ERP/1: Аналітика

Техноробочий проект

на створення та впровадження
локальної DWH-інфраструктури
для аналізу та обробки великих даних

Згідно з Постановою КМУ № 205 від 21.02.2025

Формалізація вимог за стандартом ISO/IEC/IEEE 29148:2018

2026 © Системи Електронного Урядування

Зміст

Зміст

1. Загальні відомості про засіб інформатизації

1.1. Найменування та підстави для розробки

Найменування засобу інформатизації: Модуль «Аналітика» (локальна DWH-інфраструктура) у складі інформаційної системи управління підприємством ERP/1.

Підстави для виконання робіт:

- Закон України «Про захист персональних даних»;
- Закон України «Про захист інформації в інформаційно-комунікаційних системах»;
- Порядок використання засобів інформатизації (Постанова КМУ № 205 від 21.02.2025);
- Програма модернізації обчислювальної IT-інфраструктури ЄСІКС.

1.2. Призначення та цілі створення засобу

Модуль призначений для виконання швидких аналітичних обчислень над локальними реєстрами, збереження історії та побудови BI-звітів on-premises на базі інтеграції з відкритими аналітичними базами даних ClickHouse та DuckDB.

Основні цілі створення:

- Організація децентралізованого DWH на базі C99-сумісного двигуна DuckDB для периферійних вузлів.
- Налаштування централізованого аналітичного кластера ClickHouse для консолідації даних.
- Розробка високопродуктивних ETL-контурів синхронізації транзакційних даних Mnesia/KVS у аналітичні структури.
- Надання BI-інтерфейсу користувачам через інтеграцію з Apache Superset.

1.3. План-графік виконання етапів робіт

Розробка та впровадження модуля «Аналітика» розділяється на такі етапи:

1. **Етап 1: Технічне проектування** — проектування схем даних (зірки/сніжинки), вибір форматів та полів синхронізації. (1 місяць).
2. **Етап 2: Реалізація C99 DuckDB NIF** — розробка інтерфейсної бібліотеки C99 для зв'язку Erlang/OTP з DuckDB без Rust. (1.5 місяці).
3. **Етап 3: Розробка ETL контуру** — реалізація процесів синхронізації `dwh_sync.erl` та вивантаження у Parquet. (1 місяць).
4. **Етап 4: Розгортання BI та кластеризація** — налаштування Apache Superset та інтеграція з центральним ClickHouse. (1.5 місяці).
5. **Етап 5: Сертифікація безпеки** — експертиза КСЗІ, налаштування шифрування LUKS/TPM 2.0 та мережевих політик Cilium. (2 місяці).

2. Відомості про робочий процес та умови експлуатації

2.1. Опис бізнес-процесів (BPMN / FSM)

Керування ETL-процесами та виконання аналітичних запитів формалізуються у вигляді кінцевих автоматів (FSM) та виконуються рушієм процесів BPE:

2.1.1. Процес ETL синхронізації (dwh_sync.erl)

Цей процес відповідає за періодичне або транзакційне перенесення даних з Mnesia/KVS у стовпчикове сховище DWH:

1. **StartSync:** Активація процесу за cron-розкладом або транзакційним тригером.
2. **ExtractMnesiaChanges:** Збір змінених записів з журналів транзакцій Mnesia за останній період.
3. **TransformData:** Приведення даних до пласкої реляційної структури, знеособлення за потреби.
4. **WriteToParquet:** Асинхронне збереження проміжних даних у файл формату Parquet на NVMe.
5. **LoadToDuckDB:** Імпорт Parquet файлу в локальну базу даних DuckDB за допомогою векторної операції копіювання (Copy).
6. **SyncCentralDWH:** Передача агрегованих даних до центрального аналітичного вузла ClickHouse.
7. **SyncCompleted:** Фіксація завершення та запис статусів.

2.1.2. Процес ad-hoc аналізу (dwh_query.erl)

Забезпечує обробку аналітичних запитів користувачів через API:

1. **ReceiveQuery:** Користувач надсилає аналітичний запит або запускає оновлення дашборду.
2. **RouteQuery:** Маршрутизація запиту (на локальну DuckDB для периферійних даних чи на ClickHouse для глобальної статистики).
3. **ExecuteVectorSQL:** Виконання SQL-запиту векторним процесором СКБД.
4. **FormatResponse:** Перетворення результату в бінарний JSON/Tuple представлення.
5. **LogQueryStats:** Збереження статистики виконання (час обробки, кількість зчитаних рядків) в аудит-журнал.

2.2. Опис ролей та прав доступу (ABAC)

Доступ до виконання SQL-запитів та перегляду дашбордів Apache Superset обмежується на базі атрибутів користувача:

```
allow(User, Action, Resource) ->
    UserRole = maps:get(role, User),
    UserDept = maps:get(department, User),
    ResDept = maps:get(department, Resource),
```

```
IsAnalyst = lists:member(UserRole, [analyst, admin]),
MatchesDept = (UserDept == ResDept),

case {IsAnalyst, Action} of
    {true, execute_query} -> true;
    {false, execute_query} -> MatchesDept;
    {_, view_dashboard} -> true;
    _ -> false
end.
```

2.3. Специфікація апаратного забезпечення

Апаратні вимоги до серверів DWH на периферійних та центральному вузлах:

Параметр	Периферійний вузол (DuckDB)	Центральний вузол (ClickHouse)
Процесор	x86-64 (min 8 Cores), AVX2	2x Intel Xeon / AMD EPYC (min 32 Cores), AVX-512
Оперативна пам'ять	16–32 GB DDR4/DDR5	128–256 GB ECC DDR5
Накопичувач	NVMe SSD (PCIe Gen4)	Hardware RAID-10 NVMe Enterprise SSD
Мережевий інтерфейс	1 Gbps Ethernet	10/25 Gbps Fiber Ethernet

2.4. Умови експлуатації та системне середовище

Периферійні DWH вузли встановлюються на робочих станціях у локальній мережі установ. Системне середовище складається з: Ubuntu LTS 24.04, Erlang/OTP 27, Elixir 1.17, бібліотек C99 (gcc/clang, make) без встановленого Rust SDK.

2.5. Детальна архітектура DWH на базі DuckDB та ClickHouse

Аналітична архітектура будується на поєднанні двох технологій стовпчикового аналізу:

1. **DuckDB (Edge OLAP)**: Локальний аналітичний двигун на базі вбудовуваної векторної технології DuckDB. Для забезпечення самостійності та ізоляції ресурсів, DuckDB інтегрується через виділений Erlang NIF, що працює в окремій групі потоків (dirty CPU schedulers) з обмеженими лімітами на використання RAM та CPU, унеможливлюючи вплив OLAP-навантаження на транзакційну OTP-частину. Масштабування рішення досягається через федеративне виконання запитів (за допомогою розширень роботи з об'єктними сховищами) та Peer-to-Peer реплікацію проміжних Parquet-файлів між периферійними вузлами для відмовостійкості.
2. **ClickHouse (Cluster DWH)**: Масштабована СКБД, яка об'єднує дані з усіх вузлів. Синхронізація з DuckDB відбувається асинхронно через вивантаження Parquet файлів.

3. Інформаційне та програмне забезпечення

3.1. Інформаційне забезпечення (Схеми та Дані)

3.1.1. Визначення структур даних (Erlang Records)

Основні реєстрові сутності аналітики:

```
-record(dwh_node_status, {
    node_id,                    % UUID вузла (PK)
    hostname,                   % Текстове ім'я хоста
    ip_address,                 % IP-адреса вузла
    has_avx512 = false,         % Наявність інструкцій AVX-512
    queries_processed = 0,      % Кількість виконаних запитів
    status = offline,           % online | offline | degraded
    last_ping_at = 0            % Timestamp пінгу
}).

-record(dwh_query_log, {
    task_id,                    % UUID задачі (PK)
    user_id,                    % UUID користувача
    query_hash,                 % SHA256 хеш SQL запиту
    execution_time_ms = 0,      % Час виконання
    read_rows = 0,              % Кількість зчитаних рядків
    query_type = adhoc,         % adhoc | report | sync
    created_at = 0
}).

-record(dwh_pipeline, {
    pipeline_id,                % UUID конвеєра (PK)
    source_table,               % Транзакційна таблиця Mnesia
    target_table,               % Аналітична таблиця DWH
    last_sync_at = 0,           % Час останньої синхронізації
    sync_interval_sec = 3600,   % Інтервал синхронізації
    status = idle               % idle | running | failed
}).

-record(dwh_storage_config, {
    storage_id,                 % UUID налаштування
    local_path,                 % Шлях до бази файлу (.db / .parquet)
    format = parquet,           % parquet | csv | native
    compression = zstd,         % zstd | lz4 | none
    allocated_mb = 0
}).
```

3.2. Програмне забезпечення (Реалізація)

3.2.1. dwh_sync.erl (DSL процесу синхронізації)

```
-module(dwh_sync).
-include("bpe.hrl").
-compile(export_all).

def() ->
    #process{
        name = 'DWH ETL Sync Workflow',
        beginEvent = 'StartSync',
        endEvent = 'SyncCompleted',
        tasks = [
            #beginEvent{name='StartSync'},
            #serviceTask{name='ExtractMnesiaChanges', module=?MODULE},
            #serviceTask{name='TransformData', module=?MODULE},
            #serviceTask{name='WriteToParquet', module=?MODULE},
            #serviceTask{name='LoadToDuckDB', module=?MODULE},
            #serviceTask{name='SyncCentralDWH', module=?MODULE},
            #endEvent{name='SyncCompleted'}
        ],
        flows = [
            #sequenceFlow{name='1', source='StartSync', target='ExtractMnesiaChanges'},
            #sequenceFlow{name='2', source='ExtractMnesiaChanges', target='TransformData'},
            #sequenceFlow{name='3', source='TransformData', target='WriteToParquet'},
            #sequenceFlow{name='4', source='WriteToParquet', target='LoadToDuckDB'},
            #sequenceFlow{name='5', source='LoadToDuckDB', target='SyncCentralDWH'},
            #sequenceFlow{name='6', source='SyncCentralDWH', target='SyncCompleted'}
        ],
        roles = [operator, system]
    }.

action({serviceTask, 'ExtractMnesiaChanges'}, Proc) ->
    % Витягнення логу транзакцій з Mnesia
    {reply, Proc};
action({serviceTask, 'WriteToParquet'}, Proc) ->
    % Збереження у тимчасовий Parquet-файл на NVMe
    {reply, Proc};
action({serviceTask, 'LoadToDuckDB'}, Proc) ->
    % Виклик C99 NIF модуля для завантаження в DuckDB
    {reply, Proc};
action(_, Proc) -> {reply, Proc}.
```

3.2.2. dwh_nif.c (C99 Erlang NIF для DuckDB)

Нижче наведено код інтеграційного модуля на чистій мові C99 для зв'язку з C-API DuckDB без застосування Rust:

```
#include "erl_nif.h"
#include <duckdb.h>
#include <string.h>

// Контекст для збереження дескриптора бази даних
typedef struct {
    duckdb_database db;
    duckdb_connection con;
} dwh_ctx_t;
```

```
static ErlNifResourceType* DWH_RES_TYPE = NULL;

// Функція очищення ресурсу при видаленні в Erlang
static void dwh_resource_cleanup(ErlNifEnv* env, void* obj) {
    dwh_ctx_t* ctx = (dwh_ctx_t*)obj;
    if (ctx->con) {
        duckdb_disconnect(&ctx->con);
    }
    if (ctx->db) {
        duckdb_close(&ctx->db);
    }
}

// Ініціалізація бази даних DuckDB
static ERL_NIF_TERM dwh_open_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[]) {
    char path[512];
    if (enif_get_string(env, argv[0], path, sizeof(path), ERL_NIF_LATIN1) <= 0) {
        return enif_make_badarg(env);
    }

    dwh_ctx_t* ctx = enif_alloc_resource(DWH_RES_TYPE, sizeof(dwh_ctx_t));
    memset(ctx, 0, sizeof(dwh_ctx_t));

    // Виклик C-API DuckDB
    if (duckdb_open(path, &ctx->db) == DuckDBError) {
        enif_release_resource(ctx);
        return enif_make_tuple2(env, enif_make_atom(env, "error"),
                                enif_make_string(env, "cannot_open_db", ERL_NIF_LATIN1));
    }

    if (duckdb_connect(ctx->db, &ctx->con) == DuckDBError) {
        duckdb_close(&ctx->db);
        enif_release_resource(ctx);
        return enif_make_tuple2(env, enif_make_atom(env, "error"),
                                enif_make_string(env, "cannot_connect_db", ERL_NIF_LATIN1));
    }

    ERL_NIF_TERM res = enif_make_resource(env, ctx);
    enif_release_resource(ctx);
    return enif_make_tuple2(env, enif_make_atom(env, "ok"), res);
}

// Виконання аналітичного запиту
static ERL_NIF_TERM dwh_query_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[]) {
    dwh_ctx_t* ctx;
    char query[2048];

    if (!enif_get_resource(env, argv[0], DWH_RES_TYPE, (void**)&ctx)) {
        return enif_make_badarg(env);
    }
    if (enif_get_string(env, argv[1], query, sizeof(query), ERL_NIF_LATIN1) <= 0) {
        return enif_make_badarg(env);
    }

    duckdb_result result;
    if (duckdb_query(ctx->con, query, &result) == DuckDBError) {
```

```
        const char* err_msg = duckdb_value_strval(&result, 0); // спрощено
        duckdb_destroy_result(&result);
        return enif_make_tuple2(env, enif_make_atom(env, "error"),
                                enif_make_string(env, err_msg ? err_msg : "query_failed", ERL_NIF_
    }

    // Отримання кількості рядків
    idx_t row_count = duckdb_row_count(&result);
    duckdb_destroy_result(&result);

    return enif_make_tuple2(env, enif_make_atom(env, "ok"), enif_make_uint64(env, (unsigned long)r
}

static ErlNifFunc nif_funcs[] = {
    {"open", 1, dwh_open_nif},
    {"query", 2, dwh_query_nif}
};

static int load(ErlNifEnv* env, void** priv_data, ERL_NIF_TERM load_info) {
    DWH_RES_TYPE = enif_open_resource_type(env, NULL, "dwh_res", dwh_resource_cleanup,
                                           ERL_NIF_RT_CREATE | ERL_NIF_RT_TAKEOVER, NULL);
    return 0;
}

ERL_NIF_INIT(dwh_nif, nif_funcs, load, NULL, NULL, NULL)
```

4. Вимоги до засобу

4.1. Функціональні вимоги

Модуль «Аналітика» повинен забезпечувати регулярне завантаження транзакційних даних, їх зберігання у стовпчиковому форматі на NVMe, виконання локальних аналітичних запитів та агрегацій у фоновому режимі та надання інтерфейсу візуалізації.

4.2. Вимоги до інтерфейсів та дизайну

- Веб-інтерфейс адміністратора конвеєрів та ETL має базуватися на Nitro/N2O.
- Клієнтська візуалізація звітів повинна бути реалізована через інтегровану панель Apache Superset у фірмовому стилі Sleek Dark Mode.

4.3. Вимоги до безпеки та захисту інформації

- Локальні файли баз даних DuckDB (.db) повинні зберігатися на NVMe розділах, зашифрованих за допомогою LUKS з прив'язкою до TPM 2.0.
- Мережева взаємодія між периферійними вузлами та центральним ClickHouse повинна шифруватися за допомогою TLS 1.3 з використанням взаємної автентифікації (mTLS).

4.4. Вимоги до продуктивності та надійності

- Максимальний час виконання регламентної синхронізації (ETL) за годину не повинен перевищувати 3 хвилин.
- Помилка виконання SQL на периферійному вузлі не повинна впливати на транзакційну працездатність ERP/1.