CRUDy OLAP

saving, revising, and slicing large forecasts on the fly

Devi Borg, Formulate by RELEX, 2022-12-01, Stockholm

RELEX

History

2013 or 2016
Formulate
founded

2018
start using
ClickHouse

2020
I join the
team

2021
PLAN
(CRUDy OLAP)

2022
acquired
by
RELEX

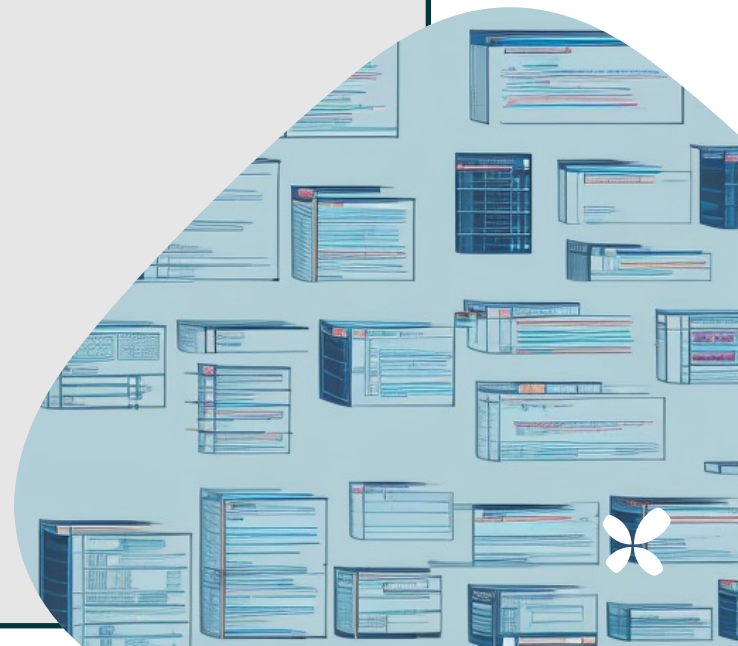
Data ingestion

- CSV/json from retailers to BigQuery
- Analyses use BQ & GCS for I/O
  - Fast models: rerun
  - Pretrained models
    - Coefficients in tables
    - Static config and storage
- Workflows: Cadence & Apache Beam
- Publish to a new set of tables in CH
- Generated queries to CH serve FE

Separation of workflow and online processing

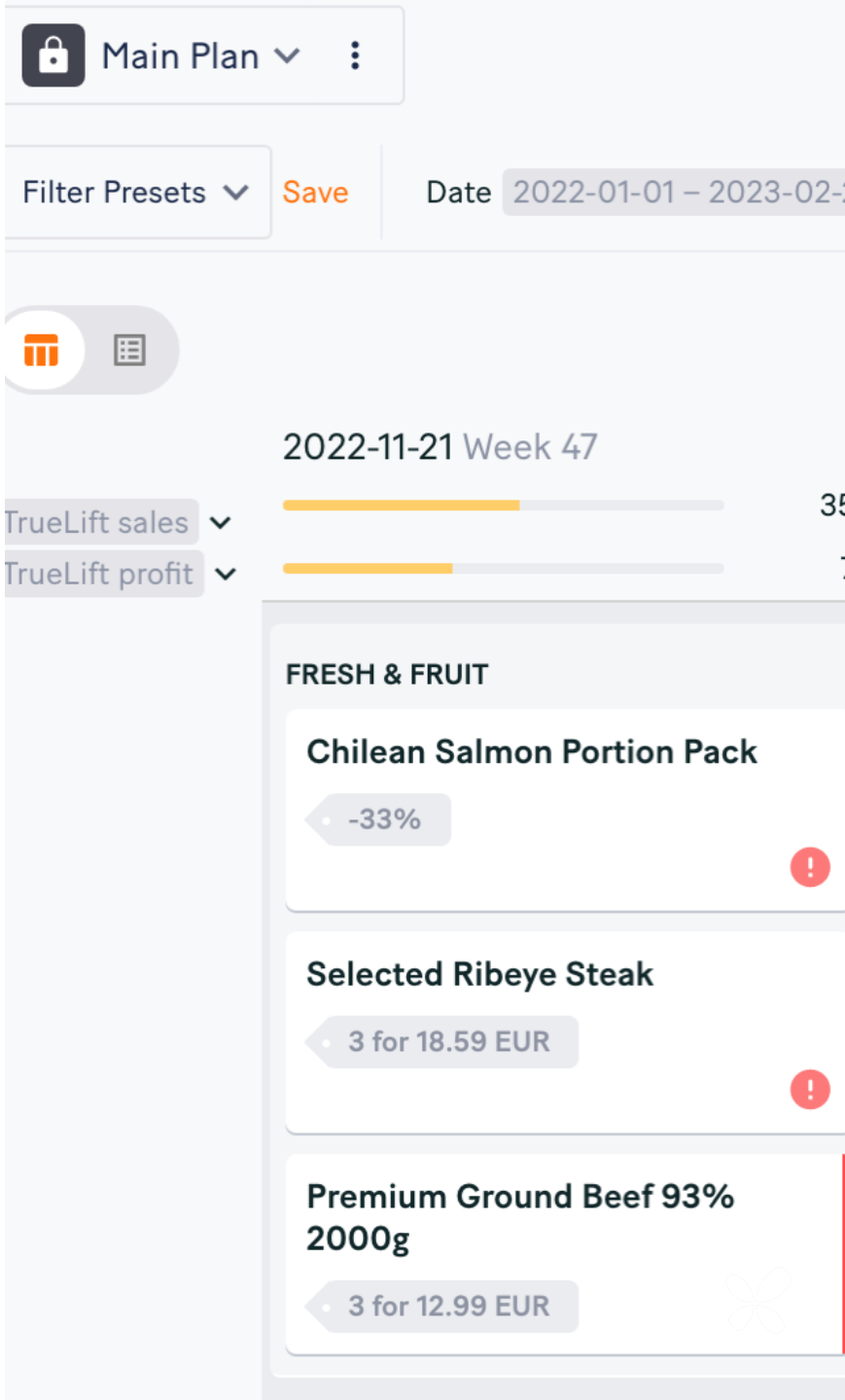
Schema (approximately)

```
CREATE TABLE promotion_RETAILER (
      {item_id, promotion_id, store_id} LowCardinality(String),
      week Date,
      {baseline,sales,gross_lift,net_lift} Float64,
      -- promotion fields: name, displays, offer specification
      -- item/store fields: name, brand, attributes
      -- secondary metrics: cross-effects
) ENGINE = ReplacingMergeTree PARTITION BY tuple()()
ORDER BY (item_id, promotion_id, store_id, week)
; CREATE TABLE promotion_dimension_RETAILER (
      -- data fields to recreate promotion
) … ORDER BY promotion_id
```



PLAN module

- Move batch→online scenario planning
- Replicate analyses without workflows
- Dual challenge of model: Input & Output
- Input: different data made available in CH
- Output: support all CRUD operations
- Focus on Output part in this talk
- Extra field `plan_id` part of all filters



Our CRUDy OLAP problem

- Many rows (1000's) for each upsert
- Update different tables
- Atomic data visibility on return
- (reduce dependence on CH atomicity)
- Want to keep schema for querying
- Keep data across weekly runs
- Conclusion: similar table + tricks
- Bonus: keep track of history

Trick 1: append only

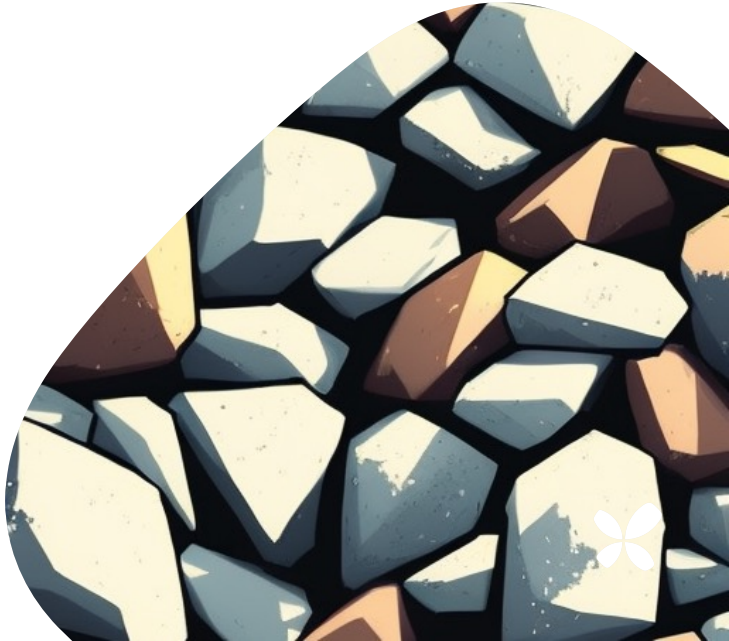
- Common and excellent pattern
- Add `version Int64` field
- Filter `version=MAX(version) OVER (key)`
- On inserts use `version = MAX(version)+1`
- Concurrent updates could get same version
- Relies on ClickHouse atomicity
- Scanning whole table

Trick 2: postgres sequence

- Concurrent updates
- Local locks incompatible with scaling instances
- postgres sequence for versions
- postgres locks to avoid concurrency
  - Take lock on row corresponding to this plan
- Hold lock for writing only, not reading or forecast

Trick 3: versions KV-table

- Started with map `promotion_id` to `version`
- Queries don’t need to search for `MAX(version)`
- Aborted partial writes ignored
- Delete by setting `version = 0`
- Later redesigned: also key by plan_id
  - Allows moving/copying promotions between plans
  - More metadata



Querying combined tables

```
(
      SELECT * FROM promotion_1
      WHERE historic
      UNION ALL
      SELECT * EXCEPT(version, plan_id)
      FROM promotion_plan_1
      WHERE
      ((plan_id, promotion_id), version) IN
            (SELECT (key, version)
            FROM promotion_versions_1)
      AND plan_id=?
) AS promotion
```

Functions to get table names expanded

RELEX

Thank you!

(ps we're hiring)

Devi Borg
devi.borg@relexsolutions.com
Robin Bartholdson
robin.bartholdson@relexsolutions.com

www.relexsolutions.com @RelexSolutions RELEX Solutions

Extras

Querying extras

- Querying evolution
  - Started off getting metrics from graphQL
  - Moved on to custom tabular API aggregating along different dimensions
- Filtering: builder types in go
  - Common filter input type generated in graphQL
  - Different functions for different dimensions
  - Refactor to use dimension tables to filter each separately
- Avoid using `SELECT *` due to schema updates
  - Dynamically generate expressions from schema type
  - When adding a new field, always add default expressions
  - Use `hasColumnInTable` to switch between default and stored value