ClickHouse

Другие движки

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

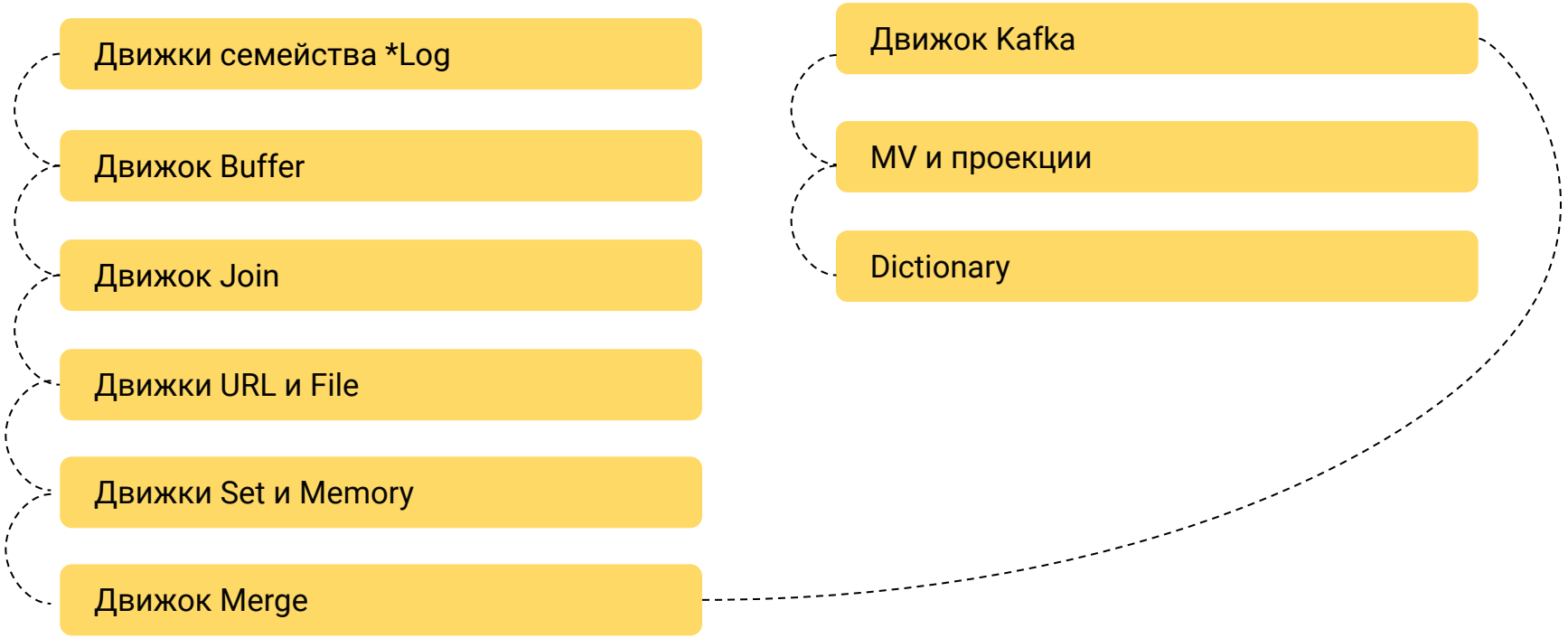
Другие движки

Цыкунов Алексей

Co-founder & CTO at Hilbert Team

- Более 20 лет опыта в проектировании и реализации отказоустойчивых и высоконагруженных информационных систем в таких отраслях как телеком и FinTech
- Автор курсов по Linux в Otus.ru
- Более 8 лет опыта оптимизации работы продуктовых команд и R&D департаментов с помощью DevOps инструментов и методик (Kubernetes, CI/CD, etc.) и облачных технологий (AWS, GCP, Azure, Yandex.Cloud)

Маршрут вебинара



Цели вебинара

К концу занятия вы сможете

1. выбирать подходящие движки для разных задач
2. настраивать базовые Data Pipelines с помощью MV

Смысл

Зачем вам это уметь

1. Для организации более быстрых запросов с помощью Buffer и Join
2.
3. Для построения простейших Data Pipelines

Движки TinyLog, Log, StripeLog

Основные особенности семейства *Log

- Основной сценарий использования - временные, промежуточные данные
  - пишется в много таблиц, а в таблицах немного записей (до 1 млн)
  - записи читаются из таблиц целиком
- Запись блокирует запись и чтение
  - чтение не блокирует другие чтения
- Мутации не поддерживаются.
- Индексы не поддерживаются.
- Дозапись в конец файла.
- Не атомарно. Может быть повреждение данных при сбоях во время записи
- Могут хранить данные в HDFS или S3

TinyLog

- Самый простой движок
- Каждый столбец хранится в отдельном файле
  - как и в Log
- Не поддерживает многопоточного чтения
  - в отличие от Log и StripeLog
  - зато записи при чтении отсортированы в порядке вставки

Log

- Расширяет возможности TinyLog многопоточным чтением
  - реализуется с помощью файла засечек
  - Засечки пишутся на каждый блок данных и содержат смещение:
    - с какого места нужно читать файл, чтобы пропустить заданное количество строк.

StripeLog

- Хранит все столбцы в одном файле
  - **data.bin** — файл с данными.
  - **index.mrk** — файл с метками. Метки содержат смещения для каждого столбца каждого вставленного блока данных.
- при INSERT добавляет блок данных в конец файла таблицы
  - записывая столбцы один за другим.

Синтаксис CREATE TABLE

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
  column1_name [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
  ...
)
ENGINE = [ StripeLog | Log | TinyLog ]
```

Пример

```
CREATE TABLE log_tbl (number UInt64) ENGINE = Log;
INSERT INTO log_tbl SELECT number FROM numbers(10);

CREATE TABLE stripe_log_tbl ENGINE = StripeLog AS
    SELECT number AS x FROM numbers(10);

CREATE TABLE tiny_log_tbl (A UInt8) ENGINE = TinyLog;
INSERT INTO tiny_log_tbl SELECT number FROM numbers(10);


select name, data_paths from system.tables
where name in ('log_tbl', 'stripe_log_tbl', 'tiny_log_tbl')
```

Движок Buffer

Buffer

- предназначен для буферизации данных в памяти
  - для ускорения вставки в другую! таблицу
  - рекомендуется также вставлять блоками
    - хотя бы по 10 записей
- При чтении из буферной таблицы, чтение происходит также и из целевой таблицы
- при разнице структур столбцов в целевой и буферной таблице
  - будут вставлены данные совпадающих столбцов
  - типы данных у столбцов должны совпадать
    - иначе INSERT завершится ошибкой
- Можно указать пустые кавычки вместо целевой таблицы
  - у вас будет просто буфер с периодической очисткой

Синтаксис CREATE TABLE

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    column1_name [type1] [DEFAULT|ALIAS expr1],
    ...
) ENGINE = Buffer(
    database, target_table,
    num_layers,
    min_time, max_time,
    min_rows, max_rows,
    min_bytes, max_bytes
    [,flush_time [,flush_rows [,flush_bytes]]]
);
```

Buffer

- num_layers— уровень параллелизма.
  - Кол-во независимых буферов. Рекомендуемое значение — 16
- Данные сбрасываются из буфера и записываются в таблицу назначения
  - если выполнены все min-условия
  - или хотя бы одно **max**-условие.
- min_time, max_time — условие на время в секундах от момента первой записи в буфер.
- min_rows, max_rows — условие на количество строк в буфере.
- min_bytes, max_bytes — условие на количество байт в буфере.
- Условия для сброса данных учитываются отдельно для каждого из буферов

Buffer: недостатки

- FINAL и SAMPLE не учитывают данные в буфере
- блокировка одного из буферов при вставке
  - может влиять на скорость чтения
- Сброс данных в порядке, отличном от ставки
  - негативный эффект на CollapsingMergeTree (нерегулируемый порядок вставки) и реплицируемые таблицы

Пример

```
create table buffer_tbl
(
     number UInt64
)
Engine=Buffer(currentDatabase(), log_table,
    /* num_layers= */  1,
    /* min_time= */    10,
    /* max_time= */    86400,
    /* min_rows= */    10000,
    /* max_rows= */    100000,
    /* min_bytes= */   0,
    /* max_bytes= */   8192,
    /* flush_time= */  86400,
    /* flush_rows= */  500,
    /* flush_bytes= */ 1024
);
```

Альтернатива движку Buffer: async_insert

- async_insert = 1 – включить асинхронные вставки (по умолчанию: 0)
- async_insert_threads - число потоков для фоновой обработки и вставки данных (по умолчанию: 16)
- wait_for_async_insert = 0/1 - ожидать или нет записи данных в таблицу
- wait_for_async_insert_timeout - время ожидания в секундах, выделяемое для обработки асинхронной вставки. 0 — ожидание отключено.
- …

Движок Join

Движок Join

- предназначен для предварительной подготовки данных для использования в операциях JOIN.
- Таблица используется в правой части секции JOIN или в функции joinGet
- Данные всегда в ОЗУ
- Данные хранятся на диске (опция persistent)
  - при аварийной остановке данные могут быть повреждены
- Поддерживаются DELETE мутации
- Нельзя использовать в GLOBAL JOIN
- Не поддерживается сэмплирование
- Не поддерживаются индексы и репликация

Синтаксис

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    fld [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    ...
) ENGINE = Join(
     [ANY | ALL],              -- строгость соединения
     [INNER | LEFT | ...],     -- тип соединения
     fld[, fld2, ...]          -- ключевые столбцы секции USING
)
SETTINGS
   persistent = [1 | 0],
   join_use_nulls = [1 | 0],
   max_rows_in_join = 0,
   max_bytes_in_join = 0,
   join_overflow_mode = [THROW | BREAK],
   join_any_take_last_row = [1 | 0];
```

Параметры движка

- **join_strictness** – строгость JOIN.
  - *ANY , ALL*
- **join_type** – тип JOIN.
  - *INNER, LEFT, RIGHT, FULL, CROSS*
- **k1[, k2, ...]** – ключевые столбцы секции USING с которыми выполняется операция JOIN.

Параметры **join_strictness** и **join_type** должны быть такими же как и в той операции JOIN, в которой таблица будет использоваться.

Если параметры не совпадают, ClickHouse не генерирует исключение и может возвращать неверные данные.

Движок Join

- persistent – хранить ли данные на диске
- join_use_nulls чем заполнять пустые ячейки полученных в ходе соединения таблиц
  - 0 - значения по умолчанию, 1 - NULL
- max_rows_in_join – ограничивает на кол-во строк в хэш таблице используемой при соединении 2х таблиц. По умолчанию 0.
- max_bytes_in_join - ограничивает на кол-во байт в хэш таблице используемой при соединении 2х таблиц. По умолчанию 0.
- join_overflow_mode – когда достигнуто ограничение по кол-ву байт или строк, этот параметр определяет действие при переполнении.
  - По умолчанию THROW - остановить запрос и прервать операцию.
  - Значение BREAK - прерывает операцию без исключения.
- join_any_take_last_row – определяет какие строки присоединять при совпадении.
  - 0 - первая найденная в правой таблице.
  - 1 – последняя найденная в правой таблице.

Пример

```
CREATE TABLE main_data
(
    id UInt32,
    desc_id UInt32
)
ENGINE = TinyLog;

CREATE TABLE desc_data (
    desc_id UInt32,
    desc String
)
engine = Join(ANY, INNER , desc_id);
```

Пример

```
INSERT INTO main_data VALUES (1,10), (2,20), (3,30)

INSERT INTO desc_data VALUES (10, 'mysql'),(20, 'pg'),(30, 'ch');

SELECT * FROM main_data ANY LEFT JOIN desc_data USING (desc_id);

SELECT * FROM main_data ANY INNER JOIN desc_data USING (desc_id);

SELECT id, joinGet(desc_data, 'desc', toUInt32(desc_id)) as description
FROM main_data;
```

Движки URL/File

Движок URL

- Для работы с данными на удаленном сервере
  - Запросы INSERT и SELECT транслируются в POST и GET запросы.
  - например, можно обращаться к таблице в другом ClickHouse через http
- Поддерживается многопоточная запись и чтение
- Движок не хранит данные локально
- Не поддерживаются изменение данных с помощью ALTER
- Не поддерживается сэмплирование
- Не поддерживаются индексы и репликация

Пример

```
CREATE TABLE url_engine_table
(`SIC Code` Nullable(Int64), `Description` Nullable(String))
ENGINE =
URL('https://cdn.wsform.com/wp-content/uploads/2020/06/industry_sic.csv', CSV)


SELECT * FROM url_engine_table;
```

Движок File

Управляет данными в одном файле на диске в указанном формате.

Примеры применения:

- Выгрузка данных из ClickHouse в файл.
- Преобразование данных из одного формата в другой.
  a. File(format)
- Обновление данных в ClickHouse редактированием файла на диск

Движок File

- Поддерживается одновременное выполнение множества запросов **SELECT**
- запросы **INSERT -** сериализуются
- При операции **CREATE** создает директорию
  - можно поместить туда файл и прикрепить с помощью **ATTACH**
- Поддерживается создание ещё не существующего файла при запросе **INSERT**.
- Для существующих файлов **INSERT** записывает в конец файла.
- Не поддерживается:
  - использование операций **ALTER** и **SELECT...SAMPLE**;
  - индексы;
  - репликация.

Пример

```
CREATE TABLE file_engine_table
(name String, value UInt32)
ENGINE=File(TabSeparated);

INSERT INTO file_engine_table VALUES ('test', 10), ('x',2);
```

Движки Set и Memory

Движок Set

- множество в оперативной памяти
  - реализовано на hash таблице
- используется в операторе IN
  - нельзя сделать SELECT из таблицы
- Можно вставлять данные через INSERT
  - Возможны дубликаты записей
- Данные могут храниться на диске
  - опция persistent

Движок Memory

- данные хранятся только в оперативной памяти
  - в несжатом виде
- Уместно использовать на датасетах до 10M строк
  - для достижения очень высокой скорости
- Чтение распараллеливается
- Чтение и запись не блокируют друг друга
- Индексы не поддерживаются
- подходит для GLOBAL IN

Пример

```
CREATE TABLE SX ( hbx UInt32 ) ENGINE = Set SETTINGS persistent=1
CREATE TABLE MX ( hbx UInt32 ) ENGINE = Memory

INSERT INTO MX SELECT number from numbers(300000)
INSERT INTO SX SELECT number from numbers(300000)

SELECT COUNT(*) FROM MX
SELECT COUNT(*) FROM SX

CREATE TABLE HL (id UInt32, val UInt32)
ENGINE = MergeTree ORDER BY (val);

INSERT INTO HL SELECT number, number * 10 from numbers(30000000)
SELECT count(*) FROM HL WHERE val IN SX
SELECT count(*) FROM HL WHERE val IN MX
```

Движок Merge

Движок Merge

- не имеет отношение к семейству MergeTree
- позволяет делать запрос к нескольким таблицам одновременно
  - вариация UNION
  - имена таблиц можно задать через RegExp
- ***INSERT*** не поддерживается
- ***CREATE TABLE ... Engine=Merge(db_name, tables_regexp)***
  - имя БД тоже можно указать через RegExp
- при выборке имеет виртуальный столбец ***_table***
- столбцы в таблице должны существовать в таблицах источниках

Для чего использовать

- работа с большим набором таблиц как с одной
  - например с *Log
- партиционирование существующей таблицы с данными
  - создаете новую с партициями
  - создаете Merge таблицу для чтения из старой и новой
  - запись происходит только в новую

Пример

```
CREATE TABLE log_1 (id UInt32, val UInt32) ENGINE = TinyLog;
CREATE TABLE log_2 (id UInt32, val UInt32) ENGINE = TinyLog;
INSERT INTO log_1 (id, val) select number, number * 10 FROM numbers(10000000);
INSERT INTO log_2 (id, val) select number, number * 200 FROM numbers(10000000);

CREATE TABLE log_common (id UInt32, val UInt32)
ENGINE = Merge(currentDatabase(), '^log_*');

SELECT count(*) FROM log_common;
```

Специализированные движки для интеграций

Движки для интеграций

- Clickhouse легко интегрируется с другими системами и базами данных
- Для этого у него есть ряд движков, которые обеспечивают доступ к данным в других системах
  - MySQL и MaterializedMySQL
  - PostgreSQL и MaterializedPostgreSQL
  - Hive
  - MongoDB
  - HDFS
  - Kafka
  - SQLite
  - ODBC и JDBC

MV и проекции

Materialized View

- хранят данные, которые были выбраны соответствующим запросом SELECT, указанным при создании
- содержимое может быть реплицировано
- SELECT можно делать как из MV так и из таргетной таблицы
- Принцип работы отличается от других СУБД
  - больше похоже на триггер ***AFTER INSERT***
  - Если в запросе материализованного представления есть агрегирование, оно применяется только к вставляемому блоку записей.
  - обновления в исходной таблице не влияют на данные в MV, только вставки

Синтаксис

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db.]mv_name [ON CLUSTER]
[TO [db.]table]  -- явно указать таблицу для хранения данных
[                 -- имена полей и типы (лучше указать сразу)
  ( fld_1 Type_1, fld_2 Type_2, ...)
]
[ENGINE = engine]    -- указать движок, если не указана таблица в "TO"
[POPULATE]       -- сразу загрузить данные из источника
AS
SELECT           -- SELECT преобразование
     expr(f1) as fld_1,
     expr(f2) as fld_2, ...
FROM source_tbl  -- выборка из таблицы-источника
[GROUP BY] [ORDER BY]
```

Пример с неявным созданием таблицы приемника

```
CREATE TABLE source_tbl (num UInt64)
ENGINE = Log;

CREATE MATERIALIZED VIEW otus_mv
     ENGINE = TinyLog AS
     SELECT num * num as fld
     FROM source_tbl;

INSERT INTO source_tbl SELECT number FROM numbers(10);

SELECT * FROM otus_mv WHERE fld=1;

SELECT name, uuid, engine, metadata_path FROM system.tables t WHERE name ilike
'%.inner%' or name='otus_mv'\G
```

Пример с явным созданием таблицы приемника

```
CREATE TABLE mem_target (num UInt64, fld UInt64)
ENGINE = SummingMergeTree ORDER BY (num);

CREATE MATERIALIZED VIEW my_mv
     TO mem_target
     AS
     SELECT num, num + 10 as fld
     FROM source_tbl;

SELECT * FROM my_mv;

INSERT INTO source_tbl SELECT intDiv(number,2) FROM numbers(10);

SELECT * FROM my_mv;
```

Сценарии использования MV

- Агрегация данных
- Обработка потоков данных из внешних источников
- Маршрутизация и преобразование данных
    - можно создать несколько MV для одного источника и направлять данные в разные таблицы в зависимости от условий
- Дублирование данных для изменения ключа сортировки

Проекции

- данные проекций всегда согласованы
- данные обновляются атомарно вместе с таблицей
- содержимое проекций реплицируются вместе с таблицей
- проекция может быть автоматически использована для запроса SELECT

Когда используется проекция (соблюдение всех условий):

- если выборка соответствует запросу проекции
- если 50% выбранных кусков содержат материализованные проекции. Проекции могут содержать данные, т.е. быть материализованы, либо могут быть пустые, если, например была применена команда ALTER TABLE CLEAR PROJECTION
- если количество выбранных строк меньше общего количества строк таблицы

Пример проекции

```
CREATE TABLE visits
(
   user_id UInt64,
   user_name String,
   pages_visited Nullable(Float64),
   user_agent String,
   PROJECTION projection_visits_by_user
   (
       SELECT
           user_agent,
           sum(pages_visited)
       GROUP BY user_id, user_agent
   )
)
ENGINE = MergeTree()
ORDER BY user_agent
```

Пример проекции

```
INSERT INTO visits SELECT
    number, 'test', 1.5 * (number / 2),'Android'
FROM numbers(1, 100);

INSERT INTO visits SELECT
    number,'test',1. * (number / 2),'IOS'
FROM numbers(100, 500);

SELECT
    user_agent,
    sum(pages_visited)
FROM visits
GROUP BY user_agent
```

Движок Kafka

что такое Kafka

- масштабируемая шина сообщений
- используется для построения Data Pipelines и ETL процессов
- лучше всего принцип работы продемонстрирован в данной визуализации
- основные понятия
  - **Record** – Запись, состоящая из ключа и значения
  - **Topic** – категория или имя потока куда публикуются записи
  - **Producer** – процесс публикующий данные в топик
  - **Consumer** – процесс читающий данные из топика
  - **Consumer group -** группа читателей из одного топика для балансировки нагрузки по чтению
    - Разные consumer groups читают независимо друг от друга.
  - **Offset** – позиция записи
  - **Partition** – шард топика

Движок Kafka

- одна из самых частых интеграций
- при создании таблицы указываются
- не хранит данные самостоятельно
  - предназначен для подписки на потоки данных в конкретных топиках (consumer)
    - SELECT может прочесть запись только один раз
      - поэтому имеет смысл использовать MV
  - и для публикации данных в конкретные топики (producer)

Синтаксис

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'host:port',
    kafka_topic_list = 'topic1,topic2,...',
    kafka_group_name = 'group_name',
    kafka_format = 'data_format'[,]
    [kafka_row_delimiter = 'delimiter_symbol',]
    [kafka_schema = '',]
    ….
```

Обязательные настройки движка Kafka

- **kafka_broker_list** — перечень брокеров, разделенный запятыми
- **kafka_topic_list** — перечень необходимых топиков Kafka, разделенный запятыми
- **kafka_group_name** — группа потребителя Kafka. Если необходимо, чтобы сообщения не повторялись на кластере, необходимо использовать везде одно имя группы.
- **kafka_format** — формат сообщений, например JSONEachRow.
- Опциональные параметры, такие как размер блока, количество потребителей на таблицу и потоков, подробно описаны в документации.

Пример организации пайплайна INSERT to -> kafka -> MV -> MergeTree

```
cat > test_msg.json << EOL
{"Message":"Some message","Priority":"A1"}
EOL

# отправим сообщение
cat test_msg.json | kafkacat -P -b 127.0.0.1:29092 -t OtusTopic

# проверим что оно дошло
kafkacat -C -b 127.0.0.1:29092 -t OtusTopic
```

Пример организации пайплайна INSERT to -> kafka -> MV -> MergeTree

```
CREATE TABLE kafka_tbl (
Message String,
Priority String
) ENGINE = Kafka
SETTINGS kafka_broker_list = '127.0.0.1:29092',
kafka_topic_list = 'MyOtusTopic',
kafka_group_name = 'test-consumer-group',
kafka_format = 'JSONEachRow';

SELECT * FROM default.kafka_tbl
SETTINGS stream_like_engine_allow_direct_select = 1;
```

Пример организации пайплайна INSERT to -> kafka -> MV -> MergeTree

```
CREATE TABLE from_kafka_tbl (
Message String,
Priority String
) ENGINE = MergeTree
ORDER BY Priority;

DROP VIEW IF EXISTS mv_kafka;
CREATE MATERIALIZED VIEW mv_kafka TO from_kafka_tbl AS SELECT * FROM kafka_tbl;

SELECT * FROM mv_kafka;
```

Пример организации пайплайна INSERT to -> kafka-engine -> kafka topic

```
DROP VIEW mv_kafka;
INSERT INTO kafka_tbl(Message) VALUES('test msg');

kafkacat -C \ -b 127.0.0.1:29092 \ -t MyTestTopic
```

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате