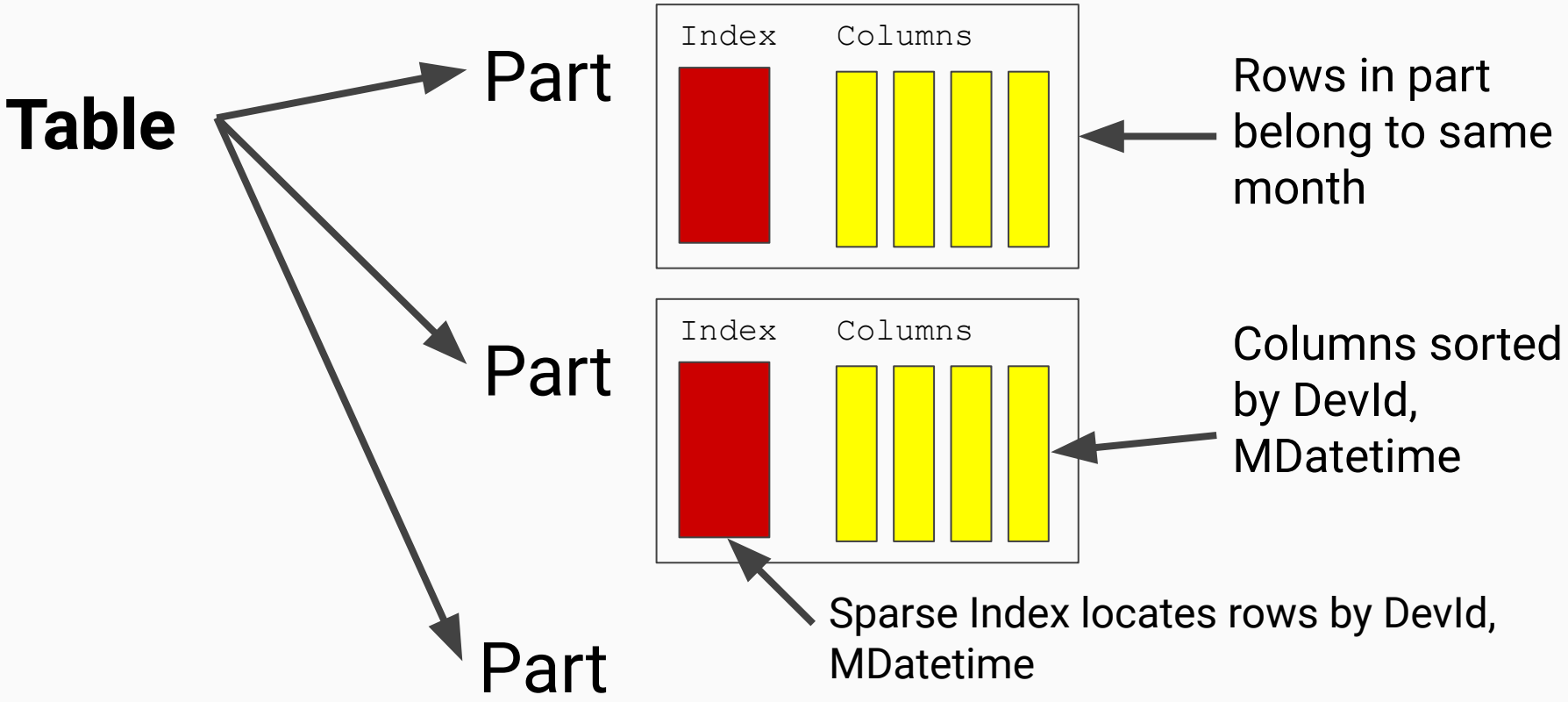
```
CREATE TABLE [IF NOT EXISTS] table_name (
    column_name1 type,
    column_name2 type,
    column_name3 type,
) ENGINE = MergeTree()
PARTITION BY partition_key
ORDER BY (sort_columns)

DROP TABLE [IF EXISTS] table_name
```

Example of table definition

```
CREATE TABLE IF NOT EXISTS sdata (
    DevId Int32,
    Type String,
    MDate Date,
    MDatetime DateTime,
    Value Float64
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(MDate)
ORDER BY (DevId, MDatetime)
```



MergeTree data layout in storage

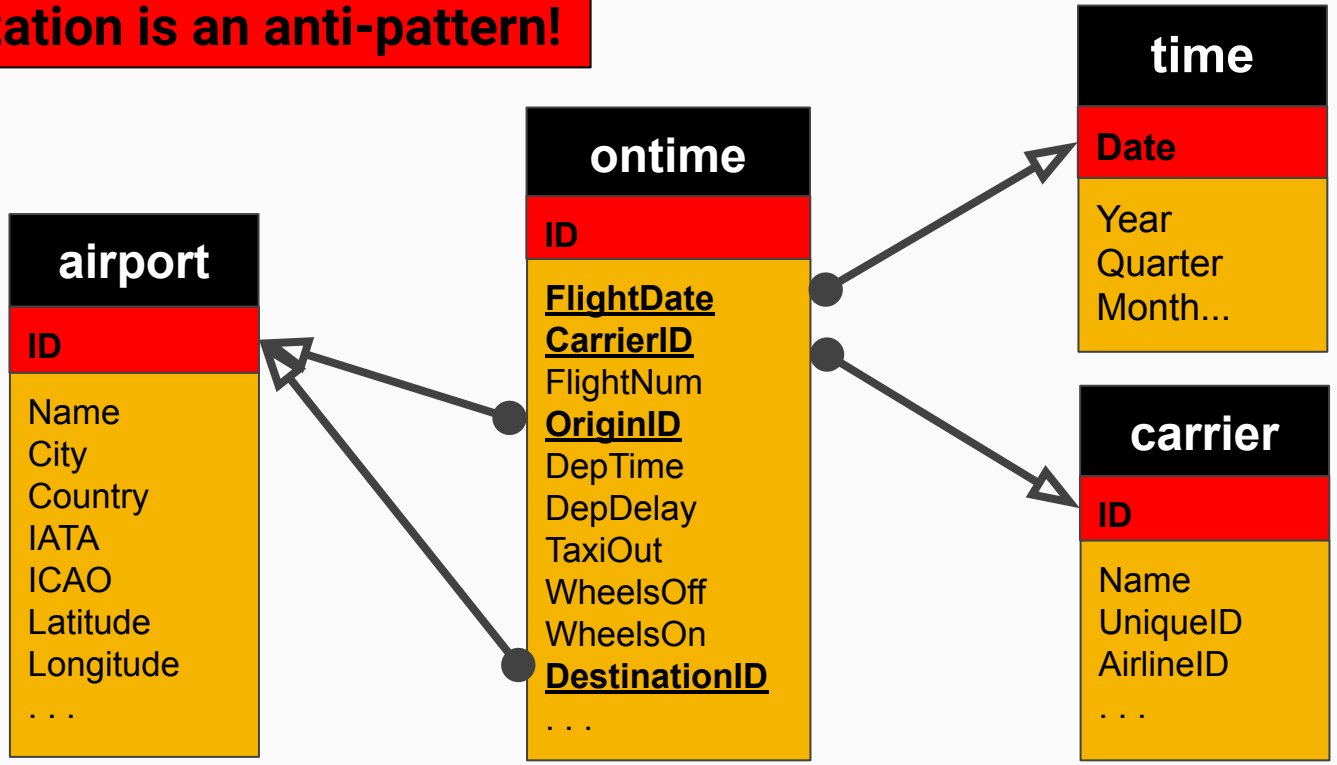


Finding tables and seeing what's in them

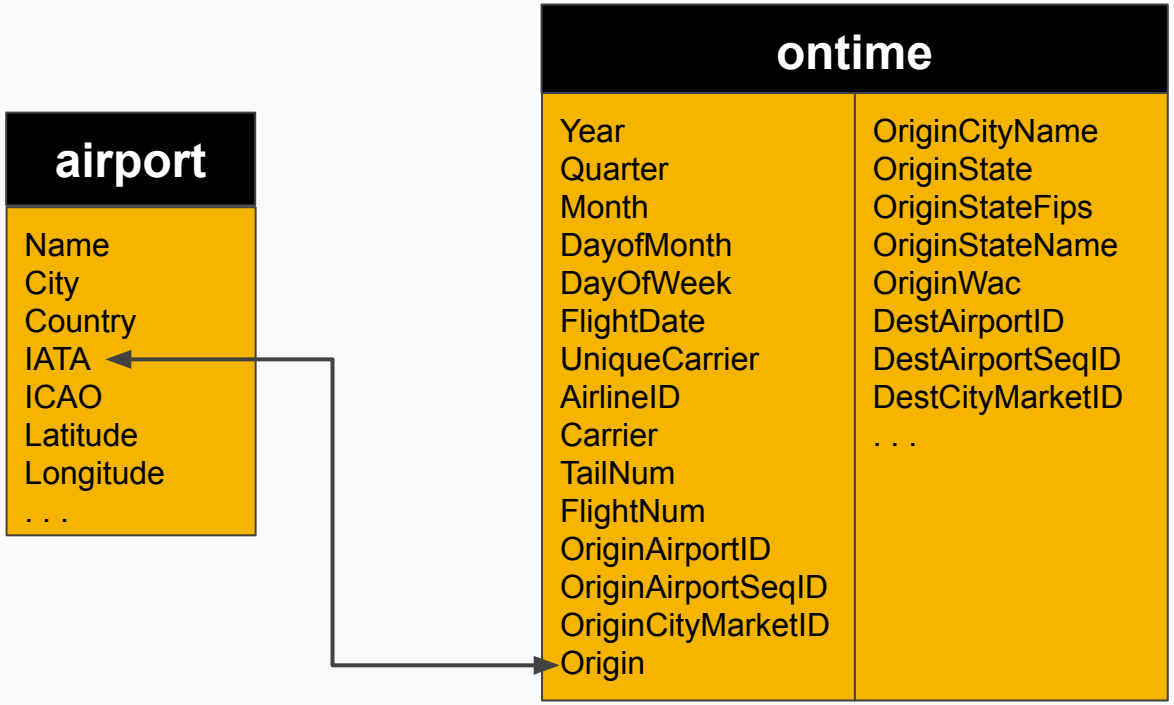
SQL Command	What it does
`show databases`	List databases
`show tables [from` *name*`]`	Show tables [in a particular database]
`describe table` *name*	Show the structure of named table

Forget about 3rd normal form

Full normalization is an anti-pattern!



Denormalization is fast and cheap



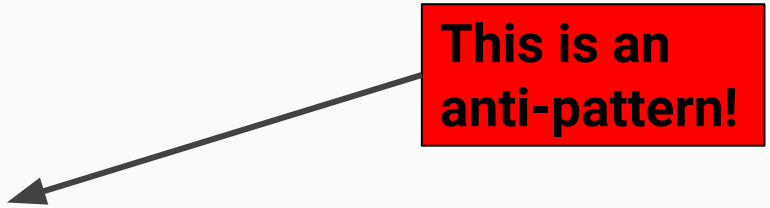
Lab Exercise #1

- Connect to ClickHouse with your favorite query tool
  - clickhouse-client
  - ClickHouse play interface
  - Any other tool you have available, such as
- Find out how many databases there are
- List the tables in the default database
- Describe the columns of the ontime database

Inserting Data

Introducing the INSERT command

```
INSERT INTO table_name
VALUES | FORMAT format_name]
<data>...


-- Example
INSERT INTO sdata VALUES
(15, 'TEMP', '2018-01-01', '2018-01-01 23:29:55', 18.0),
(15, 'TEMP', '2018-01-01', '2018-01-01 23:30:56', 18.7)
```



Use clickhouse-client to load data in bulk

Input file sdata.csv

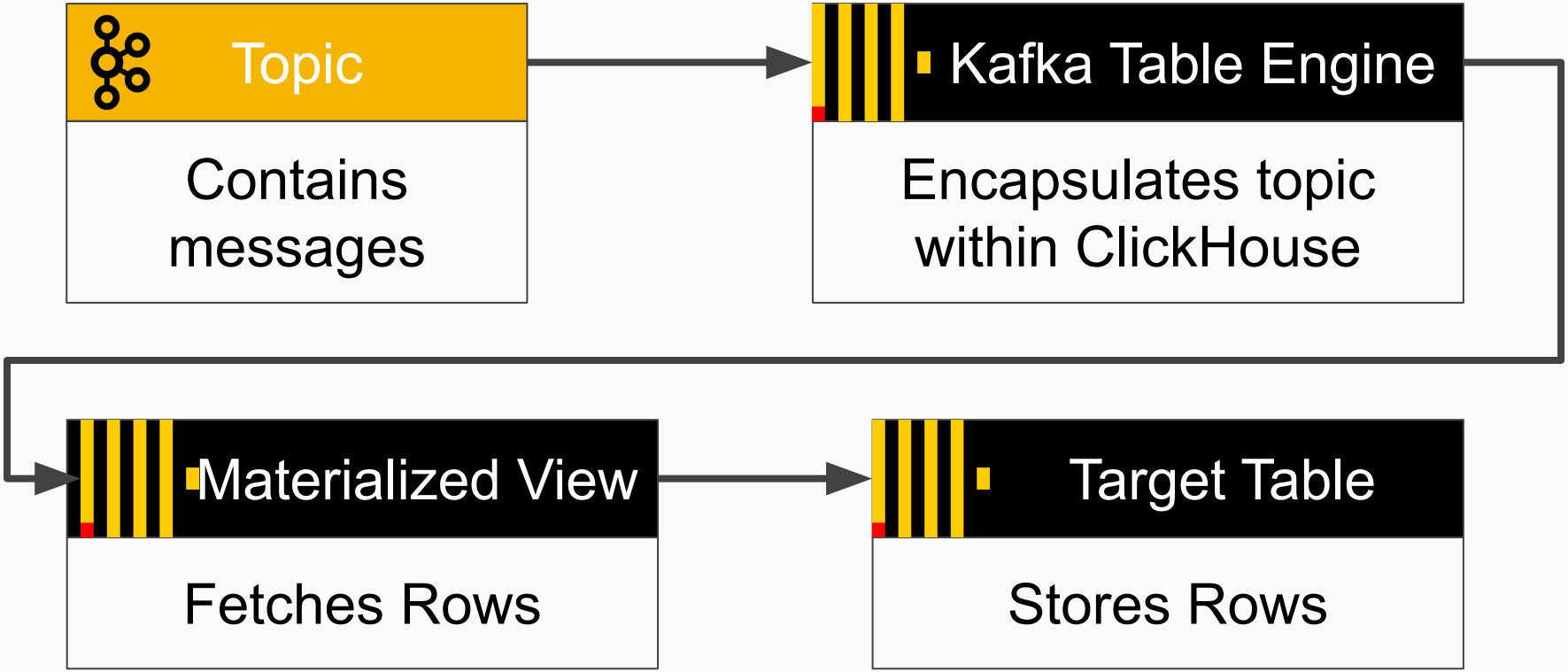
```
DevId,Type,MDate,MDatetime,Value
59,"TEMP","2018-02-01","2018-02-01 01:10:13",19.5
59,"TEMP","2018-02-01","2018-02-01 02:10:01",18.8
59,"TEMP","2018-02-01","2018-02-01 03:09:58",18.6
59,"TEMP","2018-02-01","2018-02-01 04:10:05",15.1
59,"TEMP","2018-02-01","2018-02-01 05:10:31",12.2
. . .
```

```
cat sdata.csv |clickhouse-client \
--database=sense \
--query='INSERT INTO sdata FORMAT CSVWithNames'
```

Load data from S3

```
INSERT INTO meetup.readings
SELECT * FROM
s3('https://s3.us-east-1.amazonaws.com/altinity-data-1/readings.csv',
    'CSVWithNames',
    'sensor_id Int32, time DateTime, date Date, temperature
Decimal(5,2)')
```

For real-time response, load from Kafka

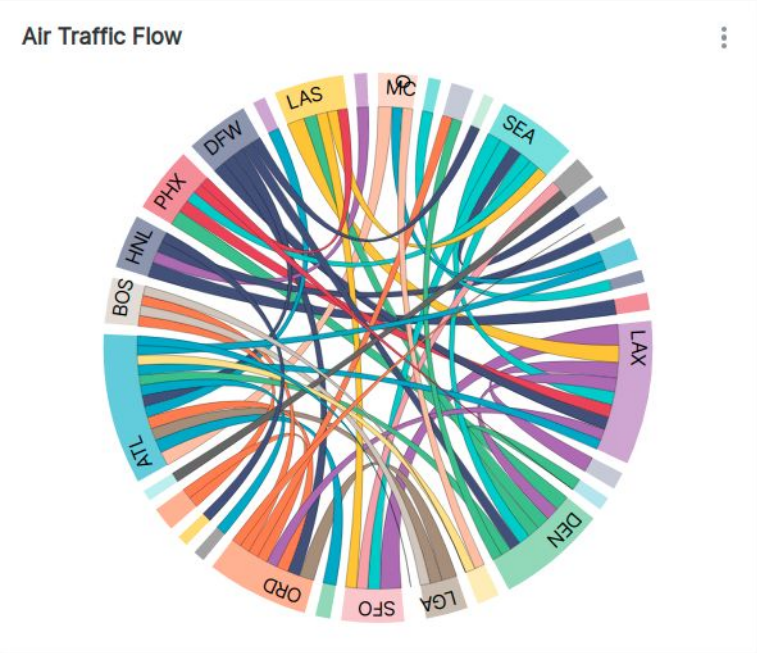


Building Reports with SELECT

Introducing the SELECT command

```
SELECT
  column_names
FROM table_name
WHERE filter_conditions
GROUP BY grouping_columns
HAVING measurement_conditions
ORDER BY ordering_columns
LIMIT limit
```

SELECT allows us to fetch interesting data



But first, an anti-pattern...

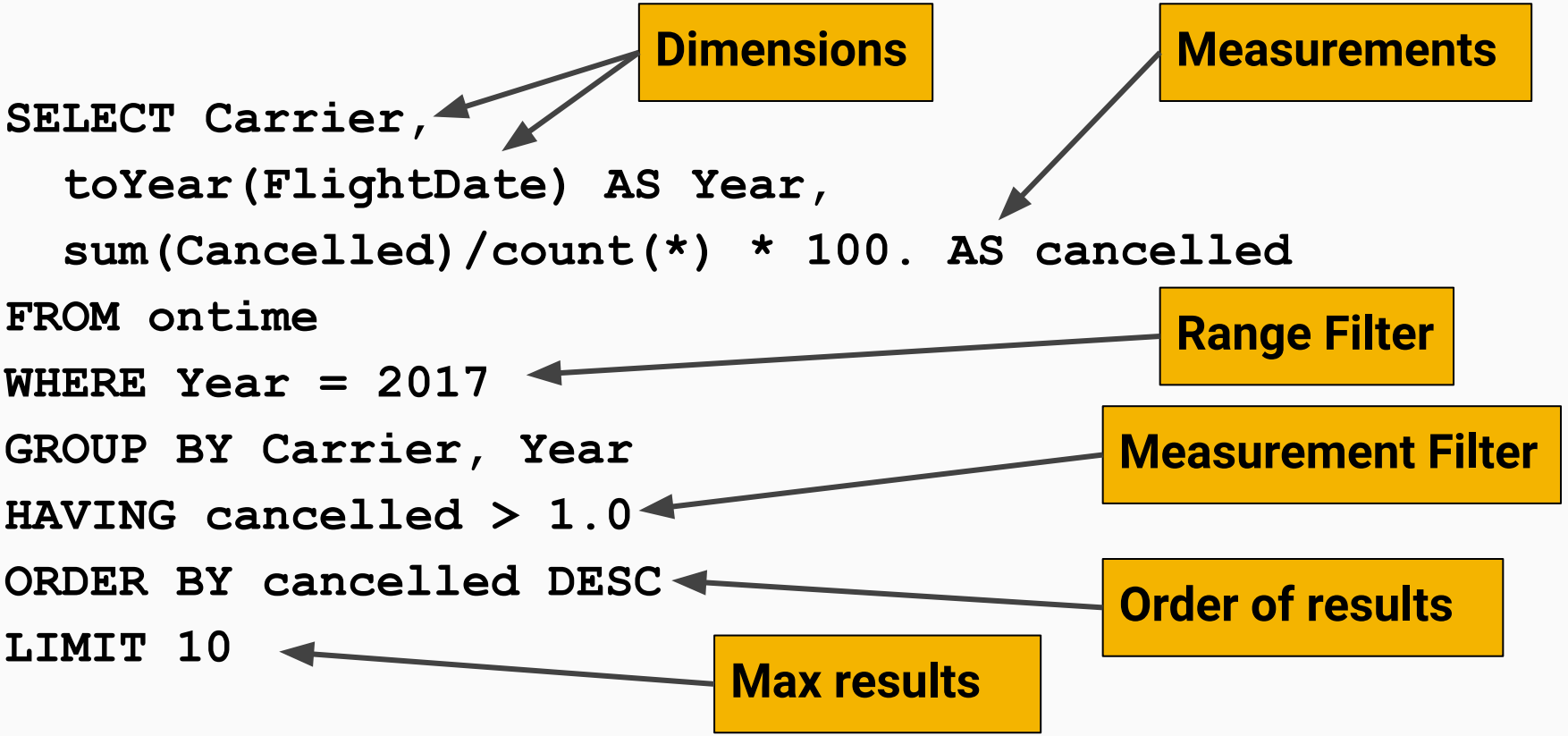
You want every column??!

```
SELECT * FROM ontime
```

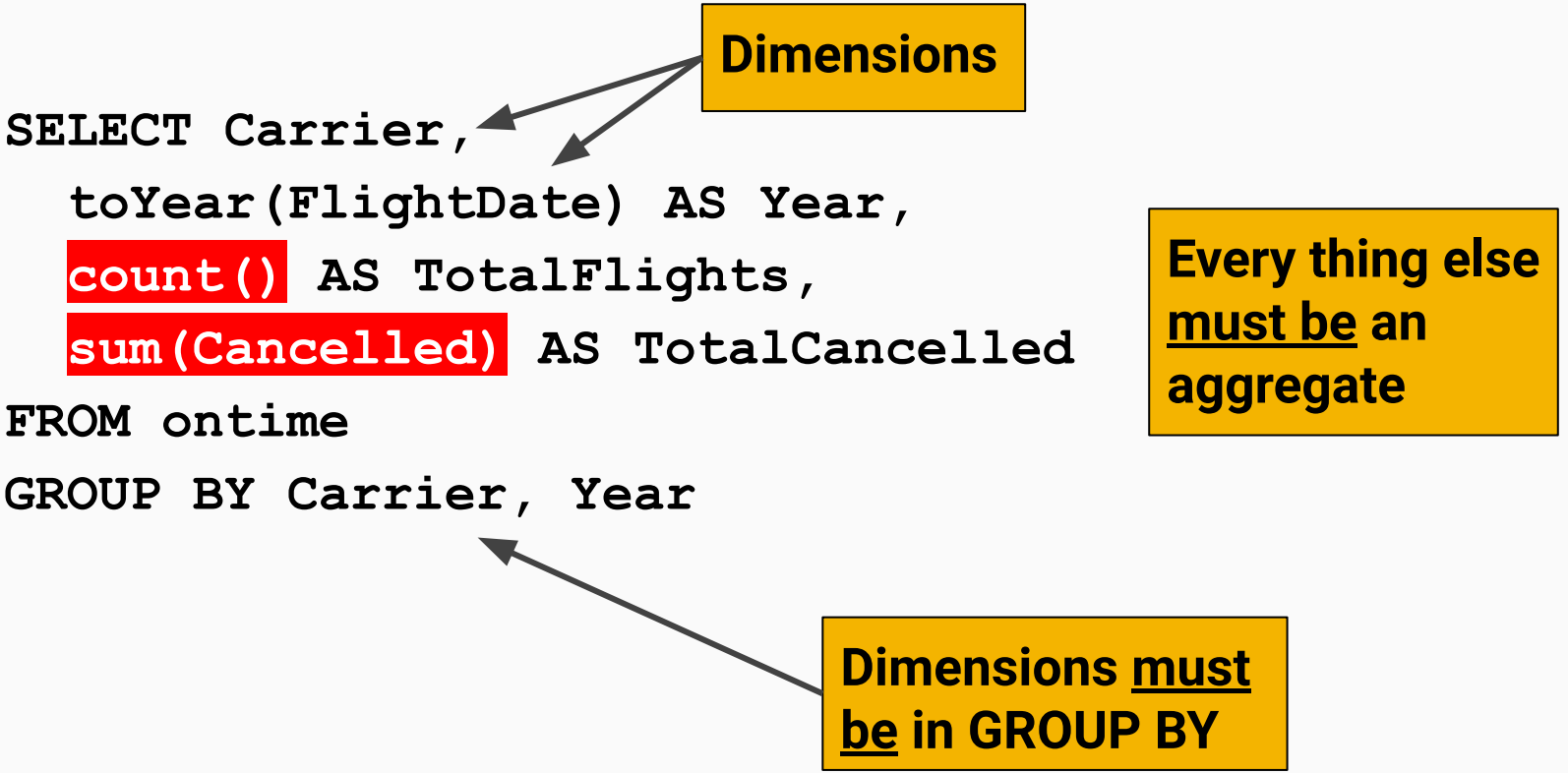
...And 196M rows?!!!

Seek insight, not data!

Who had the most cancelled flights in 2017?



Understanding GROUP BY



Standard SQL aggregate functions

Name	Meaning	Example
COUNT	Number of rows in group	COUNT()
COUNT DISTINCT	Number of distinct values in group	COUNT(DISTINCT Origin)
AVG	Mean average value in group	AVG(ArrDelayMinutes)
MIN	Minimum value in group	MIN(ArrDelayMinutes)
MAX	Maximum value in group	MAX(ArrDelayMinutes)
SUM	Sum of all value in group	SUM(ArrDelayMinutes)
any*	First value encountered in group	any(TailNum)

* Case-sensitive

Aggregates in action!

```
SELECT Carrier,
  COUNT() AS Flights,
  COUNT(DISTINCT Dest) AS Destinations,
  AVG(ArrDelayMinutes) AS AvgArrivalDelay,
  MIN(ArrDelayMinutes) AS MinArrivalDelay,
  MAX(ArrDelayMinutes) AS MaxArrivalDelay,
  SUM(ArrDelayMinutes) AS TotalArrivalDelays
FROM ontime
GROUP BY Carrier ORDER BY Carrier
```

Time is the most important dimension!

```
SELECT
  toStartOfMonth(FlightDate) AS Month,
  Carrier,
  count() AS Flights
FROM ontime
GROUP BY Month, Carrier
ORDER BY Month, Carrier
```

ClickHouse has rich date-time support

Date -- Precision to day

DateTime -- Precision to second

DateTime64 -- Precision to nanosecond

BI tools like Grafana like DateTime values

toYear(), toMonth(), toWeek(), toDayOfWeek, toDay(), toHour(), ...

toStartOfYear(), toStartOfQuarter(), toStartOfMonth(), toStartOfHour(), toStartOfMinute(), ..., toStartOfInterval()

toYYYYMM()

toYYYYMMDD()

toYYYYMMDDhhmmsss()

And many more!

WHERE clauses filter data

```
SELECT
  toStartOfMonth(FlightDate) AS Month,
  Carrier, count() AS Flights
FROM ontime
WHERE
  FlightDate BETWEEN toDate('2015-01-01')
             AND toDate('2015-06-30')
  AND Carrier != 'AA'
  AND Dest IN ('SFO', 'ORD', 'JFK')
GROUP BY Month, Carrier ORDER BY Month, Carrier
```

Standard SQL filter expressions

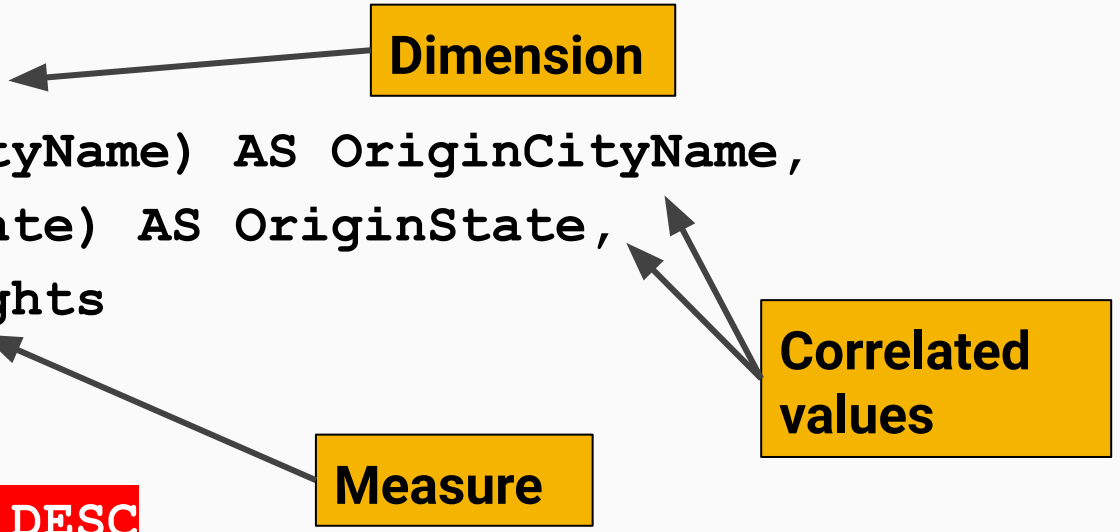
Name	Meaning	Example
= !=	Equal, not Equal	Carrier != 'AA'
> >= < <=	Greater than [or equal], less than [or equal]	ArrDelayMinutes > 10
IN	Value is in a list or subquery	Dest IN ('SFO', 'ORD', 'JFK')
BETWEEN	Value is within an inclusive interval	FlightDate BETWEEN toDate('2015-01-01') AND toDate('2015-06-30')
AND	Both conditions must be true	Carrier = ‘AA’ AND Dest = ‘SFO’
OR	At least one condition must be true	Carrier = ‘AA’ OR Dest = ‘SFO’

Filter conditions can use subqueries

```
-- How many Alaska flights served the same cities
-- as Southwest?
SELECT COUNT() AS Flights
FROM ontime o
WHERE Carrier = 'AS'
  AND toYear(FlightDate) = 2017
  AND (Origin, Dest) IN (
    SELECT Origin, Dest
    FROM ontime
    WHERE Carrier = 'WN')
```

Find “Top N”with ORDER BY and LIMIT

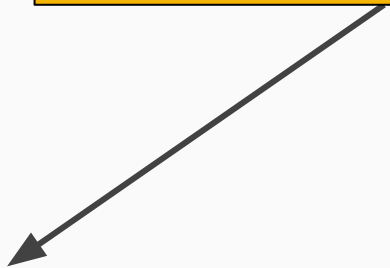
```
SELECT o.Origin,
  any(o.OriginCityName) AS OriginCityName,
  any(o.OriginState) AS OriginState,
  COUNT() as Flights
FROM ontime o
GROUP BY Origin
ORDER BY Flights DESC
LIMIT 10
```



HAVING lets you filter out measurements

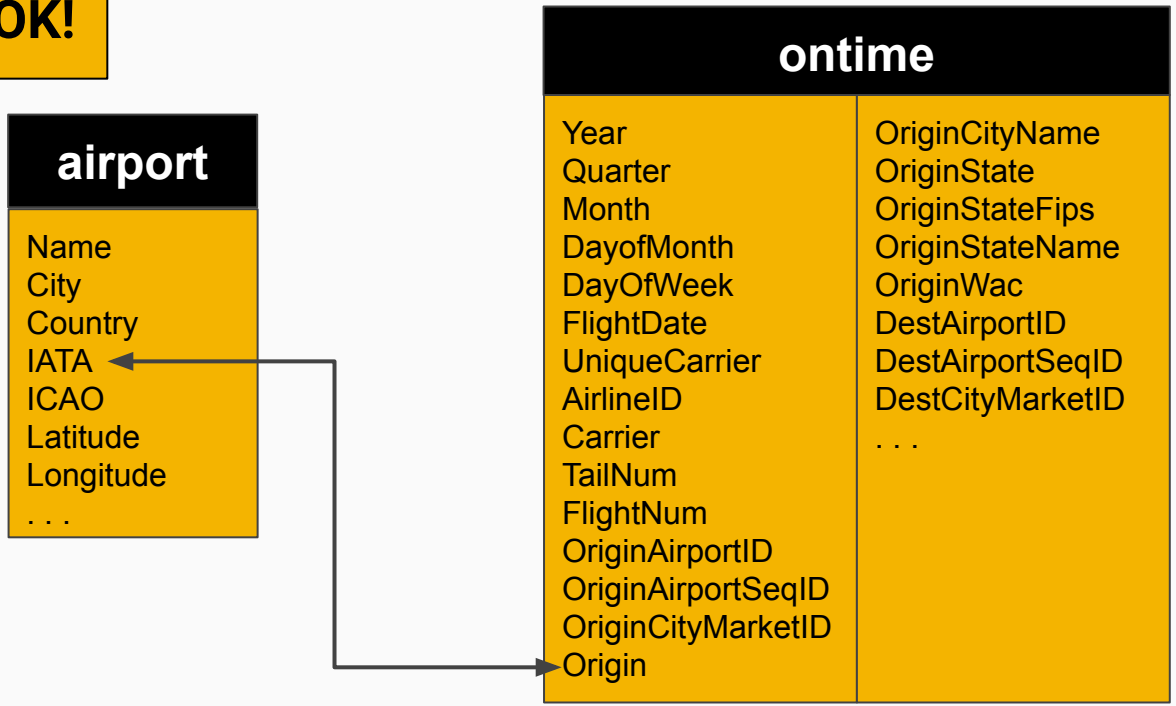
```
SELECT
   Year,
   Origin,
   COUNT() AS Flights,
   AVG(DepDelayMinutes) AS Delay
FROM ontime
GROUP BY Year, Origin HAVING Flights >= 100000
ORDER BY Year, Flights DESC
```

Ignore airports with < 100K flights per year



What if data are in multiple tables?

Hint: It's OK!



JOIN combines data between tables

```
SELECT o.Dest,
  any(a.Name) AS AirportName,
  count(Dest) AS Flights
FROM ontime o
RIGHT JOIN airports a ON a.IATA = toString(o.Dest)
GROUP BY Dest
ORDER BY Flights
DESC LIMIT 10
```

Join type

Include ontime data even if we can't find the airport name

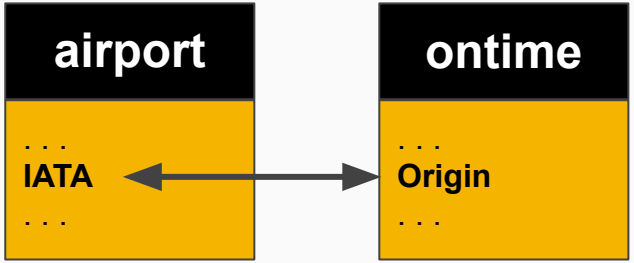
Join condition

Let's look more closely at joins

```
JOIN airports a ON a.IATA = toString(o.Dest)
```

LEFT [OUTER] JOIN

All airport rows, plus matching ontime rows



FULL [OUTER] JOIN

Matching rows, plus non-matching rows from both sides

RIGHT [OUTER] JOIN

All ontime rows, plus matching airport rows

[INNER] JOIN

Only rows that match on both sides

CROSS JOIN

Matches all rows with each other. Aka "cartesian join"

Examples with data

Table Left	
id	value
1	Left
2	Left

Table Right	
id	value
2	Right
3	Right

[INNER] JOIN

```
Id value id value
 2 Left   2 Right
```

```
SELECT l.id,  l.value,  r.id,  r.value
FROM left AS l INNER JOIN right AS r ON l.id = r.id
```

```
Id value id value
 1 Left   0 -
 2 Left   2 Right
```

LEFT [OUTER] JOIN

```
Id value id value
 1 Left   0 -
 2 Left   2 Right
 0 -      3 Right
```

FULL [OUTER] JOIN

```
Id value id value
 2 Left   2 Right
 0 -      3 Right
```

RIGHT [OUTER] JOIN

Results in real life

```
SELECT o.Dest, any(a.Name) AS AirportName, count(Dest) AS Flights
FROM ontime o
RIGHT JOIN airports a ON a.IATA = toString(o.Dest)
GROUP BY Dest ORDER BY Flights DESC LIMIT 10
```

```
┌─Dest─┬─AirportName─────────────────────────────────────┬─Flights─┐
│ ATL  │ Hartsfield Jackson Atlanta International Airport │ 10605117 │
. . .
│ LAS  │ McCarran International Airport                  │  4361486 │
└──────┴─────────────────────────────────────────────────┴──────────┘

10 rows in set. Elapsed: 2.581 sec. Processed 196.52 million rows,
982.84 MB (76.13 million rows/s., 380.75 MB/s.)
```

There's another way to do that: Subqueries

```
SELECT o.Dest, any(a.Name) AS AirportName, SUM(Flights) AS Flights
FROM (
  SELECT Dest, COUNT() AS Flights
  FROM ontime
  GROUP BY Dest
) AS o
RIGHT JOIN airports a ON a.IATA = toString(o.Dest)
GROUP BY Dest ORDER BY Flights DESC LIMIT 10
```

Subquery executes first

Performance is ~2.5x better!

You can have multiple joins per query

```
SELECT oa.State, da.State, count()
FROM default.ontime_ref AS f
LEFT JOIN default.dot_airports AS oa ON f.OriginAirportID = oa.AirportID
LEFT JOIN default.dot_airports AS da ON f.DestAirportID = da.AirportID
WHERE f.Year = 2000
GROUP BY oa.State, da.State ORDER BY count() DESC LIMIT 20
```

```
┌─oa.State─┬─da.State─┬─count()─┐
│ TX       │ TX       │  259022 │
│ CA       │ CA       │  230317 │
│ AZ       │ CA       │   64306 │
. . .
```

Table dot_airports will be scanned twice...

Lab Exercise #2

Connect to ClickHouse again and try the following exercises.

1. Create a report showing the number of flights each year.
2. What were the busiest airports in 2017 measured by number of departing flights?
    a. Can you also print the airport name?
3. Which airport had the highest departure delay in 2017?
    a. Same question as above but restrict to airports with more than 100K flights.
4. Which airline carrier has the record for most flights in a single day?
5. Can you show the airport name, number of flights, latitude, and longitude of every airport in 2020 with flights from San Francisco International Airport (SFO) or Portland (PDX)?

What’s next?

More things to learn about ClickHouse

- Optimizing table partitions and sort order
- Reducing column size with compression and codecs
- Pre-aggregating data with materialized views
- Scaling clusters using replication and sharding
- Ingesting data from Kafka
- Visualizing results using BI tools (Grafana, Superset, Tableau)

More information and references

- Community docs on ClickHouse.tech
  - Everything Clickhouse
- ClickHouse Youtube Channel
  - Piles of community videos
- Altinity Blog
  - Lots of articles about ClickHouse usage
- Altinity Webinars
  - Webinars on all aspects of ClickHouse
- ClickHouse source code on Github
  - Check out tests for examples of detailed usage

Thank you!

We're hiring

ClickHouse:
https://github.com/ClickHouse/ClickHouse

Altinity Website:
https://www.altinity.com

Altinity.Cloud
https://altinity.com/cloud