Hunting Non-Optimized Queries in ClickHouse

Strategies to find the needles in the haystack of ClickHouse queries

Yohann Jardin

Yohann Jardin

Lead Data Engineer

Optimization mindset (tech + DevEx)
Lead dev of a ClickHouse client in Scala
Top 3 researcher on CH Bug Bounty program

yohannjardin

twitch.tv/
amendile

yohannj

Amendil#6077

How do we end up hunting non-optimized queries?

Many services and users

Diverse maturity of using ClickHouse

Load on ClickHouse changes throughout the day

Scope

In scope:

- SELECT queries

Out of scope:

- Materialized views
- Dictionary loading

Dataset

System.query_log - interesting fields

```
CREATE TABLE system.query_log
(
    `type` Enum8('QueryStart' = 1, 'QueryFinish' = 2, 'ExceptionBeforeStart' = 3,
'ExceptionWhileProcessing' = 4),
    `event_date` Date,
    `event_time` DateTime,
    `query_duration_ms` UInt64,
    `read_rows` UInt64,
    `memory_usage` UInt64,
    `query` String,
    `log_comment` String,
    `ProfileEvents` Map(LowCardinality(String), UInt64),
    `ProfileEvents.Names` Array(LowCardinality(String)) ALIAS mapKeys(ProfileEvents),
    `ProfileEvents.Values` Array(UInt64) ALIAS mapValues(ProfileEvents),
    ...
)
```

System.query_log - log_comment

Initially created for clickhouse-test.

String of up to `max_query_size` characters.

```
SELECT 'Hello Singapore!' AS msg
SETTINGS log_comment = 'origin=webserver;endpoint_name=/
hello;table=none;nested_subqueries_depth=0;country_count=1'

   ┌─msg──────────┐
1. │ Hello Singapore! │
   └──────────────┘


--

SELECT
    toUInt16(extract(log_comment, ';country_count=(\\d+)')) AS country_count,
    extract(log_comment, ';table=([^;]+);') AS `table`,
    extract(log_comment, '^origin=([^;]+)') AS origin,
    extract(log_comment, ';endpoint_name=([^;]+)') AS endpoint_name
FROM system.query_log
WHERE (query_id = '354785f9-9f2c-40d3-92ac-6c9cfa159e9c') AND (type = 'QueryFinish')

   ┌─country_count─┬─table─┬─origin────┬─endpoint_name─┐
1. │             1 │ none  │ webserver │ /hello        │
   └───────────────┴───────┴───────────┴───────────────┘
```

1st strategy: Manual analysis

Manual analysis

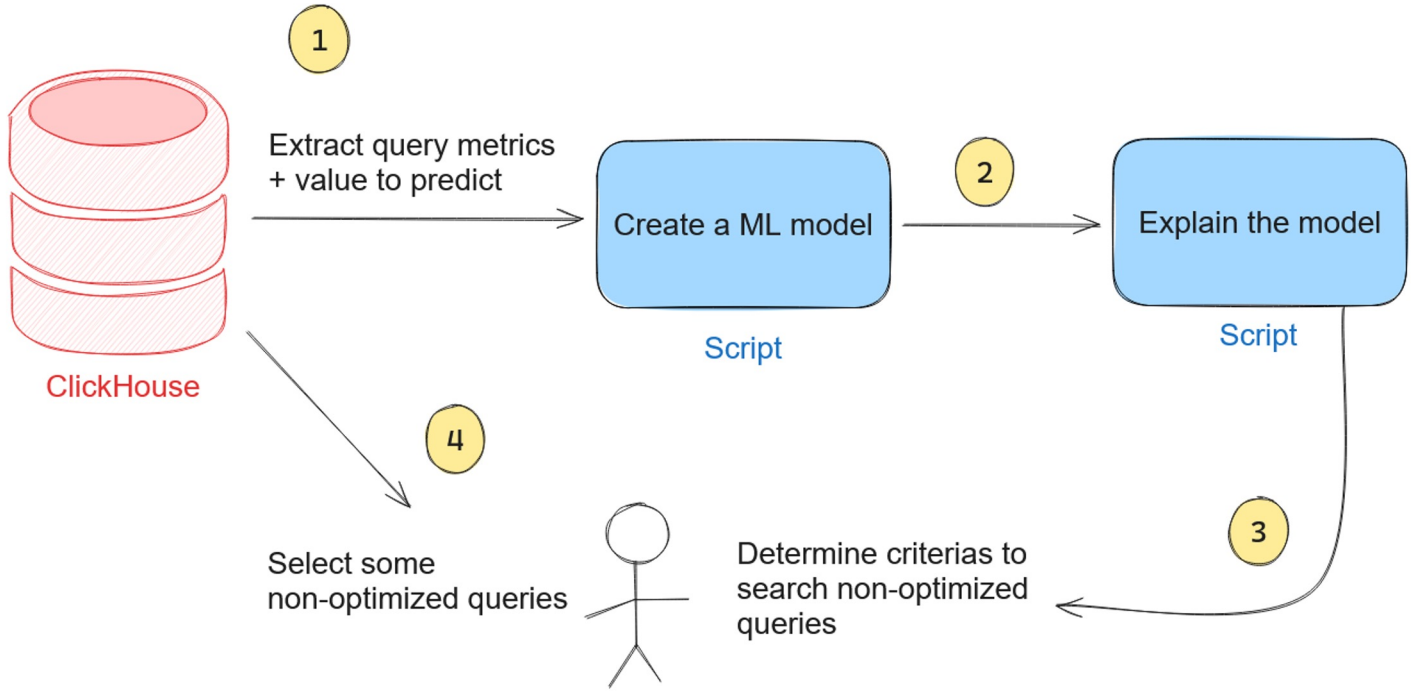
Top queries by CPU, Memory, Network

```
SELECT
    type,
    event_time,
    initial_query_id,
    query_id,
    formatReadableSize(memory_usage) AS memory,
    ProfileEvents.Values[
      indexOf(ProfileEvents.Names, 'UserTimeMicroseconds')
    ] AS userCPU,
    ProfileEvents.Values[
      indexOf(ProfileEvents.Names, 'SystemTimeMicroseconds')
    ] AS systemCPU,
    ProfileEvents.Values[
      indexOf(ProfileEvents.Names, 'NetworkReceiveElapsedMicroseconds')
    ] AS networkReceived,
    ProfileEvents.Values[
      indexOf(ProfileEvents.Names, 'NetworkSendElapsedMicroseconds')
    ] AS networkSend,
    normalizedQueryHash(query) AS normalized_query_hash
FROM clusterAllReplicas(default, system.query_log)
ORDER BY memory_usage DESC
LIMIT 10;
```

Cf https://clickhouse.com/docs/knowledgebase/find-expensive-queries
+ adding network metrics

2nd strategy: Manual analysis assisted by Machine Learning

General Idea



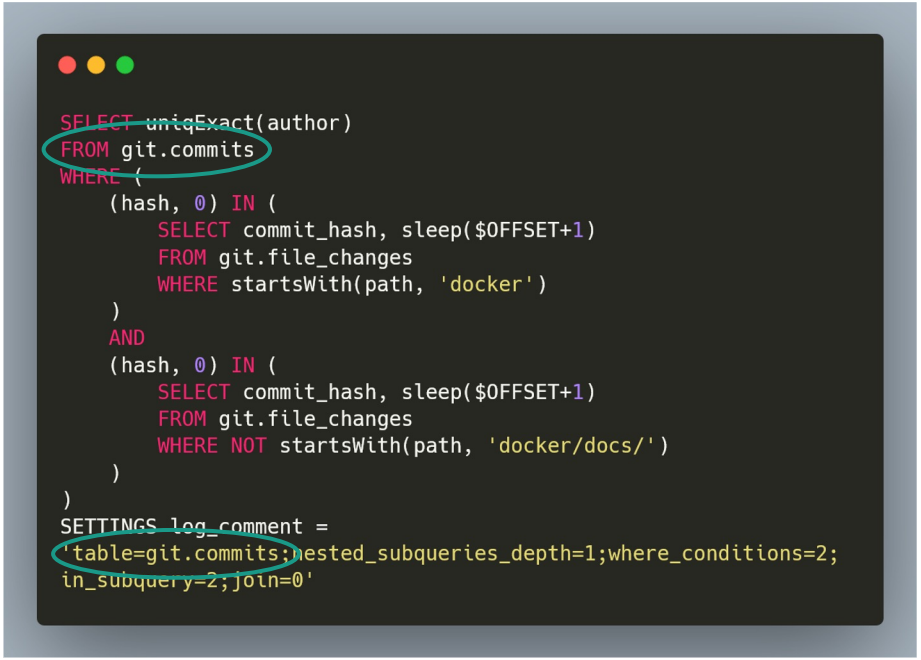
Scenario

example-datasets/github

11 different queries, each executed 100 times

4 metrics in log_comment

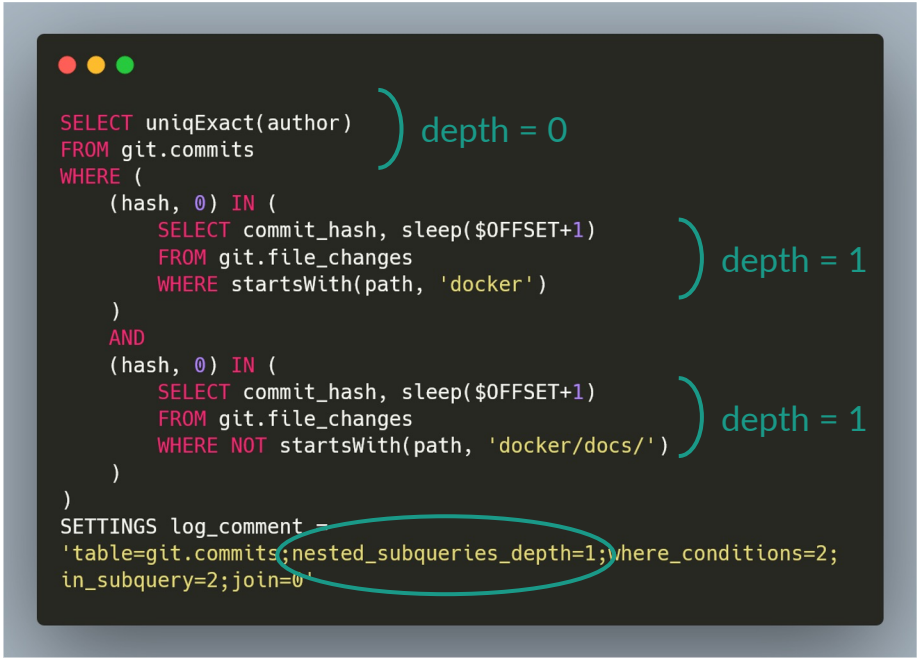
Query example and metrics



```
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE startsWith(path, 'docker')
    )
    AND
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE NOT startsWith(path, 'docker/docs/')
    )
)
SETTINGS log_comment =
'table=git.commits;nested_subqueries_depth=1;where_conditions=2;
in_subquery=2;join=0'
```

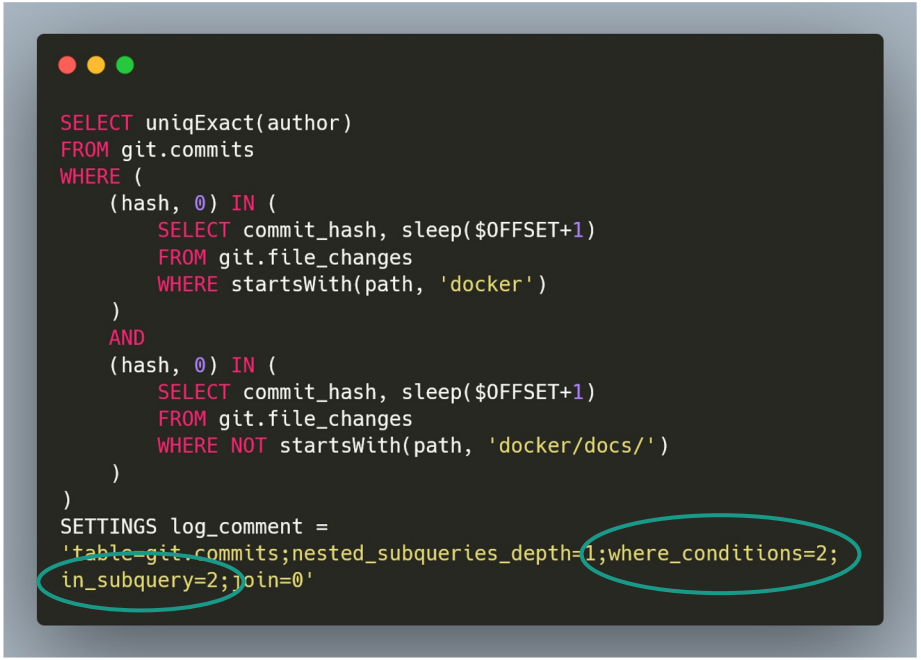
Our context, the main table, are commits of the ClickHouse repository.

Query example and metrics



Maybe nesting queries impacts performances?

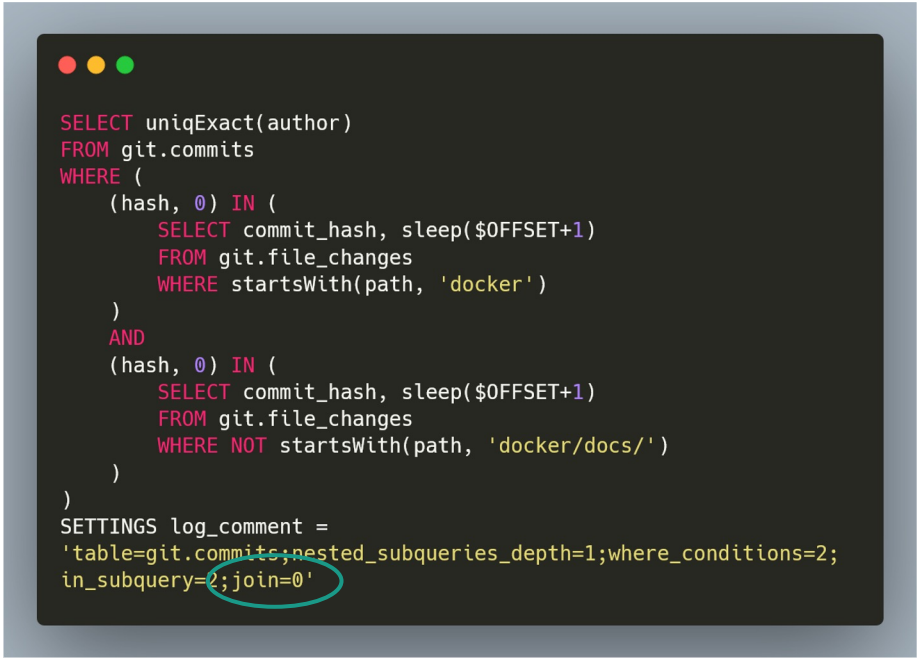
Query example and metrics



```
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE startsWith(path, 'docker')
    )
    AND
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE NOT startsWith(path, 'docker/docs/')
    )
)
SETTINGS log_comment =
'table=git.commits;nested_subqueries_depth=1;where_conditions=2;
in_subquery=2;join=0'
```

2 conditions in the main queries, both are “... IN ...” conditions

Query example and metrics



```
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE startsWith(path, 'docker')
    )
    AND
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE NOT startsWith(path, 'docker/docs/')
    )
)
SETTINGS log_comment =
'table=git.commits;nested_subqueries_depth=1;where_conditions=2;
in_subquery=2;join=0'
```

Maybe (GLOBAL LEFT, RIGHT, ...)
JOIN impacts performances?

Query example and metrics

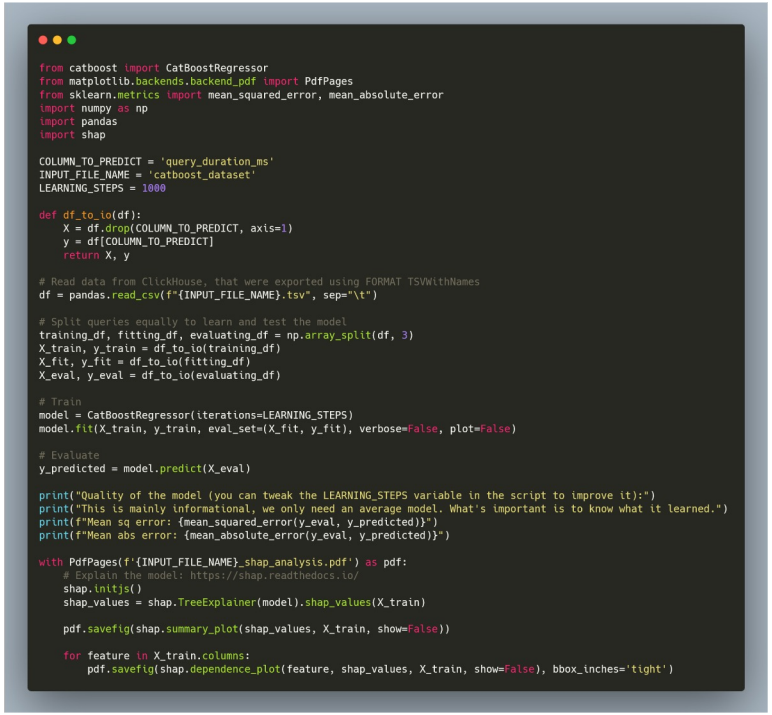
```
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE startsWith(path, 'docker')
    )
    AND
    (hash, 0) IN (
        SELECT commit_hash, sleep($OFFSET+1)
        FROM git.file_changes
        WHERE NOT startsWith(path, 'docker/docs/')
    )
)
SETTINGS log_comment =
'table=git.commits;nested_subqueries_depth=1;where_conditions=2;
in_subquery=2;join=0'
```

Artificially represents querying a huge table (file_changes) using `sleep()`.

Download dataset

```
SELECT
    extract(log_comment, ';nested_subqueries_depth=(\\d+)') AS
nested_subqueries_depth,
    extract(log_comment, ';where_conditions=([^;]+);') AS where_conditions,
    extract(log_comment, ';in_subquery=([^;]+);') AS in_subquery,
    extract(log_comment, ';join=([^;]+)$') AS join,
    query_duration_ms
FROM system.query_log
WHERE type = 'QueryFinish'
AND extract(log_comment, '^table=([^;]+)') == 'git.commits'
INTO OUTFILE 'catboost_dataset.tsv'
FORMAT TSVWithNames
```

Learning & Explaining



```python
from catboost import CatBoostRegressor
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import pandas
import shap

COLUMN_TO_PREDICT = 'query_duration_ms'
INPUT_FILE_NAME = 'catboost_dataset'
LEARNING_STEPS = 1000

def df_to_io(df):
    X = df.drop(COLUMN_TO_PREDICT, axis=1)
    y = df[COLUMN_TO_PREDICT]
    return X, y

# Read data from ClickHouse, that were exported using FORMAT TSVWithNames
df = pandas.read_csv(f"{INPUT_FILE_NAME}.tsv", sep="\t")

# Split queries equally to learn and test the model
training_df, fitting_df, evaluating_df = np.array_split(df, 3)
X_train, y_train = df_to_io(training_df)
X_fit, y_fit = df_to_io(fitting_df)
X_eval, y_eval = df_to_io(evaluating_df)

# Train
model = CatBoostRegressor(iterations=LEARNING_STEPS)
model.fit(X_train, y_train, eval_set=(X_fit, y_fit), verbose=False, plot=False)

# Evaluate
y_predicted = model.predict(X_eval)

print("Quality of the model (you can tweak the LEARNING_STEPS variable in the script to improve it):")
print("This is mainly informational, we only need an average model. What's important is to know what it learned.")
print(f"Mean sq error: {mean_squared_error(y_eval, y_predicted)}")
print(f"Mean abs error: {mean_absolute_error(y_eval, y_predicted)}")

with PdfPages(f'{INPUT_FILE_NAME}_shap_analysis.pdf') as pdf:
    # Explain the model: https://shap.readthedocs.io/
    shap.initjs()
    shap_values = shap.TreeExplainer(model).shap_values(X_train)

    pdf.savefig(shap.summary_plot(shap_values, X_train, show=False))

    for feature in X_train.columns:
        pdf.savefig(shap.dependence_plot(feature, shap_values, X_train, show=False), bbox_inches='tight')
```

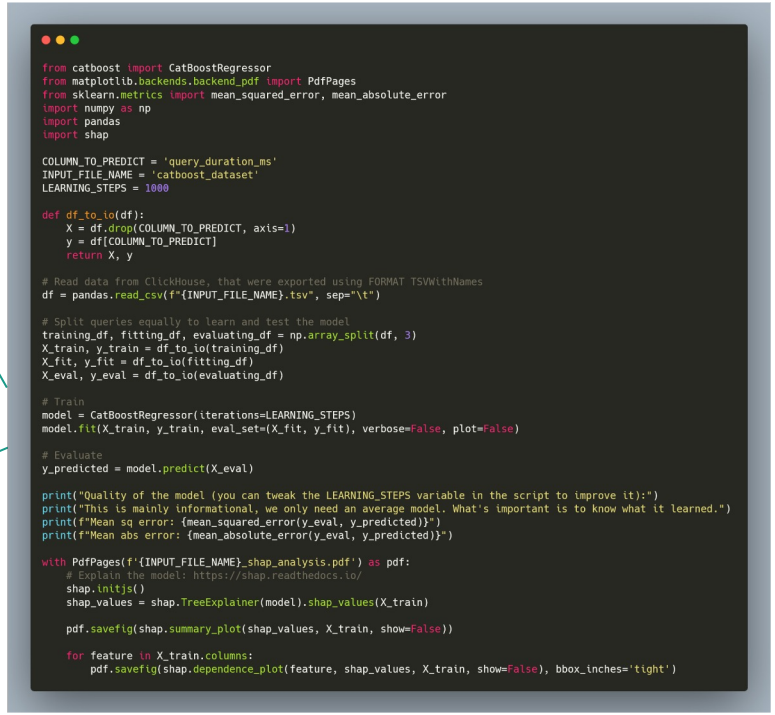
Learning & Explaining

```
# Read data from ClickHouse, that were exported using FORMAT TSVWithNames
df = pandas.read_csv(f"{INPUT_FILE_NAME}.tsv", sep="\t")

# Split queries equally to learn and test the model
training_df, fitting_df, evaluating_df = np.array_split(df, 3)
X_train, y_train = df_to_io(training_df)
X_fit, y_fit = df_to_io(fitting_df)
X_eval, y_eval = df_to_io(evaluating_df)
```

```
from catboost import CatBoostRegressor
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import pandas
import shap

COLUMN_TO_PREDICT = 'query_duration_ms'
INPUT_FILE_NAME = 'catboost_dataset'
LEARNING_STEPS = 1000

def df_to_io(df):
    X = df.drop(COLUMN_TO_PREDICT, axis=1)
    y = df[COLUMN_TO_PREDICT]
    return X, y

# Read data from ClickHouse, that were exported using FORMAT TSVWithNames
df = pandas.read_csv(f"{INPUT_FILE_NAME}.tsv", sep="\t")

# Split queries equally to learn and test the model
training_df, fitting_df, evaluating_df = np.array_split(df, 3)
X_train, y_train = df_to_io(training_df)
X_fit, y_fit = df_to_io(fitting_df)
X_eval, y_eval = df_to_io(evaluating_df)

# Train
model = CatBoostRegressor(iterations=LEARNING_STEPS)
model.fit(X_train, y_train, eval_set=(X_fit, y_fit), verbose=False, plot=False)

# Evaluate
y_predicted = model.predict(X_eval)

print("Quality of the model (you can tweak the LEARNING_STEPS variable in the script to improve it):")
print("This is mainly informational, we only need an average model. What's important is to know what it learned.")
print(f"Mean sq error: {mean_squared_error(y_eval, y_predicted)}")
print(f"Mean abs error: {mean_absolute_error(y_eval, y_predicted)}")

with PdfPages(f'{INPUT_FILE_NAME}_shap_analysis.pdf') as pdf:
    # Explain the model: https://shap.readthedocs.io/
    shap.initjs()
    shap_values = shap.TreeExplainer(model).shap_values(X_train)

    pdf.savefig(shap.summary_plot(shap_values, X_train, show=False))

    for feature in X_train.columns:
        pdf.savefig(shap.dependence_plot(feature, shap_values, X_train, show=False), bbox_inches='tight')
```

Learning & Explaining

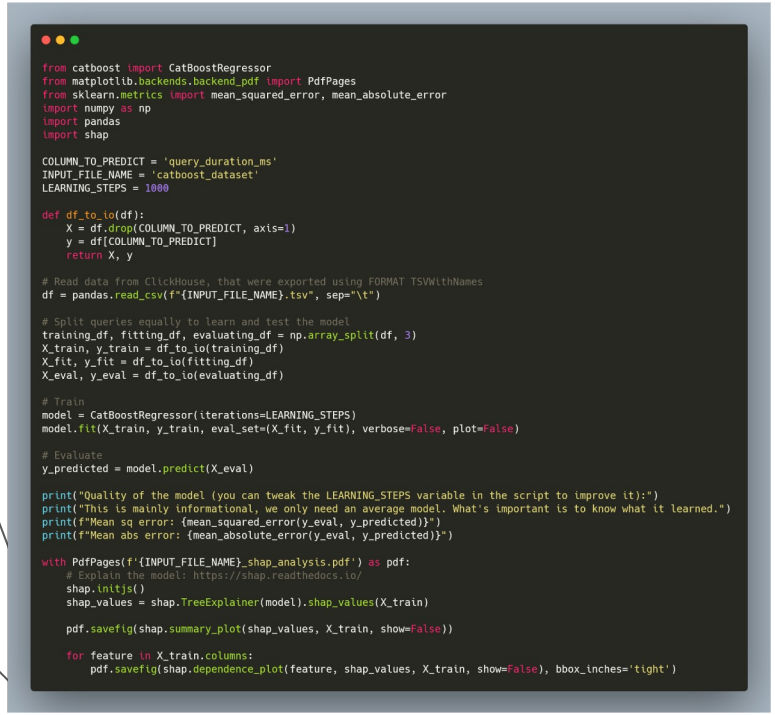
```
# Train
model = CatBoostRegressor(iterations=LEARNING_STEPS)
model.fit(
  X_train,
  y_train,
  eval_set=(X_fit, y_fit),
  verbose=False,
  plot=False
)
```



```
from catboost import CatBoostRegressor
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import pandas
import shap

COLUMN_TO_PREDICT = 'query_duration_ms'
INPUT_FILE_NAME = 'catboost_dataset'
LEARNING_STEPS = 1000

def df_to_io(df):
    X = df.drop(COLUMN_TO_PREDICT, axis=1)
    y = df[COLUMN_TO_PREDICT]
    return X, y

# Read data from ClickHouse, that were exported using FORMAT TSVWithNames
df = pandas.read_csv(f"{INPUT_FILE_NAME}.tsv", sep="\t")

# Split queries equally to learn and test the model
training_df, fitting_df, evaluating_df = np.array_split(df, 3)
X_train, y_train = df_to_io(training_df)
X_fit, y_fit = df_to_io(fitting_df)
X_eval, y_eval = df_to_io(evaluating_df)

# Train
model = CatBoostRegressor(iterations=LEARNING_STEPS)
model.fit(X_train, y_train, eval_set=(X_fit, y_fit), verbose=False, plot=False)

# Evaluate
y_predicted = model.predict(X_eval)

print("Quality of the model (you can tweak the LEARNING_STEPS variable in the script to improve it):")
print("This is mainly informational, we only need an average model. What's important is to know what it learned.")
print(f"Mean sq error: {mean_squared_error(y_eval, y_predicted)}")
print(f"Mean abs error: {mean_absolute_error(y_eval, y_predicted)}")

with PdfPages(f'{INPUT_FILE_NAME}_shap_analysis.pdf') as pdf:
    # Explain the model: https://shap.readthedocs.io/
    shap.initjs()
    shap_values = shap.TreeExplainer(model).shap_values(X_train)

    pdf.savefig(shap.summary_plot(shap_values, X_train, show=False))

    for feature in X_train.columns:
        pdf.savefig(shap.dependence_plot(feature, shap_values, X_train, show=False), bbox_inches='tight')
```

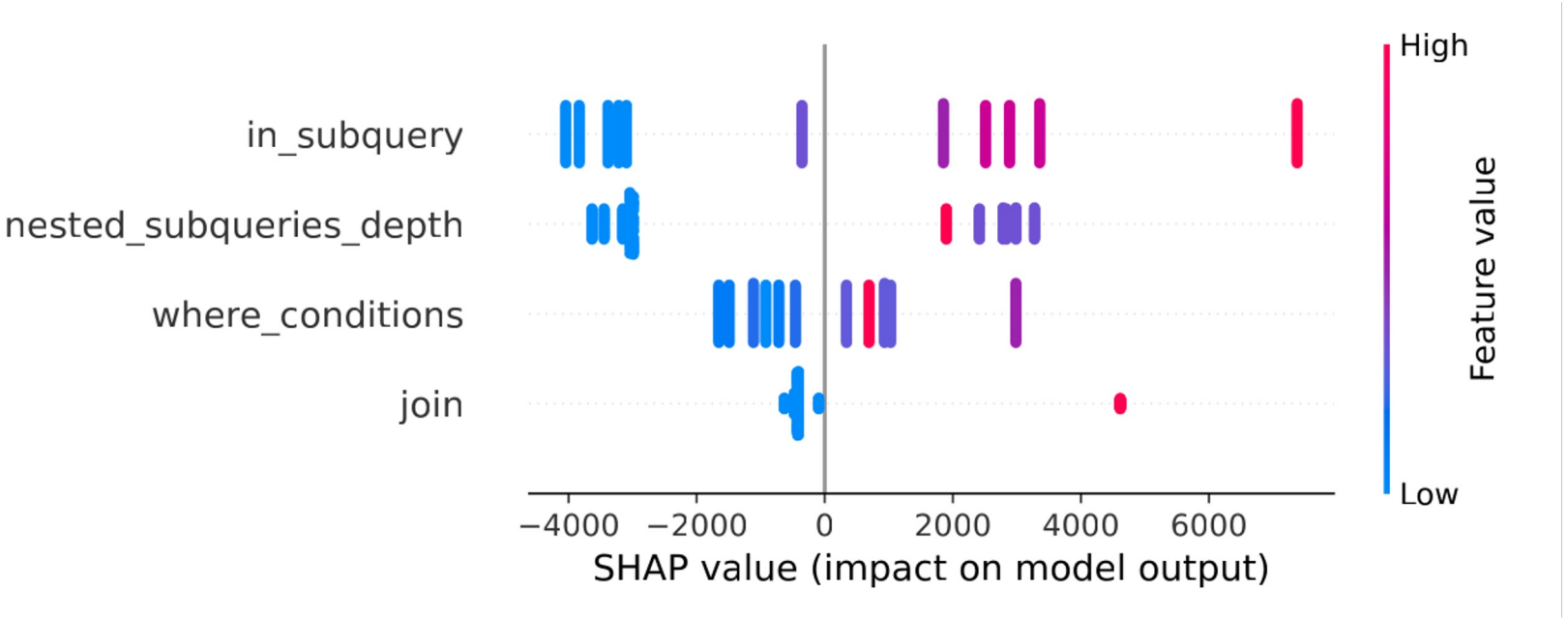
Learning & Explaining

```
with PdfPages(f'{INPUT_FILE_NAME}_shap_analysis.pdf') as pdf:
    # Explain the model: https://shap.readthedocs.io/
    shap.initjs()
    shap_values = shap.TreeExplainer(model).shap_values(X_train)

    pdf.savefig(shap.summary_plot(shap_values, X_train, show=False))

    for feature in X_train.columns:
        pdf.savefig(
          shap.dependence_plot(feature, shap_values, X_train, show=False),
          bbox_inches='tight'
        )
```



```
from catboost import CatBoostRegressor
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
import pandas
import shap

COLUMN_TO_PREDICT = 'query_duration_ms'
INPUT_FILE_NAME = 'catboost_dataset'
LEARNING_STEPS = 1000

def df_to_io(df):
    X = df.drop(COLUMN_TO_PREDICT, axis=1)
    y = df[COLUMN_TO_PREDICT]
    return X, y

# Read data from ClickHouse, that were exported using FORMAT TSVWithNames
df = pandas.read_csv(f"{INPUT_FILE_NAME}.tsv", sep="\t")

# Split queries equally to learn and test the model
training_df, fitting_df, evaluating_df = np.array_split(df, 3)
X_train, y_train = df_to_io(training_df)
X_fit, y_fit = df_to_io(fitting_df)
X_eval, y_eval = df_to_io(evaluating_df)

# Train
model = CatBoostRegressor(iterations=LEARNING_STEPS)
model.fit(X_train, y_train, eval_set=(X_fit, y_fit), verbose=False, plot=False)

# Evaluate
y_predicted = model.predict(X_eval)

print("Quality of the model (you can tweak the LEARNING_STEPS variable in the script to improve it):")
print("This is mainly informational, we only need an average model. What's important is to know what it learned.")
print(f"Mean sq error: {mean_squared_error(y_eval, y_predicted)}")
print(f"Mean abs error: {mean_absolute_error(y_eval, y_predicted)}")

with PdfPages(f'{INPUT_FILE_NAME}_shap_analysis.pdf') as pdf:
    # Explain the model: https://shap.readthedocs.io/
    shap.initjs()
    shap_values = shap.TreeExplainer(model).shap_values(X_train)

    pdf.savefig(shap.summary_plot(shap_values, X_train, show=False))

    for feature in X_train.columns:
        pdf.savefig(shap.dependence_plot(feature, shap_values, X_train, show=False), bbox_inches='tight')
```

Interpreting results

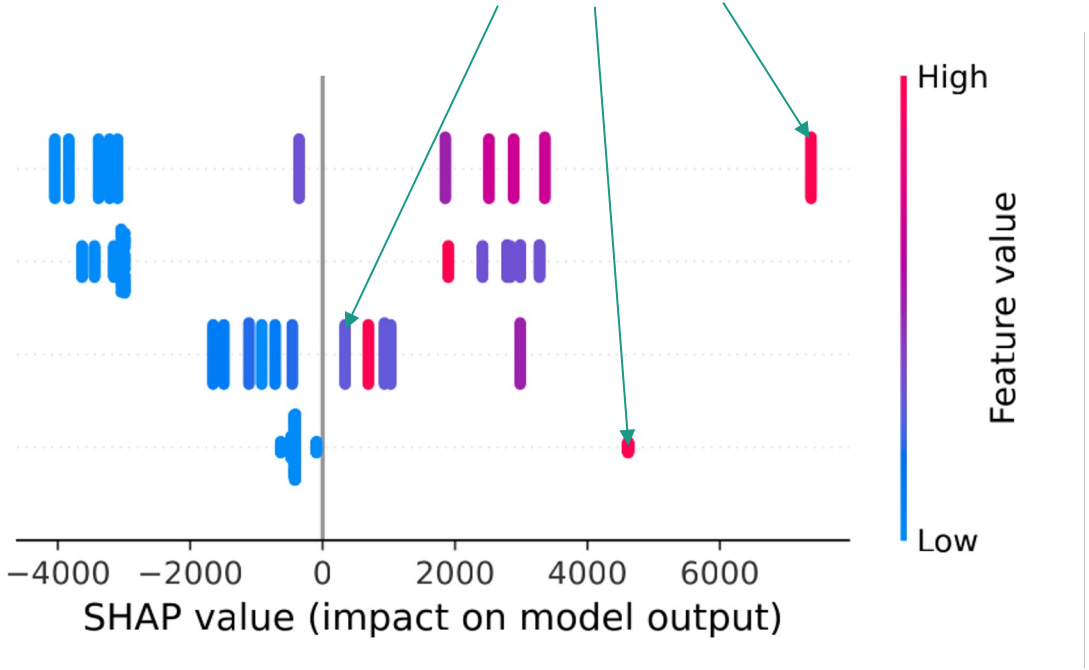


Interpreting results

Those are actually dots (lots of them stack together).
One dot = 1 row in our dataset!

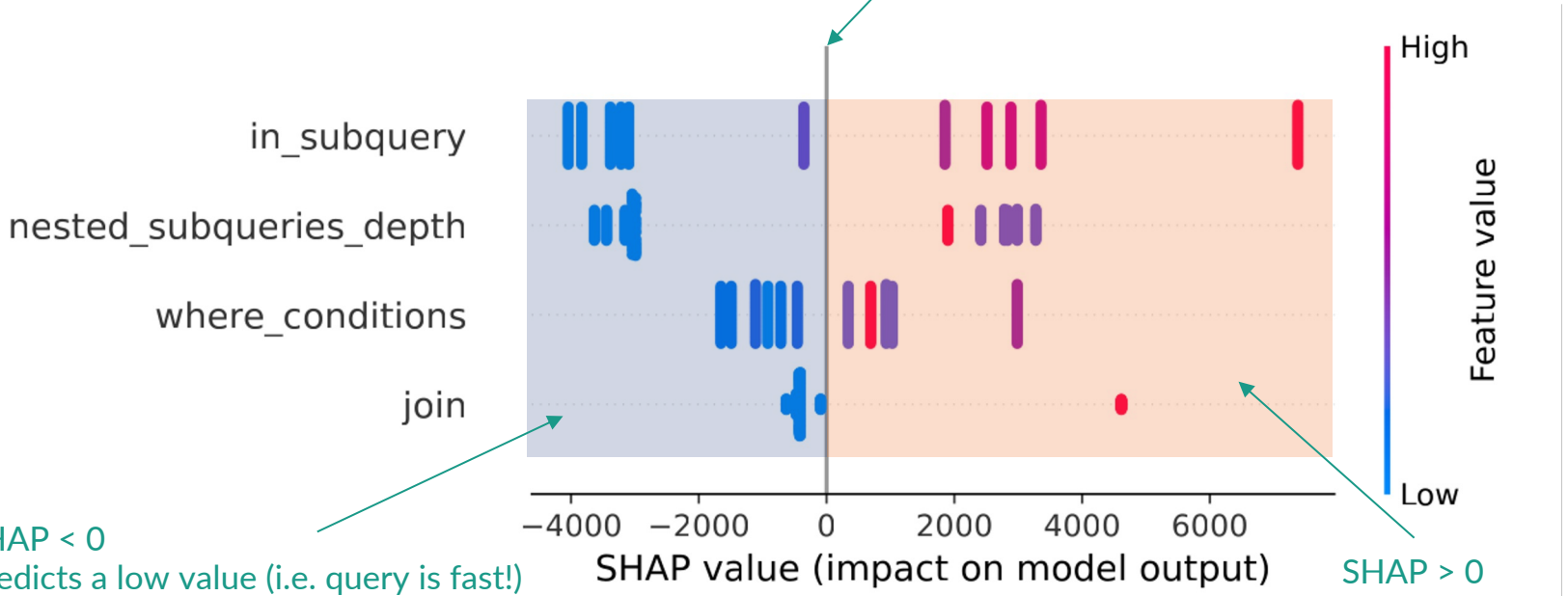
Our four metrics

- in_subquery
- nested_subqueries_depth
- where_conditions
- join



Interpreting results

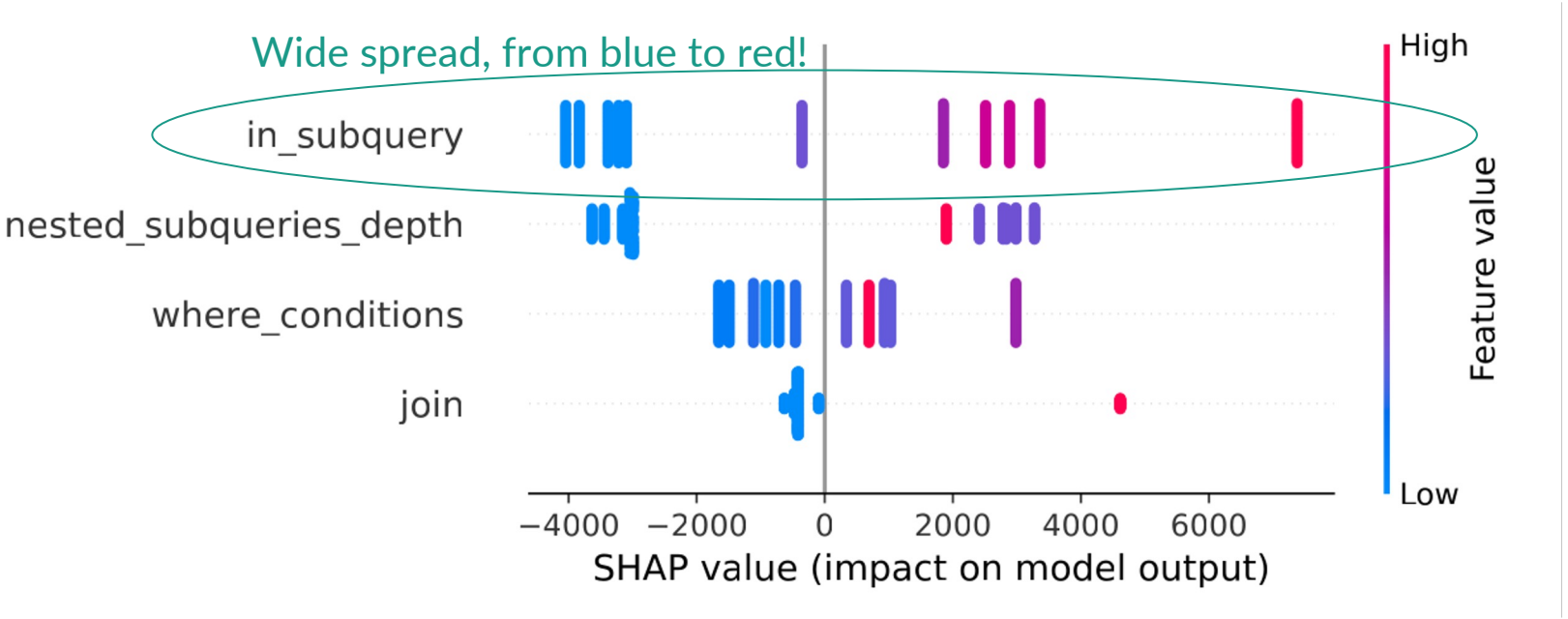
SHAP = 0
no impact on the prediction



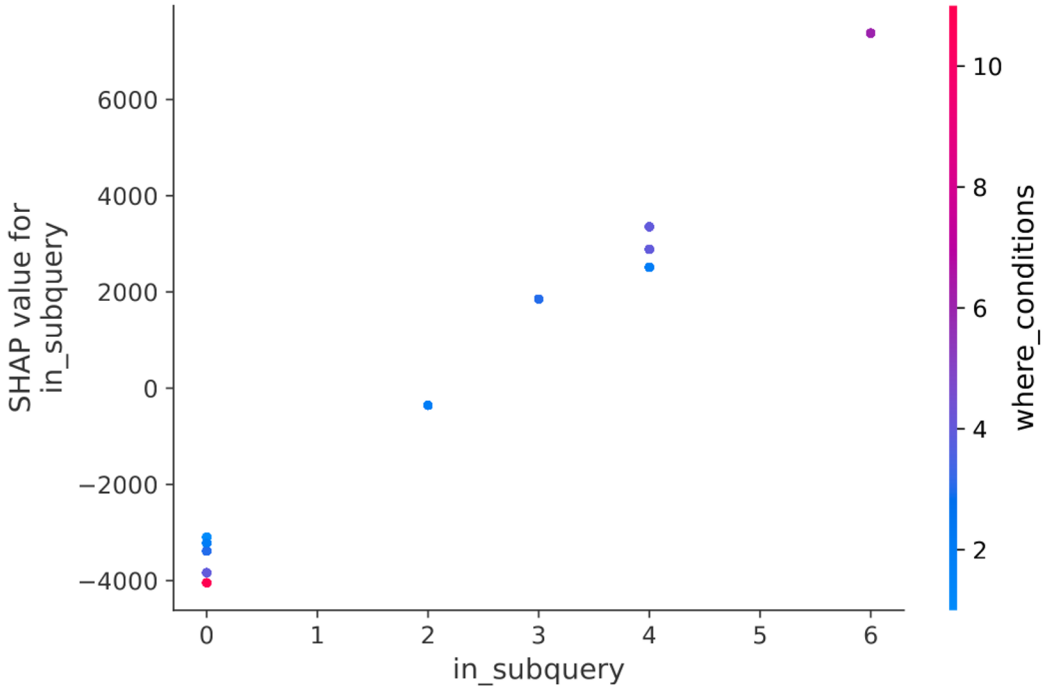
SHAP < 0
Predicts a low value (i.e. query is fast!)

SHAP > 0
Predicts a high value
(i.e. query is slow!)

Interpreting results



Interpreting results – in_subquery



Looks linear

Important SHAP value variations

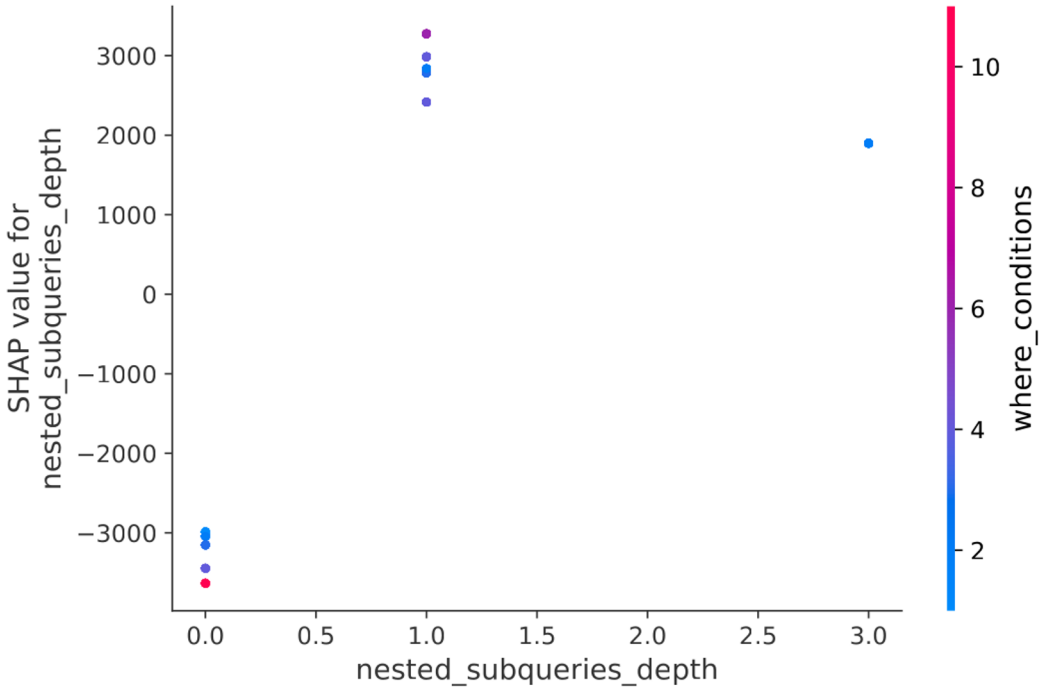
=> We want to check queries having `in_subquery=6`

Interpreting results – nested_subqueries_depth

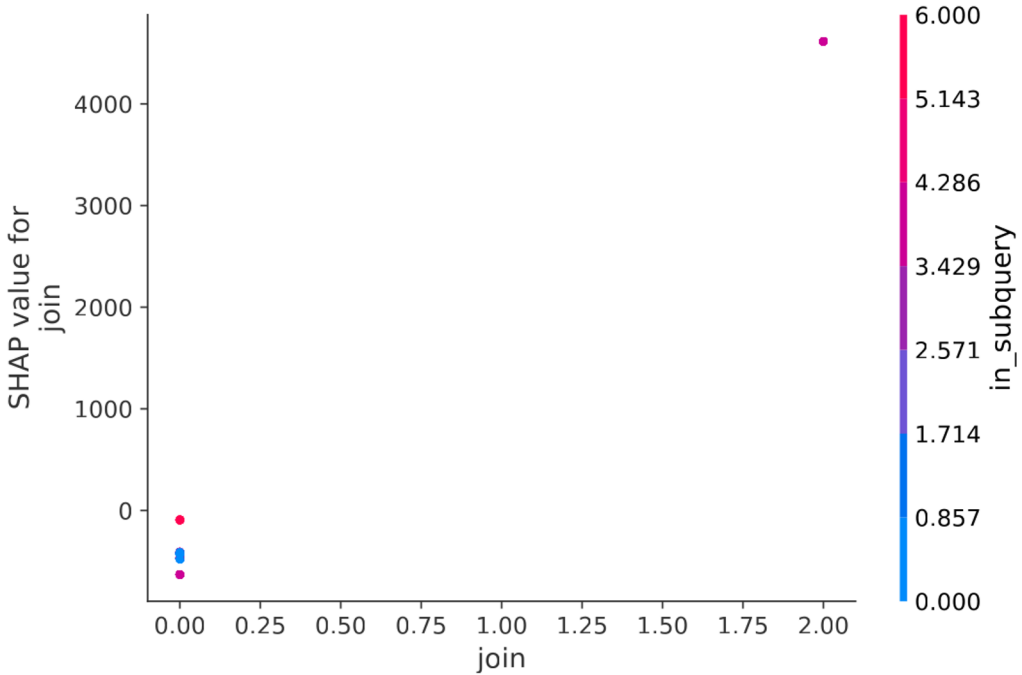
High value as soon as we have a subquery, but stays stable when we have more nesting.

=> We want to check queries having `nested_subqueries_depth>=1`

However this case will necessarily be covered by our check on `in_subquery=6`.



Interpreting results – join



Do we have enough queries with a join?

SHAP value variation is much lower than others.

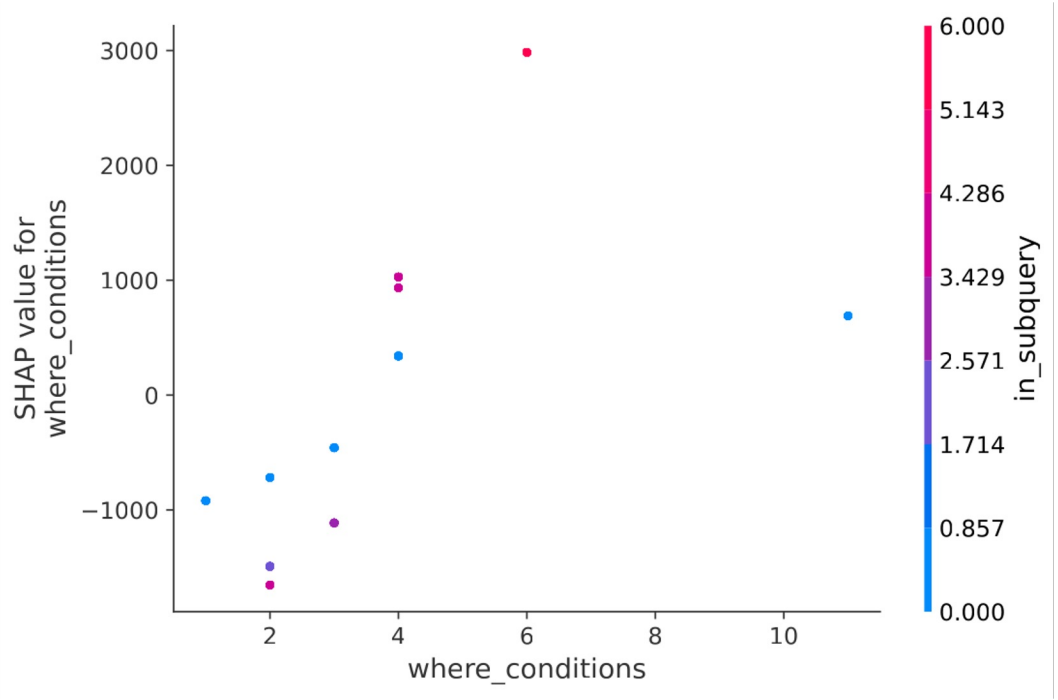
=> We don't want to spend time checking this feature

Interpreting results – where_conditions

Doesn't seem to have much impact. But maybe there are some timeouts?

SHAP value variation is almost nonexistent.

=> We don't want to spend time checking this feature

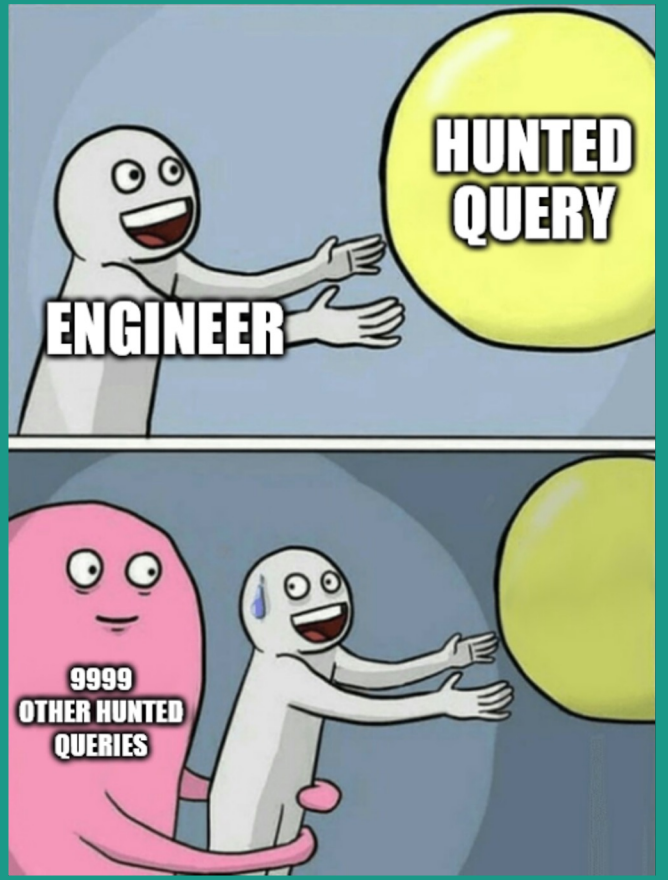


Retrieving non-optimized queries

```sql
SELECT query
FROM system.query_log
WHERE type = 'QueryFinish'
AND extract(log_comment, '^table=([^;]+)') == 'git.commits'
AND toUInt8(extract(log_comment, ';in_subquery=([^;]+);')) = 6
ORDER BY query_duration_ms DESC
LIMIT 1
```

```sql
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE startsWith(path, 'docker'))
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE NOT startsWith(path, 'docker/docs/'))
) OR (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE startsWith(path, 'tests'))
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE NOT startsWith(path, 'tests/ci/'))
) OR (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE startsWith(path, 'utils'))
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE NOT startsWith(path, 'utils/backup'))
)
SETTINGS log_comment = 'table=git.commits;nested_subqueries_depth=1;where_conditions=6;in_subquery=6;join=0'
```

From hunted to optimized



Is it worth to optimize a particular query?

How many other queries have that pattern?

What impact would an optimization have?

How much time will be needed to fix that pattern?

How many other queries have that pattern

```
SELECT
    countIf(
      hash_IN > 2
      AND hash_IN - startswith_path - not_startswith_path = 0
    ) AS optimizable_queries,
    round(optimizable_queries / count(), 2) AS ratio_optimizable_queries
FROM
(
    SELECT
        countSubstrings(query, '(hash, 0) IN') AS hash_IN,
        countSubstrings(query, 'WHERE startsWith(path') AS startswith_path,
        countSubstrings(query, 'WHERE NOT startsWith(path') AS not_startswith_path
    FROM system.query_log
    WHERE type IN ['QueryStart', 'ExceptionBeforeStart']
    AND extract(log_comment, '^table=([^;]+)') = 'git.commits'
    AND event_date >= today() - 30
)

┌─optimizable_queries─┬─ratio_optimizable_queries─┐
│                 300 │                      0.27 │
└─────────────────────┴───────────────────────────┘
```

What impact might it have – Proof of Concept

1. Rewrite the query
    a. Think of different ways you already know
    b. Take the time to browse through ClickHouse's documentation
    c. Ask for help on ClickHouse's official Telegram group
2. Benchmark each idea / improvement
3. Estimate the impact
    a. on the cluster
    b. on specific features of your product

What impact might it have – Query rewrite

```
SELECT uniqExact(author)
FROM git.commits
WHERE (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE startsWith(path, 'docker'))
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE NOT startsWith(path, 'docker/docs/'))
) OR (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WHERE startsWith(path, 'tests'))
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WH
) OR (
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WH
    AND
    (hash, 0) IN (SELECT commit_hash, sleep(0.048+1) FROM git.file_changes WH
)
SETTINGS log_comment = 'table=git.commits;nested_subqueries_depth=1;where_cor
```

```
SELECT uniqExact(author)
FROM git.commits
WHERE (hash, 0) IN (
    SELECT commit_hash, sleep(0.048+1)
    FROM git.file_changes
    WHERE (startsWith(path, 'docker') AND NOT startsWith(path, 'docker/docs/'))
    OR (startsWith(path, 'tests') AND NOT startsWith(path, 'tests/ci/'))
    OR (startsWith(path, 'utils') AND NOT startsWith(path, 'utils/backup'))
)
```

What impact might it have – Benchmark result

```
localhost:19000, queries: 14, QPS: 0.045, RPS: 98025.267,
MiB/s: 4.520, result RPS: 0.045, result MiB/s: 0.000.

0.000%          21.111 sec.
10.000%         21.117 sec.
20.000%         22.140 sec.
30.000%         22.145 sec.
40.000%         22.145 sec.
50.000%         22.149 sec.
60.000%         22.150 sec.
70.000%         22.150 sec.
80.000%         22.150 sec.
90.000%         22.166 sec.
95.000%         22.166 sec.
99.000%         22.169 sec.
99.900%         22.169 sec.
99.990%         22.169 sec.
```

```
localhost:19000, queries: 57, QPS: 0.189, RPS: 81590.517,
MiB/s: 3.965, result RPS: 0.189, result MiB/s: 0.000.

0.000%          5.273 sec.
10.000%         5.275 sec.
20.000%         5.276 sec.
30.000%         5.277 sec.
40.000%         5.278 sec.
50.000%         5.278 sec.
60.000%         5.279 sec.
70.000%         5.280 sec.
80.000%         5.282 sec.
90.000%         5.286 sec.
95.000%         5.287 sec.
99.000%         5.293 sec.
99.900%         5.294 sec.
99.990%         5.294 sec.
```

How much time will be needed to fix the pattern

If you want to make good use of your time, you've got to know what's most important and then give it all you've got.

— Lee Iacocca —

Happy hunting

Slides:

yohannjardin

twitch.tv/
amendile

yohannj

Amendil#6077

Credits

Icons:

- Cluster computing icons created by Canticons - Flaticon
- Dataset icons created by Muhammad Atif - Flaticon
- Deep learning icons created by Becris - Flaticon
- Different icons created by Freepik - Flaticon
- Deposit icons created by Freepik - Flaticon
- Kpi icons created by Uniconlabs - Flaticon
- Mature icons created by Chanut-is-Industries - Flaticon
- Resilience icons created by Flowicon - Flaticon
- Scale icons created by Freepik - Flaticon