ClickHouse для инженеров и архитекторов БД

otus.ru

● REC **Проверить, идет ли запись**

Меня хорошо видно && слышно?

Тема вебинара

Clickhouse и dbt

Андрей Поляков

Senior Java/Groovy Developer/ Data Engineer

Об опыте:

- В отрасли бекенд разработки на java я более 8 лет.
- Занимался fullstack разработкой приложений, разработкой высоко нагруженных compute-grid систем, а также микросервисов и etl-пайплайнов.
- Сейчас в роли старшего разработчика работаю над сервисами платежных систем в Unlimint.

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Вопросы вижу в чате, могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или задайте вопрос

Маршрут вебинара

- Мотивация: SQL в ETL/ELT
- Конфигурация dbt, структура проекта
- Специфичная для Clickhouse конфигурация
- Подготовка инфраструктуры
- Рефлексия

Цели вебинара

1. Познакомиться с Data Build Tool – мультитул для работы с DWH;
2. Рассмотреть основные возможности и принципы dbt
3. Понять, как инструменты подобные dbt могут помочь инженерам и аналитикам при работе с clickhouse;

Data Build Tool (dbt)

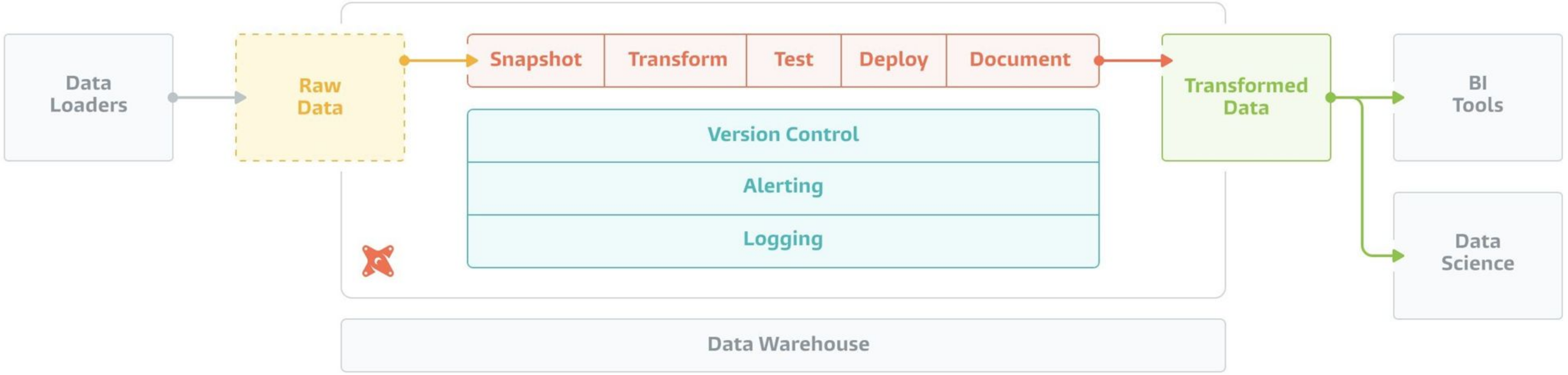
Мотивация: SQL в ETL/ELT

```
CREATE PROCEDURE etl_example AS
BEGIN
-- Extract data from the source table
SELECT * INTO #temp_table FROM source_table;
-- Transform data
UPDATE #temp_table
SET column1 = UPPER(column1),
column2 = column2 * 2;
-- Load data into the target table
INSERT INTO target_table
SELECT * FROM #temp_table;
END
```

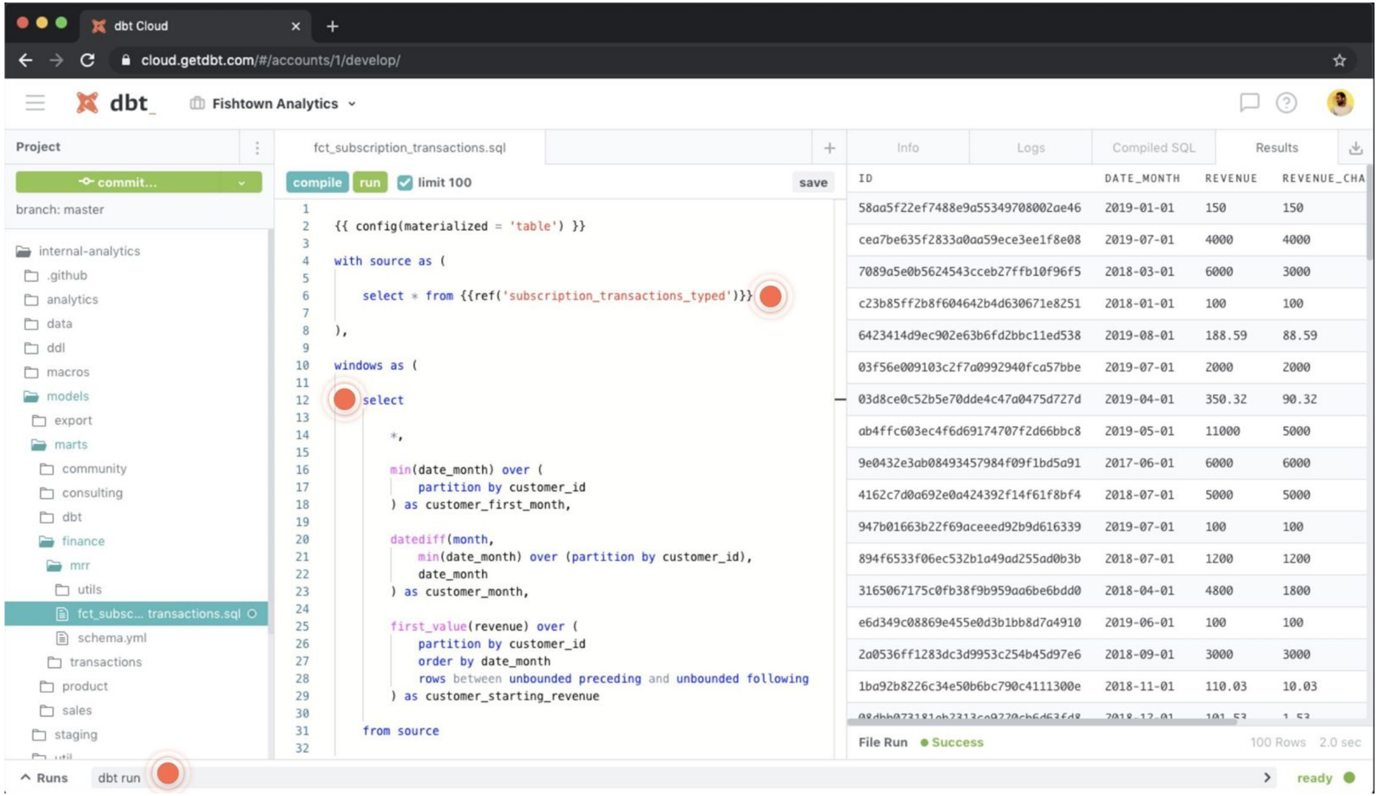
Мотивация: ETL-инструменты

```python
from airflow import DAG
from airflow.operators.clickhouse_operator import ClickhouseOperator
from datetime import datetime, timedelta

default_args = {
     'owner': 'me',
     'start_date': datetime(2022, 1, 1),
     'depends_on_past': False,
     'retries': 1,
     'retry_delay': timedelta(minutes=5)}
dag = DAG('simple_dag',
     default_args=default_args,
     schedule_interval=timedelta(hours=1))
task1 = ClickhouseOperator(task_id='get_data',
     sql='select * from table',
     dag=dag)
task2 = ClickhouseOperator(task_id='insert_data',
     sql='insert into table values()',
     retries=3,
     dag=dag)
task1 >> task2
```

Data Build Tool – T in ELT



Модели – всё есть SELECT



```
{{ config(materialized = 'table') }}

with source as (

    select * from {{ref('subscription_transactions_typed')}}

),

windows as (

    select

        *,

        min(date_month) over (
            partition by customer_id
        ) as customer_first_month,

        datediff(month,
            min(date_month) over (partition by customer_id),
            date_month
        ) as customer_month,

        first_value(revenue) over (
            partition by customer_id
            order by date_month
            rows between unbounded preceding and unbounded following
        ) as customer_starting_revenue

    from source
```

ID	DATE_MONTH	REVENUE	REVENUE_CHA
58aa5f22ef7488e9a55349708002ae46	2019-01-01	150	150
cea7be635f2833a0aa59ece3ee1f8e08	2019-07-01	4000	4000
7089a5e0b5624543cceb27ffb10f96f5	2018-03-01	6000	3000
c23b85ff2b8f604642b4d630671e8251	2018-01-01	100	100
6423414d9ec902e63b6fd2bbc11ed538	2019-08-01	188.59	88.59
03f56e009103c2f7a0992940fca57bbe	2019-07-01	2000	2000
03d8ce0c52b5e70dde4c47a0475d727d	2019-04-01	350.32	90.32
ab4ffc603ec4f6d69174707f2d66bbc8	2019-05-01	11000	5000
9e0432e3ab08493457984f09f1bd5a91	2017-06-01	6000	6000
4162c7d0a692e0a424392f14f61f8bf4	2018-07-01	5000	5000
947b01663b22f69aceeed92b9d616339	2019-07-01	100	100
894f6533f06ec532b1a49ad255ad0b3b	2018-07-01	1200	1200
3165067175c0fb38f9b959aa6be6bdd0	2018-04-01	4800	1800
e6d349c08869e455e0d3b1bb8d7a4910	2019-06-01	100	100
2a0536ff1283dc3d9953c254b45d97e6	2018-09-01	3000	3000
1ba92b8226c34e50b6bc790c4111300e	2018-11-01	110.03	10.03
[illegible]	2018-12-01	101.53	1.53

File Run ● Success 100 Rows 2.0 sec

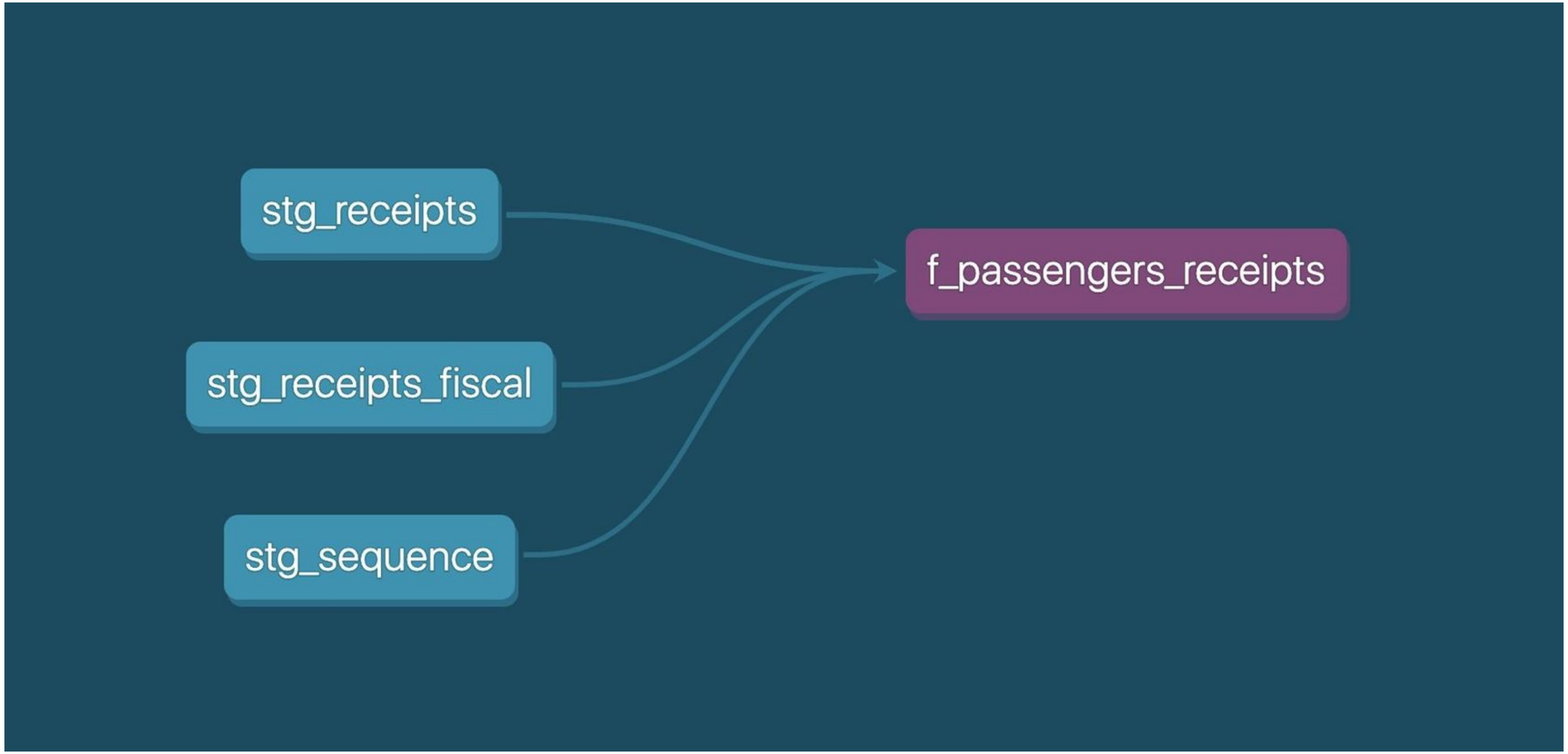
SQL + Jinja

Что делает этот код?

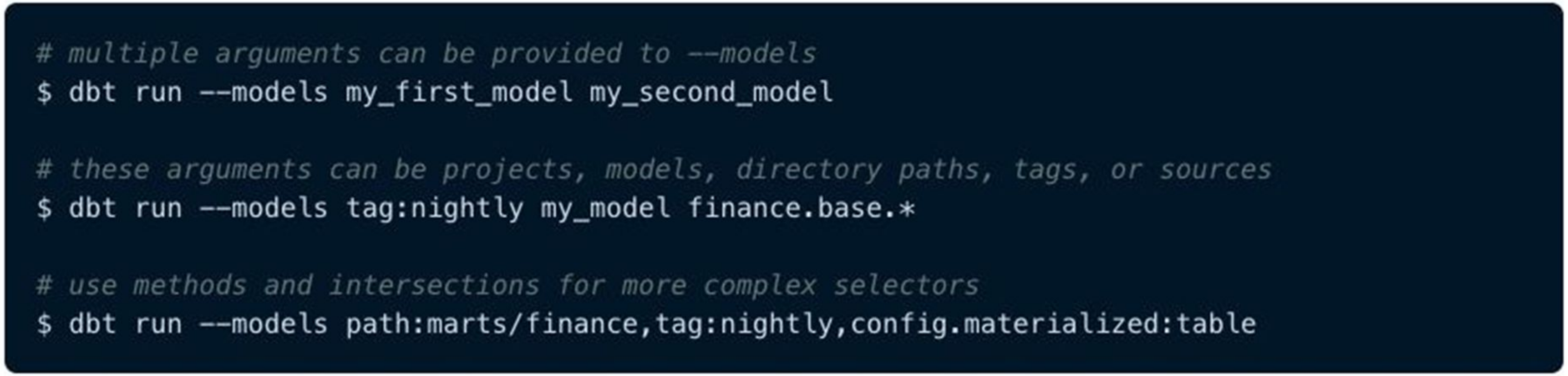
```
select
*
from event_tracking.events
{% if target.name == 'dev' %}
where created_at >= dateadd('day', -3, current_date)
{% endif %}
```

Сроки выполнения: 1 мин (пишем в чат)

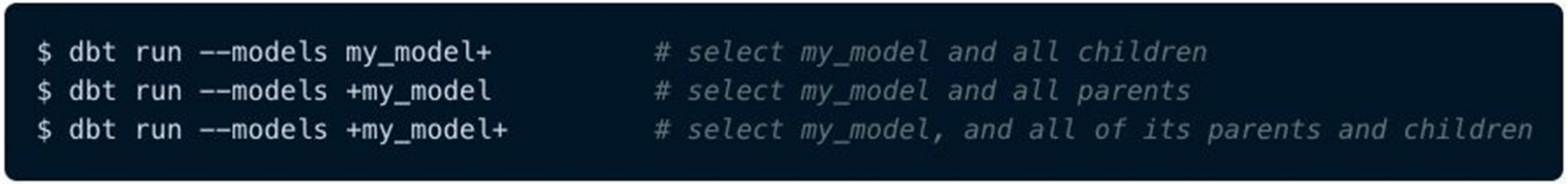
Графы исполнения моделей DAGs



Node selection syntax

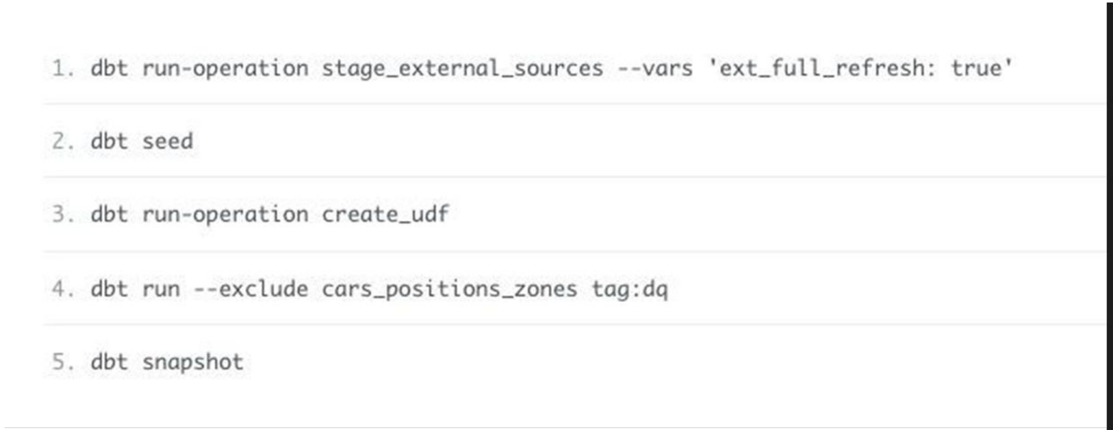


```
# multiple arguments can be provided to --models
$ dbt run --models my_first_model my_second_model

# these arguments can be projects, models, directory paths, tags, or sources
$ dbt run --models tag:nightly my_model finance.base.*

# use methods and intersections for more complex selectors
$ dbt run --models path:marts/finance,tag:nightly,config.materialized:table
```

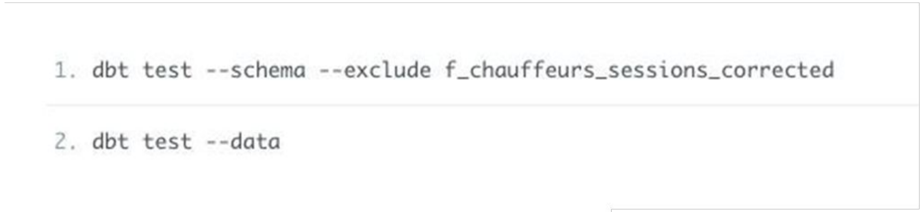


```
$ dbt run --models my_model+          # select my_model and all children
$ dbt run --models +my_model          # select my_model and all parents
$ dbt run --models +my_model+         # select my_model, and all of its parents and children
```

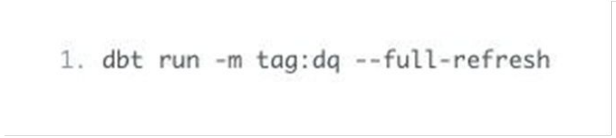
Tagging models and running subgraphs



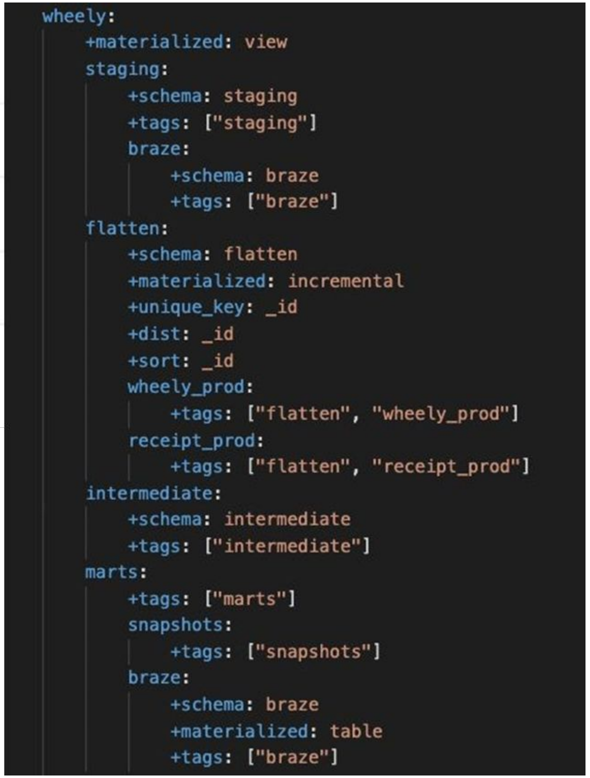
```
dbt run-operation stage_external_sources --vars 'ext_full_refresh: true'
dbt seed
dbt run-operation create_udf
dbt run --exclude cars_positions_zones tag:dq
dbt snapshot
```



```
dbt test --schema --exclude f_chauffeurs_sessions_corrected
dbt test --data
```



```
dbt run -m tag:dq --full-refresh
```



```
wheely:
    +materialized: view
    staging:
        +schema: staging
        +tags: ["staging"]
        braze:
            +schema: braze
            +tags: ["braze"]
    flatten:
        +schema: flatten
        +materialized: incremental
        +unique_key: _id
        +dist: _id
        +sort: _id
        wheely_prod:
            +tags: ["flatten", "wheely_prod"]
        receipt_prod:
            +tags: ["flatten", "receipt_prod"]
    intermediate:
        +schema: intermediate
        +tags: ["intermediate"]
    marts:
        +tags: ["marts"]
        snapshots:
            +tags: ["snapshots"]
        braze:
            +schema: braze
            +materialized: table
            +tags: ["braze"]
```

Конфигурация dbt

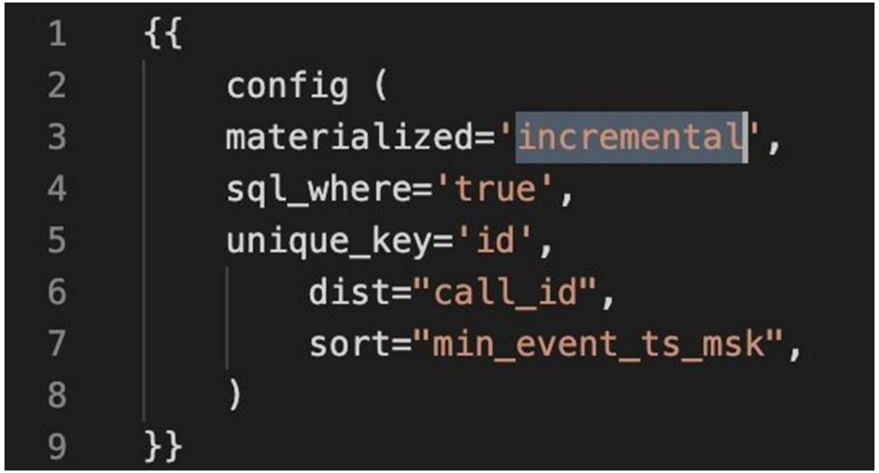
dbt project - metadata

```
name: acme_corp
profile: acme corp
version: '1.0'

require-dbt-version: ">=0.14.0"

source-paths: ["models"]
analysis-paths: ["analysis"]
test-paths: ["tests"]
data-paths: ["data"]
macro-paths: ["macros"]

target-path: "target"
clean-targets:
    - "target"
    - "dbt_modules"
```

Настройки подключения

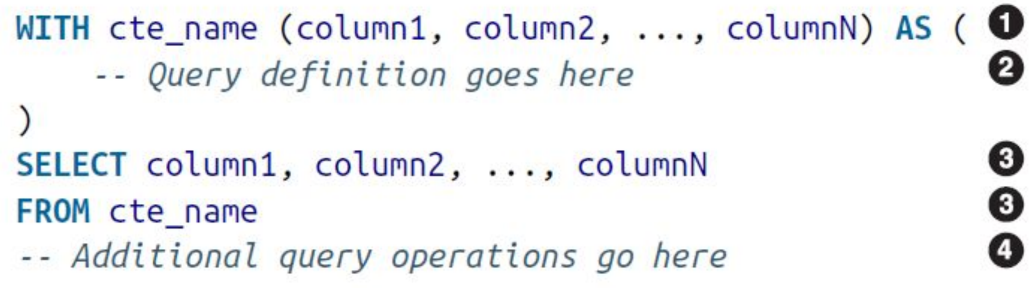
```yaml
acme_corp:
 outputs:
   dev:
     type: postgres
     threads: 8
     host: [hostname]
     user: [username]
     pass: [password]
     port: 5439
     dbname: [database name]
     schema: dbt_[username]  # e.g. dbt_alice
 target: dev
```

Полная и инкрементальная загрузка



```
{{
    config (
    materialized='incremental',
    sql_where='true',
    unique_key='id',
        dist="call_id",
        sort="min_event_ts_msk",
    )
}}
```

CTE



```
WITH cte_name (column1, column2, ..., columnN) AS (   ❶
    -- Query definition goes here                     ❷
)
SELECT column1, column2, ..., columnN                 ❸
FROM cte_name                                         ❸
-- Additional query operations go here                ❹
```

Что такое CTE? Для чего они нужны?

Сроки выполнения: 1 мин (пишем в чат)

CTE

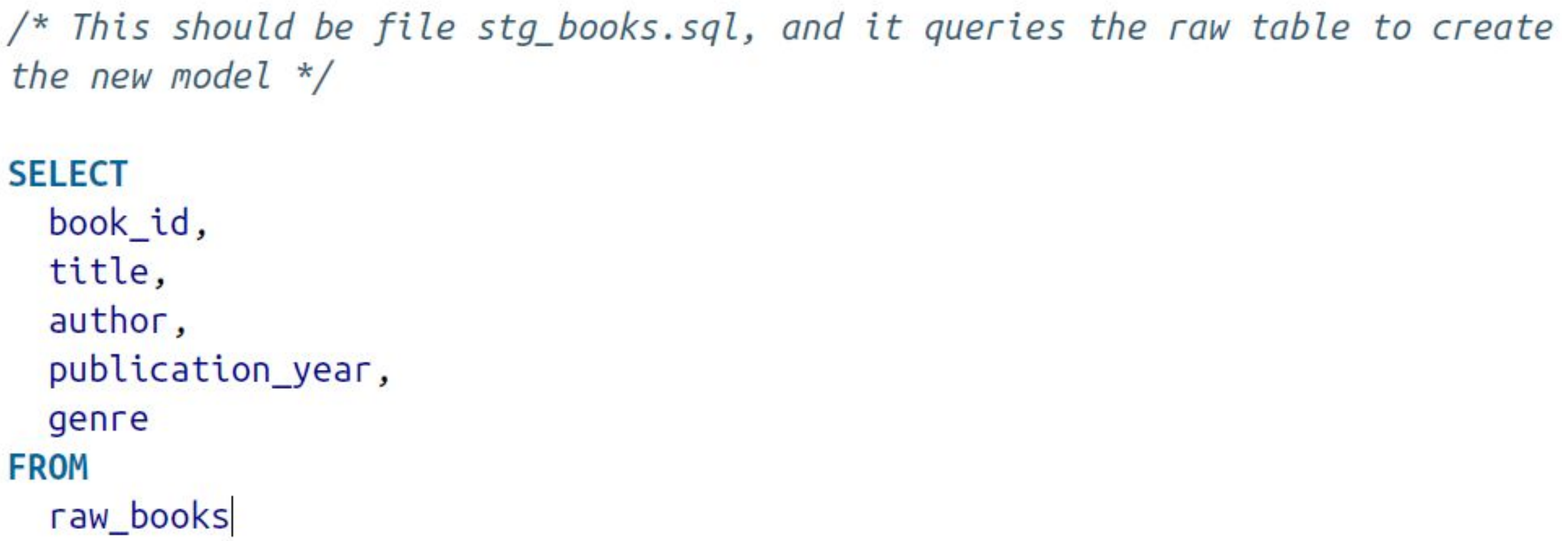
Без CTE

```sql
SELECT pb.book_id,
       pb.title,
       pb.author,
       s.total_sales
FROM (
    SELECT book_id,
           title,
           author
    FROM books
    WHERE rating >= 4.6
) AS pb
JOIN sales s ON pb.book_id = s.book_id
WHERE s.year = 2022
ORDER BY s.total_sales DESC
LIMIT 5;
```

С CTE

```sql
WITH popular_books AS (
    SELECT book_id,
           title,
           author
    FROM books
    WHERE rating >= 4.6
),
best_sellers AS (
    SELECT pb.book_id,
           pb.title,
           pb.author,
           s.total_sales
    FROM popular_books pb
    JOIN sales s ON pb.book_id = s.book_id
    WHERE s.year = 2022
    ORDER BY s.total_sales DESC
    LIMIT 5
)
SELECT *
FROM best_sellers;
```

Модели Stage



```sql
/* This should be file stg_books.sql, and it queries the raw table to create
the new model */

SELECT
  book_id,
  title,
  author,
  publication_year,
  genre
FROM
  raw_books
```

Модели Intermediate

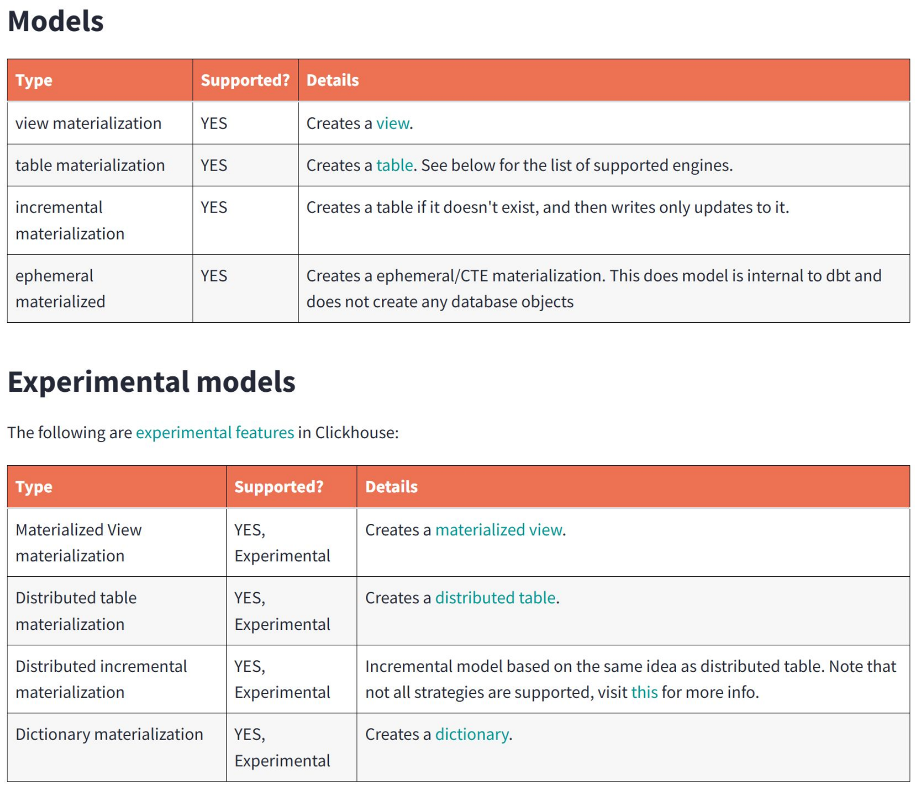
```
-- This should be file int_book_authors.sql

-- Reference the staging models
WITH
  books AS (
    SELECT *
    FROM {{ ref('stg_books') }}
  ),
  authors AS (
    SELECT *
    FROM {{ ref('stg_authors') }}
  )

-- Combine the relevant information
SELECT
  b.book_id,
  b.title,
  a.author_id,
  a.author_name
FROM
  books b
JOIN
  authors a ON b.author_id = a.author_id
```

Модели Mart

```
-- This should be file mart_book_authors.sql

{{
  config(
    materialized='table',
    unique_key='author_id',
    sort='author_id'
  )
}}

WITH book_counts AS (
  SELECT
    author_id,
    COUNT(*) AS total_books
  FROM {{ ref('int_book_authors') }}
  GROUP BY author_id
)
SELECT
  author_id,
  total_books
FROM book_counts
```

Параметры dbt-clickhouse

Database ClickHouse configurations | dbt Developer Hub



Models

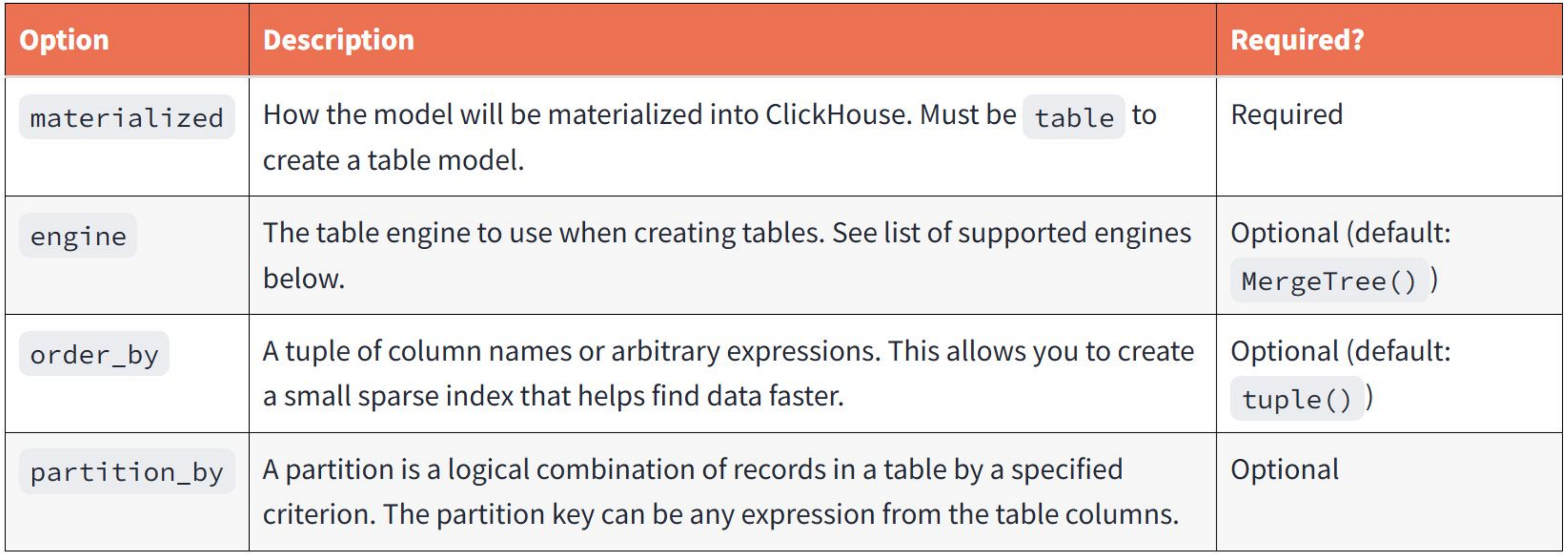
Type	Supported?	Details
view materialization	YES	Creates a view.
table materialization	YES	Creates a table. See below for the list of supported engines.
incremental materialization	YES	Creates a table if it doesn't exist, and then writes only updates to it.
ephemeral materialized	YES	Creates a ephemeral/CTE materialization. This does model is internal to dbt and does not create any database objects

Experimental models

The following are experimental features in Clickhouse:

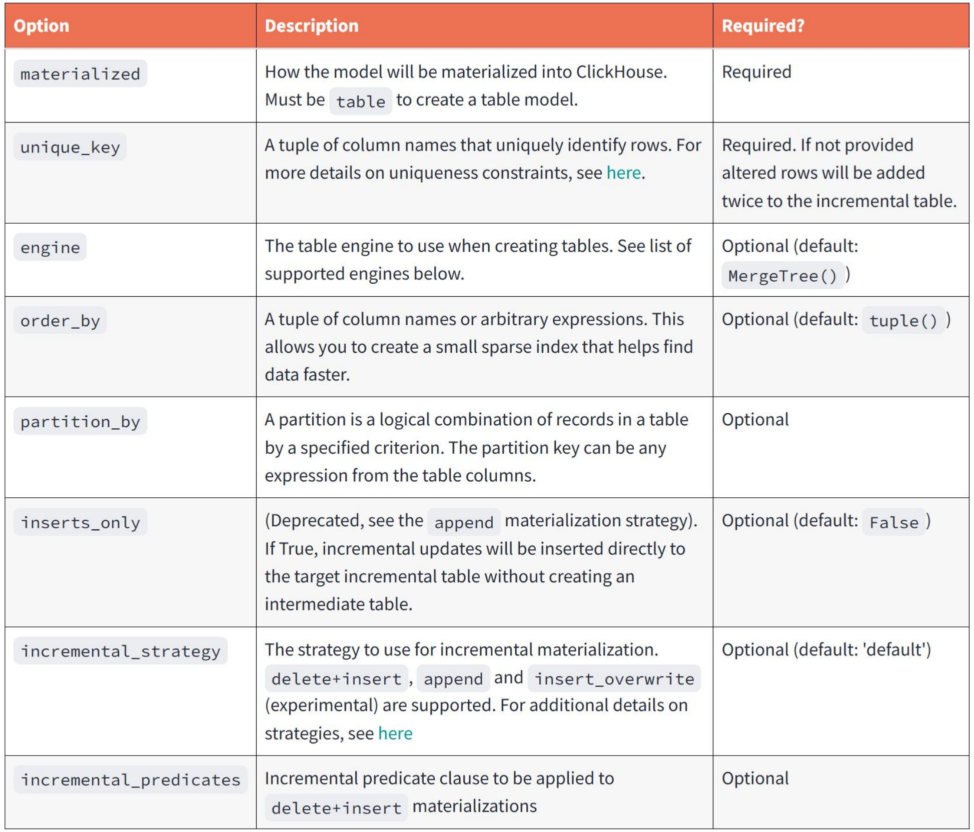
Type	Supported?	Details
Materialized View materialization	YES, Experimental	Creates a materialized view.
Distributed table materialization	YES, Experimental	Creates a distributed table.
Distributed incremental materialization	YES, Experimental	Incremental model based on the same idea as distributed table. Note that not all strategies are supported, visit this for more info.
Dictionary materialization	YES, Experimental	Creates a dictionary.

Table configurations



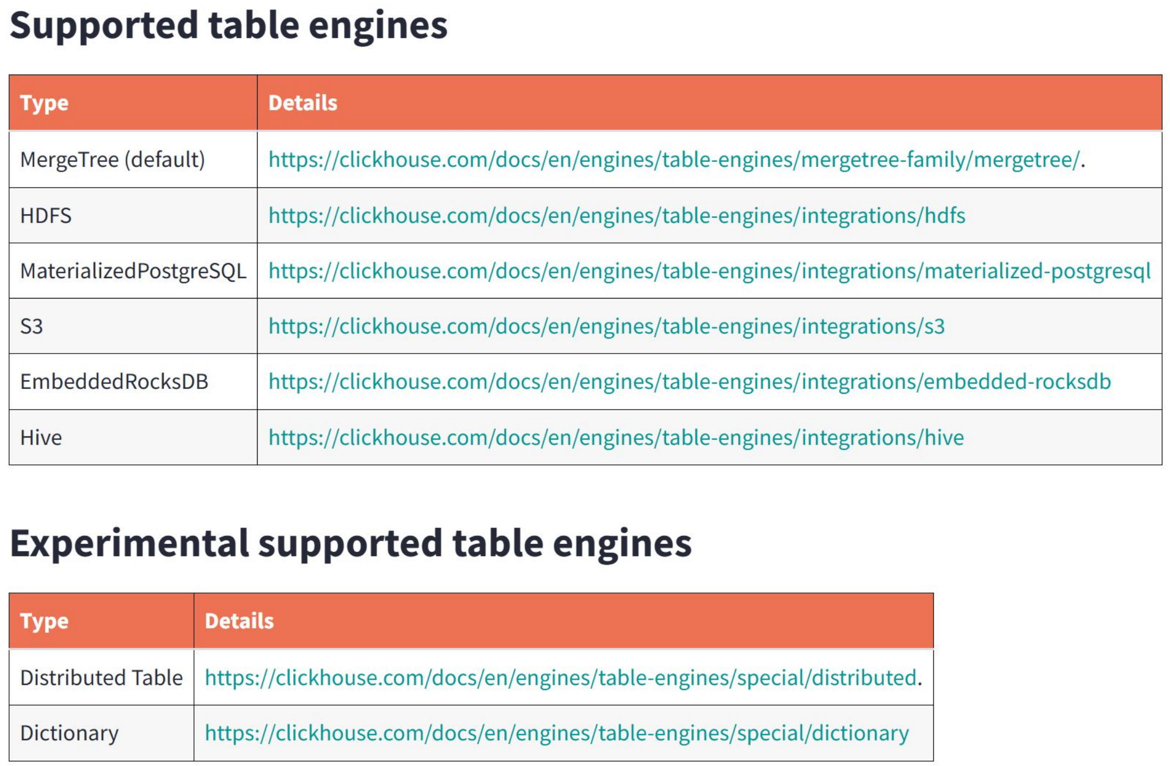
Option	Description	Required?
`materialized`	How the model will be materialized into ClickHouse. Must be `table` to create a table model.	Required
`engine`	The table engine to use when creating tables. See list of supported engines below.	Optional (default: `MergeTree()`)
`order_by`	A tuple of column names or arbitrary expressions. This allows you to create a small sparse index that helps find data faster.	Optional (default: `tuple()`)
`partition_by`	A partition is a logical combination of records in a table by a specified criterion. The partition key can be any expression from the table columns.	Optional

Incremental table configurations



Option	Description	Required?
`materialized`	How the model will be materialized into ClickHouse. Must be `table` to create a table model.	Required
`unique_key`	A tuple of column names that uniquely identify rows. For more details on uniqueness constraints, see here.	Required. If not provided altered rows will be added twice to the incremental table.
`engine`	The table engine to use when creating tables. See list of supported engines below.	Optional (default: `MergeTree()`)
`order_by`	A tuple of column names or arbitrary expressions. This allows you to create a small sparse index that helps find data faster.	Optional (default: `tuple()`)
`partition_by`	A partition is a logical combination of records in a table by a specified criterion. The partition key can be any expression from the table columns.	Optional
`inserts_only`	(Deprecated, see the `append` materialization strategy). If True, incremental updates will be inserted directly to the target incremental table without creating an intermediate table.	Optional (default: `False`)
`incremental_strategy`	The strategy to use for incremental materialization. `delete+insert` , `append` and `insert_overwrite` (experimental) are supported. For additional details on strategies, see here	Optional (default: 'default')
`incremental_predicates`	Incremental predicate clause to be applied to `delete+insert` materializations	Optional

Table engines



Supported table engines

Type	Details
MergeTree (default)	https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree/.
HDFS	https://clickhouse.com/docs/en/engines/table-engines/integrations/hdfs
MaterializedPostgreSQL	https://clickhouse.com/docs/en/engines/table-engines/integrations/materialized-postgresql
S3	https://clickhouse.com/docs/en/engines/table-engines/integrations/s3
EmbeddedRocksDB	https://clickhouse.com/docs/en/engines/table-engines/integrations/embedded-rocksdb
Hive	https://clickhouse.com/docs/en/engines/table-engines/integrations/hive

Experimental supported table engines

Type	Details
Distributed Table	https://clickhouse.com/docs/en/engines/table-engines/special/distributed.
Dictionary	https://clickhouse.com/docs/en/engines/table-engines/special/dictionary

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Список материалов для изучения

1. dbt Getting Started Tutorial
2. dbt Documentation
3. dbt FAQ
4. How we structure our dbt projects
5. The Modern Data Stack: Past, Present, and Future
6. Five principles that will keep your data warehouse organized
7. The Analytics Engineering Guide

Делитесь своими материалами в telegram

Рефлексия

Рефлексия

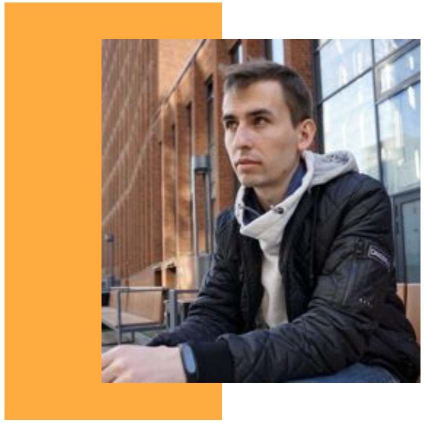
С какими впечатлениями уходите с вебинара?

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

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары



Алексей Железной

Senior Data Engineer *в Wildberries*
Магистратура - ФКН ВШЭ

Руководитель курсов ***DWH Analyst, ClickHouse для инженеров и архитекторов БД*** *в OTUS*

Преподаватель курсов ***Data Engineer, DWH Analyst, PostgreSQL*** *и пр. в OTUS*

LinkedIn