1027 predictive models in 10 seconds

A journey of discovery and astonishment

David Pardo - Corunet

Who?

- David Pardo @dei_biz
- Co-owner @corunet. 43 people.
- Data and metadata management for retail sector
- You haven’t heard of us…
- …but we’ve worked with your data

The problem

- 250 million records / 60GB
- Up to seven variables (country, family, type, brand...)
- one week bid
- exploration vs. production
- no big data. Neither small...

We know what we need. Answer two questions:

- What should have happened yesterday? (Anomalies)
- What's going to happen within two weeks? (Forecast)

The easy part.

Experience with temporal series

Weka denseInstances

We (think we) know the variables:

- country
- dayOfYear
- dayOfWeek
- daysFromSales/daysToSales
- specialDays (BF/Singles day/...)

The easy part. Weka

We only need a few thousand files like this to feed the model generator:

```
[
        {“date”:”2016-01-01”, “sales”: 7},
        {“date”:”2016-01-02”, “sales”: 11},
        …
]
```

By type, family, buyer, country. How many? Good question

250 million records? You can solve that with a few indexes

*Spoiler: you can’t

When you've got a hammer…

Let's import data into postgres and query it. Copy is fast, isn't it?

```
copy sales ("id","time","country"...) from 'd:\tmp\data.csv' DELIMITER ',' CSV HEADER;
```

Single transaction. No way.

Cut it in batches

<20K insert/s

It's gonna take quite a few hours… can we try something meanwhile?

We've got RAM, let's put it to use

- Python + pandas
- Spark
- Apache beam?
- Hadoop + parquet
- ...
- ¯_(ツ)_/¯

clickwhat?

Let’s have a coffee while spark counts rows… wait… Somebody told me there was a new columnar database

```
deb http://repo.yandex.ru/clickhouse/deb/stable/main/
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv E0C56BD4
sudo apt-get update
sudo apt-get install clickhouse-client clickhouse-server
```

It works!

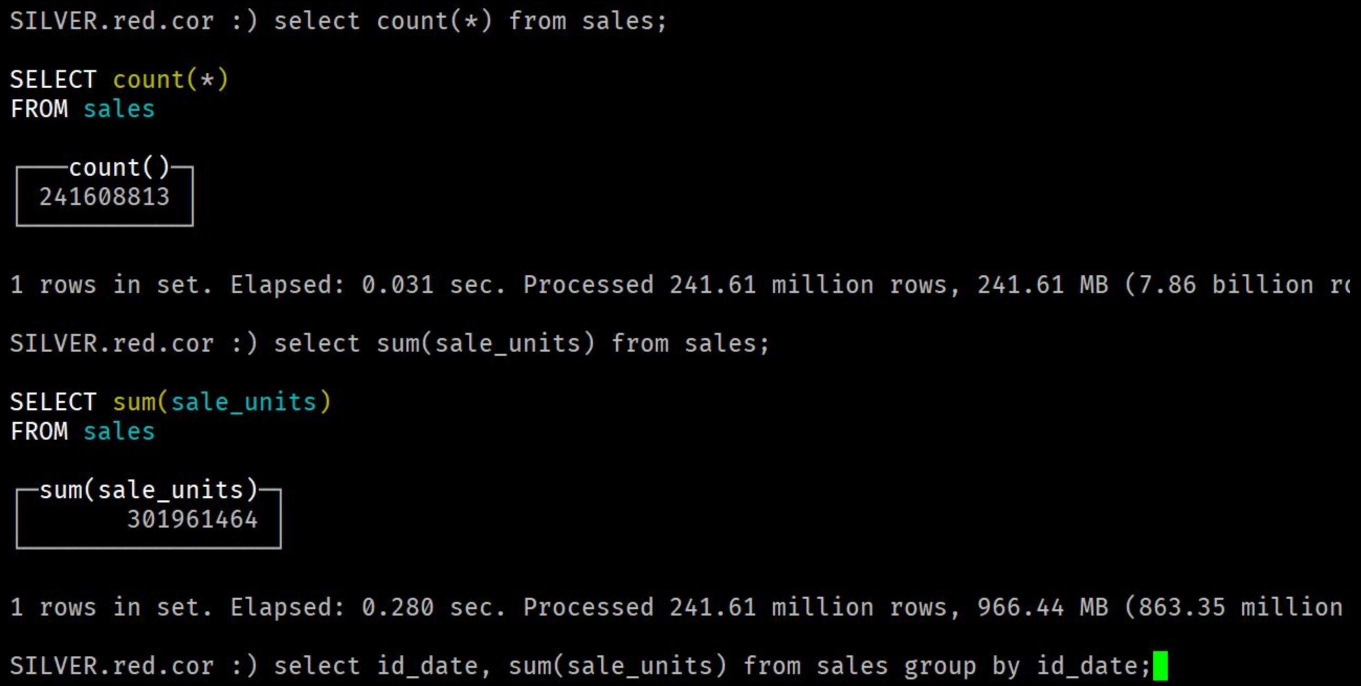
Importing CSV data

```
CREATE TABLE default.sales
(
    id_date Date,
    time String,
    country Int32,
    country_name String,
    country_iso String,
    wh_code Int32,
    wh_name String,
    category UInt32,
    sku String,
    section_code UInt8,
    product_type String,
    product_name String,
    color UInt8,
    sale_units UInt32,
    sale_amount Float32,
    product_family_code UInt8,
    product_family_name String,
    product_class String,
    product_class_name String,
    product_line String,
    piece UInt8
)
ENGINE = MergeTree(id_date, (id_date, product_family_code, sku), 8192)
```

```
tail -n +2 data.csv | time clickhouse-client --query="INSERT INTO ventas FORMAT CSV"
```

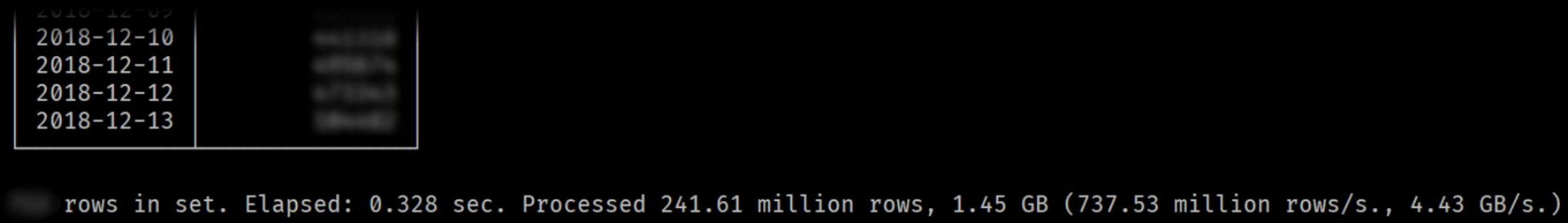
2:37.82s elapsed

You had my curiosity
Now you have my attention



```
SILVER.red.cor :) select count(*) from sales;

SELECT count(*)
FROM sales

┌─count()─┐
│ 241608813 │
└───────────┘

1 rows in set. Elapsed: 0.031 sec. Processed 241.61 million rows, 241.61 MB (7.86 billion ro

SILVER.red.cor :) select sum(sale_units) from sales;

SELECT sum(sale_units)
FROM sales

┌─sum(sale_units)─┐
│       301961464 │
└─────────────────┘

1 rows in set. Elapsed: 0.280 sec. Processed 241.61 million rows, 966.44 MB (863.35 million

SILVER.red.cor :) select id_date, sum(sale_units) from sales group by id_date;
```

What?



```
│  2018-12-09  │   [redacted]  │
│  2018-12-10  │   [redacted]  │
│  2018-12-11  │   [redacted]  │
│  2018-12-12  │   [redacted]  │
│  2018-12-13  │   [redacted]  │
└──────────────┴───────────────┘

[redacted] rows in set. Elapsed: 0.328 sec. Processed 241.61 million rows, 1.45 GB (737.53 million rows/s., 4.43 GB/s.)
```

0.328s
Out of the box
One node. No configuration

How many models?

```sql
SELECT
    country_name,
    section_code,
    product_family_name,
    product_line
FROM sales
GROUP BY
    country_name,
    section_code,
    product_family_name,
    product_line
```

```
 Showed first 10000.

15516 rows in set. Elapsed: 4.479 sec.
```

Way too many. Let's reduce it a bit

```
SELECT
    sum(sale_units) AS total,
    country_name,
    section_code,
    product_family_name,
    product_line
FROM sales
GROUP BY
    country_name,
    section_code,
    product_family_name,
    product_line
HAVING sum(sale_units) > [redacted]
ORDER BY total DESC
```

So, 1027 queries:

```
SELECT
    sum(sale_units) AS total,
    country_name,
    section_code,
    product_family_name,
    product_line
FROM sales
GROUP BY
    country_name,
    section_code,
    product_family_name,
    product_line
HAVING sum(sale_units) > [redacted]
ORDER BY total DESC
```

```
1027 rows in set. Elapsed: 4.708 sec. Processed 241.61 million row
```

Good enough. We can work it out!

- First we get all possible combinations of (COUNTRY, SECTION, FAMILY, LINE)
- We grab the data GROUPed BY date for each one
- Then we feed the dense model generator
- 1027+1 queries (100% CPU) @10s/query -> ~3 hours + latency
- Total time (5 h) with chart generation for the most complex series. Good enough

It's alive!

Thank you?

But you said 10 seconds…

1027*713 = 732.251 rows

- How long would it take to get the full set?

The full query

```
SELECT
    country_name_corrected,
    section_code AS section,
    product_line AS product_line,
    product_family_name AS family,
    id_date AS date,
    sales AS sales,
    total_units AS section_sales,
    sales / total_units AS share
FROM
(
    SELECT
        country,
        section_code,
        product_line,
        product_family_name,
        id_date,
        sum(sale_units) AS sales
    FROM sales
    GROUP BY
        country,
        section_code,
        product_line,
        product_family_name,
        id_date
) AS per_family
ANY INNER JOIN
(
    SELECT
        country,
        multiIf(country = 1, 'XXX', country = 2, 'YYY', country = 7, 'ZZZ', country = 437, 'AAA', country = 599, 'TTT', country_name) AS country_name_corrected,
        section_code,
        id_date,
        sum(sale_units) AS total_units
    FROM sales
    GROUP BY
        country,
        country_name_corrected,
        section_code,
        id_date
) AS per_section USING (country, section_code, id_date)
ORDER BY id_date ASC
```

One million rows

```
 Showed first 10000.

794980 rows in set. Elapsed: 9.799 sec. Processed 483.22 million rows, 16.92 GB (49.32 million rows/s., 1.73 GB/s.)
```

The results:

The results:

Conclusions

- Better than dataframes for data exploration
- One size fits all. DEV & PRO
- Perfect for dense model generation
- If I wasn’t already married, I’d marry it.

Thank you!

@dei_biz. DMs open