ConFoo 2024

Transactions? Pah! Joins? Overrated!

Adventures in fast, big data

Robert Hodges - Altinity CEO

A brief message from our sponsor…

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Altinity Engineering

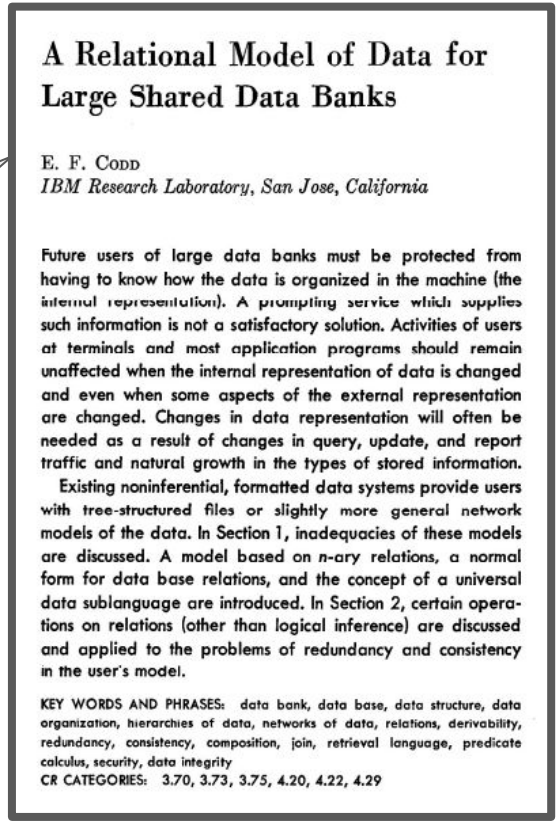
More database geeks with centuries of experience in DBMS and applications

ClickHouse support and services: Altinity.Cloud and Altinity Stable Builds
Authors of Altinity Kubernetes Operator for ClickHouse

Where the dream started…

Future users of large databanks must be protected from having to know how the data is organized in the machine (the internal representation).

Edgar F. Codd
June 1970



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 needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

What it means

```
SELECT max(temperature)
FROM test.readings_multi
WHERE (sensor_id = 2555) AND (msg_type = 'reading')
```

```
┌─max(temperature)─┐
│           138.23 │
└──────────────────┘
```

What it means

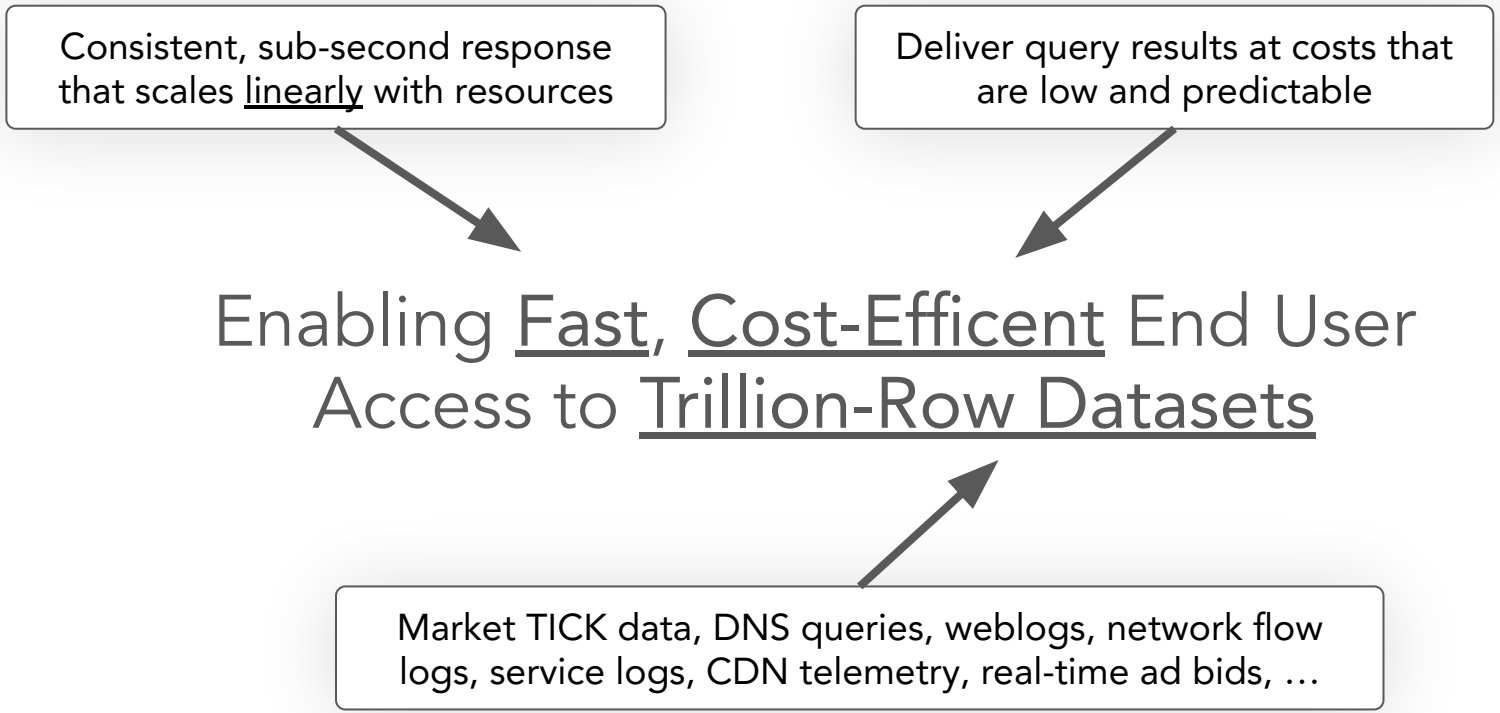
```
SELECT max(temperature)
FROM test.readings_multi
WHERE (sensor_id = 2555) AND (msg_type = 'reading')
```


```
┌─max(temperature)─┐
│           138.23 │
└──────────────────┘
```

Reality imposes a different way of thinking

Demo Time!

Modern analytic systems are a new game with new goals



Size matters

So does speed

Introducing ClickHouse, a real-time data warehouse

Understands SQL

Runs on bare metal to cloud

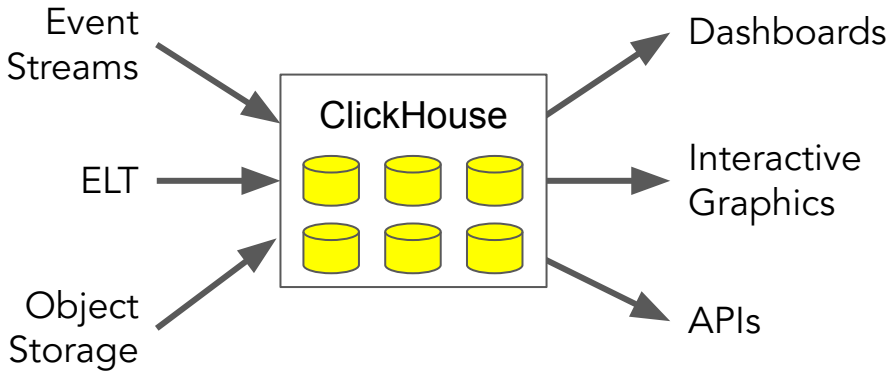
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

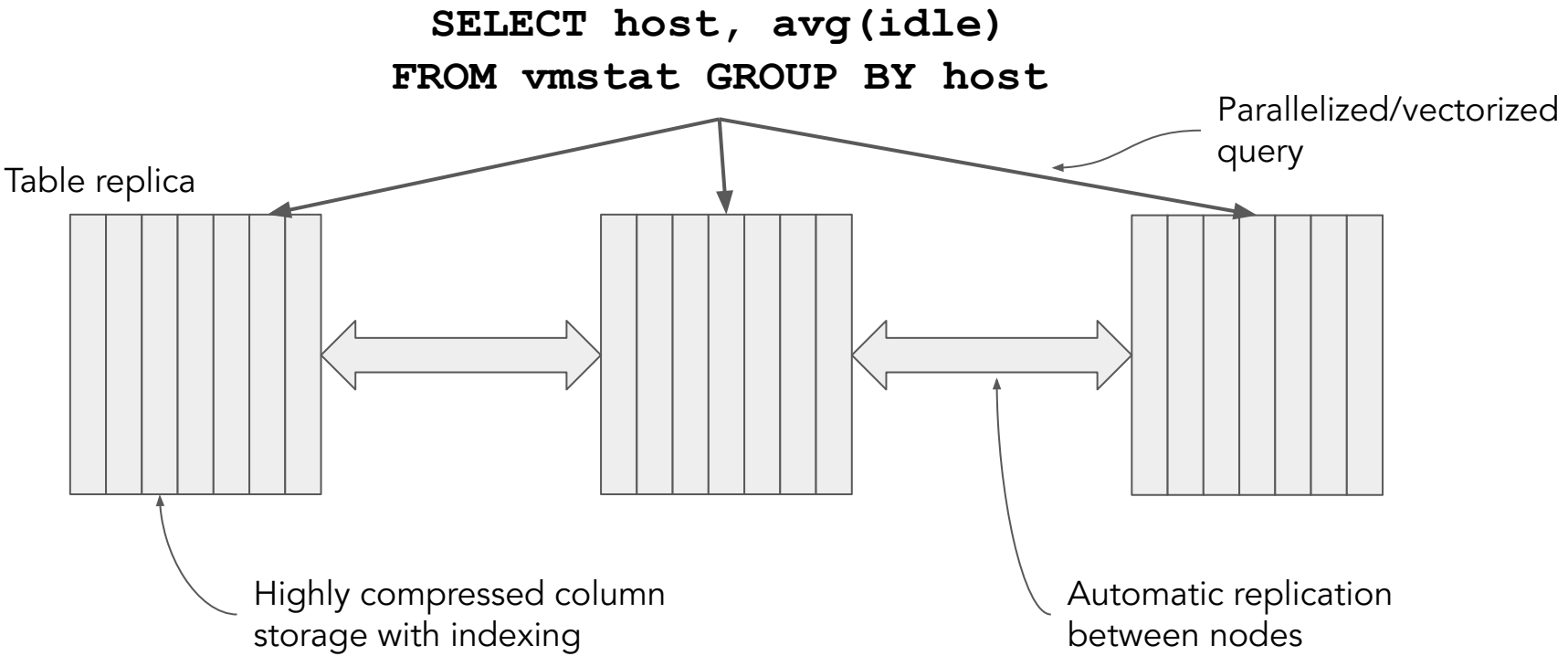
Scales to many petabytes

Is Open source (Apache 2.0)

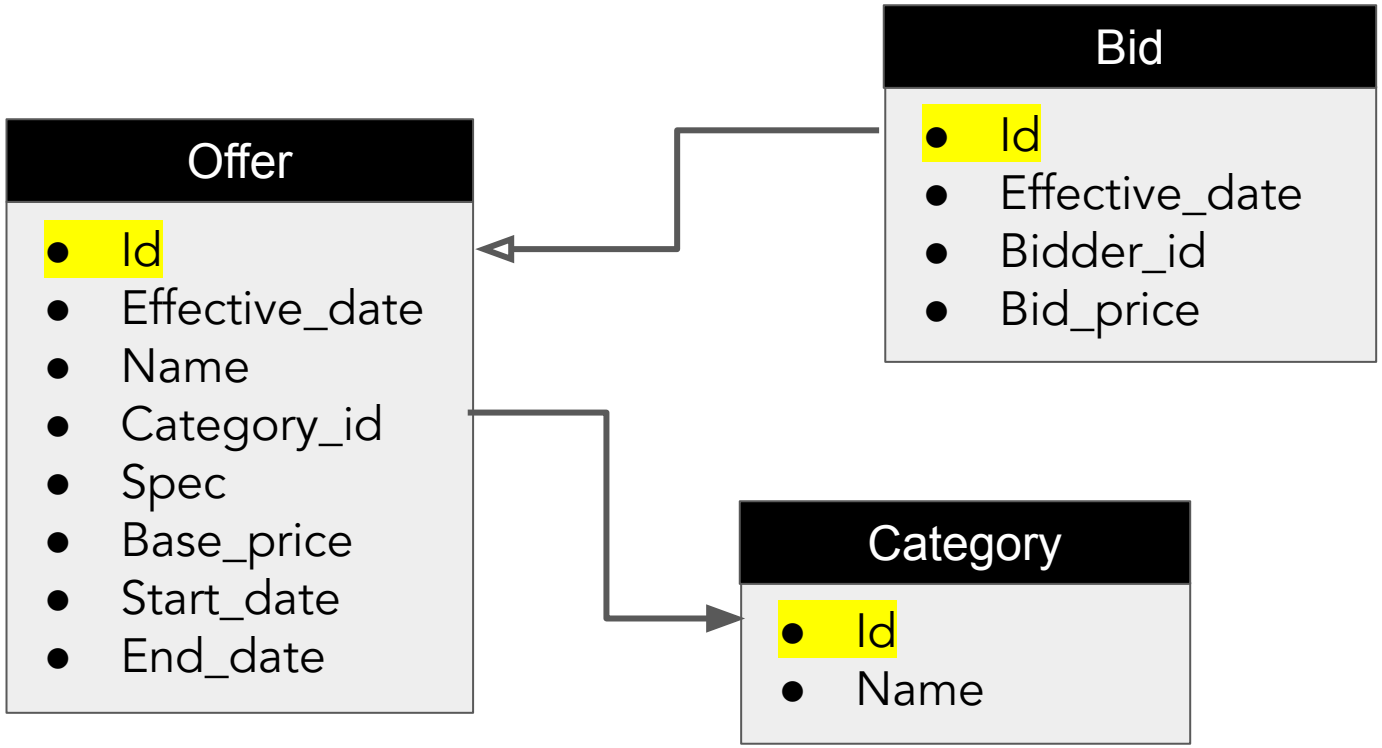


It's a popular engine for real-time analytics

ClickHouse optimizes for fast response on large datasets



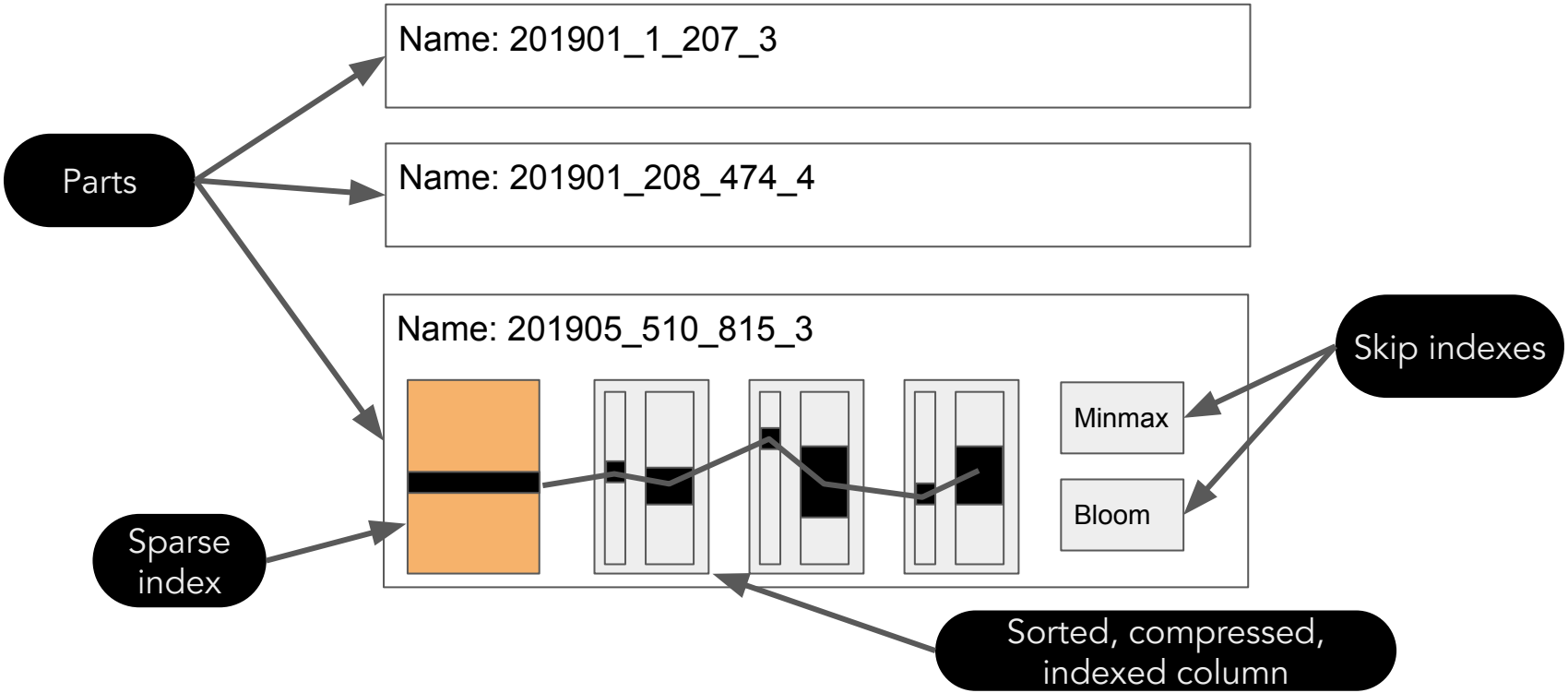
Example: a product auction site



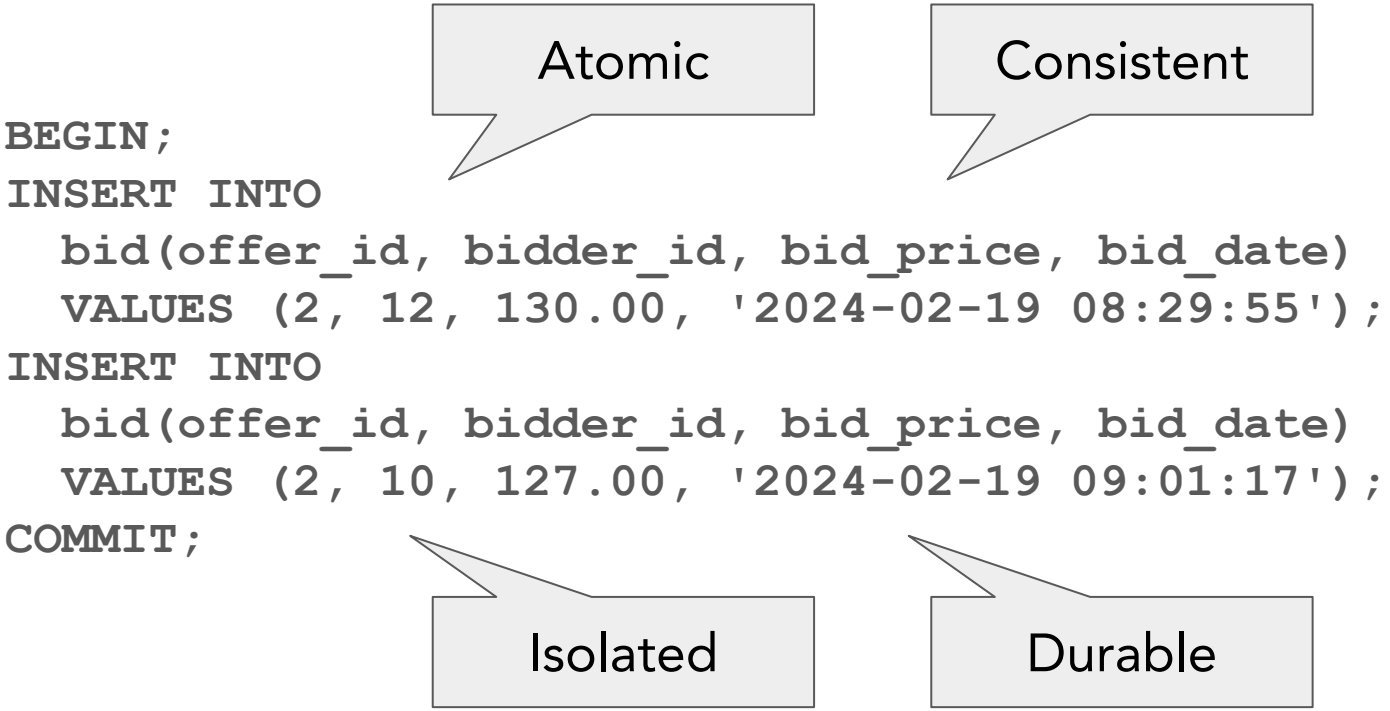
Modeling tables in ClickHouse

```
CREATE TABLE bid (
    id Int64,
    bidder_id Int64,
    offer_id Int64,
    bid_price Float32,
    bid_date DateTime,
)
Engine=MergeTree
PARTITION BY toDate(bid_date)
ORDER BY (bidder_id, product_id, bid_date)
```

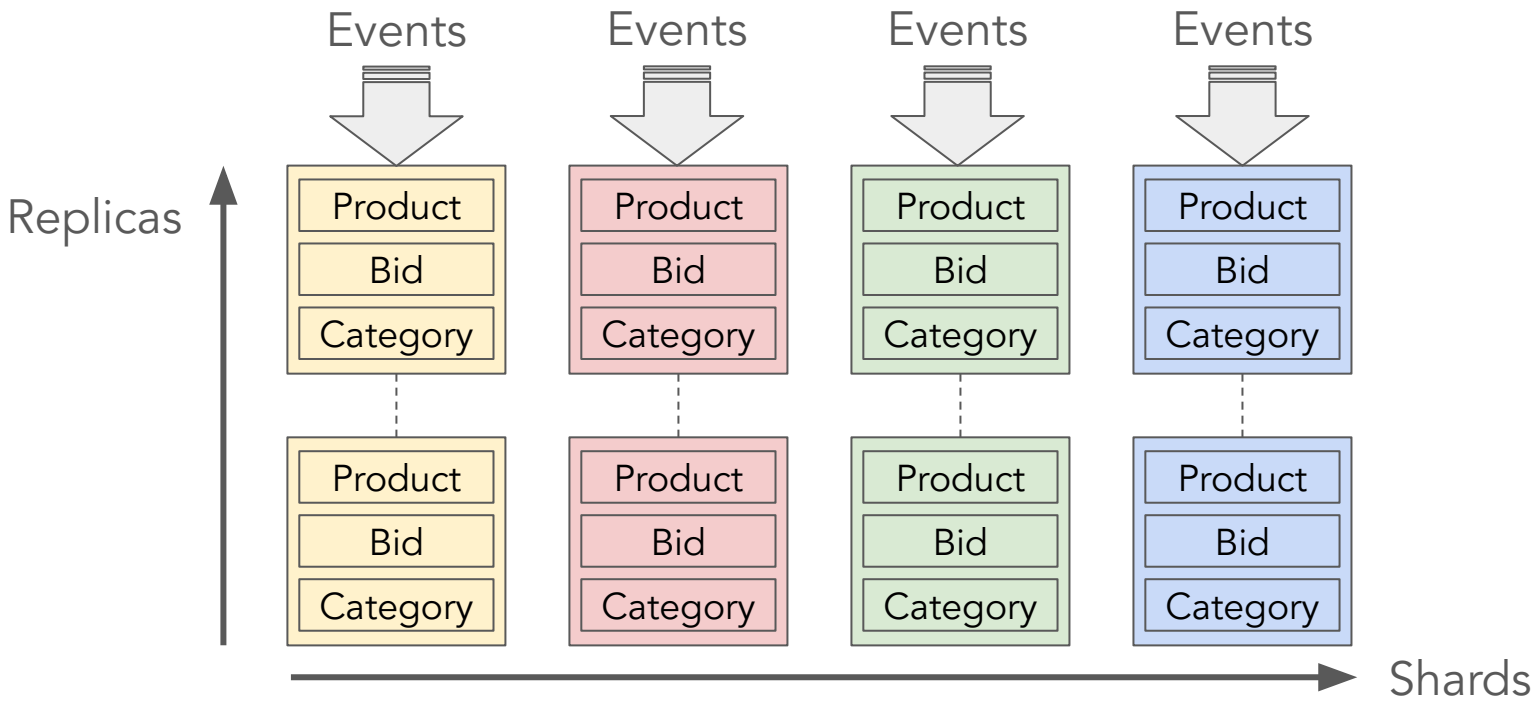
Table organization in ClickHouse analytic databases



We've become accustomed to SQL



ACID is expensive in large systems with rapid ingest

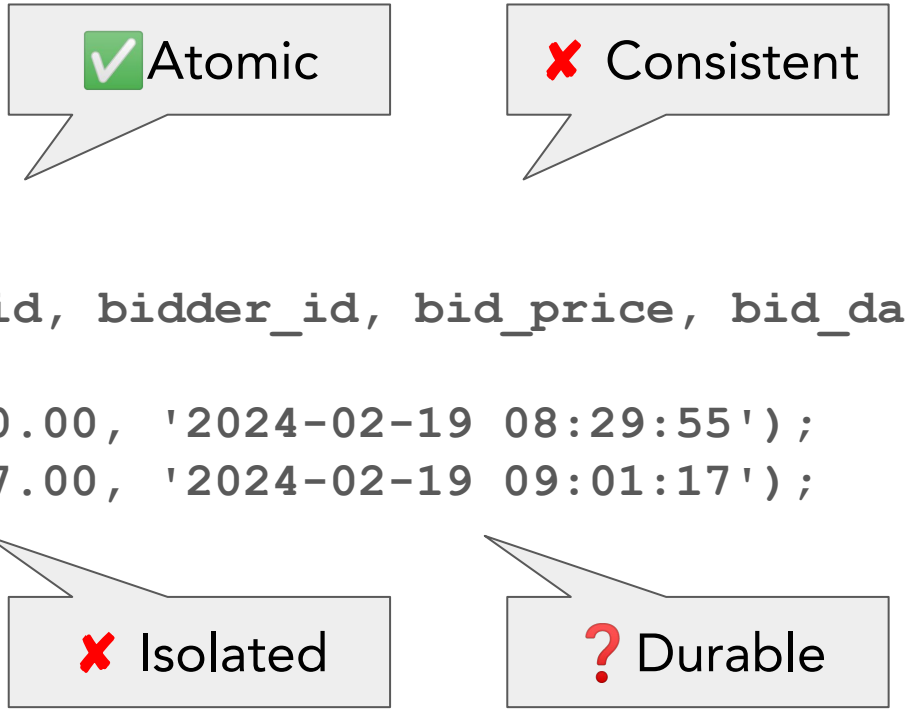


Ideas from distributed systems to deal with scale

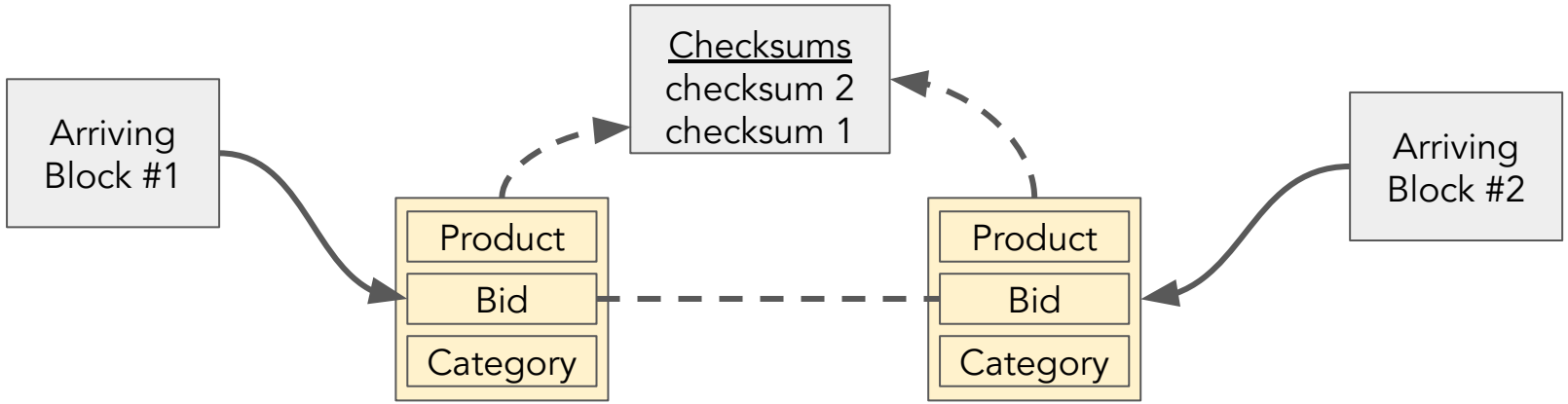
Immutable data

Eventual consistency

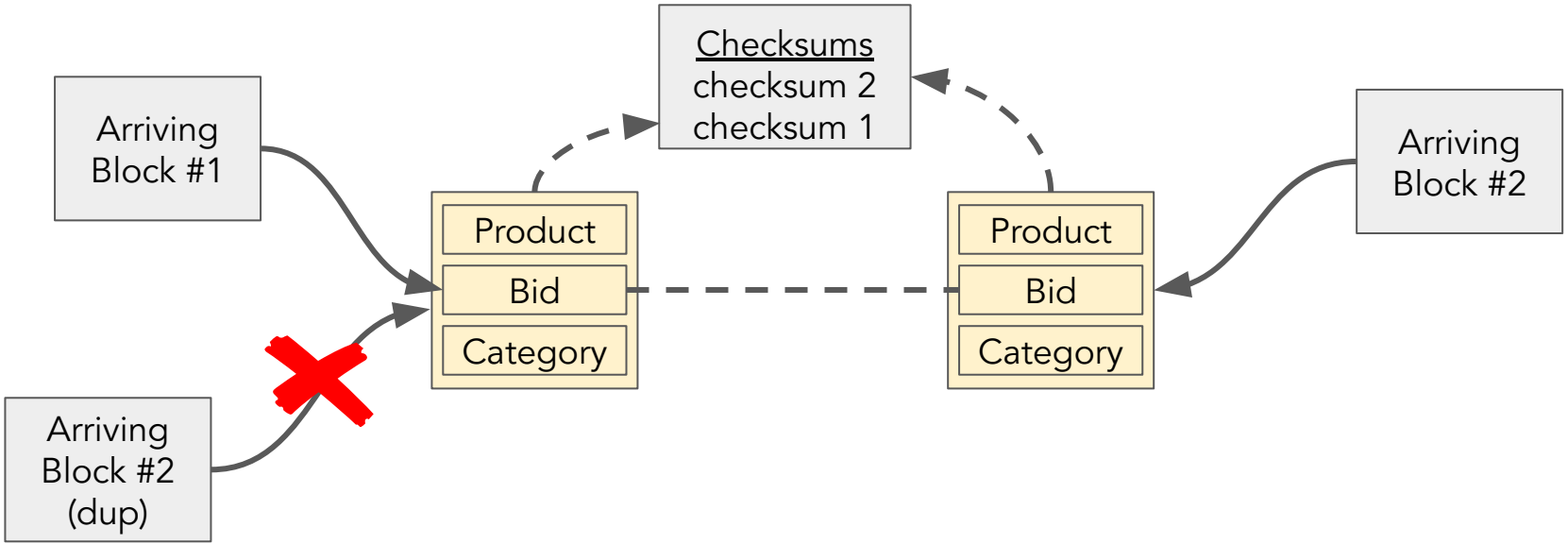
What the insert looks like in ClickHouse



Use block checksums to avoid duplicate data from failures



Redundant blocks are rejected



What about updating and deleting rows?

```
UPDATE bid
   SET bid_price = 135.00
     WHERE id = 14;

DELETE bid
   WHERE id = 14;
```

There's a table type for that!

```
CREATE TABLE bid_rmt (
    id Int64,
    bidder_id Int64,
    offer_id Int64,
    offer_effective_date DateTime,
    bid_price Float32,
    effective_date DateTime DEFAULT NOW(),
    is_deleted UInt8 DEFAULT 0)
Engine=ReplacingMergeTree(effective_date, is_deleted)
PARTITION BY toDate(offer_effective_date)
ORDER BY (bidder_id, id)
```

Version and flag for deleted rows

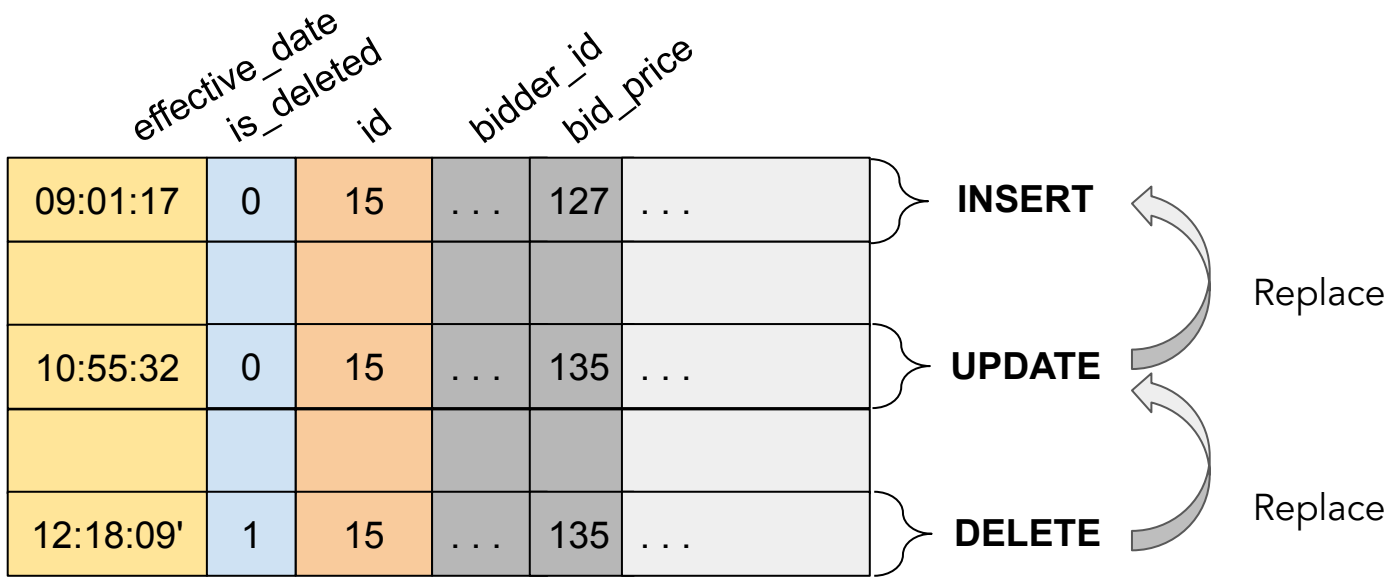
Deduplicate on these columns

Updating and deleting data with inserts only

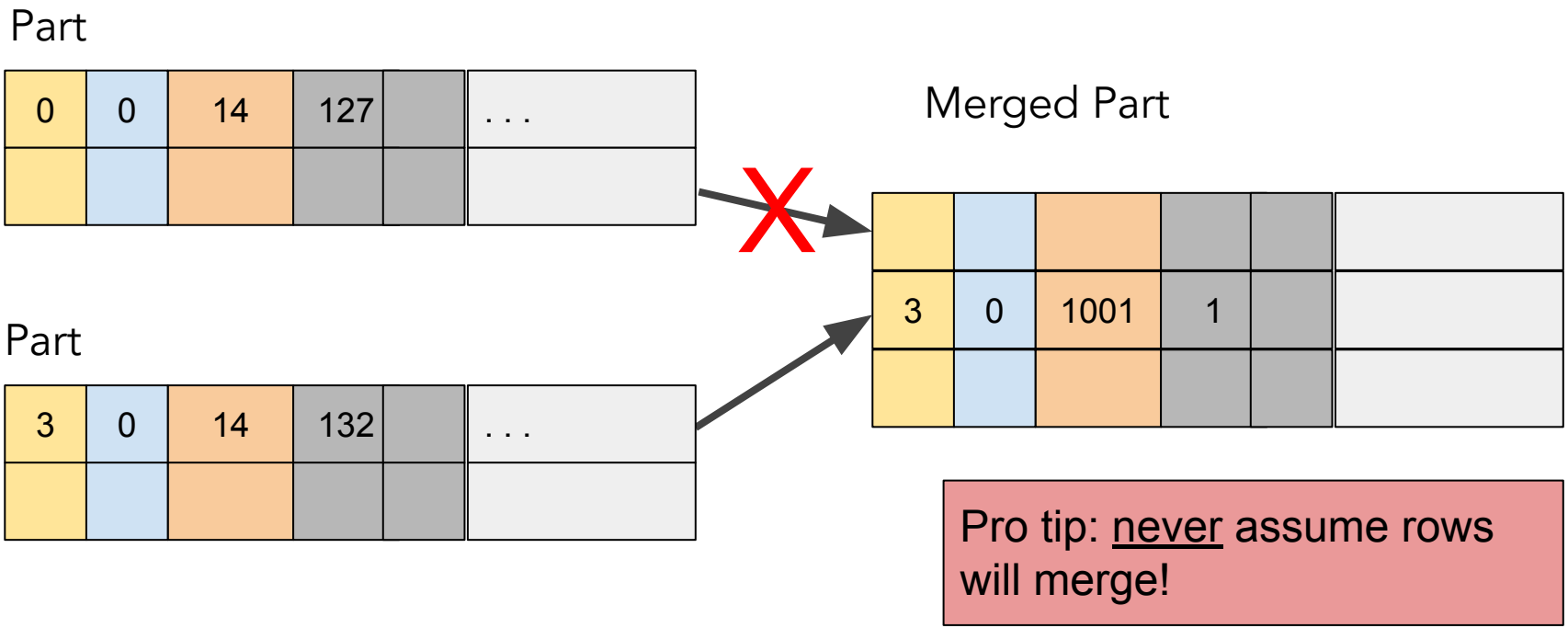
```
-- Add a row
INSERT INTO bid_rmt(id, offer_id, bidder_id...effective_date)
VALUES (15, 5, 10, 127.00,...,'2024-02-19 09:01:17');

-- Update a row
INSERT INTO bid_rmt(id, offer_id, bidder_id...effective_date)
VALUES (15, 5, 10, 135.00,...,'2024-02-20 10:55:32');

-- Delete a row
INSERT INTO bid_rmt(id, offer_id,...effective_date, is_deleted)
VALUES (15, 5, 10, 135.00,...,'2024-02-20 12:18:09', 1);
```

How ReplacingMergeTree works

Eventually consistent replacement of rows



This is where eventual consistency kicks in…



We need to resolve the inconsistencies in the query

```
SELECT id, bid_price, effective_date, is_deleted
  FROM bid_rmt ORDER BY id, effective_date;
```

id	bid_price	effective_date	is_deleted
14	130	2024-02-19 08:29:55	0
15	127	2024-02-19 09:01:17	0
15	135	2024-02-20 10:55:17	0
15	135	2024-02-20 12:55:17	1

```
SELECT id, bid_price, effective_date, is_deleted
  FROM bid_rmt ORDER BY id, effective_date SETTINGS final = 1;
```

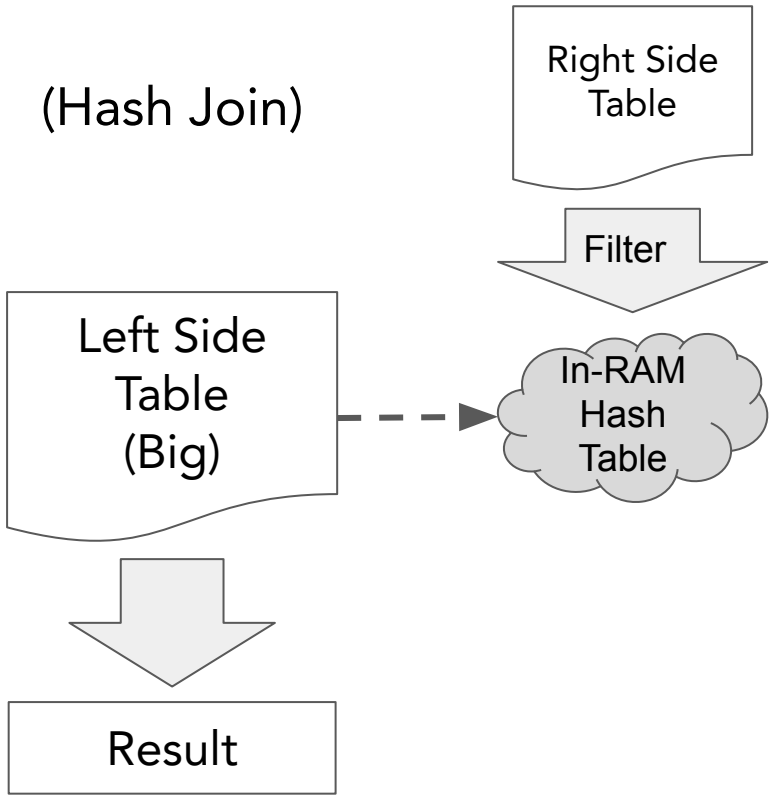
id	bid_price	effective_date	is_deleted
14	130	2024-02-19 08:29:55	0

What about joins between tables?

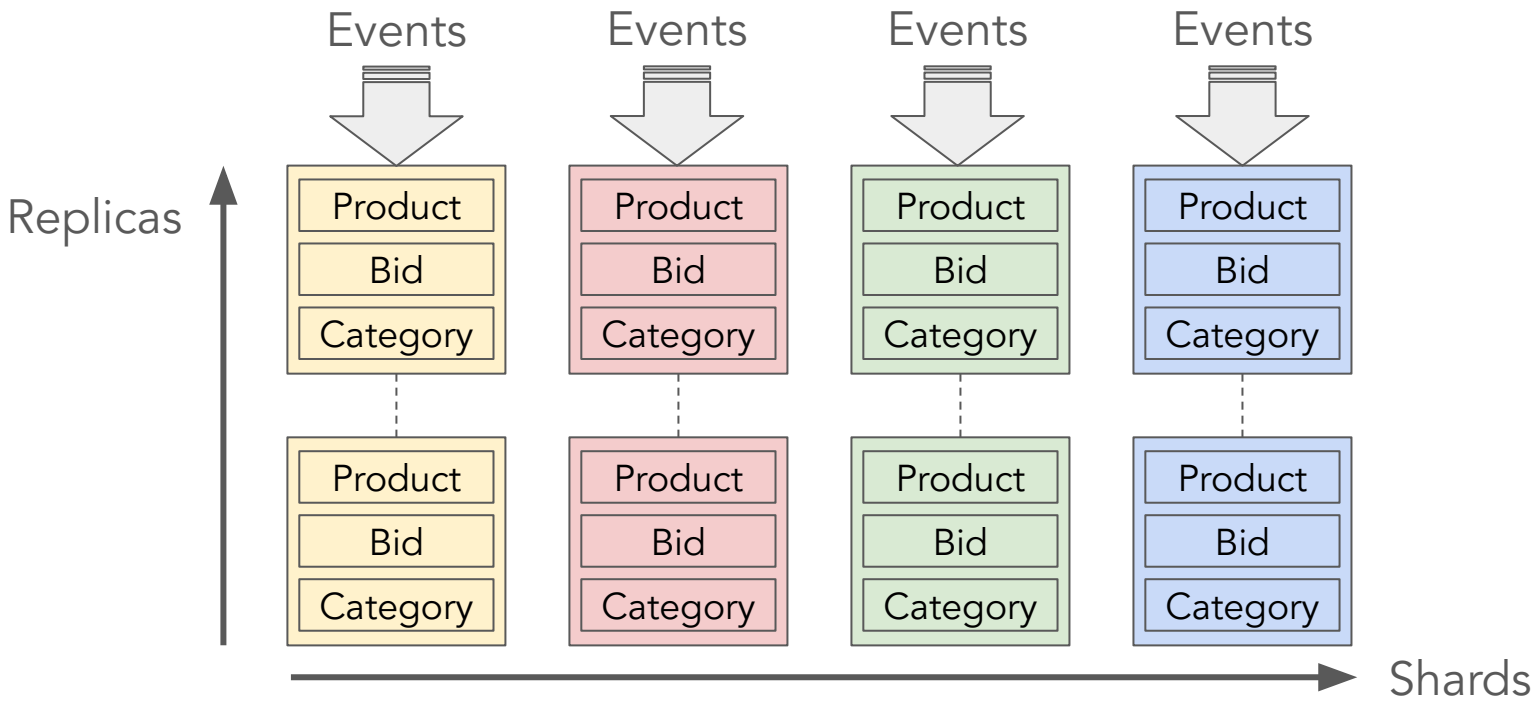
```
SELECT o.id,
       any(base_price) as starting_pric,
       min(b.bid_price) as min,
       max(b.bid_price) as max
  FROM bid_rmt b
    JOIN offer_rmt o ON o.id=b.offer_id
      WHERE o.id = 5
        GROUP BY o.id;
```

id	starting_price	min	max
5	125	127	135

How joins work in ClickHouse



Joins are expensive and unpredictable in large systems



How can we make joins really fast on large data sets?

Create “joins” that can work in a single scan over many hosts

Hint 1: Put everything in a single table

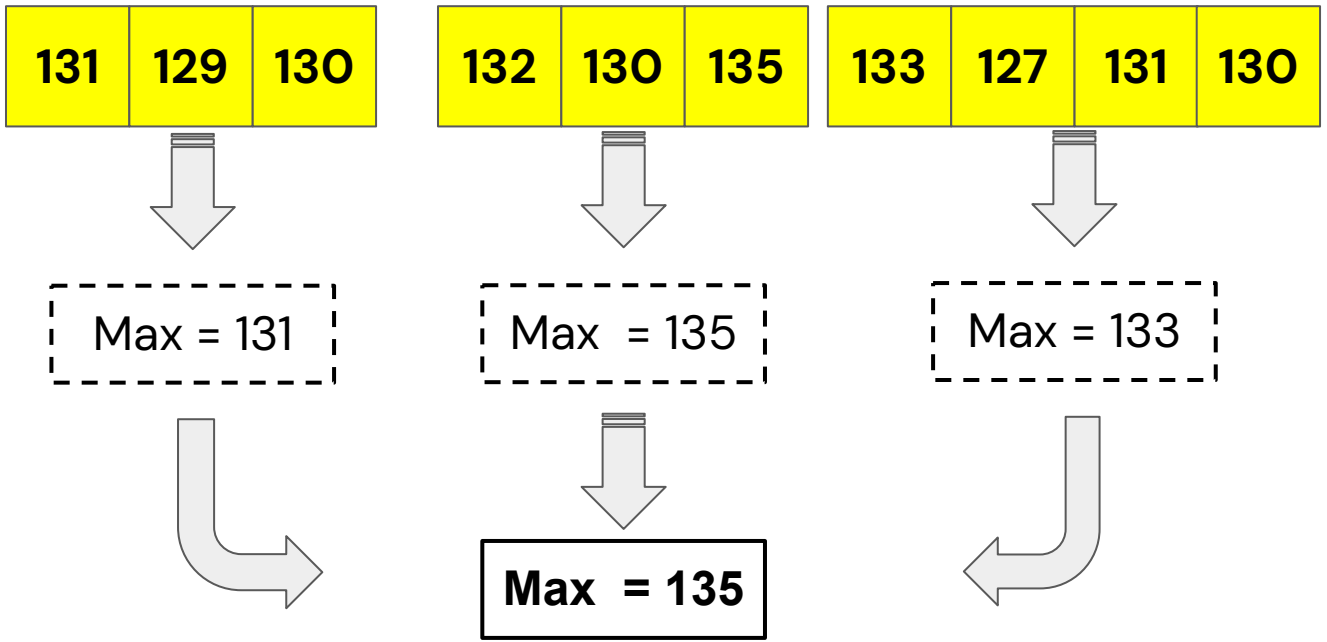
```
CREATE TABLE offer_bid_big (
    record enum('offer'=1, 'bid'=2), effective_date DateTime,
    -- Product offer values.
    offer_id Int64, name String,
    category String, spec String,
    base_price Float32,
    -- Product bid values.
    bid_id Int64, bidder_id Int64,
    bid_price Float32,
)
Engine=MergeTree
PARTITION BY toDate(effective_date)
PRIMARY KEY (category, name, offer_id)
ORDER BY (category, name, offer_id, effective_date)
```

Hint 2: Use aggregation instead of joins!

No need to move data

Parallelizes!

Intermediate results are reusable

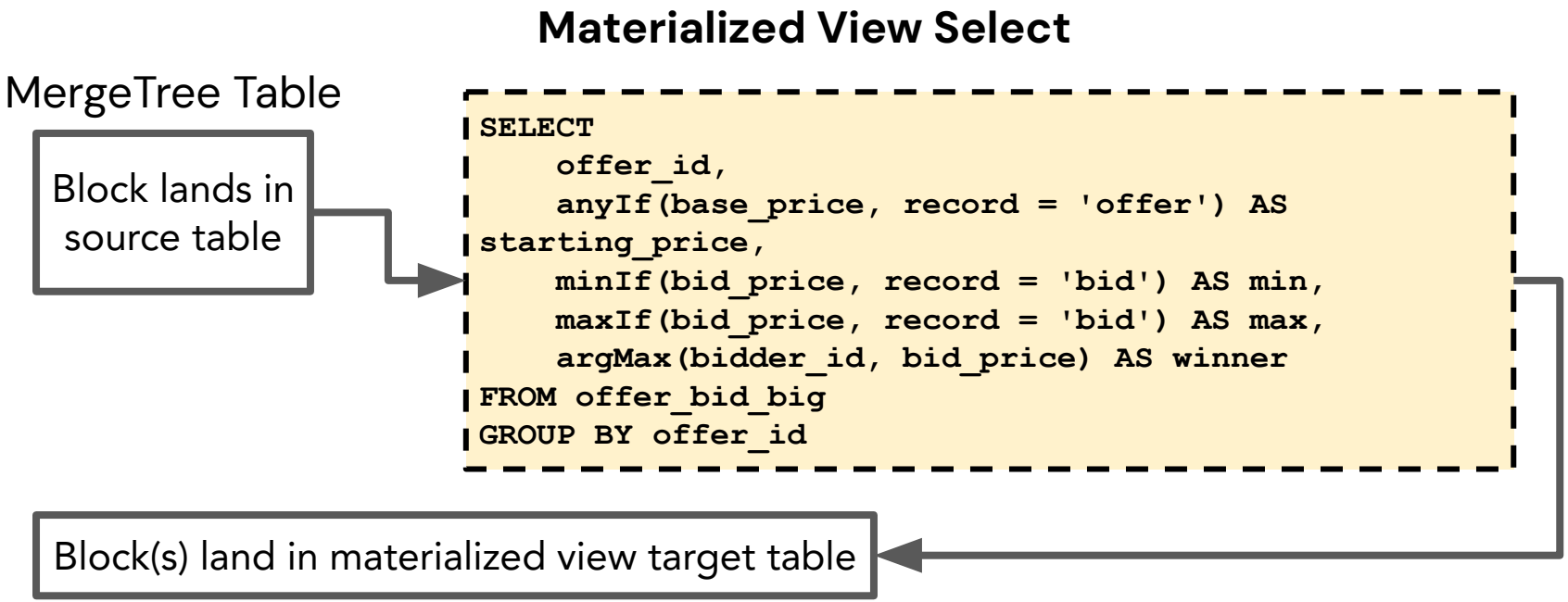


And here's the "SELECT" code…

```
SELECT
    offer_id,
    anyIf(base_price, record = 'offer') AS starting_price,
    minIf(bid_price, record = 'bid') AS min,
    maxIf(bid_price, record = 'bid') AS max,
    argMax(bidder_id, bid_price) AS winner
FROM offer_bid_big
WHERE offer_id = 5
GROUP BY offer_id
```

```
┌─offer_id─┬─starting_price─┬─min─┬─max─┬─winner─┐
│        5 │            125 │ 129 │ 135 │    100 │
└──────────┴────────────────┴─────┴─────┴────────┘
```

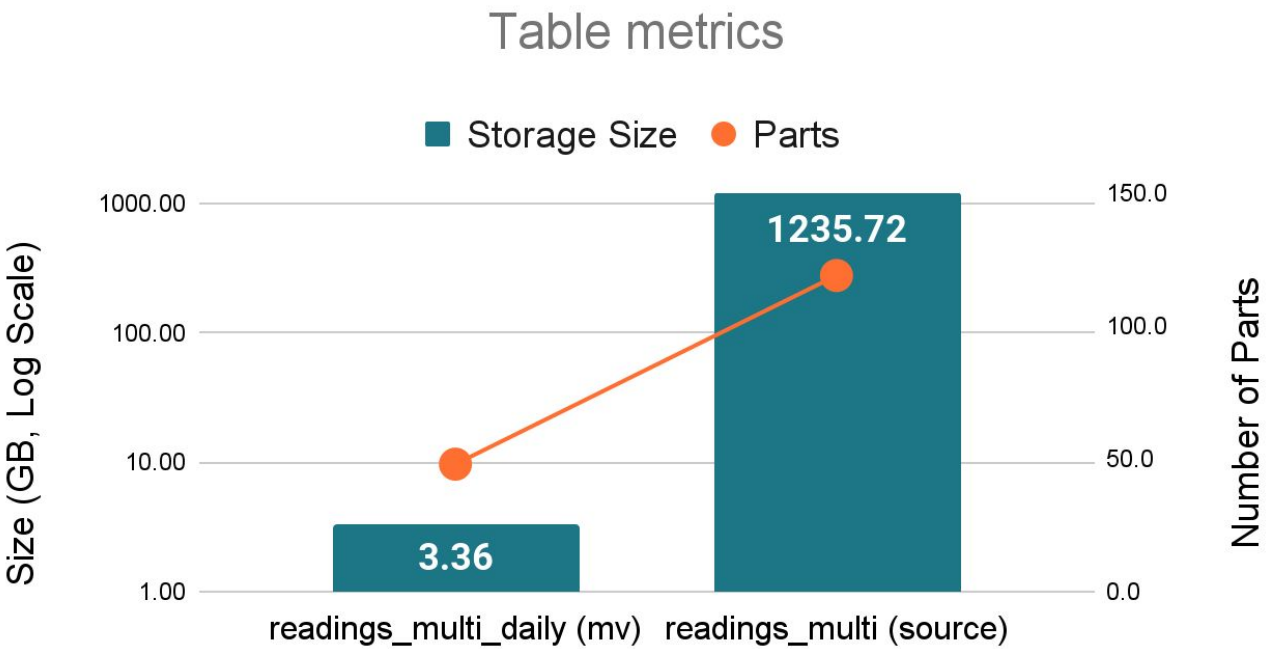
Use materialized views to pre-compute popular results



And here's code for the materialized view…

```
CREATE MATERIALIZED VIEW bidding_view
Engine = AggregatingMergeTree() ORDER BY (offer_id)
AS
SELECT
    offer_id,
    anyIf(base_price, record = 'offer') AS starting_price,
    minIf(bid_price, record = 'bid') AS min,
    maxIf(bid_price, record = 'bid') AS max,
    argMax(bidder_id, bid_price) AS winner
FROM offer_bid_big
GROUP BY offer_id;
```

Comparison of source table to typical materialized view



Welcome to the world of fast, big data

- Transactions and joins are costly in large scale analytic apps
- Think like a distributed systems engineer to succeed
  - Immutable data - Everything is an INSERT
  - Eventual consistency - Fix up data consistency when reading
- Real-time analytic databases like ClickHouse offer solutions
  - Columnar storage allows wide tables
  - Specialized techniques take the place of transactions
  - Aggregation can replace joins

We're no longer in Edgar Codd's world. Data location is everything.

Where can I find out more?

Samples: https://github.com/Altinity/clickhouse-sql-examples

Altinity Blog and YouTube Channel - https://altinity.com

- ClickHouse ReplacingMergeTree Explained: The Good, The Bad, and The Ugly

ClickHouse official docs – https://clickhouse.com/docs/

Getting started with ClickHouse

ClickHouse Community Builds
Monthly builds
LTS builds every 6 months
(1 year of support)
https://clickhouse.com

Altinity Stable Builds

Prod-ready LTS builds only
(3 years of support)
https://docs.altinity.com

```
# Ubuntu example
sudo apt-get install -y clickhouse-server clickhouse-client
sudo systemctl start clickhouse-server
```

Thank you!

Robert Hodges - Altinity
https://altinity.com

Email: rhodges at altinity dot com
Slack: @Robert Hodges (Kubernetes, ClickHouse, AltinityDB)
LinkedIn: https://www.linkedin.com/in/berkeleybob2105/