HW5

Запуск

1) добавление конфигов в `confid.d`:

- `macros.xml`:

```
<clickhouse>
    <macros>
        <shard>1</shard>
        <replica>1</replica>
    </macros>
</clickhouse>
```

- `zookeeper.xml`:

```
<clickhouse>
    <zookeeper>
        <node>
            <host>st1</host>
            <port>2181</port>
        </node>
        <session_timeout_ms>30000</session_timeout_ms>
        <operation_timeout_ms>10000</operation_timeout_ms>
    </zookeeper>
    <!-- Allow to execute distributed DDL queries (CREATE, DROP, ALTER, RENAME) on cluster.
    <!-- Works only if ZooKeeper is enabled. Comment it out if such functionality isn't requ
    <distributed_ddl>
        <!-- Path in ZooKeeper to queue with DDL queries -->
        <path>/clickhouse/task_queue/ddl</path>
    </distributed_ddl>
</clickhouse>
```

2) Запуск контейнера `docker compose up --build -d`

3) Проверка корректность работы кластера ClickHouse совместно с ZooKeeper:

```
docker exec -it clickhouse clickhouse-client
```

```
SELECT *
FROM system.zookeeper
WHERE path = '/';
```

Вывод:

```
Query id: 530df58c-3d67-41fb-a841-3ca444a0167c

    name     value  path
1.  zookeeper          /
```

2. clickhouse /

2 rows in set. Elapsed: 0.002 sec.

4) ПОдключение DBeaver:

- Можно прокинуть порт по ssh если машина не в локальной сети: `ssh -L 8123:localhost:8123 USER@HOST`
- параметры подключения:

- Host: localhost (IP)
- Port: 8123
- Database: default
- User name: admin
- Password: admin

Задания:

Упражнение 1. Запуск и проверка подключения

1. Создать базу данных:

```
CREATE DATABASE IF NOT EXISTS hw1_mt;
SHOW DATABASES;
```

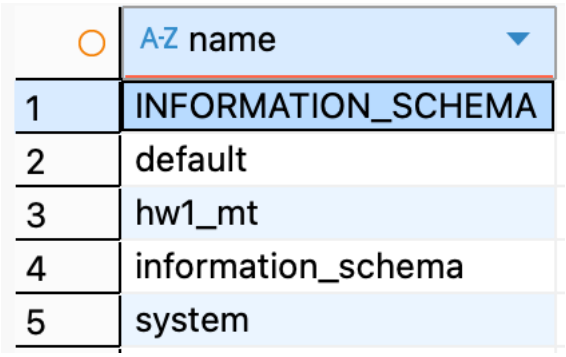


Figure 1: ER diagram

2. В созданной базе данных создать таблицу с движком MergeTree.

```
CREATE TABLE hw1_mt.events
(
    int_val   UInt32,
    uuid_val  UUID,
    dt_val    DateTime,
    str_val   LowCardinality(String)
)
```

```
ENGINE = MergeTree
ORDER BY (dt_val);
```

3. В таблице должно быть не менее четырёх столбцов, при этом необходимо использовать различные типы данных (каждый столбец — своего типа).

```
DESCRIBE TABLE hw1_mt.events;
```

	A-Z name	A-Z type
1	int_val	UInt32
2	uuid_val	UUID
3	dt_val	DateTime
4	str_val	LowCardinality(String)

Figure 2: ER diagram

4. Заполнить таблицу тестовыми данными (генерацию данных можно выполнить по примеру, приведенному в приложении 1 или по аналогии).

```
INSERT INTO hw1_mt.events
SELECT
    q.int_val,
    q.uuid_val,
    q.dt_val,
    q.str_val
FROM
(
    SELECT
        modulo(rand(), 999) + 1 AS int_val,
        generateUUIDv4() AS uuid_val,
        now() - INTERVAL rand() / 1000 SECOND AS dt_val,
        rand() / 500000 AS int_val_2,
        multiIf(
            int_val_2 <= 1500, 'A',
            int_val_2 <= 3000, 'B',
            int_val_2 <= 4500, 'C',
            int_val_2 <= 6000, 'D',
            int_val_2 <= 7300, 'E',
            'F'
        ) AS str_val
    FROM numbers(2_000_000)
) q;

SELECT *
```

```
FROM hw1_mt.events
LIMIT 10;
```

int_val	uuid_val	dt_val	str_val
267	c9ab6e15-1821-4fec-b54e-c73a4dacbf79	2025-10-27 19:54:36	F
346	a56a7ec3-4be7-4ded-aaf7-cdce402580c8	2025-10-27 19:54:40	F
780	3a5e35ea-0f88-4781-abeb-e1c175d9abeb	2025-10-27 19:54:42	F
292	d49e8130-031a-4120-a05d-38913f3ded23	2025-10-27 19:54:43	F
318	0c7f2991-18c8-4857-926b-62a96cc4d1bd	2025-10-27 19:54:43	F
885	ce5e63fa-4978-46b4-a408-deb9f1d0a203	2025-10-27 19:54:52	F
501	025b9b98-35b6-4b37-bb4a-9d95bf2a8956	2025-10-27 19:54:53	F
76	da0d159a-cd0c-4d70-b878-8c3a3b262704	2025-10-27 19:54:56	F
958	7b717ad1-b5a4-4402-a252-c9a9c3fb601d	2025-10-27 19:55:07	F
488	9abf9c4c-4597-4d90-9f82-584505d72a1d	2025-10-27 19:55:09	F

Figure 3: ER diagram

5. Выполнить любой аналитический запрос к созданной таблице, который включает: – группировку по одному из столбцов; – использование агрегатных функций (например, подсчёт количества уникальных значений и т.п.).

```
SELECT
    str_val,
    count() AS total_rows,
    uniq(int_val) AS uniq_int_vals,
    min(dt_val) AS min_time,
    max(dt_val) AS max_time
FROM hw1_mt.events
GROUP BY str_val
ORDER BY total_rows DESC;
```

	str_val	total_rows	uniq_int_vals	min_time	max_time
1	A	350,086	999	2025-12-07 20:37:25	2025-12-16 12:57:20
2	C	349,479	999	2025-11-20 11:57:26	2025-11-29 04:17:18
3	D	348,896	999	2025-11-11 19:37:24	2025-11-20 11:57:21
4	B	348,881	999	2025-11-29 04:17:27	2025-12-07 20:37:18
5	E	302,639	999	2025-11-04 07:04:06	2025-11-11 19:37:19
6	F	300,019	999	2025-10-27 19:54:36	2025-11-04 07:04:00

Figure 4: ER diagram

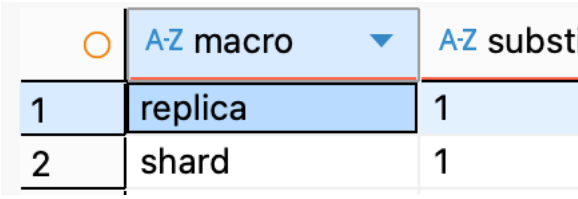
6. Проверить содержимое следующих системных таблиц ClickHouse:

   – `system.clusters`:

   ```
   SELECT *
    FROM system.clusters;
   ```

	A-Z cluster	123 shard_num	A-Z shard_name	123 shard_weight	123 internal_replication	123 replica_num
1	default	1		1	0	1

Figure 5: ER diagram

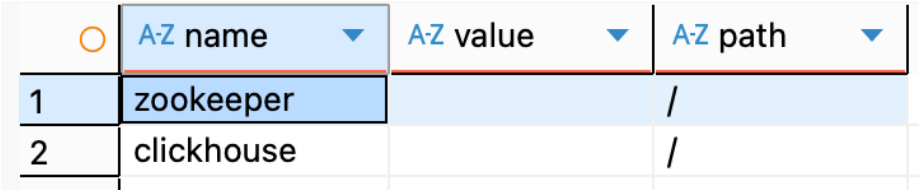


	A-Z macro	A-Z subst
1	replica	1
2	shard	1

```
– system.macros; SELECT *    FROM system.macros;
```

```
– system.zookeeper;
```

```
SELECT *
 FROM system.zookeeper
 WHERE path = '/';
```



	A-Z name	A-Z value	A-Z path
1	zookeeper		/
2	clickhouse		/

Figure 6: ER diagram

```
– system.distributed_ddl_queue;
```

```
SELECT *
 FROM system.distributed_ddl_queue;
```

Пусто

```
– system.replication_queue;
```

```
SELECT *
 FROM system.replication_queue;
```

Пусто

```
– system.trace_log.
```

```
SELECT *
 FROM system.trace_log;
```

7. Проверить работу следующих функций: – `getMacro` (получение значения макроса из конфигурации);

```
SELECT
    getMacro('shard')   AS shard,
```

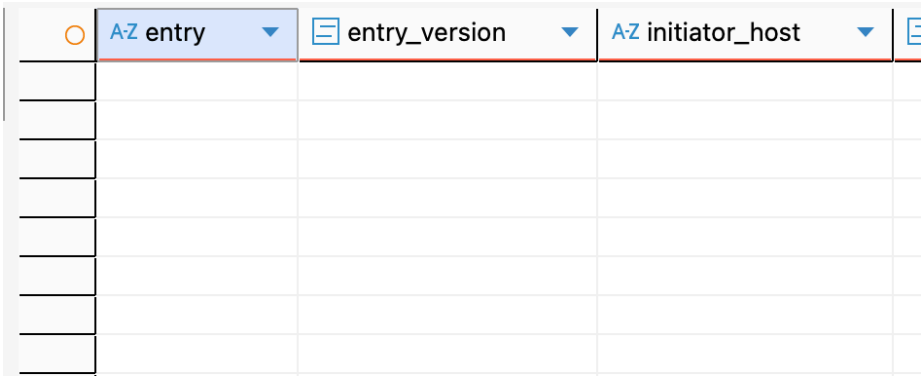


Figure 7: ER diagram

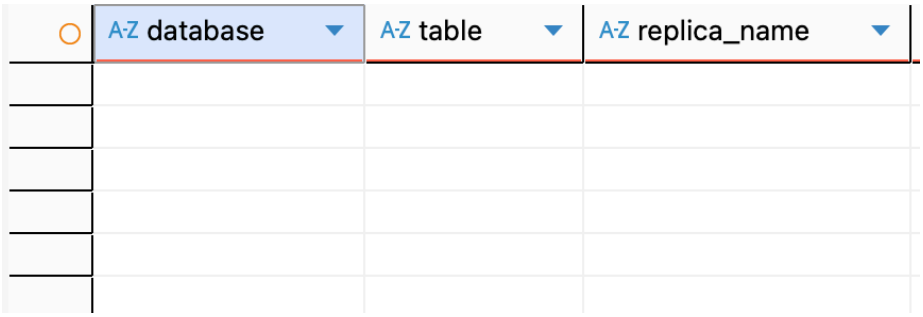
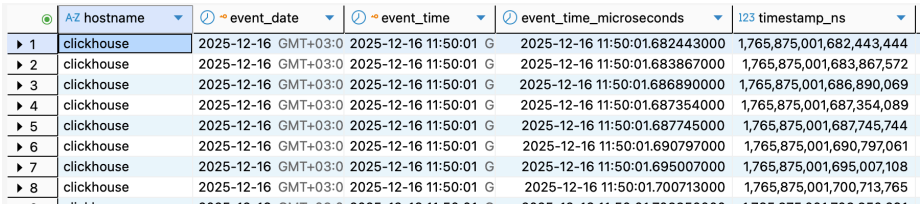


Figure 8: ER diagram



	A-Z hostname	event_date	event_time	event_time_microseconds	123 timestamp_ns
▸ 1	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.682443000	1,765,875,001,682,443,444
▸ 2	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.683867000	1,765,875,001,683,867,572
▸ 3	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.686890000	1,765,875,001,686,890,069
▸ 4	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.687354000	1,765,875,001,687,354,089
▸ 5	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.687745000	1,765,875,001,687,745,744
▸ 6	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.690797000	1,765,875,001,690,797,061
▸ 7	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.695007000	1,765,875,001,695,007,108
▸ 8	clickhouse	2025-12-16 GMT+03:0	2025-12-16 11:50:01 G	2025-12-16 11:50:01.700713000	1,765,875,001,700,713,765

Figure 9: ER diagram

```
getMacro('replica') AS replica;
```

Figure 10: ER diagram

– `clusterAllReplicas` (использование функции для распределённых запросов по репликам кластера).

```
SELECT *
FROM clusterAllReplicas('default', system.one);
```

Одна нода -> одна запись

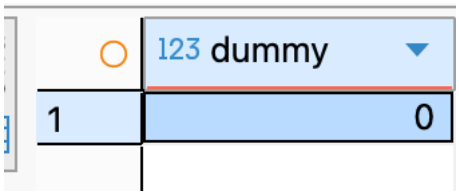


Figure 11: ER diagram

8. С использованием системных таблиц проанализировать и зафиксировать следующие метрики: – ;

```
SELECT
    query,
    query_duration_ms,
    read_rows,
    read_bytes,
    memory_usage
FROM system.query_log
WHERE
    type = 'QueryFinish'
    AND query LIKE '%FROM hw1_mt.events%'
ORDER BY event_time DESC
LIMIT 5;
```

– объём дискового пространства, занимаемый таблицей: в сжатом виде; в несжатом виде; объём, занимаемый первичным индексом;

```
SELECT
    table,
    round(sum(bytes_on_disk) / 1024 / 1024, 2) AS compressed_mb,
```

	query	query_duration_ms	read_rows	read_bytes	memory_usage
1	SELECT¶ query,¶ query_duration_ms,¶ re	2	243	79,251	8,904,350
2	SELECT¶ query,¶ query_duration_ms,¶ re	2	240	84,291	7,871,297
3	SELECT¶ query,¶ query_duration_ms,¶ re	2	236	71,917	6,826,404
4	SELECT¶ str_val,¶ count() AS total_rows,¶	6	2,000,000	18,000,000	27,000,169
5	SELECT¶ str_val,¶ count() AS total_rows,¶	6	2,000,000	18,000,000	27,000,329

Figure 12: ER diagram

```
    round(sum(data_uncompressed_bytes) / 1024 / 1024, 2) AS uncompressed_mb,
    sum(primary_key_bytes_in_memory) AS primary_key_b
FROM system.parts
WHERE database = 'hw1_mt'
AND table = 'events'
GROUP BY table;
```

	table	compressed_mb	uncompressed_mb	primary_key_b
1	events	43.07	47.69	988

Figure 13: ER diagram

– объём дискового пространства по каждому столбцу таблицы: в сжатом виде; в несжатом виде.

```
SELECT
    column,
    round(sum(column_bytes_on_disk) / 1024 / 1024, 3) AS compressed_mb,
    round(sum(column_data_uncompressed_bytes) / 1024 / 1024, 3) AS uncompressed_mb
FROM system.parts_columns
WHERE database = 'hw1_mt'
AND table = 'events'
GROUP BY column
ORDER BY compressed_mb DESC;
```

	column	compressed_mb	uncompressed_mb
1	uuid_val	30.644	30.518
2	dt_val	6.931	7.629
3	int_val	5.482	7.629
4	str_val	0.01	1.911

Figure 14: ER diagram

Упражнение 2. Размер вставки

9. Создать две таблицы с движком MergeTree.

   создаем таблицы:

```
CREATE TABLE hw1_mt.mt_small
(
    int_val   UInt32,
    uuid_val  UUID,
    dt_val    DateTime,
    str_val   LowCardinality(String)
)
ENGINE = MergeTree
ORDER BY dt_val;


CREATE TABLE hw1_mt.mt_buffered
(
    int_val   UInt32,
    uuid_val  UUID,
    dt_val    DateTime,
    str_val   LowCardinality(String)
)
ENGINE = MergeTree
ORDER BY dt_val;
```

10. Для второй таблицы создать дополнительную таблицу с движком Buffer, через которую будет выполняться вставка данных.

```
CREATE TABLE hw1_mt.mt_buffer
(
    int_val   UInt32,
    uuid_val  UUID,
    dt_val    DateTime,
    str_val   LowCardinality(String)
)
ENGINE = Buffer(
    hw1_mt,
    mt_buffered,
    16,   #
    10,   #
    60,   #
    100000,  #
    1000000,  #
    10000000, #
    100000000 #
);
```

11. С помощью скрипта запустить одновременную автоматическую вставку записей в обе таблицы. Вставка в первую таблицу MergeTree должна выполняться мелкими пачками.
    Вставка во вторую таблицу MergeTree должна выполняться крупными

пачками с использованием буферизации через таблицу с движком Buffer.

Параллельный запуск:

```
#!/bin/bash

CLICKHOUSE="docker exec -i clickhouse clickhouse-client -u admin --password admin"

insert_small() {
for i in {1..300}; do
    $CLICKHOUSE <<'SQL'
INSERT INTO hw1_mt.mt_small
SELECT
    modulo(rand(), 999) + 1,
    generateUUIDv4(),
    now() - INTERVAL rand() / 1000 SECOND,
    multiIf(
        rand() / 500000 <= 1500, 'A',
        rand() / 500000 <= 3000, 'B',
        rand() / 500000 <= 4500, 'C',
        rand() / 500000 <= 6000, 'D',
        rand() / 500000 <= 7300, 'E',
        'F'
    )
FROM numbers(5000);
SQL
    sleep 0.5
done
}

insert_buffer() {
for i in {1..300}; do
    $CLICKHOUSE <<'SQL'
INSERT INTO hw1_mt.mt_buffer
SELECT
    modulo(rand(), 999) + 1,
    generateUUIDv4(),
    now() - INTERVAL rand() / 1000 SECOND,
    multiIf(
        rand() / 500000 <= 1500, 'A',
        rand() / 500000 <= 3000, 'B',
        rand() / 500000 <= 4500, 'C',
        rand() / 500000 <= 6000, 'D',
        rand() / 500000 <= 7300, 'E',
        'F'
    )
FROM numbers(5000);
```

```
SQL
    sleep 0.5
done
}

insert_small & insert_buffer & wait
```

Итог:

```
SELECT
    table,
    count() AS parts,
    sum(rows) AS rows
FROM system.parts
WHERE database = 'hw1_mt'
AND table IN ('mt_small', 'mt_buffered')
AND active
GROUP BY table;
```

```
SELECT
    table,
    count() AS parts,
    sum(rows) AS rows
FROM system.parts
WHERE database = 'hw1_mt'
AND table IN ('mt_small', 'mt_buffered')
AND active
GROUP BY table;
```

	A-Z table	123 parts	123 rows
1	mt_small	8	3,100,000
2	mt_buffered	7	3,100,000

Figure 15: ER diagram

Количество активных и неактивных партиций:

```
SELECT
    table,
    active,
    count() AS parts_count,
    sum(rows) AS rows_count
FROM system.parts
WHERE database = 'hw1_mt'
AND table IN ('mt_small', 'mt_buffered')
```

```
GROUP BY table, active
ORDER BY table, active DESC;
```

	A-Z table	123 active	123 parts_count	123 rows_count
1	mt_buffered	1	7	3,100,000
2	mt_buffered	0	54	3,840,000
3	mt_small	1	8	3,100,000
4	mt_small	0	356	7,460,000

Figure 16: ER diagram

Упражнение 3. Оптимизация Order by

15. Создать таблицу в соответствии с выданным описанием (см. скрипт в приложении).

```
DROP TABLE IF EXISTS default.person_data;

CREATE TABLE default.person_data (
id          UInt64,
region      LowCardinality(String),
date_birth  Date,
gender      UInt8,
is_marital  UInt8,
dt_create   DateTime DEFAULT now()
)
ENGINE = MergeTree()
ORDER BY (date_birth);
```

16. Заполнить таблицу сгенерированными данными, используя предоставленный скрипт.

```
INSERT INTO default.person_data(id, region, date_birth, gender, is_marital)
SELECT
    q.id,
    q.region,
    toStartOfDay(q.date_birth) AS date_birth,
    q.gender,
    q.is_marital
FROM
(
    SELECT
        rand() AS id,
        modulo(id, 70) + 20 AS n,
        toString(n) AS region,
        floor(randNormal(10000, 1700)) AS k,
```

```
        toDate('1970-01-01') + INTERVAL k DAY AS date_birth,
        if(modulo(id, 3) = 1, 1, 0) AS gender,
        if((n + k) % 3 = 0 AND dateDiff('year', date_birth, now()) > 18, 1, 0) AS is_ma
    FROM numbers(100_000_000)
) q;
```

	id	region	date_birth	gender	is_marital	dt_create
1	3,760,627,853	23	1971-02-03	0	0	2025-12-16 14:18:12
2	502,986,895	25	1972-01-08	1	1	2025-12-16 14:18:12
3	2,906,785,379	49	1972-02-26	0	0	2025-12-16 14:18:12
4	1,557,838,636	66	1972-04-10	1	0	2025-12-16 14:18:12
5	2,160,920,827	57	1972-08-22	1	0	2025-12-16 14:18:12
6	3,628,025,648	78	1972-11-27	0	0	2025-12-16 14:18:12
7	366,571,897	47	1972-12-17	1	1	2025-12-16 14:18:12
8	342,652,889	39	1973-07-07	0	0	2025-12-16 14:18:12
9	601,640,193	83	1973-12-14	0	0	2025-12-16 14:18:12
10	3,251,701,488	48	1974-01-03	0	0	2025-12-16 14:18:12
11	4,237,862,525	35	1974-03-21	0	1	2025-12-16 14:18:12
12	3,080,305,178	68	1974-04-17	0	1	2025-12-16 14:18:12
13	4,237,695,974	84	1974-04-26	0	0	2025-12-16 14:18:12
14	431,455,649	29	1974-06-10	0	1	2025-12-16 14:18:12

Figure 17: ER diagram

17. Выполнить оптимизацию таблицы таким образом, чтобы в результате остался только один кусок (одна партиция).

```
OPTIMIZE TABLE default.person_data FINAL;
```

```
SELECT
    count() AS active_parts,
    sum(rows) AS rows
FROM system.parts
WHERE database = 'default'
AND table = 'person_data'
AND active;
```

	active_parts	rows
1	1	100,000,000

Figure 18: ER diagram

18. Выполнить указанные запросы (см. приложение) и для каждого запроса: – определить среднее время выполнения на серии запусков (не менее 5); – зафиксировать количество обработанных строк (параметр PROCESSED, возвращаемый clickhouse-client); – проверить объём памяти, занимаемый первичным ключом таблицы.

- объем памяти:

```
SELECT
    table,
    round(sum(primary_key_bytes_in_memory) / 1024 / 1024, 2) AS primary_key_mb
FROM system.parts
WHERE database = 'default'
AND table = 'person_data'
AND active
GROUP BY table;
```

```
![ER diagram](./report_data/task_18_0.png)
```

a) Первый запрос:

```
SELECT t.region,
    countIf(gender = 1 AND date_diff('year', t.date_birth, now()) BETWEEN 20 AND 40) AS cnt
    countIf(gender = 0 AND date_diff('year', t.date_birth, now()) BETWEEN 18 AND 30) AS cnt
FROM default.person_data t
WHERE t.date_birth BETWEEN toDate('2000-01-01') AND toDate('2000-01-31')
AND t.region IN ('20', '25', '43', '59')
GROUP BY t.region;
```

Вывод: - **Processed 622.59 thousand rows**

```
```
     region cnt_male cnt_female
1.   43   2976   5791
2.   25   2923   5844
3.   59   2929   5870
4.   20   2953   5979

4 rows in set. Elapsed: 0.004 sec. Processed 622.59 thousand rows, 2.49 MB (176.41 million r
Peak memory usage: 4.12 MiB.
```
```

- Среднее время = 3 ms.

```
SELECT
    query_duration_ms,
    read_rows,
    read_bytes
FROM system.query_log
WHERE query LIKE '%FROM default.person_data%'
AND type = 'QueryFinish'
ORDER BY event_time DESC
LIMIT 5;
```

b) Второй запрос:

	123 query_duration_ms	123 read_rows	123 read_bytes
1	3	622,592	2,490,368
2	3	622,592	2,490,368
3	3	622,592	2,490,368
4	3	622,592	2,490,368
5	3	622,592	2,490,368

Figure 19: ER diagram

```
SELECT countIf(gender = 1 AND date_diff('year', t.date_birth, now()) BETWEEN 20 AND 40) AS c
    countIf(gender = 0 AND date_diff('year', t.date_birth, now()) BETWEEN 18 AND 30) AS cnt_
FROM default.person_data t
WHERE t.is_marital = 1
AND t.region IN ('80')
GROUP BY t.region;
```

Вывод: - **Processed 97.47 million rows**

```
```
 cnt_male cnt_female
1.   152916   214788

1 row in set. Elapsed: 0.034 sec. Processed 97.47 million rows, 347.23 MB (2.84 billion rows
Peak memory usage: 17.85 MiB.
```
```

- среднее время = 33.6 ms.

```
SELECT
query_duration_ms,
read_rows,
read_bytes
FROM system.query_log
WHERE query LIKE '%FROM default.person_data%'
AND type = 'QueryFinish'
ORDER BY event_time DESC
LIMIT 5;
```

19. Оптимизировать ORDER BY и PRIMARY KEY таблицы так, чтобы: – объём памяти, занимаемый первичным индексом, был минимален; – метрики выполнения запросов (время и объём обрабатываемых данных) улучшились.

    - СОздаем таблицу, меняем ORDER BY на region:

```
CREATE TABLE default.person_data_opt (
```

	123 query_duration_ms	123 read_rows	123 read_bytes
1	34	97,468,416	347,232,765
2	34	97,468,416	347,232,765
3	33	97,468,416	347,232,765
4	33	97,468,416	347,232,765
5	34	97,468,416	347,232,765

Figure 20: ER diagram

```
id          UInt64,
region      LowCardinality(String),
date_birth  Date,
gender      UInt8,
is_marital  UInt8,
dt_create   DateTime
)
ENGINE = MergeTree()
ORDER BY (region, date_birth);

INSERT INTO default.person_data_opt
SELECT *
FROM default.person_data;
```

a) Первый запрос:

```
SELECT t.region,
    countIf(gender = 1 AND date_diff('year', t.date_birth, now()) BETWEEN 20 AND 40) AS
    countIf(gender = 0 AND date_diff('year', t.date_birth, now()) BETWEEN 18 AND 30) AS
FROM default.person_data_opt t
WHERE t.date_birth BETWEEN toDate('2000-01-01') AND toDate('2000-01-31')
AND t.region IN ('20', '25', '43', '59')
GROUP BY t.region;
```

Вывод:

- **Processed 65.54 thousand rows** меньше чем **Processed 622.59 thousand rows**
- **817.39 KiB.** меньше чем **4.12 MiB**

```
    region  cnt_male  cnt_female
1.  43          2976        5791
2.  25          2923        5844
3.  59          2929        5870
4.  20          2953        5979
```

```
4 rows in set. Elapsed: 0.004 sec. Processed 65.54 thousand rows, 262.19 KB (15.14
Peak memory usage: 817.39 KiB.
```

– Среднее время = 3 ms.

```
SELECT
    query_duration_ms,
    read_rows,
    read_bytes
FROM system.query_log
WHERE query LIKE '%FROM default.person_data_opt%'
AND type = 'QueryFinish'
ORDER BY event_time DESC
LIMIT 5;
```

	123 query_duration_ms	123 read_rows	123 read_bytes
1	4	65,536	262,192
2	3	65,536	262,192
3	3	65,536	262,192
4	3	65,536	262,192
5	3	65,536	262,192

Figure 21: ER diagram

b) Второй запрос:

```
SELECT countIf(gender = 1 AND date_diff('year', t.date_birth, now()) BETWEEN 20 AND
    countIf(gender = 0 AND date_diff('year', t.date_birth, now()) BETWEEN 18 AND 30
FROM default.person_data_opt t
WHERE t.is_marital = 1
AND t.region IN ('80')
GROUP BY t.region;
```

Вывод:

- **Processed 1.50 million rows** меньше чем **Processed 97.47 million rows**
- **3.94 MiB.** меньше чем **17.85 MiB**

```
 cnt_male  cnt_female
1.     152916       214788


1 row in set. Elapsed: 0.006 sec. Processed 1.50 million rows, 7.43 MB (261.67 mill
Peak memory usage: 3.94 MiB.
```

- среднее время = 5 ms.

```
SELECT
```

```
    query_duration_ms,
    read_rows,
    read_bytes
FROM system.query_log
WHERE query LIKE '%FROM default.person_data%'
AND type = 'QueryFinish'
ORDER BY event_time DESC
LIMIT 5;
```

	123 query_duration_ms	123 read_rows	123 read_bytes
1	6	1,499,136	7,427,054
2	4	1,499,136	7,427,054
3	5	1,499,136	7,427,054
4	5	1,499,136	7,427,054
5	5	1,499,136	7,427,054

Figure 22: ER diagram

Упражнение 4. Сжатие данных

20-22. На основании таблицы, полученной в результате выполнения задания 2, создать новую таблицу с той же структурой и теми же данными. Подобрать кодеки сжатия для столбцов новой таблицы таким образом, чтобы выполнялись следующие условия: • общий объём, занимаемый таблицей на диске, уменьшился не менее чем на 25 % по сравнению с исходной таблицей; • среднее время выполнения обоих рассматриваемых запросов не увеличилось более чем на 10–15 %, при этом желательно, чтобы оно улучшилось.

- Создаем таблицу с кодеками:

```
CREATE TABLE default.person_data_codec (
    id          UInt64 CODEC(ZSTD(3)),
    region      LowCardinality(String) CODEC(LZ4),
    date_birth  Date CODEC(Delta, LZ4),
    gender UInt8,
    is_marital UInt8,
    dt_create   DateTime CODEC(Delta, LZ4)
)
ENGINE = MergeTree()
ORDER BY (date_birth);
```

- Переносим данные со старой таблицы и сливаем партиции:

```
INSERT INTO default.person_data_codec
SELECT *
FROM default.person_data;
```

```
OPTIMIZE TABLE default.person_data FINAL;
OPTIMIZE TABLE default.person_data_codec FINAL;
```

- Замеряем объем

```
SELECT
    table,
    round(sum(bytes_on_disk) / 1024 / 1024, 2) AS size_mb
FROM system.parts
WHERE database = 'default'
  AND table IN ('person_data', 'person_data_codec')
  AND active
GROUP BY table;
```

Улучшение на 24% по памяти

	table	size_mb
1	person_data	788.65
2	person_data_codec	600.76

а) Первый запрос - без изменений:

	query_duration_ms
1	3
2	3
3	3
4	3
5	3

Figure 23: ER diagram

б) Второй запрос - ухудшение на 10%:

	123 query_duration_ms	123 read_rows	123 read_bytes
1	37	97,468,416	347,232,162
2	36	97,468,416	347,232,162
3	37	97,468,416	347,232,162
4	37	97,468,416	347,232,162
5	38	97,468,416	347,232,162

Figure 24: ER diagram