Использование Clickhouse для обработки данных по оценке лояльности клиентов и визуализация результатов в BI-системе

Проект учебной курсовой OTUS «Clickhouse для инженеров и архитекторов БД»

Цель: построить бизнес-приложение базе хранилища данных Clickhouse для анализа клиентских метрик CSI (уровень удовлетворенности клиентов), NPS (индекс лояльности клиентов) и CES

Оглавление

Состав и технологии:

1. PostgreSQL: Содержатся и поступают новые результаты опросов клиентов;

2. Clickhouse: обработка данных, хранение результатов обработки, агрегаций и витрин для аналитики в BI;

3. PixBI: система бизнес-анализа и визуализации данных.

PostgreSQL: Датасеты и генерация данных

Создание и заполнение таблицы с каналами опросов — chanels

```
CREATE TABLE chanels (
    id INTEGER PRIMARY KEY,
    chanel VARCHAR(50) NOT NULL
);
insert into chanels (id, chanel) values (1, 'chat');
insert into chanels (id, chanel) values (2, 'email');
insert into chanels (id, chanel) values (3, 'social media');
insert into chanels (id, chanel) values (4, 'phone');
insert into chanels (id, chanel) values (5, 'site');
```

	chanel_id [PK] integer	chanel character varying (50)
1	1	chat
2	2	email
3	3	social media
4	4	phone
5	5	site

Создание и заполнение таблицы с типами сервисов – services

```
CREATE TABLE services (
    id INTEGER PRIMARY KEY,
```

```
    services VARCHAR(50) NOT NULL
);
insert into services (id, services) values (1, 'request_service');
insert into services (id, services) values (2, 'phone_service');
insert into services (id, services) values (3, 'personal_cabinet_service');
insert into services (id, services) values (4, 'office_service');
insert into services (id, services) values (5, 'ai_assistant_service');
```

	service_id [PK] integer	services character varying (50)
1	1	request_service
2	2	phone_service
3	3	personal_cabinet_servi…
4	4	office_service
5	5	ai_assistant_service

Создание и заполнение таблицы с городами проведения опросов — city

```
CREATE TABLE city (
    id INTEGER PRIMARY KEY,
    city VARCHAR(50) NOT NULL
);
insert into city (id, city) values (1, 'Москва');
insert into city (id, city) values (2, 'Санкт-Петербург');
insert into city (id, city) values (3, 'Калининград');
insert into city (id, city) values (4, 'Хабаровск');
insert into city (id, city) values (5, 'Сочи');
insert into city (id, city) values (6, 'Екатеринбург');
insert into city (id, city) values (7, 'Омск');
insert into city (id, city) values (8, 'Уфа');
insert into city (id, city) values (9, 'Воронеж');
insert into city (id, city) values (10, 'Нижний Новгород');
```

```
insert into city (id, city) values (11, 'Великий Новгород');
insert into city (id, city) values (12, 'Ярославль');
insert into city (id, city) values (13, 'Владивосток');
insert into city (id, city) values (14, 'Петрозаводск');
insert into city (id, city) values (15, 'Выборг');
insert into city (id, city) values (16, 'Новороссийск');
```

	city_id [PK] integer	city character varying (50)
1	1	Москва
2	2	Санкт-Петербург
3	3	Калининград
4	4	Хабаровск
5	5	Сочи
6	6	Екатеринбург
7	7	Омск
8	8	Уфа
9	9	Воронеж
10	10	Нижний Новгород

Создание и заполнение таблицы с ответами клиентов — answers

```
create table answers (
    respondent_id INTEGER PRIMARY KEY,
    gender VARCHAR(20) NOT NULL,
    answer_date_time TIMESTAMP NOT NULL,
    birth_date DATE NOT NULL,
    city_id INTEGER NOT NULL,
    chanel_id INTEGER NOT NULL,
    service_id INTEGER NOT NULL,
    csi INTEGER CHECK (csi BETWEEN 1 AND 10),
    ces INTEGER CHECK (ces BETWEEN 1 AND 7),
    nps INTEGER CHECK (nps BETWEEN 0 AND 10)
);
```

Создание индексов для answers для оптимизации запросов.

```
CREATE INDEX idx_answers_city_id ON answers(city_id);

CREATE INDEX idx_answers_chanel_id ON answers(chanel_id);

CREATE INDEX idx_answers_service_id ON answers(service_id);

CREATE INDEX idx_answers_date ON answers(answer_date_time);
```

Настраиваем параметры для вставки данных в таблицу answers.

```
SET maintenance_work_mem = '1GB';

SET max_parallel_workers = 4;

SET work_mem = '256MB';
```

Удаление индексов перед большой вставкой.

```
DROP INDEX IF EXISTS idx_answers_city_id, idx_answers_chanel_id, idx_answers_service_id,
idx_answers_date;
```

Заполнение таблицы answers сгенерированными данными

```
DO $$

DECLARE

   batch_size INTEGER := 100000;  -- Размер пакета

   total_rows INTEGER := 20000000; -- 20 млн записей

   start_id INTEGER;

   i INTEGER;

BEGIN

--Получаем стартовый ID

   SELECT COALESCE(MAX(respondent_id), 0) INTO start_id FROM answers;

   RAISE NOTICE 'Начинаем генерацию 20 млн записей, начиная с ID: %', start_id + 1;

   RAISE NOTICE 'Размер пакета: %, всего пакетов: %', batch_size, total_rows/batch_size;

   FOR i IN 1..(total_rows/batch_size) LOOP

      INSERT INTO answers (

         respondent_id, gender, answer_date_time, birth_date,

         city_id, chanel_id, service_id, csi, ces, nps

      )

      SELECT

         -- ID
```

```
start_id + (i-1)*batch_size + generate_series(1, batch_size),
-- Пол
CASE
    WHEN random() < 0.49 THEN 'Male'
    WHEN random() < 0.49 THEN 'Female'
    ELSE 'Non-binary'
END as gender,
-- Дата ответа (последние 2 года)
NOW() - (random() * 730)::integer * INTERVAL '1 day' +
(random() * 86400)::integer * INTERVAL '1 second',
-- Дата рождения (18-80 лет)
CURRENT_DATE - (18 + random() * 62 * 365)::integer * INTERVAL '1 day',
-- Город (1-16)
(random() * 15 + 1)::integer,
-- Канал (1-5)
(random() * 4 + 1)::integer,
-- Услуга (1-5)
(random() * 4 + 1)::integer,
-- CSI с нормальным распределением
GREATEST(1, LEAST(10, (7 + (random() - 0.5) * 4)::integer)),
-- CES с нормальным распределением
GREATEST(1, LEAST(7, (4 + (random() - 0.5) * 3)::integer)),
-- NPS с реалистичным распределением
CASE
    WHEN random() < 0.15 THEN 0  -- Детракторы
    WHEN random() < 0.4 THEN floor(random() * 6)::integer  -- 1-6
    WHEN random() < 0.7 THEN 7  -- Нейтралы
    WHEN random() < 0.85 THEN 8  -- Промоутеры
    WHEN random() < 0.95 THEN 9
    ELSE 10
END;
```

```
    -- Прогресс
    IF i % 10 = 0 THEN
        RAISE NOTICE 'Обработано: % млн записей', (i * batch_size) / 1000000;
        COMMIT;
    END IF;
  END LOOP;
  RAISE NOTICE 'Генерация завершена! Всего записей: %', total_rows;
END $$;
```

Восстановление индексов

```
CREATE INDEX idx_answers_city_id ON answers(city_id);
CREATE INDEX idx_answers_chanel_id ON answers(chanel_id);
CREATE INDEX idx_answers_service_id ON answers(service_id);
CREATE INDEX idx_answers_date ON answers(answer_date_time);
ANALYZE answers;
```

Проверяем размер таблицы

```
SELECT
  pg_size_pretty(pg_total_relation_size('answers')) as total_size,
  pg_size_pretty(pg_relation_size('answers')) as table_size,
  pg_size_pretty(pg_indexes_size('answers')) as indexes_size;
```

	total_size text	table_size text	indexes_size text
1	3907 MB	1611 MB	2296 MB

Проверяем количество записей

```
SELECT COUNT(*) as total_records FROM answers;
```

	total_records bigint
1	20000000

Проверка распределения данных

```
SELECT
  COUNT(*) as total,
  MIN(respondent_id) as min_id,
  MAX(respondent_id) as max_id,
  COUNT(DISTINCT respondent_id) as unique_ids
```

FROM answers;

	total bigint	min_id integer	max_id integer	unique_ids bigint
1	20000000	1	20000000	20000000

Просматриваем вставленные в таблицу answers данные

	respondent_id [PK] integer	gender character varying (20)	answer_date_time timestamp without time zone	birth_date date	city_id integer	chanel_id integer	service_id integer	csi integer	ces integer	nps integer
1	1	Non-binary	2024-11-13 02:01:04.429038	1977-02-12	2	2	2	8	4	9
2	2	Female	2024-11-05 23:22:14.429038	1985-04-30	14	4	1	6	3	7
3	3	Male	2025-02-23 23:59:55.429038	1964-01-01	15	3	1	9	3	2
4	4	Male	2025-04-24 09:37:11.429038	2013-04-09	10	2	4	6	5	7
5	5	Female	2024-08-20 16:37:32.429038	1983-12-27	4	2	5	7	5	7
6	6	Non-binary	2024-04-17 21:15:43.429038	2006-10-17	8	2	5	7	5	8
7	7	Male	2024-02-14 21:29:53.429038	1979-11-28	5	3	2	7	4	4

Формирование хранилища данных в Clickhouse

Создание словарей - загрузка данных из PGSQL

chanel_dict_v1 — словарь каналов опросов

CREATE DICTIONARY chanel_dict_v1
(chanel_id Int64,
chanel String
)
Primary Key chanel_id
Source (Postgresql(host 'localhost' PORT 5432 USER 'myuser_bi' PASSWORD 'myuser_bi' DB 'mydb_bi' table 'chanels' schema 'public'))
Lifetime(min 300 max 600)
layout(FLAT());

Проверка загруженных значений словаря

123 chanel_id	A-Z chanel
1	chat
2	email
3	social media
4	phone
5	site
6	unknown

services_dict_v1 — словарь сервисов, по качеству которых проходили опросы

CREATE DICTIONARY services_dict_v1
(service_id Int64,
services String
)

Primary Key service_id
Source (Postgresql(host 'localhost' PORT 5432 USER 'myuser_bi' PASSWORD 'myuser_bi' DB 'mydb_bi' table 'services' schema 'public'))
Lifetime(min 300 max 600)
layout(FLAT())

Проверка загруженных значений словаря
select * from services_dict_v1

	service_id	services
1	1	request_service
2	2	phone_service
3	3	personal_cabinet_service
4	4	office_service
5	5	ai_assistant_service

city_dict_v1 — словарь городов, в которых проходили опросы

CREATE DICTIONARY city_dict_v1
(city_id Int64,
city String
)
Primary Key city_id
Source (Postgresql(host 'localhost' PORT 5432 USER 'myuser_bi' PASSWORD 'myuser_bi' DB 'mydb_bi' table 'city' schema 'public'))
Lifetime(min 300 max 600)
layout(FLAT())

Проверка загруженных значений словаря
select * from city_dict_v1

	city_id	city
1	1	Москва
2	2	Санкт-Петербург
3	3	Калининград
4	4	Хабаровск
5	5	Сочи
6	6	Екатеринбург

Проверка словарей через таблицу system.dictionaries

	database	name	uuid	status	origin	type	key.names	key.types	attribute.names	
1	BI	chanel_dict_v1	fd28c3d8-2f6f	LOADED	fd28c3d8-2f6f-4252-9ce5-8fdd0dc9d	Flat	chanel_id	UInt64	chanel	S
2	BI	services_dict_v1	29c8d131-8a6	LOADED	29c8d131-8a63-478c-b0db-4b64bfc5	Flat	service_id	UInt64	services	S
3	BI	city_dict_v1	b7a0504b-0bb	LOADED	b7a0504b-0bbf-4e9e-ab9a-f0077f518	Flat	city_id	UInt64	city	S

Создание и обновление сырой таблицы со всеми опросами BI.mv_answers

```
create TABLE IF NOT EXISTS BI.answers_raw
(
        respondent_id UInt64,
    gender String,
    answer_date_time DateTime,
    birth_date Date,
    city_id Int64,
    chanel_id Int64,
    service_id Int64,
    csi Int64,
    ces Int64,
    nps Int64
)
ENGINE = ReplacingMergeTree(respondent_id)
PARTITION BY toYYYYMM(answer_date_time)
ORDER BY (respondent_id, answer_date_time)

DROP MATERIALIZED VIEW IF EXISTS BI.mv_answers
CREATE MATERIALIZED VIEW IF NOT EXISTS BI.mv_answers
REFRESH EVERY 1 DAY
APPEND
TO BI.answers_raw as
SELECT
respondent_id,
    gender,
    answer_date_time,
    birth_date,
    city_id,
    chanel_id,
    service_id,
    csi,
    ces,
    nps
    FROM postgresql(
    'localhost:5432',
    'mydb_bi',
    'answers',
    'myuser_bi',
    'myuser_bi'
)
WHERE answer_date_time>coalesce((SELECT MAX(answer_date_time) FROM
BI.answers_raw), toDateTime('1970-01-01 00:00:00'))
;
```

Формирование MV BI.respondent_answers_wide_mv с полными данными по результатам опросов

```
CREATE MATERIALIZED VIEW IF NOT EXISTS BI.respondent_answers_wide_mv
(
    respondent_id UInt64,
    answer_date_time DateTime,
    answer_year UInt16,
    answer_YYYY_MM_DD UInt32,
    answer_YYYY_MM UInt32,
    birth_date Date,
    age UInt8,
    age_group String,
    gender String,
    chanel_name String,
    service_name String,
    city_name String,
    ces Float32,
    csi Float32,
    nps Int8,
    version DateTime
)
ENGINE = MergeTree()
PARTITION BY answer_YYYY_MM
ORDER BY (answer_date_time, chanel_name, service_name, city_name)
SETTINGS index_granularity = 8192 POPULATE
AS
SELECT
    answ.respondent_id,
    answ.answer_date_time,
    toYear(answ.answer_date_time) as answer_year,
    toYYYYMMDD(answ.answer_date_time) as answer_YYYY_MM_DD,
    toYYYYMM(answ.answer_date_time) as answer_YYYY_MM,
    answ.birth_date,
    age('year', answ.birth_date, today()) AS age,
    case
          when age('year', answ.birth_date, today()) BETWEEN 0 AND 18 then 'До 18'
          when age('year', answ.birth_date, today()) BETWEEN 19 AND 25 then 'До 25'
          when age('year', answ.birth_date, today()) BETWEEN 26 AND 35 then 'До 35'
          when age('year', answ.birth_date, today()) BETWEEN 36 AND 50 then 'До 50'
          when age('year', answ.birth_date, today()) BETWEEN 51 AND 65 then 'До 65'
          when age('year', answ.birth_date, today()) BETWEEN 66 AND 75 then 'До 75'
          else '76+'
     end as age_group,
    gender,
    dictGetString('chanel_dict_v1', 'chanel', answ.chanel_id) AS chanel_name,
    dictGetString('services_dict_v1', 'services', answ.service_id) AS service_name,
    dictGetString('city_dict_v1', 'city', answ.city_id) AS city_name,
    ces,
    csi,
```

```
    nps,
    now() AS version
FROM BI.mv_answers AS answ;
```

Формирование агрегированного MV BI.csi_metrics_mv с расчетами метрик CSI

```
CREATE MATERIALIZED VIEW IF NOT EXISTS BI.csi_metrics_mv
ENGINE = AggregatingMergeTree()
PARTITION BY answer_YYYY_MM
ORDER BY (answer_YYYY_MM, answer_year, chanel_name, service_name, age_group)
POPULATE
AS
SELECT
answer_YYYY_MM,
    answer_year,
    chanel_name,
    service_name,
    age_group,
    countIf(csi >= 8) as csi_satisfied_count,
    count(*) as total_responses,
    round(csi_satisfied_count * 100.0 / total_responses, 1) as csi_score
FROM BI.respondent_answers_wide_mv
GROUP BY answer_YYYY_MM, answer_year, chanel_name, service_name, age_group;
```

Формирование агрегированного MV BI.nps_metrics_mv с расчетами метрик NPS

```
CREATE MATERIALIZED VIEW IF NOT EXISTS BI.nps_metrics_mv
ENGINE = AggregatingMergeTree()
PARTITION BY answer_YYYY_MM
ORDER BY (answer_YYYY_MM, answer_year, chanel_name, service_name, age_group)
POPULATE
AS
SELECT
answer_YYYY_MM,
    answer_year,
    chanel_name,
    service_name,
    age_group,
    count(*) as total_responses,
    -- Категории NPS
    sumIf(1, nps >= 9) as promoters,      -- 9-10: Промоутеры
    sumIf(1, nps BETWEEN 7 AND 8) as passives, -- 7-8: Нейтральные
    sumIf(1, nps <= 6) as detractors,     -- 0-6: Детракторы
    -- Расчет NPS
    round((promoters - detractors) * 100.0 / total_responses, 1) as nps_score,
       -- Проценты
    round(promoters * 100.0 / total_responses, 1) as promoter_percent,
    round(passives * 100.0 / total_responses, 1) as passive_percent,
```

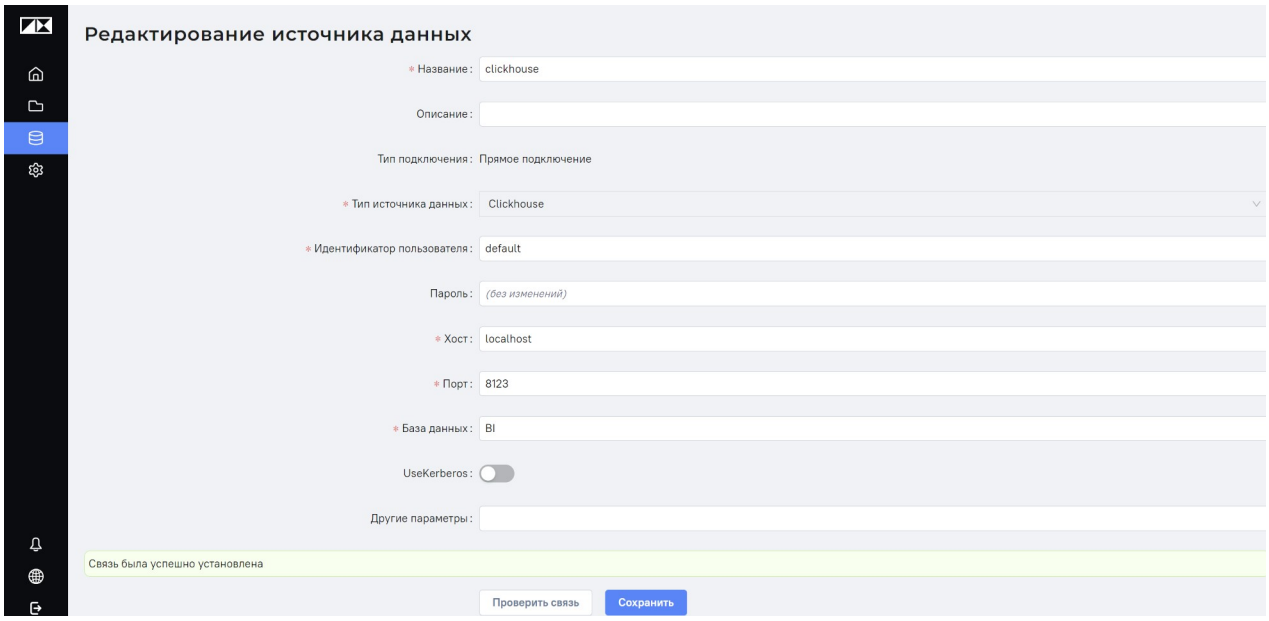
```
    round(detractors * 100.0 / total_responses, 1) as detractor_percent,
    round(avg(nps), 2) as avg_nps,
    min(nps) as min_nps,
    max(nps) as max_nps,
    round(stddevPop(nps), 2) as nps_std_dev
FROM BI.respondent_answers_wide_mv
GROUP BY answer_YYYY_MM, answer_year, chanel_name, service_name, age_group;
```

Формирование агрегированного MV BI.ces_metrics_mv с расчетами метрик CES

```
CREATE MATERIALIZED VIEW IF NOT EXISTS BI.ces_metrics_mv
ENGINE = AggregatingMergeTree()
PARTITION BY answer_YYYY_MM
ORDER BY (answer_YYYY_MM, answer_year, chanel_name, service_name, age_group)
POPULATE
AS
SELECT
answer_YYYY_MM,
    answer_year,
    chanel_name,
    service_name,
    age_group,
    count(*) as total_responses,
    -- Основной CES
    round(avg(ces), 2) as ces_score,
    -- Распределение оценок
    countIf(ces = 1) as effort_very_low,
    countIf(ces = 2) as effort_low,
    countIf(ces = 3) as effort_medium_low,
    countIf(ces = 4) as effort_medium,
    countIf(ces = 5) as effort_medium_high,
    countIf(ces = 6) as effort_high,
    countIf(ces = 7) as effort_very_high,
    -- Процент низких усилий (1-2)
    round((effort_very_low + effort_low) * 100.0 / total_responses, 1) as low_effort_percent,
    -- Процент высоких усилий (6-7)
    round((effort_high + effort_very_high) * 100.0 / total_responses, 1) as high_effort_percent
FROM BI.respondent_answers_wide_mv
GROUP BY answer_YYYY_MM, answer_year, chanel_name, service_name, age_group;
```

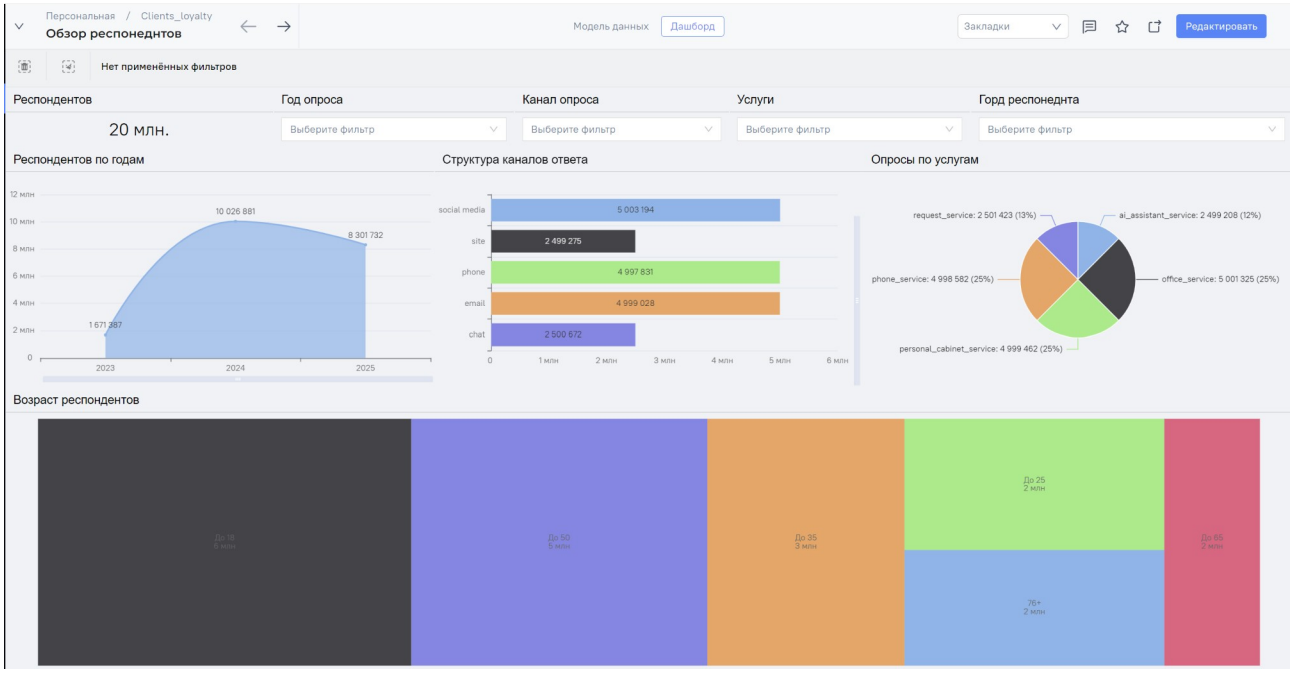
Интеграция с PIX BI

1. Формирование подключения к Clickhouse

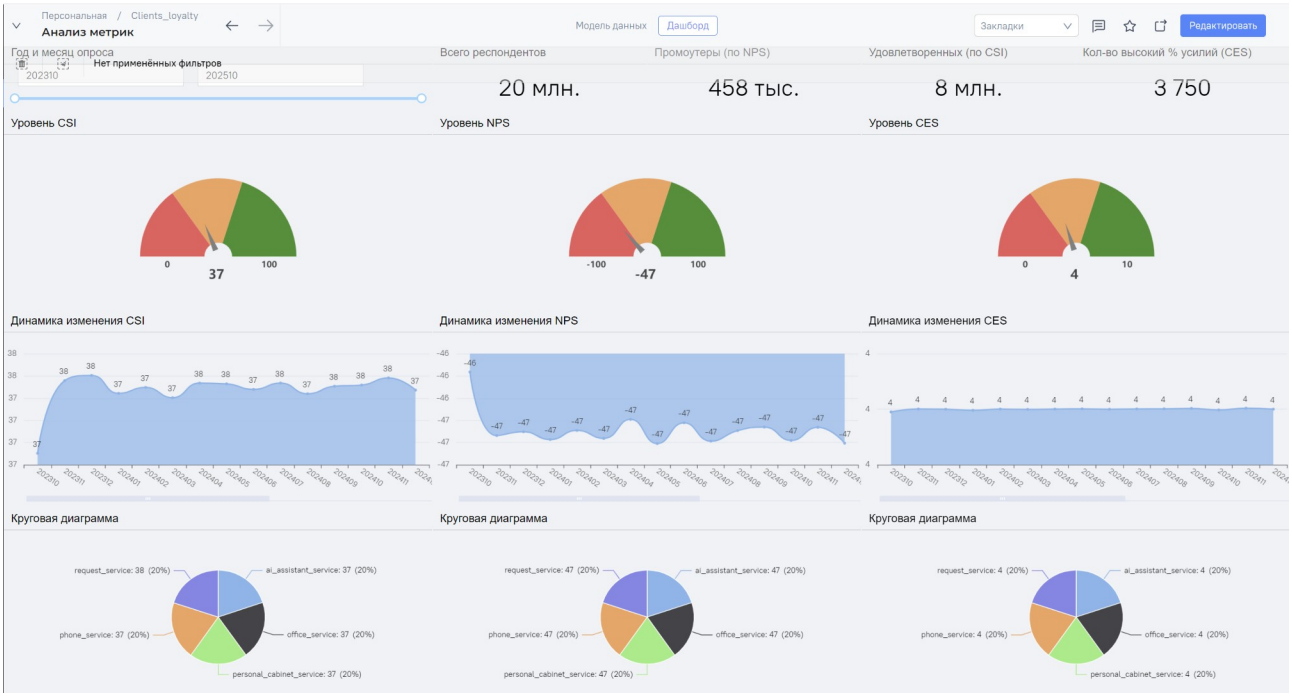


2. Формирование модели данных для дашборда по оценке лояльности клиентов — Clients_loyalty

3. Дашборд «Обзор респондентов»



4. Дашборд «Анализ метрик»



Мониторинг Clickhouse

Создаем базу данных для хранения данных мониторинга
CREATE DATABASE IF NOT EXISTS system_monitoring;

Мониторинг настраивается по следующим системным таблицам:
system.dictionaries — мониторинг словарей

system.query_log — мониторинг запросов
system.disks — мониторинг дисков
system.processes — мониторинг процессов
system.tables — мониторинг таблиц

Создание витрины для мониторинга словарей

```
CREATE VIEW system_monitoring.dictionaries_status AS
SELECT
    name as dictionary_name,
    type,
    status,
    round(loading_duration, 2) as loading_sec,
    formatReadableSize(bytes_allocated) as memory_size,
    element_count as elements,
    last_exception,
    last_successful_update_time as last_update,
    round(age('second', last_successful_update_time, now()) / 3600, 1) as hours_since_update,
    -- Статус здоровья
    multiIf(
        status = 'LOADED' AND hours_since_update <= 24, 'OK',
        status = 'LOADED' AND hours_since_update <= 72, 'WARNING',
        status = 'LOADED', 'STALE',
        status = 'FAILED', 'ERROR',
        '○ UNKNOWN'
    ) as health_status
FROM system.dictionaries
ORDER BY bytes_allocated DESC;
```

Создание витрины для мониторинга запросов

```
CREATE VIEW system_monitoring.query_performance AS
SELECT
    toDate(event_time) as query_date,
    query,
    query_id,
    user,
    address as client_ip,
    -- Производительность
    round(query_duration_ms / 1000, 2) as duration_sec,
    read_rows,
    formatReadableSize(read_bytes) as read_size,
    written_rows,
    formatReadableSize(written_bytes) as written_size,
    result_rows,
    formatReadableSize(result_bytes) as result_size,
    memory_usage,
    -- Анализ сложности
    multiIf(
        query_duration_ms > 30000, 'SLOW',
```

```
        query_duration_ms > 10000, 'MEDIUM',
        query_duration_ms > 1000, 'FAST',
        'INSTANT'
    ) as performance_category,
    -- Тип запроса
    multiIf(
        lower(query) LIKE '%select%', 'SELECT',
        lower(query) LIKE '%insert%', 'INSERT',
        lower(query) LIKE '%alter%', 'ALTER',
        lower(query) LIKE '%create%', 'CREATE',
        'OTHER'
    ) as query_type
FROM system.query_log
WHERE event_time >= now() - 3600  -- последний час
  AND type = 'QueryFinish'
ORDER BY query_duration_ms DESC;
```

Создание витрины мониторинга дисков

```
CREATE VIEW system_monitoring.disks_usage AS
SELECT
    name as disk_name,
    path,
    type,
    -- Использование пространства
    formatReadableSize(free_space) as free_space_size,
    formatReadableSize(total_space) as total_space_size,
    (total_space - free_space) as used_space_bytes,
    formatReadableSize(total_space - free_space) as used_space,
    round((total_space - free_space) * 100.0 / total_space, 1) as used_percent,
   -- Анализ загрузки
    multiIf(
        used_percent >= 90, 'CRITICAL',
        used_percent >= 80, 'HIGH',
        used_percent >= 70, 'WARNING',
        'NORMAL'
    ) as disk_status,
    -- Рекомендации
    multiIf(
        used_percent >= 90, 'Немедленно очистите место',
        used_percent >= 80, 'Рассмотрите очистку данных',
        used_percent >= 70, 'Мониторьте использование',
        'В пределах нормы'
    ) as recommendation
FROM system.disks
ORDER BY used_percent DESC
;
```

Создание витрины мониторинга процессов

```
CREATE VIEW system_monitoring.active_processes AS
SELECT
    query_id,
    user,
    address as client_ip,
    query,
    -- Время выполнения
    round(elapsed, 2) as elapsed_sec,
    is_cancelled as cancelled,
    -- Потребление ресурсов
    formatReadableSize(memory_usage) as memory_used,
    read_rows,
    formatReadableSize(read_bytes) as read_data,
    written_rows,
    formatReadableSize(written_bytes) as written_data,
    -- Анализ длительных запросов
    multiIf(
        elapsed > 300, 'CRITICAL',
        elapsed > 60, 'LONG',
        elapsed > 10, 'MEDIUM',
        'NORMAL'
    ) as execution_status,
    -- Действия
    multiIf(
        elapsed > 300 AND NOT is_cancelled, 'Рассмотрите KILL QUERY',
        elapsed > 60, 'Мониторьте выполнение',
        'В пределах нормы'
    ) as action_required
FROM system.processes
ORDER BY elapsed DESC;
```

Создание витрины мониторинга таблиц

```
CREATE VIEW system_monitoring.tables_metrics AS
SELECT
    database,
    name as table_name,
    engine,
    -- Размеры данных (используем правильные названия колонок)
    total_bytes,
    formatReadableSize(total_bytes) as total_size,
    -- Для расчета сжатия используем доступные колонки
    total_bytes as compressed_size_bytes,
    formatReadableSize(total_bytes) as compressed_size,
    -- Приблизительный расчет несжатых данных (если нет точных данных)
    total_bytes * 3 as estimated_uncompressed_bytes,  -- примерный коэффициент
    formatReadableSize(total_bytes * 3) as estimated_uncompressed_size,
    -- Статистика строк
```

```
    total_rows as rows_count,
  formatReadableSize(parts) as avg_part_size,
    parts,
    -- Расчет коэффициента сжатия на основе оценок
    round(3.0, 1) as estimated_compression_ratio,  -- примерное значение
    -- Анализ эффективности на основе доступных метрик
    multiIf(
        total_rows > 0 AND total_bytes / total_rows < 100, 'EXCELLENT',  -- мало байт на строку
        total_rows > 0 AND total_bytes / total_rows < 500, 'GOOD',
        total_rows > 0 AND total_bytes / total_rows < 1000, 'AVERAGE',
        'POOR'
    ) as efficiency_status,
   -- Рекомендации по оптимизации
    multiIf(
        total_rows > 10000000 AND engine LIKE '%MergeTree%', 'Рассмотрите
партиционирование',
        total_rows > 0 AND total_bytes / total_rows > 1000, 'Проверьте структуру данных',
        parts > 100, 'Слишком много партиций',
        total_rows = 0, 'Пустая таблица',
        'Оптимально'
    ) as optimization_tip,
    -- Дополнительные метрики
    round(total_bytes * 1.0 / NULLIF(total_rows, 0), 2) as bytes_per_row
FROM system.tables
WHERE database NOT IN ('system', 'information_schema')
ORDER BY total_bytes DESC;
```

Создание сводной витрины общего состояния системы

```
CREATE VIEW system_monitoring.overall_health AS
WITH
disk_stats AS (
    SELECT
        countIf(used_percent >= 80) as critical_disks,
        count(*) as total_disks
    FROM system_monitoring.disks_usage
),
process_stats AS (
    SELECT
        countIf(elapsed_sec > 60) as long_running_queries,
        count(*) as active_queries
    FROM system_monitoring.active_processes
),
dict_stats AS (
    SELECT
        countIf(health_status LIKE '%ERROR%') as failed_dictionaries,
        count(*) as total_dictionaries
    FROM system_monitoring.dictionaries_status
```

```
),
table_stats AS (
    SELECT
        countIf(efficiency_status LIKE '%POOR%') as poor_compression_tables,
        count(*) as total_tables
    FROM system_monitoring.tables_metrics
)
SELECT
    'System Health Report' as report_name,
    now() as report_time,
    -- Сводка по дискам
    critical_disks,
    total_disks,
    round(critical_disks * 100.0 / total_disks, 1) as disk_health_score,
    -- Сводка по запросам
    long_running_queries,
    active_queries,
    round(long_running_queries * 100.0 / NULLIF(active_queries, 0), 1) as query_health_score,
    -- Сводка по словарям
    failed_dictionaries,
    total_dictionaries,
    round(failed_dictionaries * 100.0 / total_dictionaries, 1) as dict_health_score,
    -- Сводка по таблицам
    poor_compression_tables,
    total_tables,
    round(poor_compression_tables * 100.0 / total_tables, 1) as table_health_score,
    -- Общий статус системы
    multiIf(
        critical_disks > 0 OR failed_dictionaries > 5, 'CRITICAL',
        long_running_queries > 3 OR poor_compression_tables > 10, 'WARNING',
        'HEALTHY'
    ) as overall_status,
    -- Приоритетные действия
    multiIf(
        critical_disks > 0, 'СРОЧНО: Освободите место на диске',
        failed_dictionaries > 5, 'Восстановите словари',
        long_running_queries > 3, 'Оптимизируйте медленные запросы',
        poor_compression_tables > 10, 'Улучшите сжатие таблиц',
        'Система в норме'
    ) as priority_action
FROM disk_stats, process_stats, dict_stats, table_stats;
```

Создание витрины для алертов

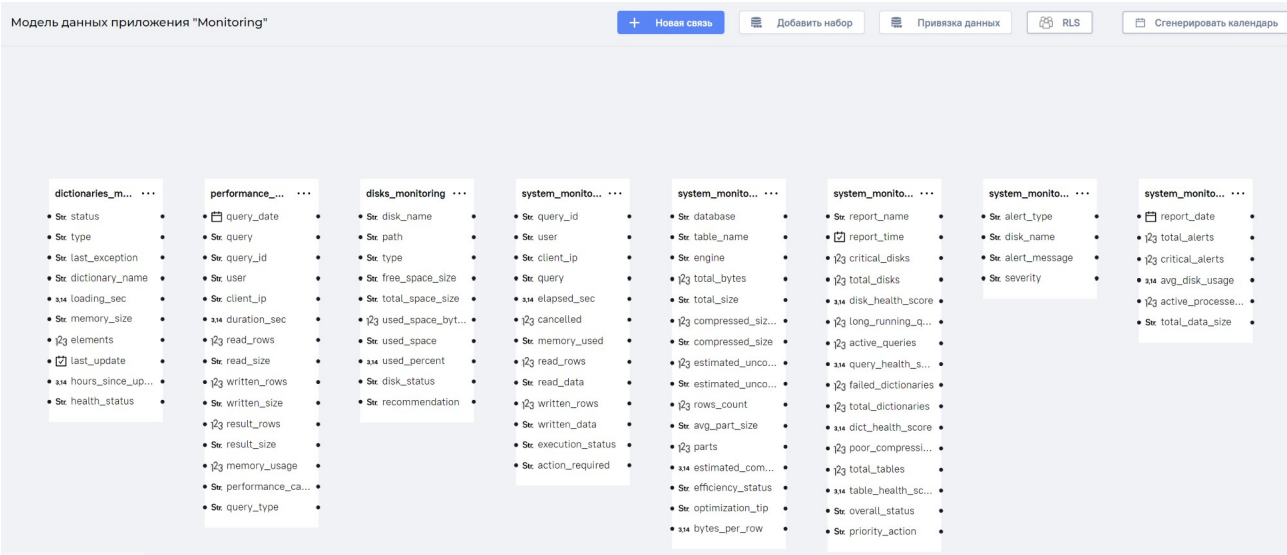
```
CREATE VIEW system_monitoring.alerts AS
SELECT
    'DISK_USAGE' as alert_type,
    disk_name,
    'Использование диска: ' || toString(used_percent) || '%' as alert_message,
    'HIGH' as severity
```

```
FROM system_monitoring.disks_usage
WHERE used_percent >= 80
UNION ALL
SELECT
    'LONG_QUERY' as alert_type,
    query_id,
    'Долгий запрос: ' || toString(elapsed_sec) || ' сек' as alert_message,
    'MEDIUM' as severity
FROM system_monitoring.active_processes
WHERE elapsed_sec > 60
UNION ALL
SELECT
    'DICTIONARY_FAILED' as alert_type,
    dictionary_name,
    'Словарь не обновляется: ' || last_exception as alert_message,
    'HIGH' as severity
FROM system_monitoring.dictionaries_status
WHERE status = 'FAILED' OR hours_since_update > 72
UNION ALL
SELECT
    'POOR' as alert_type,
    table_name,
    'Плохое сжатие: ' || toString(estimated_compression_ratio) || ':1' as alert_message,
    'LOW' as severity
FROM system_monitoring.tables_metrics
WHERE estimated_compression_ratio < 3
ORDER BY
    CASE severity
        WHEN 'HIGH' THEN 1
        WHEN 'MEDIUM' THEN 2
        WHEN 'LOW' THEN 3
    END,
    alert_type;
```

Создание витрины для ежедневного отчета

```
CREATE VIEW system_monitoring.daily_report AS
SELECT
    toDate(now()) as report_date,
    (SELECT count(*) FROM system_monitoring.alerts) as total_alerts,
    (SELECT count(*) FROM system_monitoring.alerts WHERE severity = 'HIGH') as
critical_alerts,
    (SELECT round(avg(used_percent), 1) FROM system_monitoring.disks_usage) as
avg_disk_usage,
    (SELECT count(*) FROM system_monitoring.active_processes) as active_processes_count,
    (SELECT formatReadableSize(sum(total_bytes))
     FROM system_monitoring.tables_metrics) as total_data_size;
```

Передача данных мониторинга в PIX BI для дашбордов



CHANGELOG

- Версия Clickhouse – 25.9.4.58
- Версия PostgreSQL - PostgreSQL 16.10 (Ubuntu 16.10-1.pgdg24.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0, 64-bit
- Версия PIX BI - 1.31.10