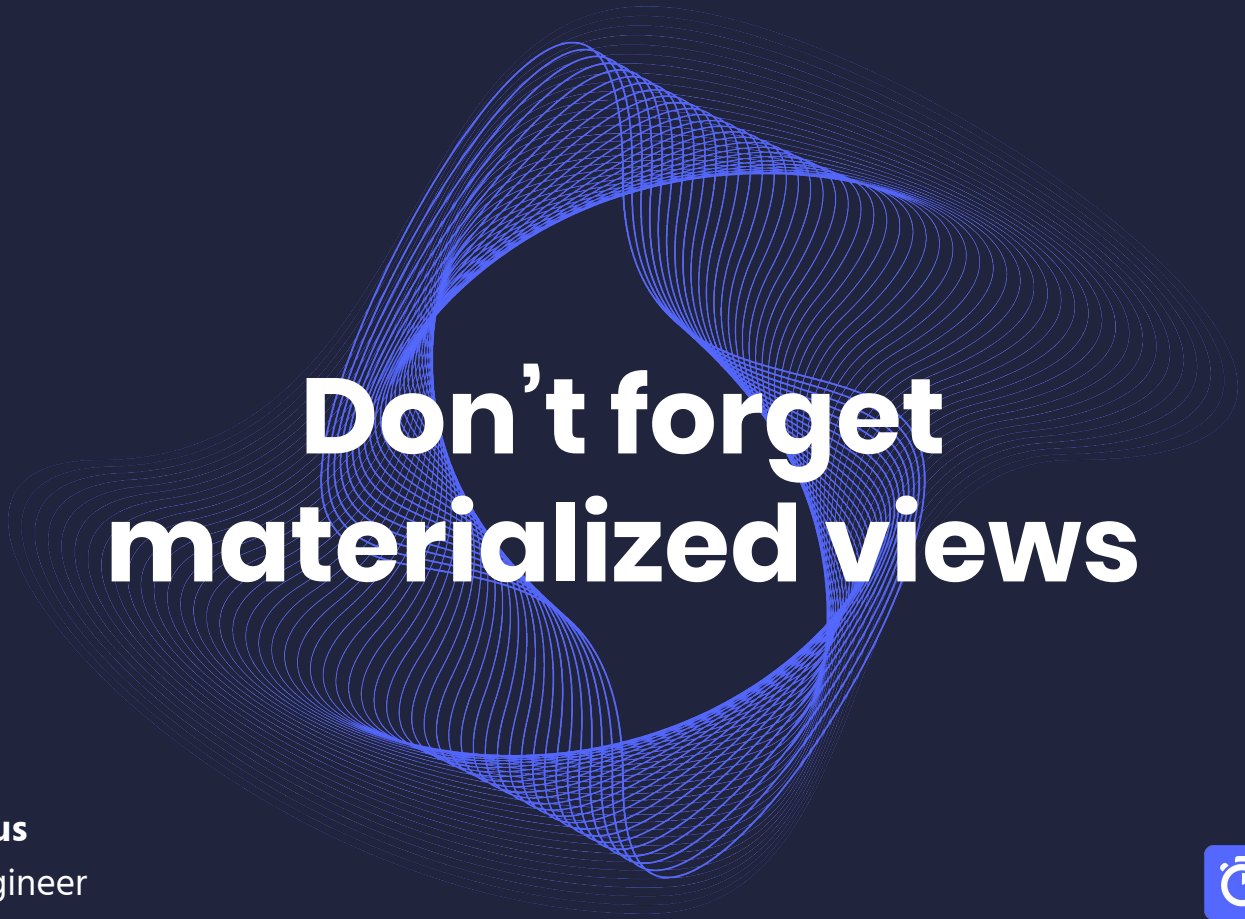


Don’t forget materialized views

Stephanie Baltus
Sr. Software engineer

algolia

Agenda

1. About
2. A bit of theory
3. Concrete use case
4. Wrap-Up

algolia

About

Me

- Loves cats, sneakers, sport
- 11 years in data ecosystem
  - Consulting, Leboncoin, JobTeaser, ManoMano
  - Currently software engineer at Algolia
- In love with PG since 2013

twitter : @steph_baltus
blog: honest.engineering

Algolia

algolia

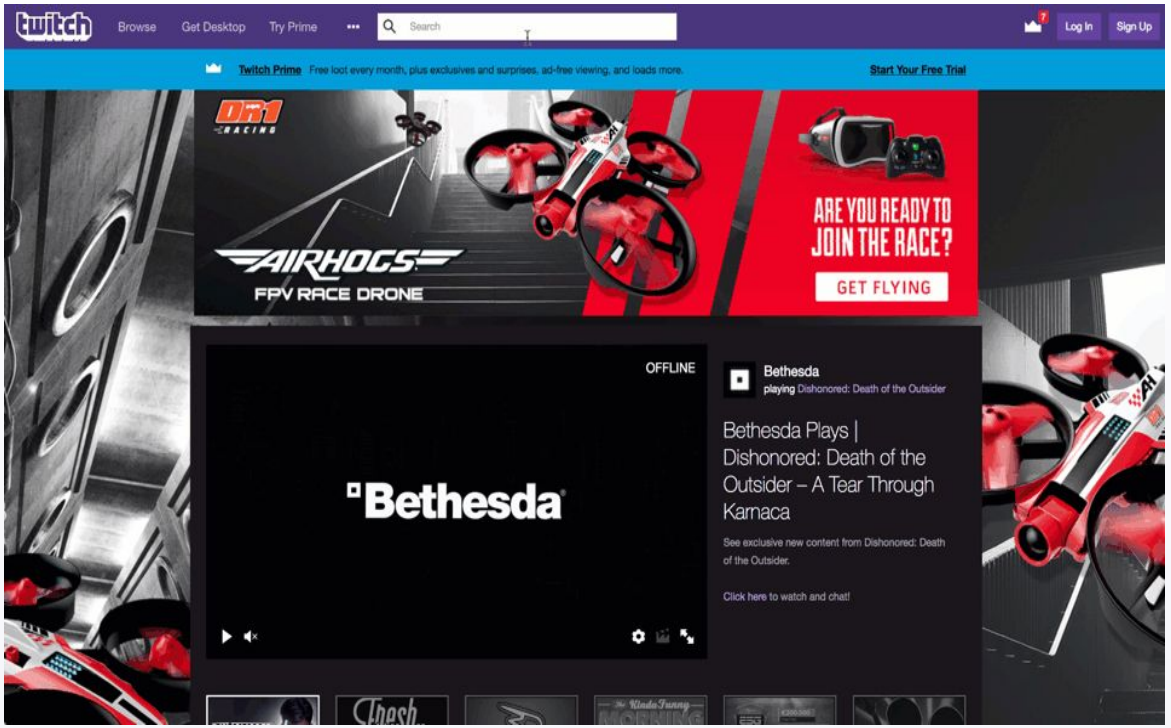
As you type speed

Relevance

Developer Experience

Algolia

algolia



Algolia

algolia

CONFIDENTIAL

300+
employees

16
regions

70+
datacenters

100+
Countries

200B+
API calls / month

algolia.com/careers

We're hiring!

algolia

algolia

A bit of theory

Views

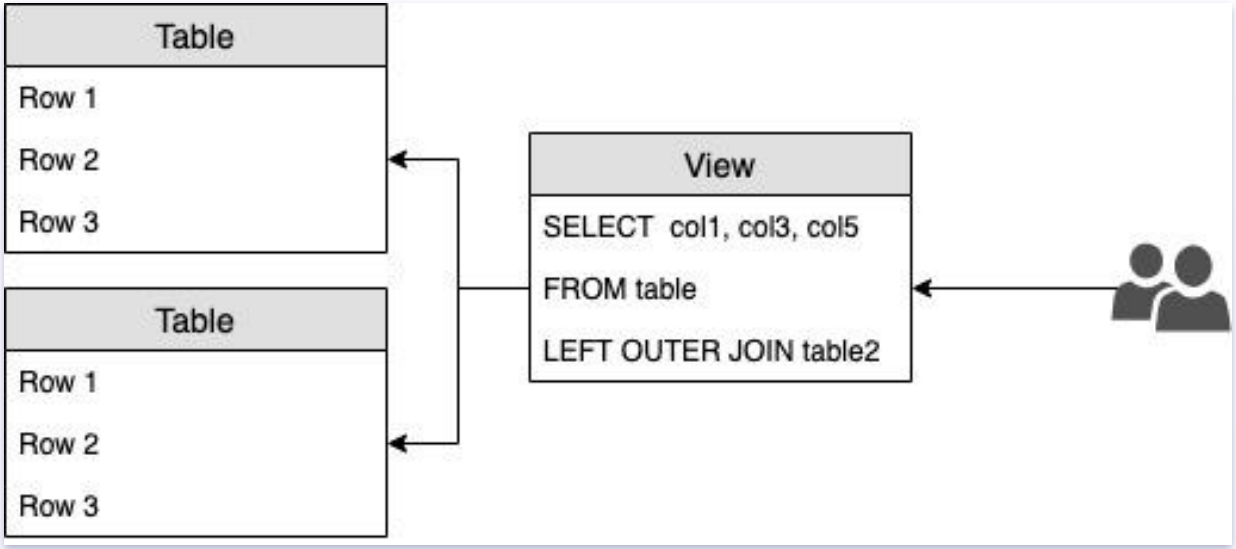


Table
Row 1
Row 2
Row 3
Table
Row 1
Row 2
Row 3
View
SELECT col1, col3, col5
FROM table
LEFT OUTER JOIN table2

MatViews

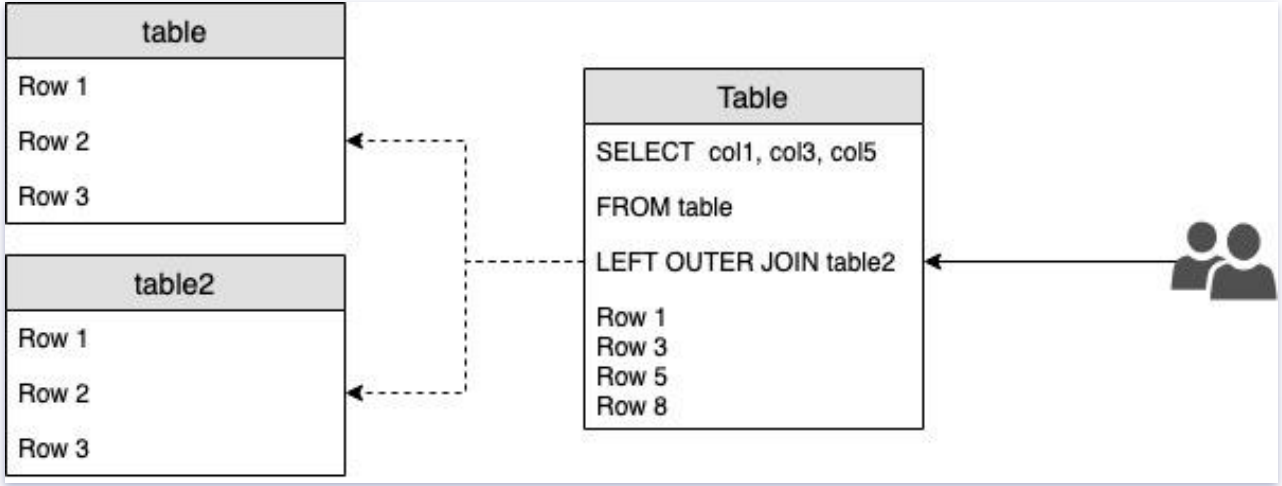
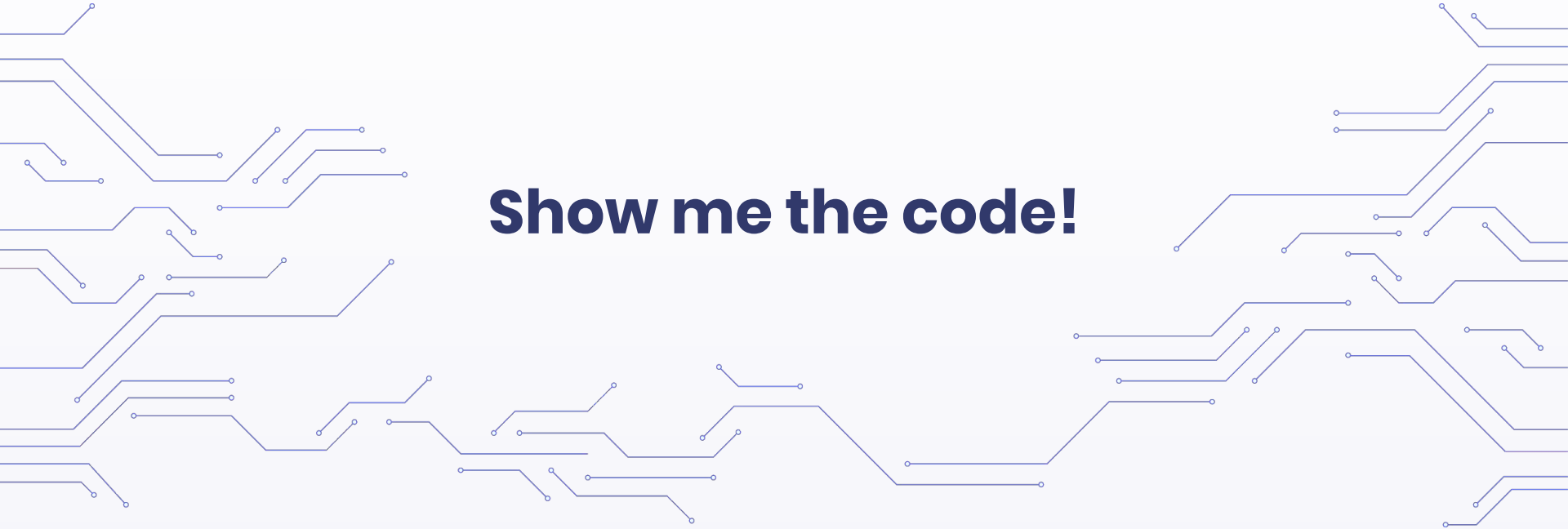


table
Row 1
Row 2
Row 3
table2
Row 1
Row 2
Row 3
Table
SELECT col1, col3, col5
FROM table
LEFT OUTER JOIN table2
Row 1
Row 3
Row 5
Row 8

MatViews

algolia

- Added in 9.3 (2013)
- Results are persisted
- On-demand update
- Behaves like a table
  - Indices creation
  - Key constraints
  - Maintenance operations
  - Supports Joins


Show me the code!

createas.c

algolia

```
/* This code supports both CREATE TABLE AS and CREATE MATERIALIZED VIEW */
    is_matview = (into->viewQuery != NULL);
    relkind = is_matview ? RELKIND_MATVIEW : RELKIND_RELATION;
…
/* Create the "view" part of a materialized view. */
    if (is_matview)
    {
        /* StoreViewQuery scribbles on tree, so make a copy */
        Query    *query = (Query *) copyObject(into->viewQuery);

        StoreViewQuery(intoRelationAddr.objectId, query, false);
        CommandCounterIncrement();
    }
```

src/backend/commands/createas.c

Example of use cases

algolia

- Cache slow query results
- Cache foreign data wrapper results
- No data freshness constraint
- Let the data engineers mess up the source table(s) with [no] consequence

Concrete Use case

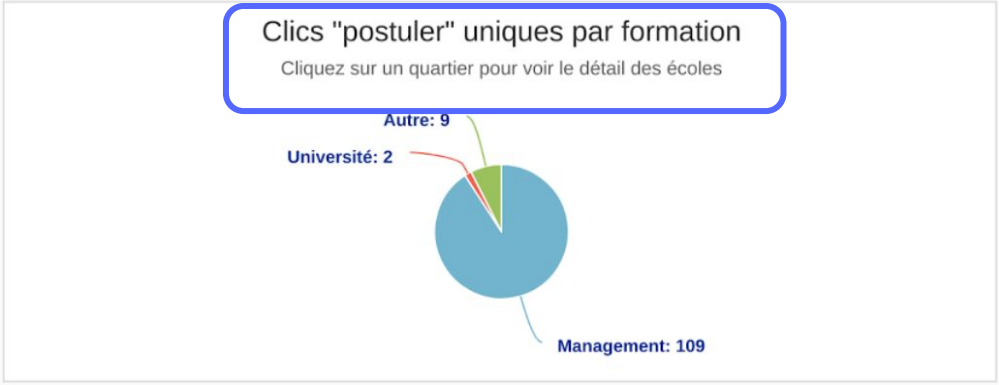
JobTeaser

algolia

Développeur commercial en Alternance - Toute France Inactif

1231 (575)
NOMBRE DE VUES (UNIQUES)

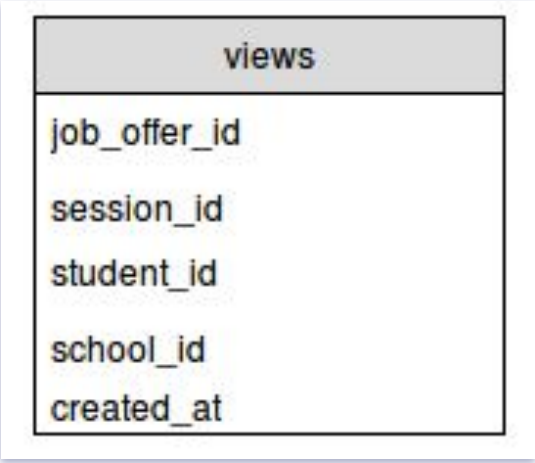
120
NOMBRE DE CLICS UNIQUES



Current situation

algolia

Table schema



Pain points

- 50 millions rows
- Refreshed once a day
- 2 aggregation levels
- Painfully slow (~ 5min)

Query plan

algolia

```
GroupAggregate (cost=12117964.73..12662692.48 rows=161615 width=28) (actual
time=203293.527..294512.778 rows=1515142 loops=1)
Group Key: fom.job_offer_id
Buffers: shared hit=7956 read=1247936, temp read=999414 written=999414

-> Sort (cost=12117964.73..12253742.63 rows=54311160 width=24) (actual
time=203293.371..222702.438 rows=53529386 loops=1)
         Sort Key: fom.job_offer_id
         Sort Method: external merge Disk: 1825640kB
         Buffers: shared hit=7948 read=1247936, temp read=999414 written=999414
         -> Seq Scan on agg.fresh_offer_metrics fom (cost=0.00..1798992.60
rows=54311160 width=24) (actual time=0.667..118503.122 rows=53529386 loops=1)
              Buffers: shared hit=7945 read=1247936

Planning time: 0.920 ms
Execution time: 295166.518 ms
```

First “brilliant” idea

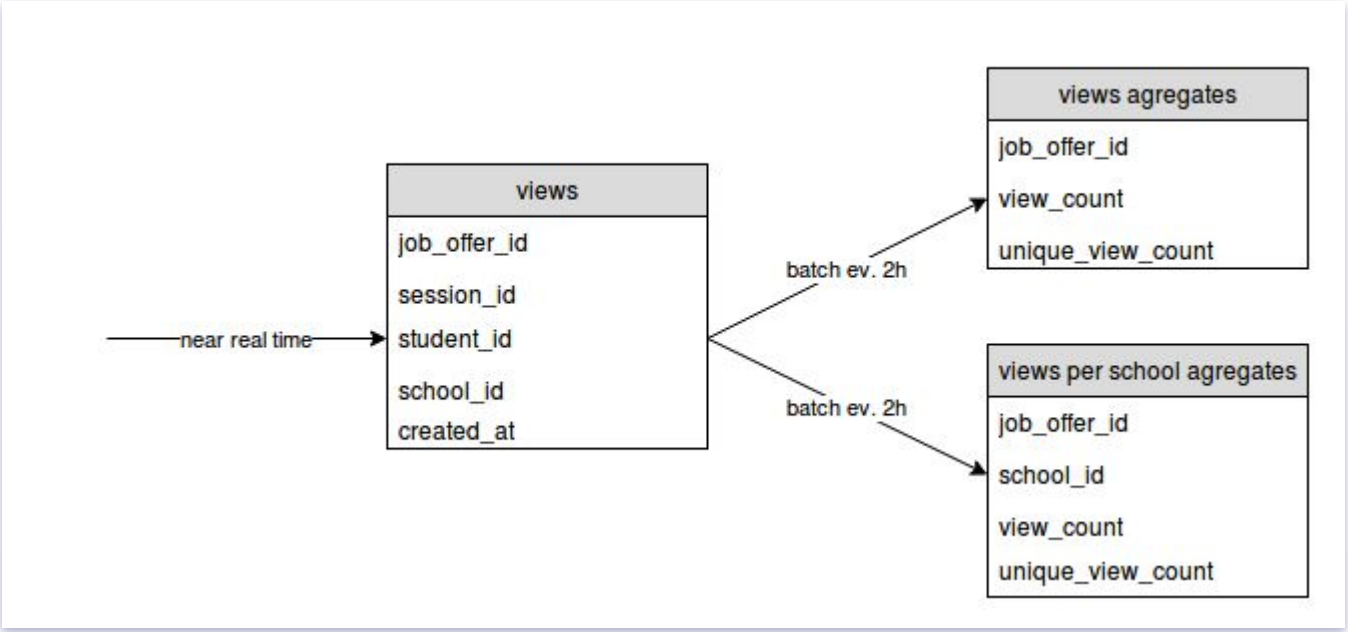
Aggregated table and upsert strategy

Aggregated tables + Upsert strategy

algolia

- 2 aggregates tables : 1 per aggregation level
- Leverage real time data by refreshing every 2h
- Upsert: DELETE + INSERT in a transaction

Aggregated tables + Upsert strategy



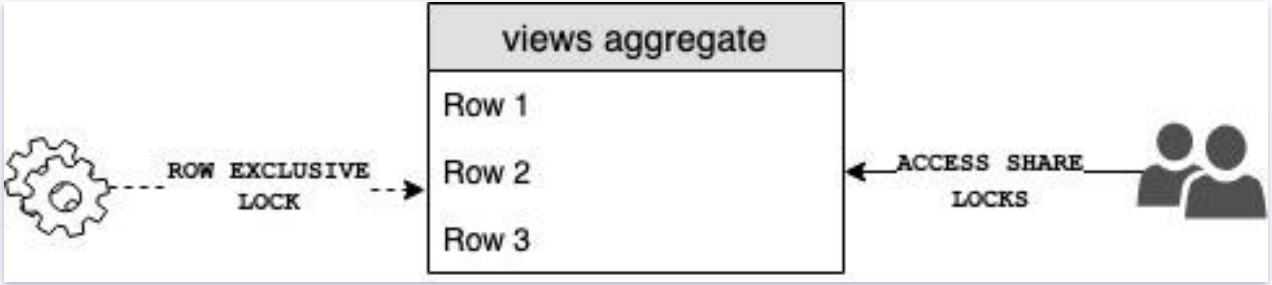
Upsert Strategy

algolia

job_offer_id	view_count	uq_view_count
45	5430	3798
150	66	55

job_offer_id	view_count	uq_view_count
10	999	666
45	5001	3456
150	42	42
320	54321	5112

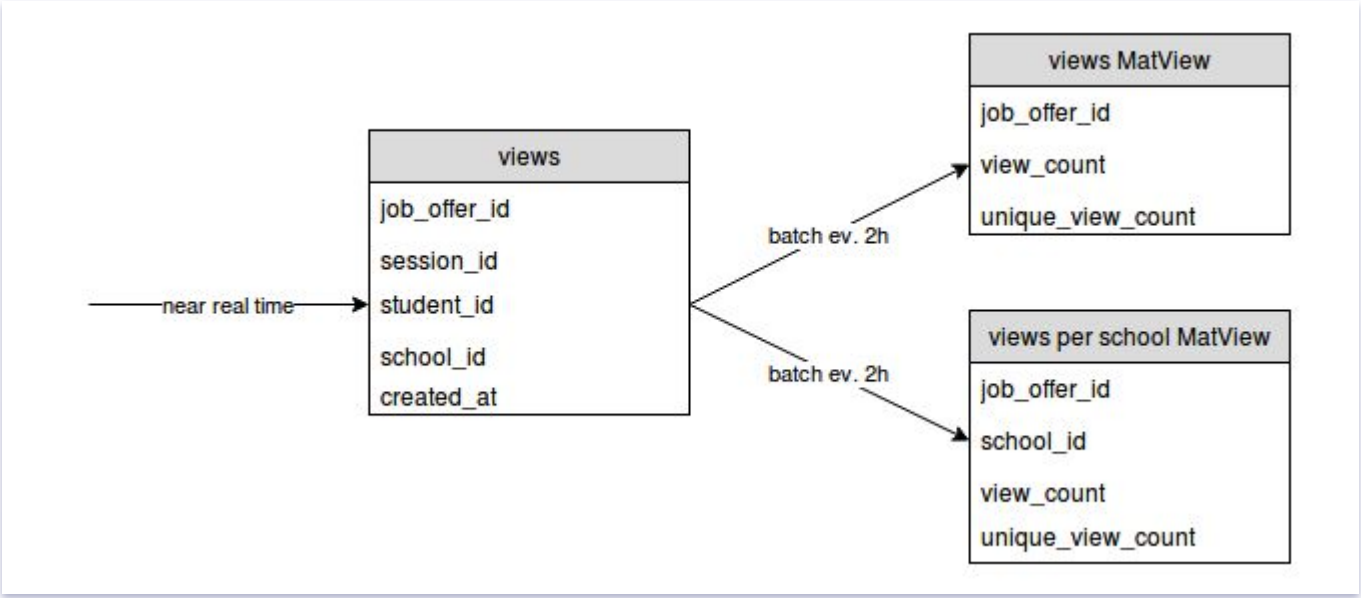
Which leads to locks



Second idea

... poor execution

Same, but with MatViews



Easy as CREATE TABLE AS

algolia

```
CREATE MATERIALIZED VIEW IF NOT EXISTS
job_offer_views_mv AS
SELECT job_offer_id
  , COUNT(*) AS view_count
  , COUNT(distinct session_id) AS unique_view_count
FROM views
GROUP BY job_offer_id;
```

Query plan: without MatView

```
GroupAggregate (cost=12117964.73..12662692.48 rows=161615 width=28) (actual
time=203293.527..294512.778 rows=1515142 loops=1)
Group Key: fom.job_offer_id
Buffers: shared hit=7956 read=1247936, temp read=999414 written=999414

-> Sort (cost=12117964.73..12253742.63 rows=54311160 width=24) (actual
time=203293.371..222702.438 rows=53529386 loops=1)
         Sort Key: fom.job_offer_id
         Sort Method: external merge Disk: 1825640kB
         Buffers: shared hit=7948 read=1247936, temp read=999414 written=999414
         -> Seq Scan on agg.fresh_offer_metrics fom (cost=0.00..1798992.60
rows=54311160 width=24) (actual time=0.667..118503.122 rows=53529386 loops=1)
              Buffers: shared hit=7945 read=1247936

Planning time: 0.920 ms
Execution time: 295166.518 ms
```

Query plan: with MatView

algolia

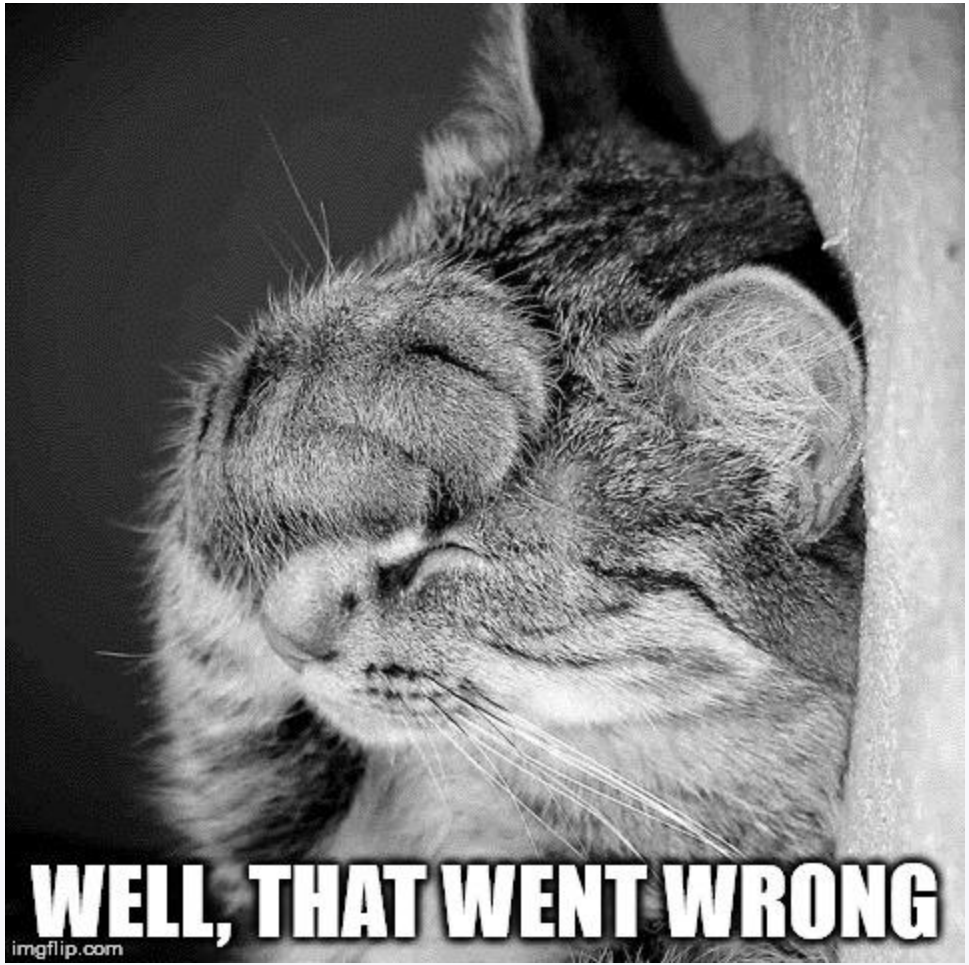
```
QUERY PLAN

------------------------------------------------------------------------------------------
Index Scan using fomv_uq_jo_id on agg.fresh_offer_metrics_view (cost=0.43..8.45
rows=1 width=20) (actual time=1.045..1.045 rows=0 loops=1)

Index Cond: (fresh_offer_metrics_view.job_offer_id = 150)

Buffers: shared hit=2 read=1

Planning time: 0.182 ms
```

Execution time: **1.066 ms**

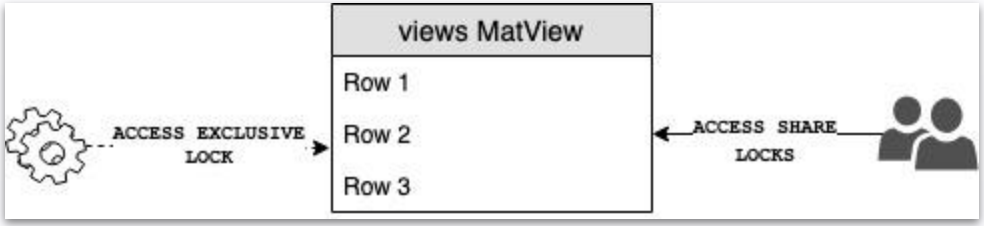
algolia

```
REFRESH MATERIALIZED VIEW job_offer_views_mv;
```

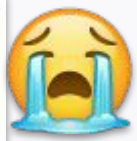
algolia



Locks ! Locks Everywhere



Tip: Only an `ACCESS EXCLUSIVE` lock blocks a `SELECT` (without `FOR UPDATE/SHARE`) statement.

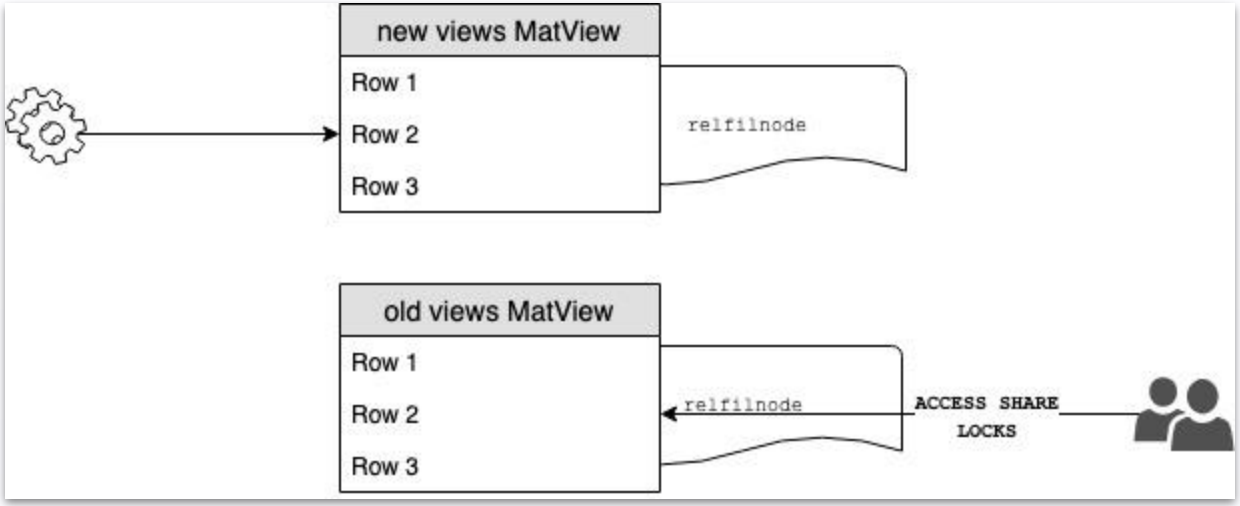


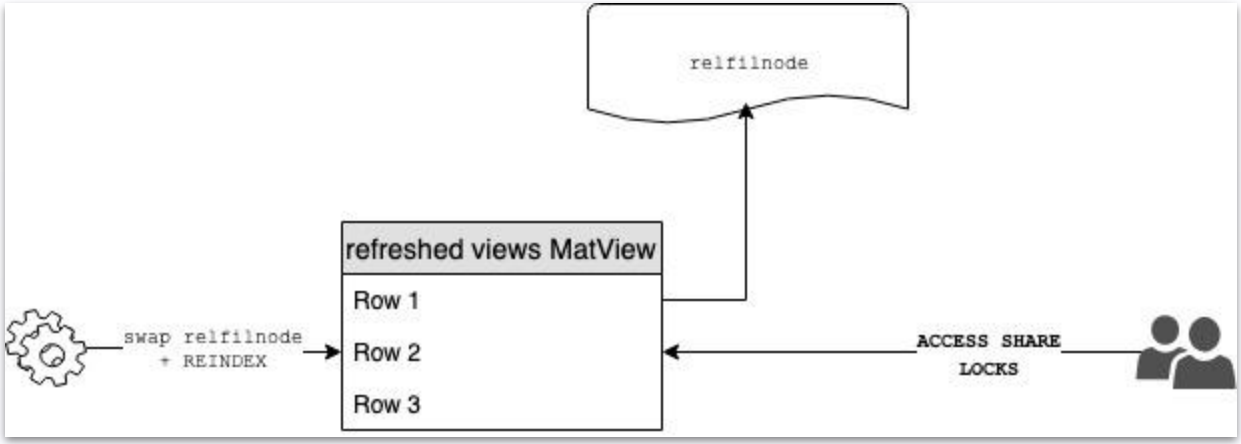
It's all in the code!

algolia

```
/*
* ExecRefreshMatView -- execute a REFRESH MATERIALIZED VIEW command
*
* This refreshes the materialized view by creating a new table and swapping
* the relfilenodes of the new table and the old materialized view, so the OID
* of the original materialized view is preserved. Thus we do not lose GRANT
* nor references to this materialized view.
…
* Indexes are rebuilt too, via REINDEX. Since we are effectively bulk-loading
* the new heap, it's better to create the indexes afterwards than to fill them
* incrementally while we load.
…
* /
    /* Determine strength of lock needed. */
    concurrent = stmt->concurrent;
    lockmode = concurrent ? ExclusiveLock : AccessExclusiveLock;
```

src/backend/commands/matview.c


new views MatView
Row 1
Row 2
Row 3
relfilnode
old views MatView
Row 1
Row 2
Row 3
relfilnode
ACCESS SHARE
LOCKS


relfilnode
refreshed views MatView
Row 1
Row 2
Row 3
swap relfilnode
+ REINDEX
ACCESS SHARE
LOCKS

REFRESH CONCURRENTLY

The magic of CONCURRENTLY

algolia

- Allows concurrent selects statements
- Requires at least one unique index
- Can be faster or slower than simple REFRESH

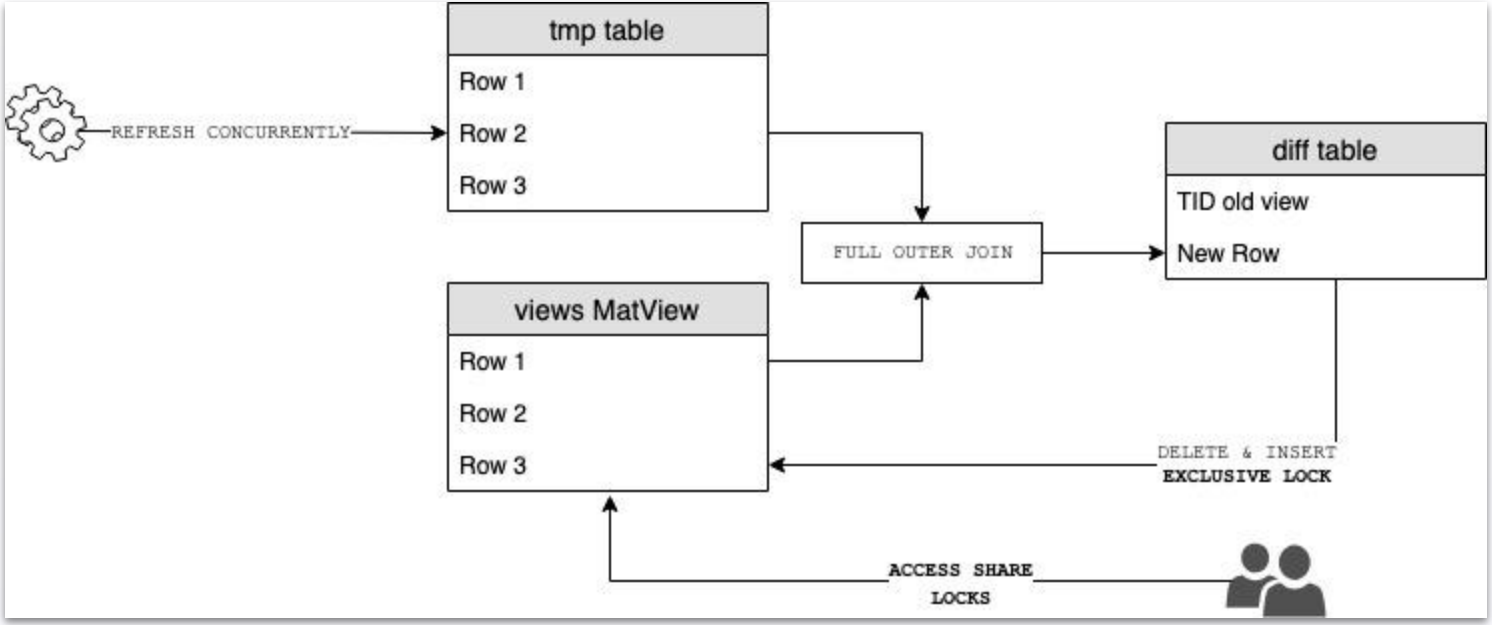
It's all in the code!

algolia

```
if (concurrent)
    …
        refresh_by_match_merge(matviewOid, OIDNewHeap, relowner, save_sec_context);
    else
        refresh_by_heap_swap (matviewOid, OIDNewHeap, relpersistence);
...
/*
* Refresh a materialized view  with transactional semantics, while allowing concurrent reads.
* ...
* It performs a  full outer join against the old version of
* the data, producing "diff" results. This join cannot work if there are any
* duplicated rows in either the old or new versions, in the sense that every
* column would compare as equal between the two rows.
* …
* Once we have the diff table, we perform set-based DELETE and INSERT
* operations against the materialized view, and discard both temporary
* tables.
```

src/backend/commands/matview.c

The magic behind CONCURRENTLY



Retro

algolia

- Carefully read the doc
- Read the code when in doubt, it’s easy!

Wrap-up

Sum-Up

algolia

- MatViews can replace tables by caching slow queries results
- They’re not refreshed automatically
- Be careful with refresh strategy you choose
  - REFRESH requires ACCESS EXCLUSIVE LOCK and replaces the underlying table
  - REFRESH CONCURRENTLY requires a UNIQUE INDEX and proceeds by a diff

Pro tip: Use `pg_cron` extension to refresh the view

```
SELECT cron.schedule('0 10 * * *', 'REFRESH MATERIALIZED VIEW CONCURRENTLY ...');
SELECT cron.unschedule(43);
```

Stephanie Baltus - Sr software engineer

@steph_baltus honest.engineering

algolia