Data Council Austin / March 28, 2024

# Why streaming SQL?

*(and the semantics of applying SQL to unbounded data)*

Micah Wylde
co-founder, Arroyo
@mwylde

arroyo

# Agenda

- A Brief History of Query Languages
- But first, some math
- Dataflow semantics
- Update semantics
- So why SQL?
- Questions

| | |
| --- | --- |
| **2012** | Built my first real-time data product in Storm |
| **2014-2018** | Streaming systems to fight fraud at Sift |
| **2018-2022** | Flink team lead at Lyft and Splunk |
| **2023** | Co-founded Arroyo to bring streaming to everyone |

arroyo

# A Brief History of Query Languages

arroyo

# Databases

```
       05 FUNCN                PIC X(4)   VALUE 'GN  '.
       05 SEG-IO-AREA          PIC X(45).
       05 SSA                  PIC X(09) VALUE 'PATIENT'.
   01 CONSTANTS.
       05 C-GB                 PIC X(2)  VALUE 'GB'.
   01 SWITCHES.
       05 S-FLAG-BIT            PIC X(01) VALUE LOW-VALUES.
         88 S-FLAG                       VALUE HIGH-VALUES.
   LINKAGE SECTION.
   01 PCBMASK.
       05 DBD-NAME               PIC X(08).
       05 SEG-ID                 PIC X(02).
       05 STATS                  PIC X(02).
       05 PROCOPT                PIC X(04).
       05 FILLER                 PIC X(04).
       05 SEGMENT-NAME           PIC X(08).
       05 LENGTH-FDBK            PIC S9(05) COMP.
       05 NUMBER-SENSEGS         PIC S9(05) COMP.
       05 KEY-FDBK-AREA          PIC X(21).
  ******************************************************************
  *                 PROCEDURE DIVISION                             *
  ******************************************************************
   PROCEDURE DIVISION.
       ENTRY 'DLITCBL' USING PCBMASK
       PERFORM MAIN-PARA
       PERFORM FINAL-PARA.
   MAIN-PARA.
       CALL 'CBLTDLI' USING FUNCN,
                 PCBMASK,
                 SEG-IO-AREA.
       DISPLAY 'SEGMENT NAME '  SEGMENT-NAME
       DISPLAY 'SEGMENT AREA '  SEG-IO-AREA
       PERFORM UNTIL S-FLAG
       MOVE SPACES TO SEG-IO-AREA
       CALL 'CBLTDLI' USING FUNCN,
                         PCBMASK,
                         SEG-IO-AREA
       IF STATS NOT = C-GB
           DISPLAY 'SEGMENT NAME '  SEGMENT-NAME
           DISPLAY 'SEGMENT AREA '  SEG-IO-AREA
       END-IF
       IF STATS = C-GB
           DISPLAY 'NON BLANK FROM GN CALL ' STATS
           MOVE HIGH-VALUES TO S-FLAG-BIT
       END-IF
       END-PERFORM.
   FINAL-PARA.
        DISPLAY  'END OF PROGRAM'
        GOBACK.
   A100-EXIT.
        EXIT.
```

IMS
ISAM
COBOL

**1960s**

arroyo

# Databases

Relational model (Codd, 1970)
SQL (mid-70s)




## A Relational Model of Data for Large Shared Data Banks

E. F. Codd
*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be

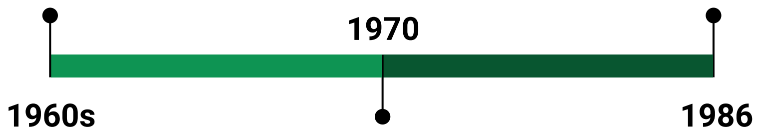
The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

# Databases

```
SELECT
FROM
WHERE
GROUP BY
HAVING
INSERT
UPDATE
DELETE
CREATE TABLE
CREATE VIEW
```

ANSI SQL




arroyo

# Databases

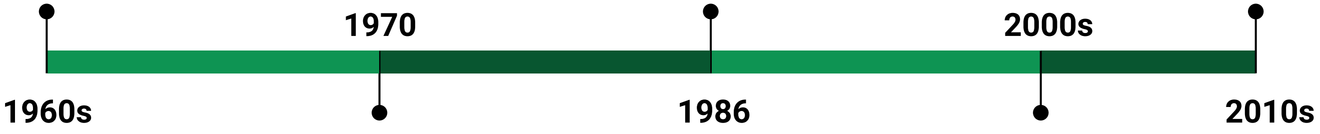
```
let $cat := doc("catalog.xml")/catalog
for $dept in distinct-values($cat/product/@dept)
return  <li>Department: {if ($dept = "ACC")
                        then "Accessories"
                        else if ($dept = "MEN")
                             then "Menswear"
                             else if ($dept = "WMN")
                                  then "Womens"
                                  else ()
               }  ({$dept})</li>
```

- Graphs: SPARQL, Cypher, Gremlin
- XML???

# Databases

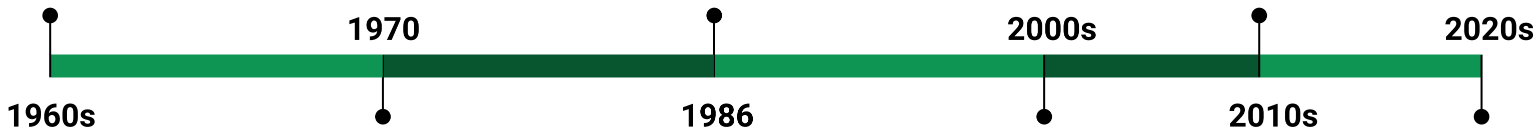
- KV stores
- DB-specific query languages (CQL, MongoDB, AQL, …)

```
db.users.aggregate(
  [
    { $project :
       {
         month_joined : { $month : "$joined" },
         name : "$_id",
         _id : 0
       }
    },
    { $sort : { month_joined : 1 } }
  ]
)
```



## Databases

# SQL



# Big Data

## MapReduce

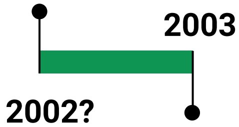
```
public static class Map extends Mapper<LongWritable, Text, Text, ArrayWritable> {
    public void map(
            LongWritable key,
            Text value,
            Context context
    ) throws IOException, InterruptedException {
        String[] lineParts = value.toString().split(":::");
        String title = lineParts[2];
        String authorsString = lineParts[1];

        String[] authors = authorsString.split("::");
        List<String> words = new ArrayList<String>();
        StringTokenizer tokenizer = new StringTokenizer(title);

        while (tokenizer.hasMoreTokens()) {
            value.set(StringUtils.stripAccents(tokenizer.nextToken()));
            words.add(value.toString().toLowerCase());
        }

        for (String author : authors) {
            String[] arrayWords = words.stream().toArray(String[]::new);
            context.write(new Text(author), new TextArrayWritable(arrayWords));
        }
    }
}

public static class Reduce extends Reducer<Text, TextArrayWritable, NullWritable, Text> {

    private final HashSet<String> stopWords = new HashSet<String>(Arrays.asList(StopWords.STOP_WORDS));

    public void reduce(
            Text key,
            Iterable<TextArrayWritable> values,
            Context context
    ) throws IOException, InterruptedException {
        JSONObject jsonObj = new JSONObject();
        JSONArray wordsJsonArray = new JSONArray();
        LinkedHashMap<Text, Integer> wordCount = new LinkedHashMap<Text, Integer>();

        for (ArrayWritable array : values) {
            for (Writable value : array.get()) {
```

**2002?**

# Big Data

```
best: table top(3)[url: string] of referer: string weight
count: int;
line: string = input;
fields: array of string =
   saw(line, ".*GET ", "[^\t ]+",
   " HTTP/1.[0-9]\"", "[0-9]+",
   "[0-9]+", "\"[^\t ]+\"");
emit best[fields[1]] ← fields[5] weight 1;
```

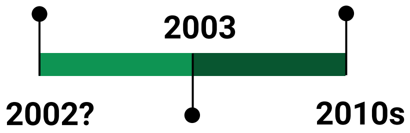
- Sawzall
- Pig Latin



# Big Data

```
rdd
    .flatMap(_.split("\\s"))
    .map(_.replaceAll(
      "[,.!?:;]", "")
    .trim
    .toLowerCase)
    .filter(!_.isEmpty)
    .map((_, 1))
    .reduceByKey(_ + _)
    .sortByKey()
```

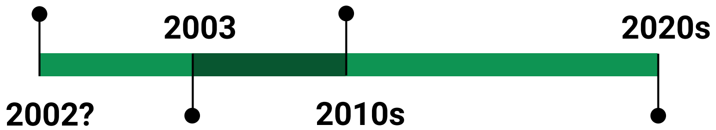
Dataframe APIs
(Flume, Spark)




arroyo

Big Data

# SQL




arroyo

# Streaming

$$
\begin{array}{l}
(\ ?A\ :\ \mathrm{Accnt}\ )\\
\quad \mathrm{Balance}(A\ \mathbf{is}\ ?A) \longrightarrow\\
\quad ((\ ?A\ :\ \mathrm{Accnt}\ )\ \mathrm{Deposit}(A\ \mathbf{is}\ ?A) \longrightarrow [\,1..12\ \mathbf{rel}\ \sim]\ (\mathrm{Deposit}(A\ \mathbf{is}\ ?A))) \longrightarrow\\
\quad \mathrm{Balance}(A\ \mathbf{is}\ ?A)
\end{array}
$$

Complex Event Processing
RAPIDE
Snoop

**1990s**

# Streaming

## Graph-construction APIs (Storm)

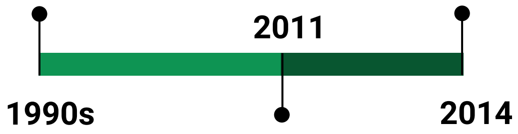
```java
public static class SplitSentence extends BaseBasicBolt {
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
      declarer.declare(new Fields("word"));
    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
      return null;
    }

    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector)
{
      String sentence = tuple.getStringByField("sentence");
      String words[] = sentence.split(" ");
      for (String w : words) {
        basicOutputCollector.emit(new Values(w));
      }
    }
}


public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
      String word = tuple.getString(0);
      Integer count = counts.get(word);
      if (count == null)
        count = 0;
      count++;
      counts.put(word, count);
      collector.emit(new Values(word, count));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
      declarer.declare(new Fields("word", "count"));
    }
}
```


arroyo

# Streaming

```
val counts = text.flatMap {
  _.toLowerCase.split("\\W+") filter { _.nonEmpty } }
  .map { (_, 1) }
  .groupBy(0)
  .sum(1)

counts.writeAsCsv(outputPath, "\n", " ")
```

## Datastream APIs (Flink, Beam)



# Streaming

# SQL


Always has been
Wait it’s all SQL?
SELECT *
FROM PLANET

# But first,
# some *Math*

# Semantics

*Describes the precise behavior of a particular program or language construct*

# Relational Algebra

Formalized operations that follow certain rules, over a class of sets, called “relations” (aka “tables”)

| set operations | relational operations | aggregate functions |
| --- | --- | --- |
| $\cup$ set union | $\sigma$ selection | sum |
| $\cap$ set intersection | $\pi$ projection | count |
| $\setminus$ set difference | $\bowtie$ joins | avg |
| $\times$ cartesian product | | max |
| | | min |

arroyo

```sql
CREATE TABLE orders (
  id        INT,
  time      TIMESTAMP,
  user_id   TEXT,
  product   TEXT,
  store     INT,
  price     FLOAT
);
```

```sql
CREATE TABLE pageviews (
  id        INT,
  time      TIMESTAMP,
  user_id   TEXT,
  page      TEXT
);
```

arroyo

```
SELECT price * 1.08 AS total_price
FROM orders
WHERE store_id = 5;
```

arroyo

# Table

| time | user | store_id | price |
|---|---|---|---|
| 12:45:01 | bob | 3 | 54.29 |
| 12:45:02 | alice | 5 | 72.55 |
| 12:45:02 | sid | 10 | 532.22 |
| 12:45:03 | scott | 5 | 242.73 |
| 12:45:03 | mary | 3 | 372.82 |
| 12:45:04 | roland | 1 | 185.21 |
| 12:45:05 | sarah | 5 | 32.58 |

78.35

262.15

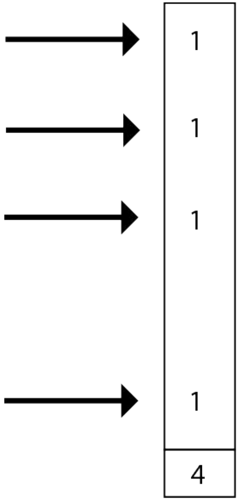
35.19

```
SELECT price * 1.08
FROM orders
WHERE store_id = 5;
```

```sql
SELECT count(*)
FROM orders
WHERE price > 100;
```

# Table

| time | user | store_id | price |
|---|---|---|---|
| 12:45:01 | bob | 124 | 54.29 |
| 12:45:02 | alice | 22 | 72.55 |
| 12:45:02 | sid | 754 | 532.22 |
| 12:45:03 | scott | 57 | 242.73 |
| 12:45:03 | mary | 523 | 372.82 |
| 12:45:04 | roland | 22 | 72.55 |
| 12:45:05 | sarah | 57 | 242.73 |




```sql
SELECT count(*)
FROM orders
WHERE price > 100;
```

# How can we apply this to streaming?

```sql
CREATE STREAM orders (
  id         INT,
  time       TIMESTAMP,
  user_id    TEXT,
  product    TEXT,
  store      INT,
  price      FLOAT
);
```

```sql
CREATE STREAM pageviews (
  id         INT,
  time       TIMESTAMP,
  user_id    TEXT,
  page       TEXT
);
```

arroyo

```sql
SELECT price * 1.08 AS total_price
FROM orders
WHERE store_id = 5;
```

arroyo


```sql
SELECT price * 1.08
FROM orders
WHERE store_id = 5
```

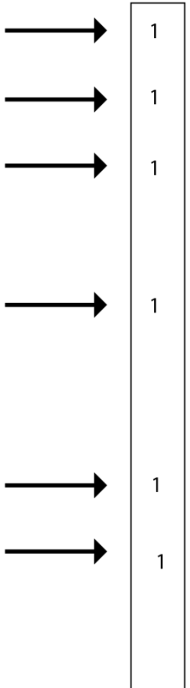
```sql
SELECT count(*)
FROM orders
WHERE price > 100;
```

arroyo

# Stream

| time | user | product | price |
|---|---|---|---|
| 12:45:01 | bob | 124 | 54.29 |
| 12:45:02 | alice | 22 | 72.55 |
| 12:45:02 | sid | 754 | 532.22 |
| 12:45:03 | scott | 57 | 242.73 |
| 12:45:03 | mary | 523 | 372.82 |
| 12:45:04 | roland | 22 | 72.55 |
| 12:45:05 | sarah | 57 | 242.73 |
| 12:45:06 | bob | 124 | 54.29 |
| 12:45:07 | alice | 22 | 72.55 |
| 12:45:08 | sid | 754 | 532.22 |
| 12:45:08 | scott | 57 | 242.73 |

⋮




```sql
SELECT count(*)
FROM orders
WHERE price > 100
```

arroyo

We need some new semantics

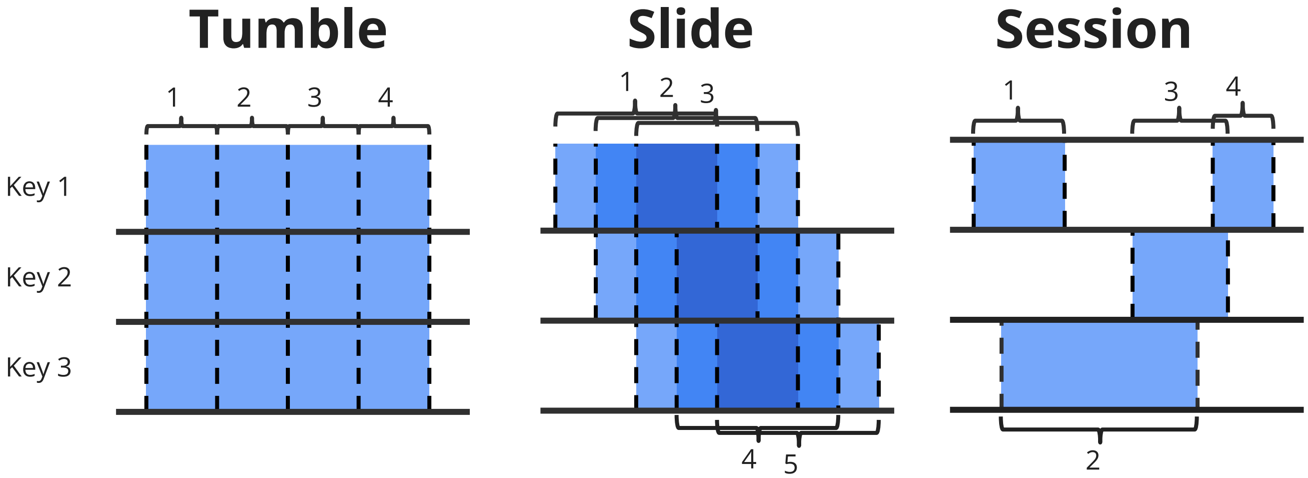
# Dataflow Semantics

# The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle

Proceedings of the VLDB Endowment, vol. 8 (2015), pp. 1792-1803

arroyo




arroyo

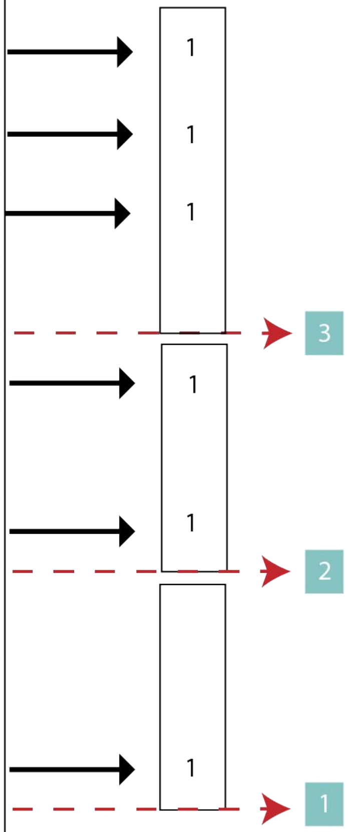
```sql
-- this query will never return
SELECT
  date_trunc('minute', time) as minute,
  count(*)
FROM orders
WHERE price > 100
GROUP BY minute;
```

arroyo

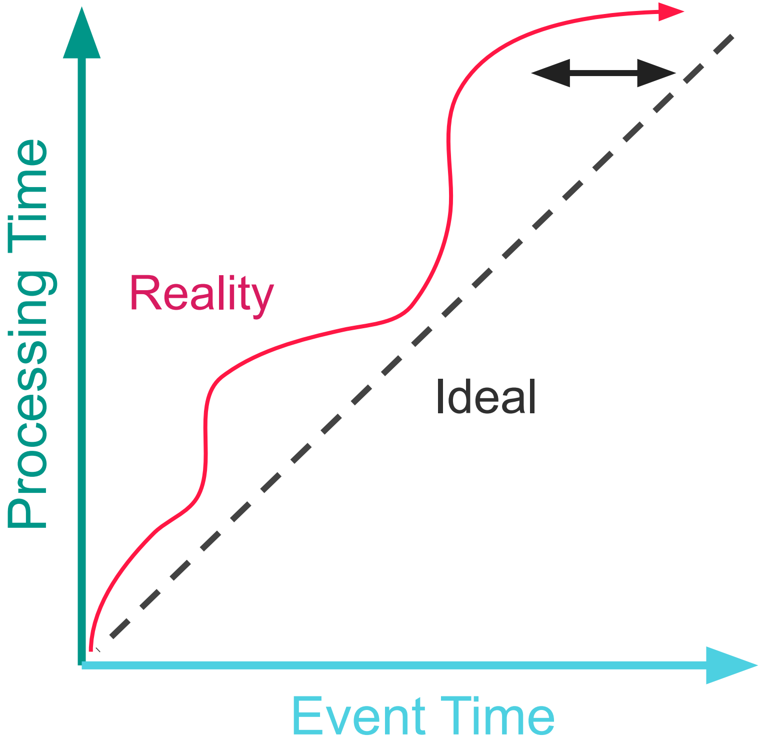
```sql
-- this will actually emit records
-- because we've placed a bound on time!
SELECT
  tumble(interval '1 minute') as minute,
  count(*)
FROM orders
WHERE price > 100
GROUP BY minute;
```

arroyo

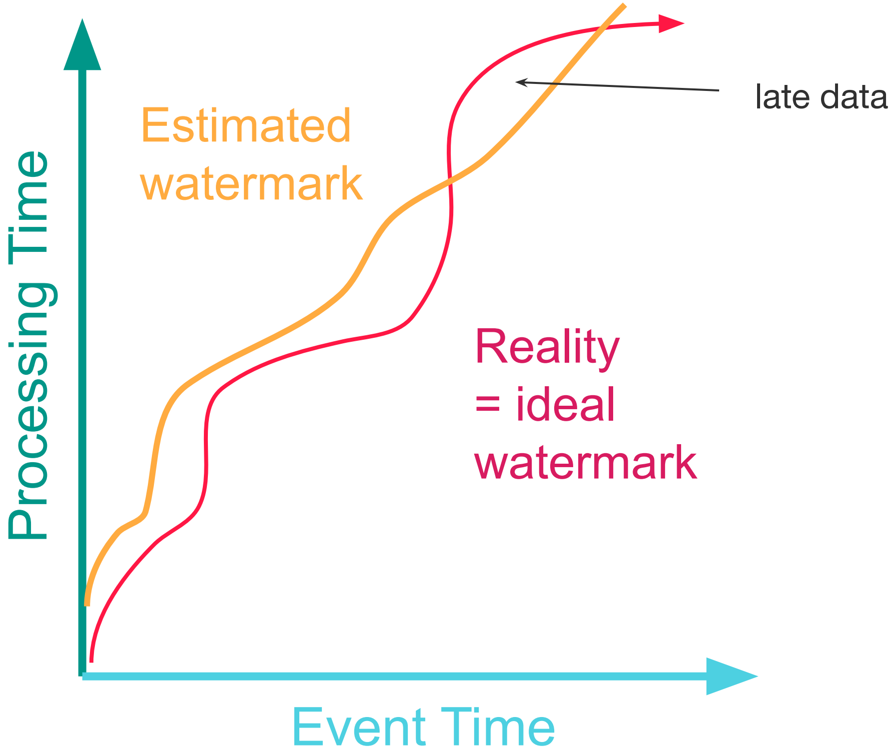
| time | user | product | price |
| --- | --- | --- | --- |
| 12:45:01 | bob | 124 | 242.73 |
| 12:45:30 | alice | 22 | 532.22 |
| 12:45:45 | sid | 754 | 128.52 |
| 12:45:49 | scott | 57 | 54.29 |
| 12:46:05 | mary | 523 | 372.82 |
| 12:46:30 | roland | 22 | 72.55 |
| 12:46:35 | sarah | 57 | 242.73 |
| 12:47:06 | bob | 124 | 54.29 |
| 12:47:07 | alice | 22 | 72.55 |
| 12:47:22 | sid | 754 | 532.22 |
| 12:48:08 | scott | 57 | 242.73 |




arroyo




arroyo




Watermark: a lower bound on event times that the system will process in the future

arroyo

```sql
CREATE VIEW pageviews_agg as (
  SELECT
    count(*) as views,
    user_id,
    tumble(interval '1 hour') as window
  FROM pageviews
  GROUP BY window, customer_id
);
```

```sql
CREATE VIEW orders_agg as (
  SELECT
    count(*) as orders,
    customer_id,
    tumble(interval '1 hour') as window
  FROM orders
  GROUP BY window, customer_id
);
```

```sql
SELECT
  O.window, O.customer_id, C.views, O.orders
FROM orders_agg as O
LEFT JOIN clicks_agg as C ON
  C.customer_id = O.customer_id AND
  C.window = O.window;
```

arroyo

# Update Semantics

```sql
SELECT
  date_trunc('hour', time) as hour,
  sum(price) as sales
FROM orders
GROUP BY hour, store_id;
```


time
store
price
12:45:01
6
54.29
create row for key (12:00, 6) with 54.29
12:00
6
54.29
12:45:03
12
372.82
create row for key (12:00, 12) with 372.82
12:00
6
54.29
12:00
12
372.82
12:45:07
6
72.55
update row for key (12:00, 12), + 72.55
12:00
6
126.84
12:00
12
372.82

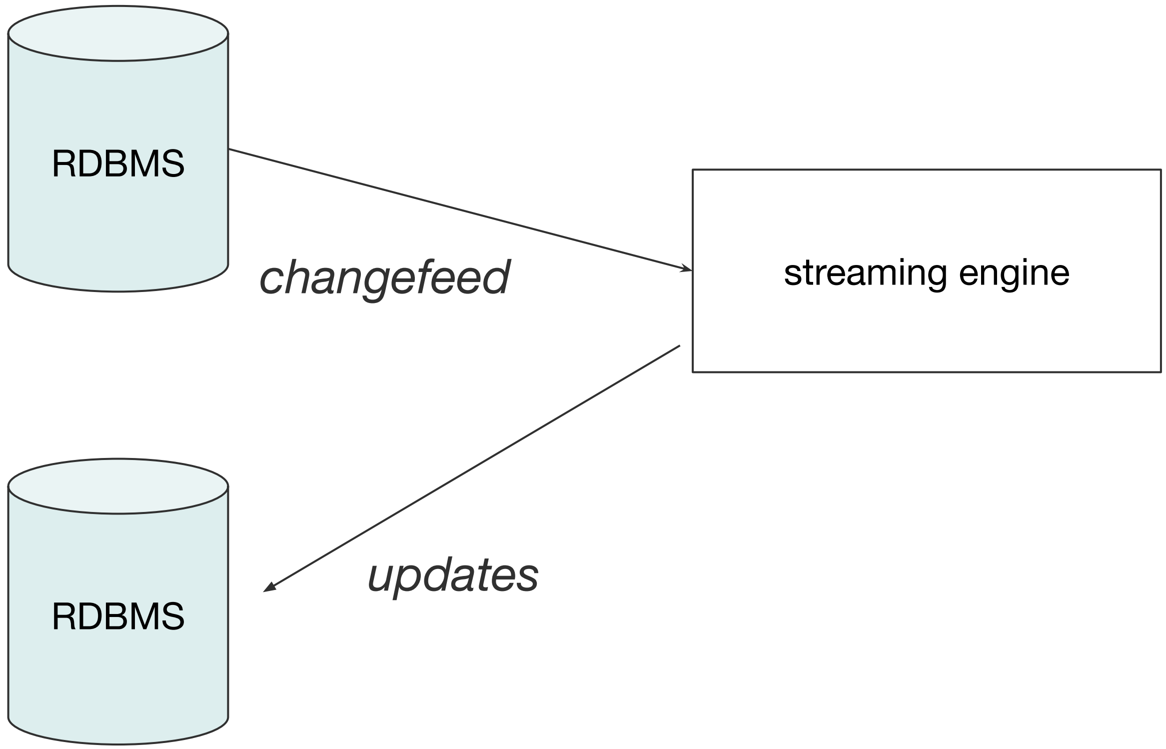
arroyo

```sql
SELECT
  date_trunc('hour', time) as hour,
  sum(price) as sales
FROM orders
GROUP BY hour, store_id
WHERE mz_now() <= timestamp + INTERVAL '1 day';
```

arroyo

```
{"before": null,
 "after":  {"hour":"2023-07-24T23:00:00","store_id":5,"sales":1000},"op":"c"}
{"before": null,
 "after":  {"hour":"2023-07-24T23:00:00","store_id":7,"sales":1372},"op":"c"}
{"before": {"hour":"2023-07-24T23:00:00","store_id":5,"sales":1000},
 "after":  {"hour":"2023-07-24T23:00:00","store_id":5,"sales":1300},"op":"u"}
...
```

arroyo




arroyo

| | Dataflow Semantics | Update Semantics |
|---|---|---|
| **Completeness** | Relies on a watermark to determine when a window is complete; data received past the watermark is dropped | Tables are incrementally updated and eventually converge to the complete result; in practice TTLs are used to constrain state sizes |
| **SQL support** | Generally requires that aggregations and joins are performed over a window | Nearly all SQL can be supported |
| **Efficiency** | Allows very efficient windowing implementations | Maintaining old data in state takes more resources, rows may need to be updated many times |
| **Usage pattern** | Generally push-driven; the streaming system pushes out results to consumers when they are ready | Generally pull-driven; consumers need to decide when they will query the results and determine for themselves whether data is complete enough |
| **Use cases** | Real-time application features, monitoring, fraud, ETL | Analytics, billing, integration with RDBMs |

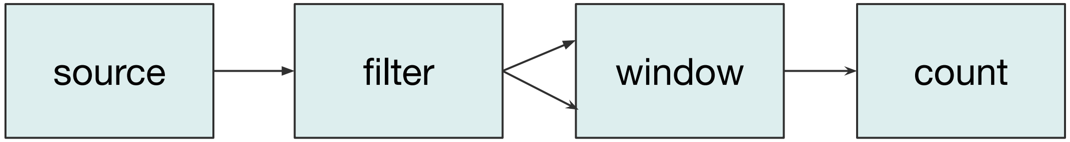
# So why streaming SQL?


arroyo

# SQL is Declarative

```sql
SELECT
  tumble(interval '1 minute'),
  store_id
  count(*)
FROM orders
WHERE price > 100
GROUP BY 1, 2;
```

```
orders
    .filter(_.price > 100)
    .key_by(_.store_id)
    .window(TumblingWindow.of(Time.minutes(1)))
    .count()
```

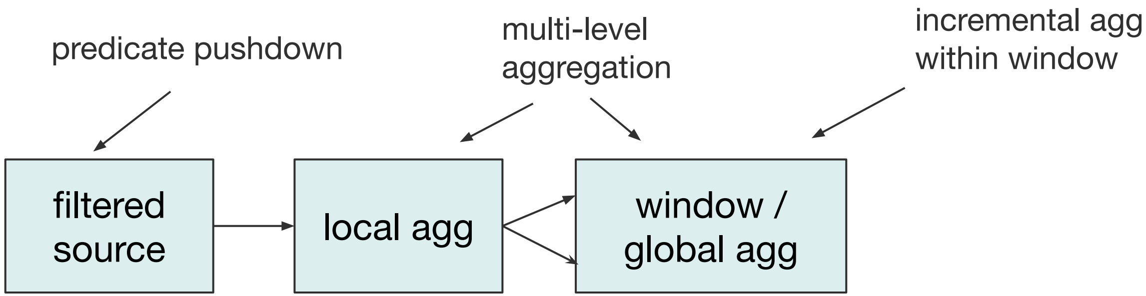



arroyo

# SQL is Declarative

That means engines can optimize

```sql
SELECT
  tumble(interval '1 minute'),
  store_id
  count(*)
FROM orders
WHERE price > 100
GROUP BY 1, 2;
```




arroyo


SQL is Flexible
IBM
360

# SQL is Flexible

Transactional databases

Analytical databases

Batch processing

Stream processing

Metric systems

Graph databases

Data lakes

…

# SQL is Extensible

materialized views

event time

user-defined functions

watermarks

new operators

aggregate functions

optimizations

windows

custom data types

distributed processing


List of relations
Type
Owner
checkpoints
cluster_info
connection_profiles
select count(*) from checkpoints;

arroyo

# questions?

micah@arroyo.dev

@mwylde

linkedin.com/u/wylde