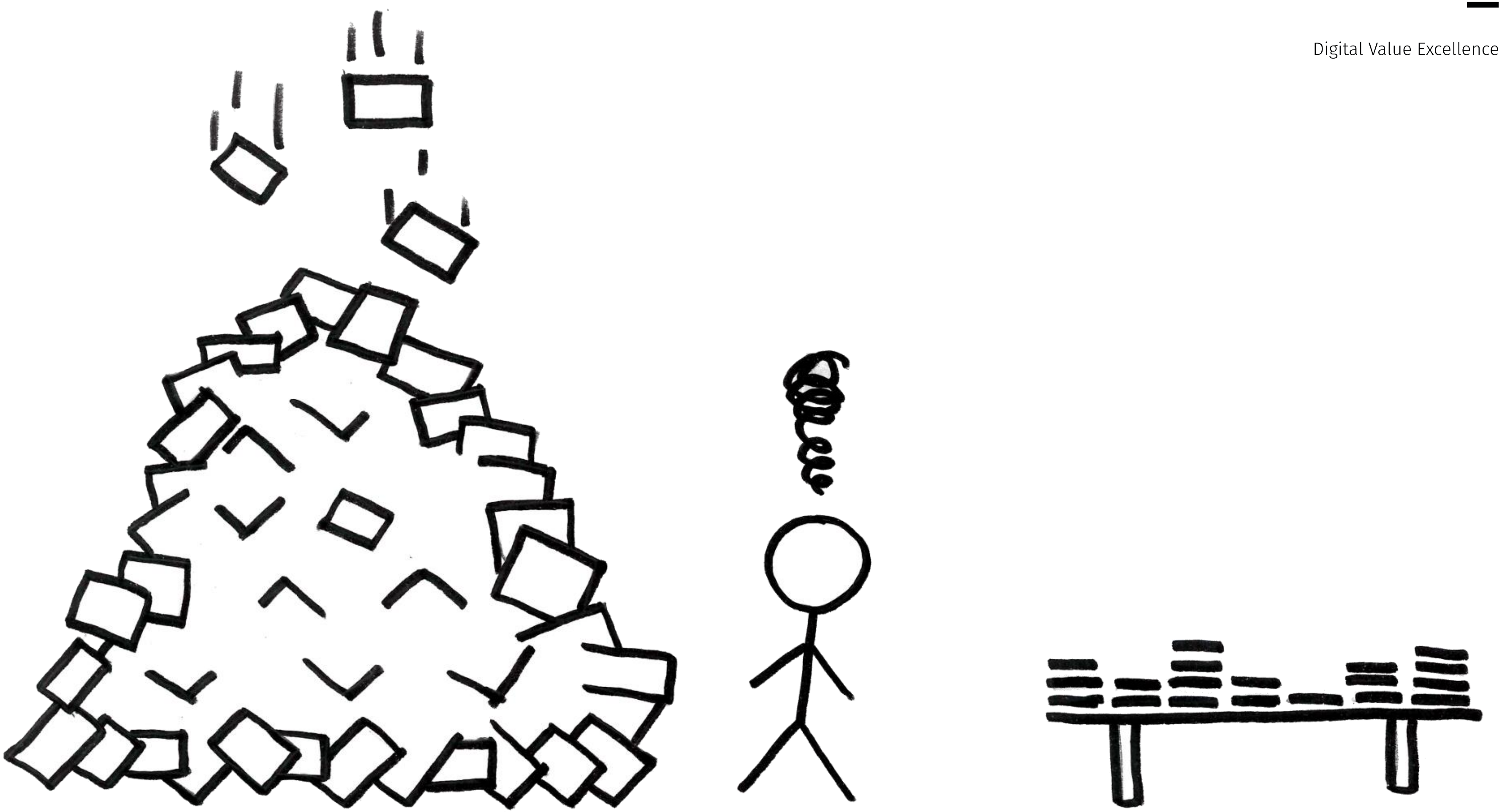
Unusual applications of a column-oriented database

Sascha Dienelt – Senior Software Developer (data warehouse, bid management)

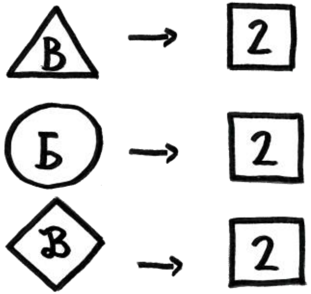
Usual applications of column-oriented DBMS

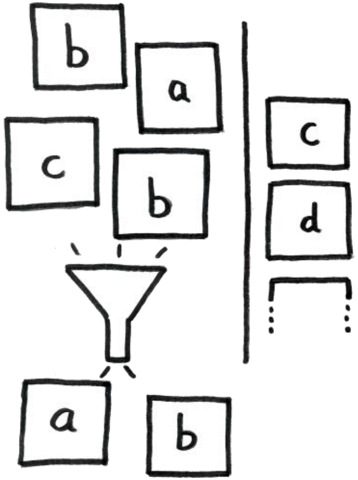
- Fetch a huge amount of time series data
- Aggregate many rows at once
- Process date ranges

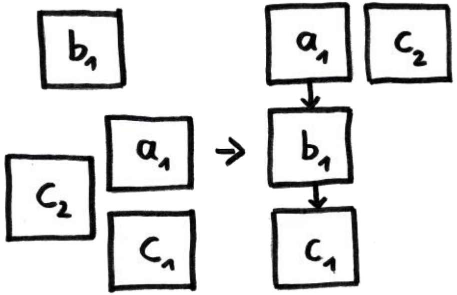


What we need to do

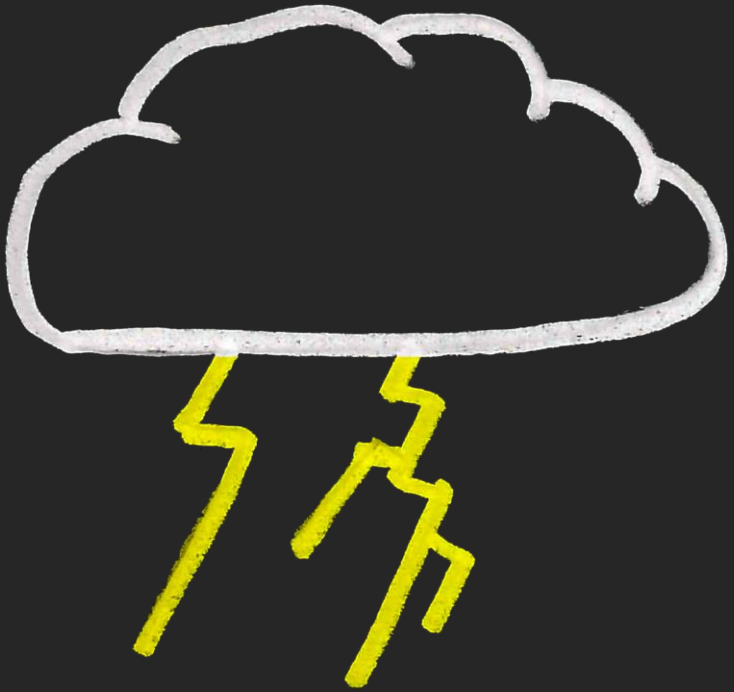
- Normalize values
- Filter duplicates
- Build chains







What we need to do



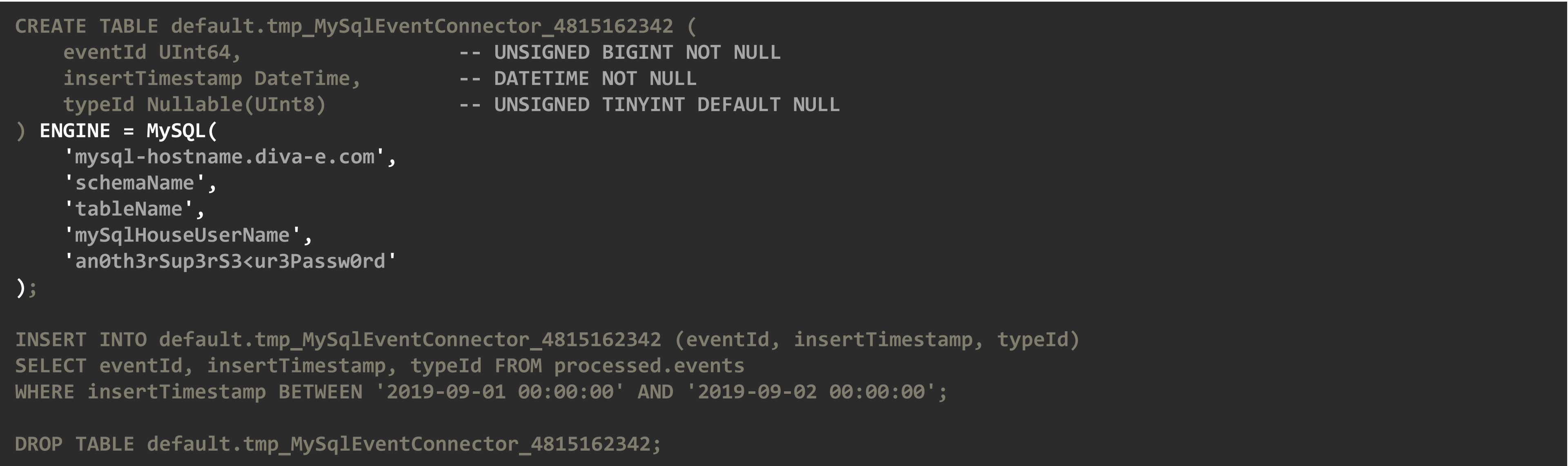
Different ClickHouse clusters

- cluster 1: raw data
- cluster 2: processed information
- Solution: remote table functions (ClickHouse <-> ClickHouse)
- Communication overhead → don’t use too extensive
- Alternative: Distributed table engine

```
INSERT INTO processed.events
    (eventId, insertTimestamp, typeId)
SELECT
    eventId,
    insertTimestamp,
    dictGetUInt16('Global_MasterData.Event_Types', 'typeId', visitParamExtractString(eventData, 'typeName')) AS typeId
FROM remoteSecure(
    'raw-clickhouse-hostname.diva-e.com',
    'rawSchemaName',
    'EventTableName',
    'clickHouseUserName',
    'sup3rS3<ur3Passw0rd'
) AS rawEvents
WHERE insertTimestamp BETWEEN '2019-09-01 00:00:00' AND '2019-09-02 00:00:00';
```

Master data and further processing in MySQL

- Master data must be kept in a RDBMS (here: MySQL)
- Implementation of very complex logic is using MySQL
- Solution: MySQL table engine (ClickHouse <-> MySQL)



```
CREATE TABLE default.tmp_MySqlEventConnector_4815162342 (
    eventId UInt64,                    -- UNSIGNED BIGINT NOT NULL
    insertTimestamp DateTime,          -- DATETIME NOT NULL
    typeId Nullable(UInt8)             -- UNSIGNED TINYINT DEFAULT NULL
) ENGINE = MySQL(
    'mysql-hostname.diva-e.com',
    'schemaName',
    'tableName',
    'mySqlHouseUserName',
    'an0th3rSup3rS3<ur3Passw0rd'
);

INSERT INTO default.tmp_MySqlEventConnector_4815162342 (eventId, insertTimestamp, typeId)
SELECT eventId, insertTimestamp, typeId FROM processed.events
WHERE insertTimestamp BETWEEN '2019-09-01 00:00:00' AND '2019-09-02 00:00:00';

DROP TABLE default.tmp_MySqlEventConnector_4815162342;
```

Client settings require different queries

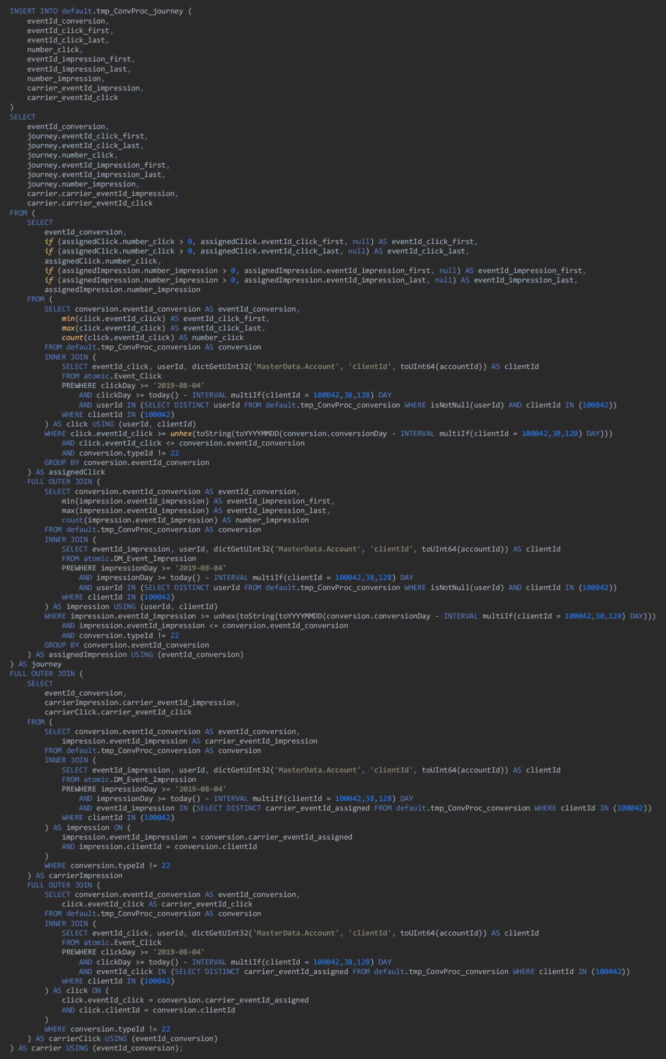
- processing many elements at once → fast
- many “small” queries → very slow
- Clients want different filters and chaining rules
- Solution: combine many steps into one query

```sql
SELECT eventId FROM processed.events
WHERE clientId = 1
    AND insertTimestamp >= now() - INTERVAL 14 DAY
    AND typeId IN (1, 2, 3);

SELECT eventId FROM processed.events
WHERE clientId = 2
    AND insertTimestamp >= now() - INTERVAL 30 DAY
    AND typeId IN (2, 3);

SELECT eventId FROM processed.events
WHERE clientId = 3
    AND insertTimestamp >= now() - INTERVAL 7 DAY
    AND typeId IN (1);
```

```sql
SELECT eventId FROM processed.events
WHERE insertTimestamp >= now() - INTERVAL multiIf(
   clientId = 1, 14,
   clientId = 2, 30,
   clientId = 3, 7,
   10
) DAY
AND multiIf(
   clientId = 1 AND typeId IN (1, 2, 3), 1,
   clientId = 2 AND typeId IN (2, 3), 1,
   clientId = 3 AND typeId IN (1), 1,
   0
);
```



```sql
INSERT INTO default.tmp_ConvProc_journey (
    eventId_conversion,
    eventId_click_first,
    eventId_click_last,
    number_click,
    eventId_impression_first,
    eventId_impression_last,
    number_impression,
    carrier_eventId_impression,
    carrier_eventId_click
)
SELECT
    eventId_conversion,
    journey.eventId_click_first,
    journey.eventId_click_last,
    journey.number_click,
    journey.eventId_impression_first,
    journey.eventId_impression_last,
    journey.number_impression,
    carrier.carrier_eventId_impression,
    carrier.carrier_eventId_click
FROM (
    SELECT
        eventId_conversion,
        if (assignedClick.number_click > 0, assignedClick.eventId_click_first, null) AS eventId_click_first,
        if (assignedClick.number_click > 0, assignedClick.eventId_click_last, null) AS eventId_click_last,
        assignedClick.number_click,
        if (assignedImpression.number_impression > 0, assignedImpression.eventId_impression_first, null) AS eventId_impression_first,
        if (assignedImpression.number_impression > 0, assignedImpression.eventId_impression_last, null) AS eventId_impression_last,
        assignedImpression.number_impression
    FROM (
        SELECT conversion.eventId_conversion AS eventId_conversion,
            min(click.eventId_click) AS eventId_click_first,
            max(click.eventId_click) AS eventId_click_last,
            count(click.eventId_click) AS number_click
        FROM default.tmp_ConvProc_conversion AS conversion
        INNER JOIN (
            SELECT eventId_click, userId, dictGetUInt32('MasterData.Account', 'clientId', toUInt64(accountId)) AS clientId
            FROM atomic.Event_Click
            PREWHERE clickDay >= '2019-08-04'
                AND clickDay >= today() - INTERVAL multiIf(clientId = 100042,38,128) DAY
                AND userId IN (SELECT DISTINCT userId FROM default.tmp_ConvProc_conversion WHERE isNotNull(userId) AND clientId IN (100042))
            WHERE clientId IN (100042)
        ) AS click USING (userId, clientId)
        WHERE click.eventId_click >= unhex(toString(toYYYYMMDD(conversion.conversionDay - INTERVAL multiIf(clientId = 100042,30,120) DAY)))
            AND click.eventId_click <= conversion.eventId_conversion
            AND conversion.typeId != 22
        GROUP BY conversion.eventId_conversion
    ) AS assignedClick
    FULL OUTER JOIN (
        SELECT conversion.eventId_conversion AS eventId_conversion,
            min(impression.eventId_impression) AS eventId_impression_first,
            max(impression.eventId_impression) AS eventId_impression_last,
            count(impression.eventId_impression) AS number_impression
        FROM default.tmp_ConvProc_conversion AS conversion
        INNER JOIN (
            SELECT eventId_impression, userId, dictGetUInt32('MasterData.Account', 'clientId', toUInt64(accountId)) AS clientId
            FROM atomic.DM_Event_Impression
            PREWHERE impressionDay >= '2019-08-04'
                AND impressionDay >= today() - INTERVAL multiIf(clientId = 100042,38,128) DAY
                AND userId IN (SELECT DISTINCT userId FROM default.tmp_ConvProc_conversion WHERE isNotNull(userId) AND clientId IN (100042))
            WHERE clientId IN (100042)
        ) AS impression USING (userId, clientId)
        WHERE impression.eventId_impression >= unhex(toString(toYYYYMMDD(conversion.conversionDay - INTERVAL multiIf(clientId = 100042,30,120) DAY)))
            AND impression.eventId_impression <= conversion.eventId_conversion
            AND conversion.typeId != 22
        GROUP BY conversion.eventId_conversion
    ) AS assignedImpression USING (eventId_conversion)
) AS journey
FULL OUTER JOIN (
    SELECT
        eventId_conversion,
        carrierImpression.carrier_eventId_impression,
        carrierClick.carrier_eventId_click
    FROM (
        SELECT conversion.eventId_conversion AS eventId_conversion,
            impression.eventId_impression AS carrier_eventId_impression
        FROM default.tmp_ConvProc_conversion AS conversion
        INNER JOIN (
            SELECT eventId_impression, userId, dictGetUInt32('MasterData.Account', 'clientId', toUInt64(accountId)) AS clientId
            FROM atomic.DM_Event_Impression
            PREWHERE impressionDay >= '2019-08-04'
                AND impressionDay >= today() - INTERVAL multiIf(clientId = 100042,38,128) DAY
                AND eventId_impression IN (SELECT DISTINCT carrier_eventId_assigned FROM default.tmp_ConvProc_conversion WHERE clientId IN (100042))
            WHERE clientId IN (100042)
        ) AS impression ON (
            impression.eventId_impression = conversion.carrier_eventId_assigned
            AND impression.clientId = conversion.clientId
        )
        WHERE conversion.typeId != 22
    ) AS carrierImpression
    FULL OUTER JOIN (
        SELECT conversion.eventId_conversion AS eventId_conversion,
            click.eventId_click AS carrier_eventId_click
        FROM default.tmp_ConvProc_conversion AS conversion
        INNER JOIN (
            SELECT eventId_click, userId, dictGetUInt32('MasterData.Account', 'clientId', toUInt64(accountId)) AS clientId
            FROM atomic.Event_Click
            PREWHERE clickDay >= '2019-08-04'
                AND clickDay >= today() - INTERVAL multiIf(clientId = 100042,38,128) DAY
                AND eventId_click IN (SELECT DISTINCT carrier_eventId_assigned FROM default.tmp_ConvProc_conversion WHERE clientId IN (100042))
            WHERE clientId IN (100042)
        ) AS click ON (
            click.eventId_click = conversion.carrier_eventId_assigned
            AND click.clientId = conversion.clientId
        )
        WHERE conversion.typeId != 22
    ) AS carrierClick USING (eventId_conversion)
) AS carrier USING (eventId_conversion);
```

High memory usage and long run times

- Lot of rows need to be covered
- Many complex joins
- Solutions:
    - Use pseudo temporary tables
        - Pro:
            - simpler queries
            - save memory
            - reusable calculations
        - Con:
            - more overhead
            - cleanup required

```
CREATE TABLE default.tmp_IntermediateResult_4815162342 (
    eventId FixedString(15),
    chainStartEventId FixedString(15),
    chainEndEventId FixedString(15)
) ENGINE = Log;
```

```
INSERT INTO default.tmp_IntermediateResult_4815162342
(eventId, chainStartEventId, chainEndEventId)
SELECT conversions.eventId,
       argMin(eventId, insertTimestamp) AS chainStartEventId,
       argMax(eventId, insertTimestamp) AS chainEndEventId
FROM processed.events AS conversions
INNER JOIN processed.events AS clicks ON (clicks.typeId = 2
    AND clicks.sessionId = conversions.sessionId)
WHERE conversions.typeId = 1;

DROP TABLE default.tmp_IntermediateResult_4815162342;
```

High memory usage and long run times

- Solutions:
  - Use clever partitioning ●
  - Filter by sorting key ○

```
CREATE TABLE raw.events (
    eventId FixedString(15),
    insertTimestamp DateTime,
    sessionId FixedString(18),
    clientId UInt32,
    eventType Enum8('click' = 1, 'conversion' = 2),
    eventData String
) ENGINE = MergeTree()
PARTITION BY toYYYYMM(insertTimestamp)
ORDER BY (sessionId, typeId, insertTimestamp, eventId);
```

```
SELECT sessionId,
    argMin(eventId, insertTimestamp) AS firstEventId,
    argMax(eventId, insertTimestamp) AS lastEventId
WHERE insertTimestamp >= '2019-08-01 00:00:00'
    AND insertTimestamp < '2019-09-01 00:00:00'
    AND typeId = 2
GROUP BY sessionId;
```

High memory usage and long run times

- Solutions:
  - Do not join plain table, join pre-filtered subselect

```
SELECT key, h.column1, l.column2
FROM default.hugeTable AS h
INNER JOIN default.veryLargeTable AS l USING (key)
WHERE h.insertDate BETWEEN '2019-01-01' AND '2019-01-31'
    AND l.insertDate BETWEEN '2019-02-01' AND '2019-02-28';
```

```
SELECT key, h.column1, l.column2
FROM (
     SELECT key, column1
     FROM default.hugeTable
     WHERE insertDate BETWEEN '2019-01-01' AND '2019-01-31'
 ) AS s
INNER JOIN (
    SELECT key, column2
    FROM default.veryLargeTable
    WHERE insertDate BETWEEN '2019-02-01' AND '2019-02-28'
) AS l USING (key);
```

Before and after

- Complex process
  - ClickHouse exports data
  - TSV files in NFS folders
  - MySQL imports data
  - Processed data is copied to ClickHouse

- Simpler process
  - No file usage anymore
  - Most processing directly in ClickHouse
  - Less copy overhead
  - MySQL: less load and storage
- Easier to maintain
- Faster low-level reporting

Questions?