Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студентки	*Білецької Анастасії Дмитрівни*
	(ПІБ)
академічної групи	*121М-20-1*
	(шифр)
спеціальності	*121 Інженерія програмного забезпечення*
	(код і назва спеціальності)
освітньої програми	*«Інженерія програмного забезпечення»*
	(назва освітньої програми)
на тему:	*Розробка та дослідження моделі фіксації етапів в Extract Transform Load системах*

_____*А.Д. Білецька*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний				
економічний	*Проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*Доц. Приходченко С.Д.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

(підпис) І.М. Удовик (прізвище, ініціали)

« » 20 21 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студентці *121м-20-1* (група) *Білецькій Анастасії Дмитрівні* (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка та дослідження моделі фіксації етапів в Extract Transform Load системах*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від ___.___.2021 р. № _____

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процеси розробки моделі фіксації етапів в Extract Transform Load системах.

Предмет досліджень – використання програмних засобів для дослідження моделей фіксації етапів в Extract Transform Load системах.

Методи дослідження: методи використання існуючих програмних пакетів для фіксації, розробка програмного забезпечення для дослідження моделей фіксації етапів в Extract Transform Load системах.

Мета роботи – розробити програмне забезпечення для дослідження моделей фіксації етапів в Extract Transform Load системах.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Новизна запропонованих рішень визначається тим, що створенно моделі руху даних, тобто послідовних чи паралельних процесів, які мають початковий процес отримання даних та кінцеву точку виводу/завантаження та зберігання кінцевої інформації, яка може бути трансформована, згрупована чи доповнена у процесах які входять до ETL-моделі.

Практична цінність результатів полягає у тому, що в результаті проведеного дослідження було спроектовано алгоритм для дослідження моделей фіксації етапів в Extract Transform Load системах.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Розробка у складі корпоративної інформаційної системи підприємства для логування процесів ETL-системи будь-якої складності, з будь-яким рівнем вкладеності та використання будь-яких технологій. Тому розробка системи, яка буде логувати процеси ETL-системи потрібна майже кожному розгалуженому підприємству для обробки процесів.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз джерел та постановка задачі	12.09.2021 - 30.09.2021
Побудова математичних моделей та розробка алгоритму	01.10.2021 - 31.10.2021
Розробка та тестування програмного забезпечення для для дослідження моделей фіксації етапів в Extract Transform Load системах	01.11.2021 - 30.12.2021

Завдання видав _____ _____
(підпис) (прізвище, ініціали)

Завдання прийняла до виконання _____ *Білецька А.Д.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2021 р.*

Термін подання кваліфікаційної роботи до ЕК *20.01.2021*

РЕФЕРАТ

Пояснювальна записка: ___ стор., ___ рис., ___ таблиці, ___ додатка, ___ джерел.

Об’єкт досліджень – процеси розробки моделі фіксації етапів в Extract Transform Load системах.

Предмет досліджень – використання програмних засобів для дослідження моделей фіксації етапів в Extract Transform Load системах.

Методи дослідження: методи використання існуючих програмних пакетів для фіксації, розробка програмного забезпечення для дослідження моделей фіксації етапів в Extract Transform Load системах.

Мета роботи – розробити програмне забезпечення для дослідження моделей фіксації етапів в Extract Transform Load системах.

Новизна запропонованих рішень визначається тим, що створенно моделі руху даних, тобто послідовних чи паралельних процесів, які мають початковий процес отримання даних та кінцеву точку виводу/завантаження та зберігання кінцевої інформації, яка може бути трансформована, згрупована чи доповнена у процесах які входять до ETL-моделі.

Практична цінність результатів полягає у тому, що в результаті проведеного дослідження було спроектовано алгоритм для дослідження моделей фіксації етапів в Extract Transform Load системах.

Розробка у складі корпоративної інформаційної системи підприємства для логування процесів ETL-системи будь-якої складності, з будь-яким рівнем вкладеності та використання будь-яких технологій. Тому розробка системи, яка буде логувати процеси ETL-системи потрібна майже кожному розгалуженому підприємству для обробки процесів.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: ЛОГУВАННЯ ПРОЦЕСІВ, ETL-СИСТЕМИ, РЕЛЯЦІЙНІ БАЗИ ДАНИХ, СХОВИЩА ДАНИХ

ABSTRACT

Explanatory note: ___ pages, ___ figures, ___ tables, ___ appendices, ___ sources.

The object of research is the processes of developing a model of fixing stages in Extract Transform Load systems.

The subject of research is the use of software for the study of models of fixation of stages in Extract Transform Load systems.

Research methods: methods of using existing software packages for fixing, software development for research of models of fixing stages in Extract Transform Load systems.

The purpose of the work is to develop software for the study of stage fixation models in Extract Transform Load systems.

The novelty of the proposed solutions is determined by the fact that created data flow models, ie sequential or parallel processes that have an initial data acquisition process and an endpoint of output / download and storage of final information that can be transformed, grouped or supplemented in ETL processes. models.

The practical value of the results is that as a result of the study, an algorithm was designed to study the models of fixation of stages in Extract Transform Load systems.

Development as a part of the corporate information system of the enterprise for logging of processes of ETL-system of any complexity, with any level of nesting and use of any technologies. Therefore, the development of a system that will log the processes of the ETL system is needed by almost every branch of the enterprise to process.

In the section "Economics" calculations of the complexity of software development, the cost of creating software and the duration of its development, as well as marketing research of the market for the software product.

Keyword list: PROCESS LOGING, ETL SYSTEMS, RELATIVE DATABASES, DATA STORAGE

ЗМІСТ

ВСТУП

На сьогодні майже всі підприємства та організації використовують у своїх системах технічних системах реляційні бази даних. Головною перевагою яких є безвідмовність, стабільність та можливість отримати дані, навіть якщо було аварійне відключення системи.

Багато підприємств використовують складні системи видобування, трансформації та завантаження даних, так звані Extract Transform Load (ETL) системи. На різних підприємствах використовуються різні технології, різні мови програмування та різні рівні складності цих систем. Головною частиною, що об’єднує ці системи – це наявність бази даних та необхідність логування процесів, тобто запису всіх кроків, всіх параметрів та результатів блоків. В різних системах може бути різна кількість вкладених процесів чи блоків, складників, різні системи по-різному повинні логуватись, десь можливо взагалі записувати лише результати виконання, а десь потрібно записувати кожний крок з детальною інформацією.

Тому розробка системи, яка буде логувати процеси ETL-системи потрібна майже кожному розгалуженому підприємству для обробки процесів.

Предмет дослідження – інформаційна розробка у складі корпоративної інформаційної системи підприємства розроблена для логування процесів ETL-системи будь-якої складності, з будь-яким рівнем вкладеності та використання будь-яких технологій.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Поняття ETL моделювання

Дані, що завантажуються з будь-яких джерел, як правило, потрібно не просто зберігати всередині однієї системи, а передавати для обробки і аналізу в інші системи. Для цього існують, так звані, сховища даних (СД або DWH – Data Warehouse). СД – це бази даних для збору і обробки різної інформації, розроблені і орієнтовані спеціально для підготовки звітів і бізнес-аналізу, з метою підтримки прийняття рішень на підприємстві. [1]

Через безліч використовуваних технологій, відмінностей в системах з точки зору архітектури та структури даних, при передачі інформації в системи споживачі, її необхідно перетворювати.

Таким чином, з огляду на вищесказане, можна виділити три етапи в процесі роботи з даними:

1. Витяг (Extract), на цій стадії відбираються і описуються дані зовнішніх джерел (починають формуватися метадані СД), які повинні зберігатися в СД (релевантні дані).

2. Перетворення (Transform), на цій стадії релевантні дані перетворюються в формат представлення даних в СД, правила перетворення зберігаються в метаданих СД, формуються ключові поля таблиць фізичної структури СД, виконується очищення даних.

3. Завантаження (Load), На цій стадії дані завантажуються в СД, виконується побудова агрегатів.

Ці три етапи і складають абревіатуру ETL – одного з основних процесів в управлінні даними при отриманні їх з множини систем джерел і завантаження в СД, з метою отримання достовірної інформації.

Процес ETL реалізується шляхом або розробки програми ETL, або створення комплексу вбудованих програмних процедур, або використання ETL-інструментарію. Додатки ETL витягають інформацію з вихідних БД джерел, перетворять її в формат, підтримуваний БД призначення, а потім завантажують в цю БД перетворені дані.

Мета будь-якого ETL-додатки полягає в тому, щоб своєчасно доставити дані із зовнішніх систем в систему, з якої працюють користувачі. Як правило, ETL-додатки використовуються при перенесенні даних зовнішніх джерел в СД систем бізнес-аналітики. Тому організація процесу ETL є складовою частиною проекту розробки практично будь-якого СД.

Часто ETL є проміжним шаром між OLTP системами і OLAP системою або сховищем даних.

OLTP (Online Transaction Processing) – поняття відноситься до транзакційних систем порівняно невеликого розміру, що обробляють великі потоки даних в реальному часі.

OLAP (Online analytical processing) – поняття відноситься до систем для динамічного побудови звітів і документів, складання складних запитів до бази даних для глибокого аналізу. [2]

ETL є невід'ємною складовою такої характеристики, як інтегрованість будь-якого процесу в базах даних. Інтегрованість означає, що дані для аналізу не беруться безпосередньо з джерел, зокрема, баз даних OLTP-систем підприємства. Вихідні дані витягуються, перевіряються, очищаються, уніфікуються і т.п., щоб задовольняти вимогам аналітика, досліджує не окремі бізнес-функції, а діяльність всього підприємства. Процес ETL дозволяє підвищити якість даних аналітичної системи та швидкість виконання аналітичних запитів до сховища даних.

Кращий спосіб поглянути на рішення ETL – це розглядати його як бізнес-процес. Бізнес-процес має вхід, вихід і одну або кілька одиниць роботи, етапи

процесу. Ці етапи, в свою чергу, також мають входи і виходи і виконують операцію по перетворенню входу в вихід.

1.2. ETL-інструменти, характеристики та елементи

Взагалі, для реалізації ETL-процесу можна використовувати більшість сучасних мов програмування. Але, якщо потрібно не просто виконати разову конвертацію даних, а мова йде про реалізацію постійного процесу інтеграції даних з декількох різнорідних джерел, то має сенс розглянути використання спеціалізованих інструментів, що полегшують автоматизацію типових операцій, підтримку основних використовуваних форматів і найбільш поширених інформаційних систем. При цьому слід брати до уваги параметри масштабованості, швидкості і розширюваності таких інструментів.

Так спираючись на базу даних, систему додатків, операційну систему, тип серверу, та його параметри обирають ту чи іншу систему ETL.

Найбільш поширеними розробниками ETL-інструментів для систем, на сьогоднішній день можна назвати Oracle, Informatica і IBM. Як правило, системи таких виробників перекривають потреби бізнесу настільки, що більшість клієнтів не використовують і половини наявного функціоналу. Тому має сенс вибирати ETL-інструменти, ґрунтуючись на необхідних завданнях рішення і наявної платформі підприємства.

Наприклад, якщо в компанії домінують системи від компанії IBM – то можна розглянути рішення Data Stage від IBM. З придбанням в 2008 році канадською компанії Cognos, у IBM з'явилося і друге рішення Data Manager, до одного з достоїнств якого можна віднести наявність OLAP, чого немає у ряду інших систем. Це дозволяє обходитися без побудови складних СД, що сильно економить кошти компанії.

При використанні Unix / Linux подібних операційних систем часто використовуються рішення від Informatica або Oracle. До переваг рішень

PowerCenter і PowerMart від Informatica можна віднести слідування за новими технологіями, регулярні релізи та, як наслідок, найбільший набір інструментів для інтеграції даних. Гнучкості системі додає можливість програмувати на мові розробки Java. Для Oracle Data Integrator характерна можливість масштабування від невеликої організації до цілого підприємства, наявність спеціальних технології для маніпулювання великими обсягами даних. Також для підтримки розробників в системі є спеціальні модулі знань для багатьох інших систем, що містять шаблони коду. [3]

Якщо в компанії переважають системи від компанії Microsoft, то ETL можуть бути побудовані на продукті SSIS від Microsoft. Даний продукт має розвинений користувальницький інтерфейс, надає широку бібліотеку стандартних компонентів потоків даних. У доповненні до них, SSIS надає інфраструктуру для створення користувацьких компонентів. Це дозволяє компаніям розробляти вузькоспеціалізовані високоефективні компоненти обробки даних. Але продукт не є кросплатформним і орієнтований на взаємодію з продуктами Miscosoft.

Служби MS SQL Server Integration Services (SSIS) представляють собою платформу для побудови високопродуктивних рішень інтеграції даних і рішень потоку операцій, включаючи операції вилучення, перетворення і завантаження (Extract, Transform, Load - ETL) для сховищ даних.

Служби SSIS містять графічні інструменти і майстри для побудови і налагодження пакетів; завдання для виконання функцій потоку операцій, таких як:

- виконання інструкцій SQL і робота з повідомленнями електронної пошти;
- джерела даних і призначення для вилучення і завантаження даних;
- перетворення для очищення, статистичної обробки, злиття і копіювання даних;

- службу управління, службу SSIS для адміністрування пакетів служб SSIS;
- API-інтерфейси для програмування об'єктної моделі служб SSIS.

До типових випадків застосування пакетів служб SSIS спільно з SSAS відносять:

- злиття даних з різнорідних сховищ даних;
- заповнення сховищ даних і вітрин даних;
- очищення і стандартизацію даних. [4]

Так закордонні автори у своїх творах вказують, що будь-яка ETL-система повинна відповідати наступним характеристикам:

1. Незалежність платформи.

Інструмент ETL повинен бути в змозі працювати на будь-якій платформі і навіть на комбінації різних платформ. Може бути, 32-розрядна операційна система працює на початковому етапі розробки, але коли обсяги даних збільшуються, а доступні пакетні вікна зменшуються, потрібно більш потужне рішення. В інших випадках розробка виконується на комп'ютері під керуванням Windows або Mac, але робочі завдання виконуються в кластері Linux. Вам не потрібно вживати спеціальних заходів, щоб врахувати це в своєму рішенні ETL.

2. Масштабованість.

Масштабованість є великою проблемою; обсяги даних ростуть з року в рік, і ваші системи повинні бути в змозі впоратися з цим. Для обробки великих обсягів даних повинні бути доступні три варіанти:

Паралелізм: дозволяє перетворенню запускати безліч потоків паралельно, використовуючи сучасні багатоядерні апаратні архітектури

Поділ: дозволяє інструменту ETL використовувати переваги конкретних схем поділу для розподілу даних по паралельних потоків.

Кластеризація: дозволяє процесу ETL розділити робоче навантаження на кілька комп'ютерів. [5]

1.3. Моделювання руху даних

Інформаційна архітектура визначає набір вимог, принципів і моделей, необхідних для гнучкого спільного використання інформації та обміну нею. [6] Моделювання інформаційної архітектури починається з прийняття високорівневих визначень і описів пристрої бізнесу, єдиних для всіх підрозділів тим самим встановлюються загальні для всієї організації стандарти представлення і опису даних. Організації повинні використовувати свої інформаційні активи відповідно до бізнес-стратегією, формуючи повномасштабні уявлення, щоб полегшити узгодження технологій з бізнес-процесами, забезпечуючи адаптивність.

Узгодженість і адаптивність є ключовими поняттями в концепції інформаційної архітектури, при цьому найціннішим активом є інформація. В основі інформаційної архітектури лежить управління метаданими, здійснюване за допомогою моделей. Досягається завдяки використанню моделей абстракції – дозволяє знизити складність, поліпшити розуміння архітектури завдяки візуальному її поданням і забезпечити управлінський вплив для підвищення узгодженості та придатності до багаторазового використання процесів і технологій на різних ділянках організації. Разом зі зростанням потреби в узгодженні IT з бізнесом збільшується і число необхідних рівнів абстракції. Зростаюча необхідність в прийнятті стандартних практик і процедур і в застосуванні загальних інструментальних засобів є наслідком фундаментального переходу від фізичного моделювання даних в інтересах окремих робочих груп до стратегічного інформаційного інжинірингу в масштабі підрозділів і організації в цілому. [7]

Тому загальне поняття та визначення будь-якої ETL-моделі є у створенні моделі руху даних, тобто послідовних чи паралельних процесів, які мають початковий процес отримання даних та кінцеву точку виводу/завантаження та

зберігання кінцевої інформації, яка може бути трансформована, згрупована чи доповнена у процесах які входять до ETL-моделі.

Загальна схема може бути представлена як на рисунку 1.1.

Рис. 1.1. Модель руху даних

Де перший блок – це джерело/ресурс(source), з якого система бере початкові дані. Другий етап – це опрацювання, агрегування та всі можливі процеси з даними. Третій етап – це завантаження даних до кінцевого сховища. Створення реферату є процесом ETL-моделі, де студент бере інформацію з різних джерел (книги, журнали, публікації, веб-сторінки), групує дані, викидає непотрібне, доповнює своїми думками та висновками та у кінцевому результаті формує doc/docx/ptx/ptsx документ для збереження та подальшого використання.

Розробка ETL-моделі, як і розробка будь-якої програми, будь-якого додатку починається з формування моделі, тобто проектування логіки та формування потоків.

Процес перетворення даних відіграє дуже важливу роль в досягненні успіху реалізації проекту СД, тому він повинен бути добре спланований. Розробка плану носить інтерактивний характер.

Спочатку створюється узагальнений план, в якому відбивається перелік систем – джерел даних і вказуються плановані цільові області даних (даних, які будуть розміщуватися в СД). Джерело цільових даних визначається на основі сформульованих бізнес-вимог до СД. Як правило, джерела даних істотно

розрізняються: від БД і текстових файлів до нотатків. Ця обставина може значно ускладнити завдання перетворення даних.

Призначення таких високорівневих описів джерел дає, з одного боку, розробникам уявлення і про систему, яка створюється, і про існуючі джерела даних, а з іншого, керівництву організації – розуміння складності, пов'язаної з процесами перетворення даних.

До складання узагальненого плану найкраще приступати, коли розроблена багатовимірна модель СД. Тоді для кожної таблиці багатовимірної схеми можна визначити таблиці – джерела даних.

Детальне планування ETL-процесу багато в чому залежить від використання обраних ETL-інструментів. До теперішнього часу розроблено досить багато таких інструментів як компаніями виробниками комплексних рішень в області СД (IBM, Oracle, MicroSoft), так і сторонніми виробниками програмного забезпечення. Тому завдання вибору відповідних ETL-інструментів повинна бути вирішена до того, як приступати до детального планування.

Програмне забезпечення цього класу призначений для вилучення, приведення до загального формату, перетворенню, очищення та завантаження даних в сховище. Існують два підходи до написання ETL-процедур:

1) їх можна написати вручну;

2) можна скористатися спеціалізованими засобами ETL.

Кожен з підходів має ряд переваг і недоліків, тому вибір того чи іншого методу реалізації процедур ETL визначається вимогами до підсистеми завантаження даних в кожному конкретному випадку.

Слід звернути увагу на вибір технології для реалізації процедур ETL, в разі, якщо однією з систем-джерел даних виступає ERP-система. Системи даного класу є найбільш складними, оскільки володіють дуже заплутаною моделлю даних і часто містять десятки тисяч таблиць. Для реалізації процедур

завантаження даних з ERP-систем в команду розробників повинен бути включений фахівець, добре знайомий з даною системою-джерелом, так як аналіз подібного роду систем з нуля займає надто тривалий час. Крім того, більшість постачальників засобів ETL надають коннектори до багатьох ERP-систем, що дозволяє імпортувати метадані ERP-систем і працювати з ними на більш високому рівні. Наявність конекторів до ERP-систем надає спеціалізованих засобів ETL велика перевага над написанням вручну процедур завантаження даних, в разі якщо в якості джерела даних виступає ERP-система.

Після виконання попереднього планування приступають до детального планування. Деталізовані плани перетворення даних складаються для всіх таблиць, які беруть участь в процесі перетворення.

Модель ETL-системи може бути створена навіть з використанням дошки та крейди у лекційній аудиторії, чи навіть на аркуші паперу. На сьогодні дуже багато є додаткових систем, в яких можна створити візуальну концепцію ETL-системи, так безкоштовними є додатки Google - draw.io, який є безкоштовним і пропонує користувачу аплікацію для створення діаграм, яке дозволяє малювати:

- Блок-схеми;
- UML;
- Діаграми сутність-зв'язок;
- Мережеві діаграми;
- Моделі бізнес-процесів;
- Організаційні схеми;
- Електричні схеми;
- Каркасні схеми і моделі;

І має такі можливості:

- Власний HTML 5 Client з повною підтримкою IE 6-8;
- Велика вбудована бібліотека елементів;

- Інтуїтивний інтерфейс за принципом перетягування;
- Функція пошуку і додавання зображень;
- Експорт у формати PNG / JPG / XML / SVG / PDF;
- Підтримка сенсорними пристроями;
- Спільна робота в реальному часі;
- Вставка діаграм в блоги і вікі-сайти. [8]

Таким чином, розробка ETL-процесу включає в себе наступні основні стадії:

1. Планування ETL-процесу.
2. Конструювання процесу заповнення таблиць вимірів.
3. Конструювання процесу заповнення таблиць фактів.
4. Вилучення даних.
5. Перетворення і очищення даних.
6. Завантаження даних.

При проектуванні процесів перетворення даних проектувальник СД повинен вирішити такі завдання:

- проаналізувати вимоги до даних СД;
- проаналізувати і описати джерела даних для СД;
- створити модель перетворення даних високого рівня;
- визначити і детально описати кожну задачу перетворення даних.

Деталізація і продуманість компонентів архітектурного ETL-рішення, служать запорукою успіху будь-якого застосування. Розбиття всього ETL-процесу завантаження типових сутностей на сукупність логічно відокремлених завдань дозволяє максимально формалізувати кожну з них, представивши у вигляді фіксованого набору операцій, і, відповідно, реалізовувати весь ETL-процес на базі заздалегідь підготовленого набору стандартних шаблонів. Це актуально з урахуванням повторюваності алгоритмів обробки даних в рамках типових сутностей. Такий підхід дозволяє домогтися чіткості в побудові ETL-

процесу, полегшує його супровід і сприяє швидкій локалізації і виявлення проблеми в разі неполадок. Природно, допускаються відхилення від стандартних шаблонів при реалізації процесів завантаження індивідуальних сутностей. Ці зміни можуть бути спрямовані, наприклад, на оптимізацію швидкості завантаження окремих сутностей. Крім цього, архітектура рішення повинна також передбачати механізми управління запуском процесу завантаження даних і окремих його складових. В їх рамках можуть бути реалізовані конкретні пропозиції щодо: взаємодії окремих операцій процесу (наприклад, передачі поточних значень параметрів); організації циклічних завантажень; ведення контрольної інформації по завантаженнях даних. [9]

1.4. Основні функції ETL-систем

Як вже згадували основними етапами є Extract, Transform, Load. Дослівний переклад: Витяг, Перетворення, Завантаження. Ці етапи і є основними функціями ETL-системи. У загальному випадку - розуміється складовий процес перенесення даних однієї програми або автоматизованої інформаційної системи в інші.

Простим визначенням ETL-системи може бути «набір процесів для отримання даних з OLTP-систем в сховище даних». Коли ми дивимося на корені ETL, це, ймовірно, життєздатне визначення, але для сучасних рішень ETL це сильно спрощує цей термін. Дані надходять не тільки з OLTP-систем, але і з веб-сайтів, простих файлів, баз даних електронної пошти, електронних таблиць і особистих баз даних, таких як Access. ETL використовується не тільки для завантаження одного сховища даних, але може мати безліч інших варіантів використання, таких як завантаження вітрин даних, створення електронних таблиць, оцінка клієнтів з використанням моделей інтелектуального аналізу даних або навіть завантаження прогнозів назад в системи OLTP. Однак основні кроки ETL все ще можна згрупувати в три розділи:

1. Витяг: вся обробка, необхідна для підключення до різних джерел даних, отримання даних з цих джерел даних і надання даних для подальших етапів обробки. Це може здатися тривіальним, але насправді може бути одним з головних перешкод в отриманні рішення ETL з нуля.

2. Перетворення: будь-яка функція, яка застосовується до витягнутим даними між витягом з джерел і завантаженням в цільові об'єкти. Ці функції можуть містити (але не обмежуються ними) такі операції:

- Рух даних.
- Перевірка даних за правилами якості даних.
- Модифікація змісту або структури даних.
- Інтеграція даних з даними з інших джерел.
- Розрахунок похідних або агрегованих значень на основі оброблених даних.

3. Завантаження: всі процеси, необхідні для завантаження даних в цільову систему. Ця частина процесу складається з набагато більшого, ніж просто масове завантаження перетворених даних в цільову таблицю. Частини процесу завантаження включають, наприклад, управління сурогатними ключами і управління таблицями вимірювань.

1.5. Постановка задачі дослідження.

Система повинна зберігати інформацію про процеси, етапи та результати виконання цих процесів. Головною її перевагою повинно бути те, що процеси можуть бути розрізнені по різноманітним системам але результати будуть доступні в одній базі даних, результати будуть доступні, прозорі та мати повну інформацію.

Дана система повинна бути розширена розробниками для більшого розгалуження та для збільшення різноманітних сутностей.

Ця система спростить роботу користувача, та буде відповідати сучасним

тенденціям. Всі дані користувача будуть збережені та кінцеві юзери зможуть користуватись даними у будь-який час, система не буде потребувати зберігання даних у оперативній пам’яті та не буде потребувати додаткового устаткування.

РОЗДІЛ 2
АНАЛІЗ ПІДХОДІВ ДО ЛОГОТУВАННЯ ETL-СИСТЕМ

2.1. Призначення логування процесів

Будь-який процес потребує логування та фіксації всіх етапів та всіх процесів. Так всі додатки можуть мати файл з даними по процесам, по транзакціям чи операціям користувача. так і всі процеси, які виконуються у ETL-системі на мій погляд повинні бути записані, фіксовані. Найчастіше, розробники використовують лише записи щодо помилок та обривів процесів, але є і такі процеси, які навіть помилки не логують. Є проекти, де логується, записується всі дії користувача, це, як правило це фінансові системи, системи заявок чи медицинські процеси, де кожна дія оператора чи системи в цілому повинна бути зафіксована, тобто записані дата та час початку процесу, кінця, що саме відбувалось, та результат.

Так, згідно Wikipedia “Журнал роботи” (англ. Logging) – форма автоматичного запису в хронологічному порядку операцій в інформаційних технологіях, процес запису інформації про події в рамках будь-якого процесу з деяким об'єктом події, наприклад, в файл реєстрації або в базу даних. У деяких програмний комплексах використовується термін "аудит", що є невірним, оскільки аудит має на увазі порівняння чогось з чимось, чого-то на предмет відповідності, наприклад, вимогам, іншими словами це кореляційний процес. [10]

Такі процеси не є обов'язковими в багатьох системах, і багато розробників та компаній не використовують запис журналів роботи у своїх системах. Але, як показує практика, такі журнали можуть бути дуже корисні. так, наприклад в будь-якій фінансовій системі - користувач знімає кошти з рахунку, може бути збій, чи шахрайські дії, чи взагалі користувач може забути про свої дії, але на

стороні системи буде чіткий запис з датою, часом, типом підтвердження логіну/користувача, сума та тип зміни балансу. І цей звіт може бути корисний для розуміння причини зміни балансу, така ж ситуація щодо будь-якої заявки, так всі заявки до державних органів логуються – записується дата та час створення заявки, користувач - його дані, система ідентифікації користувача (ЄЦП, Bank-id, логін/пароль системи заявок, інше), і кожний наступний крок проходження заявки також логується. Записується детальна інформація щодо дати/часу, зміни стану, та користувача записує.

Також дуже часто розробники програмного забезпечення включають в код обробки помилок, тобто блоки try/catch де є можливість отримувати текст помилки та систему. Найчастіше, такі помилки накопичують десь у кеші чи у log-файлі, чи виводять користувачу, чи взагалі нічого не відбувається, а метод/функція продовжує функціонування. То навіщо взагалі системи логування, так для помилок, чому логування – важлива складова будь-якої системи. Так би мовити розбір причин, що куди і як. На основі чого, можна спробувати дати відповідь на питання "чому?".

При налагоджуванні програми думаєте що за один налагоджувальний сеанс виправити всі проблеми, що виникли в рамках цього завдання. Але наша недалекоглядність не хоче вірити в те, що насправді там не одна проблема, а кілька. І за один оцінний сеанс не вийде вирішити всі ці проблеми.

Тому вам треба буде кілька разів запускати цей код в відлагоджувальному режимі, проводячи години налагодження над одним і тим же шматком коду. І це тільки ви один стільки часу витратили над цією частиною програми. Кожен член команди, кому «пощастило» працювати з цим кодом, буде змушений прожити ту ж саму історію, яку прожили ви.

Не кажучи про те, що люди в командах міняються, команди міняються і так далі. Людино-години йдуть на одне і те ж.

Можливий алгоритм вирішення проблеми:

- Розбити на окремі частини.
- Викидаємо налагодження, просто забороняємо користуватися режимом Debug.
- Аналізуємо окремі частини (придумуємо для них невалідність ситуації, граничні випадки).
- Пишемо тести на кожну окрему частину всього алгоритму.
- У тестах іноді доводиться дізнаватися проміжні дані, але ...
- Налагодження нам більш недоступна, тому встромляємо Trace в ті частини, де виникає підозра на некоректне виконання алгоритму.
- За трасування потрібно зрозуміти причину проблеми.
- Якщо не зрозуміло, то найчастіше або варто написати ще тест, або виконати трасування на один етап раніше.

І такий процес може повторюватись тисячі разів. Тому чи не простіше одноразово створити систему, яка буде логувати, записувати помилки, час їх виникнення, процес, у якому вони виникли, крок, чи рівень вкладеності, та саму помилку.

У загальному випадку – логи (запис про операцію) записуються чи у файл чи у базу даних. І тут виникають основні труднощі, наприклад є процес опрацювання та модифікації даних на стороні додатку, база даних у свою чергу в цей час без діяльності, і навпаки може бути викликана велика складна процедура чи функція з багатьма операціями всередині, і верхній процес загальний, не може знати що відбувається у середині процедури чи функції.

2.2. Основні підходи до логування

Основним підходом до логування є запис всієї необхідної інформації у кінцеву систему накопичення інформації. Так за системою зберігання інформації можна виділити наступні системи запису:

- кеш процесу
- файлова система
- база даних (локальна чи хмарна).

Зберігання даних у кеші процесу реалізується у об'єктних системах. Створюється об'єкт класу логу – куди додається у змінну типу списку: у системі .NET наслідування інтерфейсу System.Collection.IEnumerable, у системах, базованих на технологіях JAVA – це може бути Iterable. І в ці перерахування додається новий об'єкт чи структура з даними по логованій операції. Величезний мінус та недолік цього підходу є час життя цієї інформації. Тобто при перезапуску, чи вимкненні програми – всі дані будуть загублені, ще один недолік в тому, що кеш - тобто операційна пам'ять обмежена та на багато порядків менше ніж доступна пам'ять на жорсткому диску. Тобто навіть при постійній роботі програми – то через деякий час чи буде переповнення пам'яті та помилка всієї програми чи примусове очищення. І користувач не буде мати можливостей отримати інформацію щодо операцій у минулий час.

Збереження інформації, логів, у файлову систему – є дуже поширеним методом логування. Всі операції, всі дії, помилки чи транзакції записуються до файлової системи. Так у системах, працюючих на ОС Windows є можливість навіть записувати інформацію до Windows логів, та зберігати інформацію там. дуже часто використовується запис у файли *.txt у папці самої програми чи по обраному шляху користувачем чи адміністратором системи. Це можуть бути строки з записом дати/часу та додаткових атрибутів у одну строку за певним шаблоном. Чи це можливо через запис у форматі XML чи JSON. Останній варіант більш поширений, через те, що обробка та витяг з такого формату простіший та більш швидкий, ніж зчитування та парсинг строк, навіть записаних по шаблону. Багато розробників створюють цілу файлову систему з багатьма вкладеними папками, які в свою чергу можуть мати ієрархічну чи часову структури чи конкатенацію обох. Так це може бути створено дерево

каталогів, в якому папки - це може бути рік чи місяць, а в середині файли по дням, чи тижням, файлова система за датами показана на рисунку 1.1, де кожний верхній блок - це каталог, а кінцевий - це файл формату *.txt .

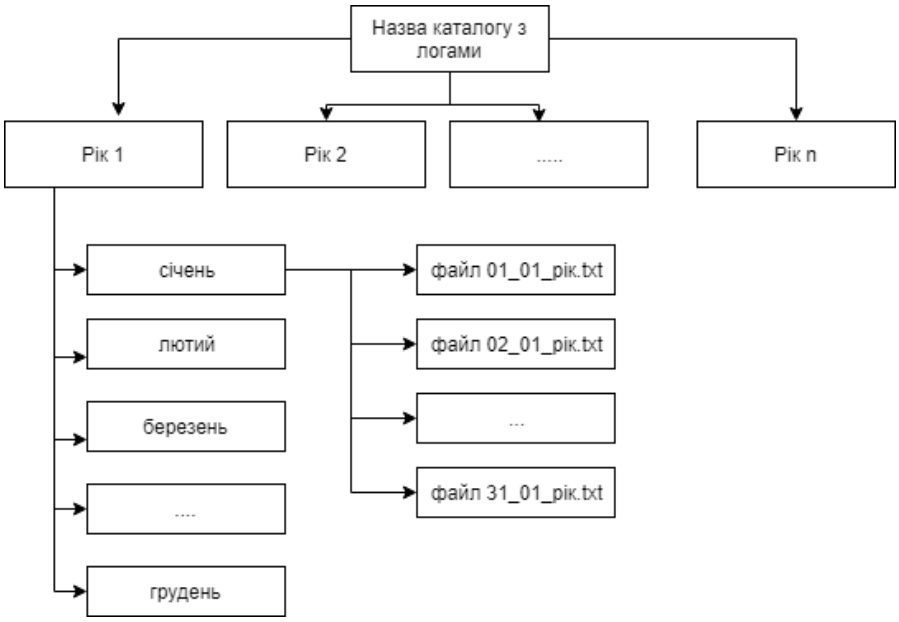

Рис. 2.1. Приклад збереження даних у файловій системі

В такому випадку завантаження даних з файлів для перегляду буде з однієї сторони простіше за рахунок більш швидкого пошуку за датою, та складніше, якщо буде потрібно витягнути дані за певний проміжок часу, тоді потрібно буде завантажувати декілька файлів та об'єднувати дані з них. рішення приймається архітектором рішення, в залежності від аналізу розгалуженості даних, прогнозів обсягів даних, прогнозів зворотного завантаження та перегляду та інших факторів. Мінусом такої системи, як вже було вказано раніше – є неможливість запису чи більш складний процес запису, при використанні складних процедур чи функцій на стороні бази даних.

Системи логування з використанням баз даних – передбачають записи операцій у базу даних. Створюється окрема таблиця, чи набір таблиць в які проводиться запис необхідних даних. Перевагою такого підходу є те, що таблиця бази даних не є тимчасовою та можливо отримувати дані за запитом чи

створення окремої системи для швидкого отримання звітів. І також що таблиця буде доступна навіть після відключення програми, чи перевантаження, і дані збережуться. Недоліком є те, що потрібно у процесах, які відбуваються не на стороні бази даних, створити підключення до бази чи використовувати наявне та викликати процедуру, функцію чи взагалі код для запису даних у ці таблиці.

За характером інформації, що записується у лог можна виділити основні типи:

1. Це логування помилок (Exceptions/errors) які виникають у процесах.

2. Логування кінцевих результатів – тобто після кожної транзакції чи закінченого процесу у систему буде записаний результат (вдалий чи невдалий).

3. Запис усіх даних до логів, тобто в цьому випадку логується кожний крок з вхідними параметрами, з кінцевим результатом, з помилками та їх описом.

Визначення та обрання типу системи обирає архітектор чи керівник проекту. Для обрання типу необхідно аналізувати можливості системи процесів, можливості системи збереження логів, необхідність ведення запису всіх кроків чи лише результату чи помилок тільки, можливість ведення логування, напруженість системи, кількість транзакцій, і т.д.

Наприклад для системи медичних установ, де програмне забезпечення виконує всі функції з пацієнтами, виконує аналіз результатів обстеження, проводить верифікацію аналізів пацієнта, та виконує дуже важливі функції, в яких неприпустимі помилки чи збої – в таких системах логуватись повинен кожний крок, кожна транзакція, чи кожен розрахунок, для того, щоб, якщо, навіть виникає питання до системи чи якась помилка, то працівники мали б можливість подивитись всі кроки і або виявити причину чи вручну зробити висновки чи аналізи, поки система буде налагоджуватись програмістами чи адміністраторами.

Чи у системі величезного національного банку, де кількість клієнтів більше ніж 10 мільйонів, записувати кожний крок та всі вхідні параметри усіх дій - буде дуже коштовно для компанії та дуже складно зі сторони підтримки. Тому такі банки логують у фінансових транзакціях тільки початок-кінець і результат загальної транзакції, а для систем заявок чи звернень, взагалі може бути реалізована система лише запису помилок, чи взагалі без системи логування.

Так SQL Server Integration Services э на сьогодні однією з найбільш розповсюджених систем для побудови ETL процесів. А об'єднання бази даних SQL Server, пакетів SSIS та програмного забезпечення на базі .NET надає користувачеві величезні можливості та функціонал.

Вбудовані можливості логування в середу Integration Services

SQL Server Служби Integration Services містять реєстратори, які можуть використовуватися для реалізації ведення журналів в пакетах, контейнерах і завданнях. При веденні журналу можна записати поточні відомості про пакет, що допоможе при кожному виконанні пакета проводити його аудит і усувати несправності. Наприклад, журнал може записати ім'я оператора, що запускає пакет, і час, в яке пакет був початий або завершений.

Ви можете налаштувати область ведення журналу на час виконання пакету на сервері Служби Integration Services. Можна також включити ведення журналу при виконанні пакета з командного рядка за допомогою програми dtexec.

Журнали асоціюються з пакетами і налаштовуються на рівні пакета. Кожне завдання або контейнер пакета може вести журнал в будь-якому журналі пакета. Можна включити ведення журналів завдань і контейнерів пакета, навіть якщо ведення журналу містить їх пакета не включено. Наприклад, можна включити ведення журналу завдання «Виконання SQL», не включаючи ведення журналу її батьківського пакета. Пакет, контейнер і завдання можуть робити

записи в декількох журналах. Можна включити ведення журналу тільки для пакета або для будь-якої індивідуальної завдання або контейнера, що міститься в пакеті.

При додаванні журналу до пакету виберіть реєстратор і місце розташування журналу. Реєстратор вказує формат для журнальних даних: наприклад, база даних служб SQL Server або текстовий файл.

Служби Integration Services включають такі реєстратори:

Постачальники журналів текстових файлів, які ведуть журнальні записи в текстових файлах ASCII в форматі значень, розділених комами (CSV). За замовчуванням для імені файлу для даного реєстратора використовується розширення LOG.

Постачальник журналу Додаток SQL Server Profiler, який записує трасування, які можуть бути переглянуті за допомогою програми SQL Server Profiler. За замовчуванням для імені файлу даного реєстратора використовується розширення TRC.

Реєстратор SQL Server, який записує елементи журналу в таблицю sysssislog бази даних SQL Server.

Постачальник служби Windows «Журнал подій», який веде журнальні записи в прикладному журналі служби Windows «Журнал подій» на локальному комп'ютері.

Постачальник журналу XML File, який записує журнальні файли в XML-файл. За замовчуванням для імені файлу даного реєстратора використовується розширення XML.

При додаванні реєстратора до пакету або програмної налаштування ведення журналу для ідентифікації реєстратора використовуйте або ProgID, або ClassID. Це робиться замість використання імен, які конструктор служб Служби SSIS відображає в діалоговому вікні Налаштування журналів служб SSIS. [11]

Тож як бачимо, виділяють 2 основні підходи до логування в SSIS:

1. Провайдери логів (logging providers).

Найпростіший варіант організації логування в пакетах SSIS - використання logging providers. Це рішення дозволяє писати логи в текстові файли, в базу даних або, наприклад, в журнал операційної системи. Плюс можна гнучко налаштовувати які події для якого компонента будуть писатися в логи. Як правило я залишаю тільки події OnError, щоб таблиця сильно не сердилася, хоча для цілей налагодження цілком можна фіксувати події OnPreExecute і OnPostExecute.

2. Обробники подій (event handlers).

Як показує практика, інформація, яку пишуть провайдери логів, не завжди достатньо. Особливо це проявляється у випадках, коли через SSIS проходить велика кількість файлів для завантаження. І на основі інформації, що надійшла від logging provider'a, не завжди можна швидко виявити файл, з яким виникли проблеми. На щастя SSIS дає можливість роботи з оброблювачами подій, що дозволяє значно розширити функціонал класичного провайдера логів. [12]

Побудова логування на базі тригерів

Система логування може бути побудована лише на стороні бази даних. Якщо є необхідність логувати лише транзакції, у яких відбуваються зміни даних тоді можливо побудувати системи запису на базі тригерів. Так, як було раніше згадано тригер - це модулі, автоматично запускаються сервером БД при певній події, наприклад видаленні даних з тієї чи іншої таблиці. [13]

Тригери – це особливі процедури, що зберігаються, автоматично виконуються при використанні бази даних певним чином. З будь-якою операцією, що викликає зміна вмісту таблиці, можна зв'язати супутню дію (тригер), яку СУБД повинна виконувати і під час кожної такої операції. Тригери можуть бути пов'язані з виконанням операцій INSERT, UPDATE і DELETE (або будь-якої їх комбінації) по відношенню до вказаних таблиць.[14]

Так найчастіше, розробники створюють вбудовані тригери з логуванням операцій за базою даних, у цих процесах записується дата та час, користувач, ім'я таблиці і що відбувається (додається, видаляється чи змінюється інформація в базі даних). Але такий підхід не зовсім коректний, тому що все одно додатково необхідно створювати блоки try-catch на кожну операцію і блоках catch, у разі виникнення помилок записувати дані у таблиці логів. Ще одним мінусом є те, що у тригерах доступні лише дані з таблиць, які піддаються змінам, тобто жодним чином немає можливостей визначити начального користувача, який намагається провести зміни і немає розуміння операції, у ході якої відбувається зміна даних у таблиці.

Тому використовування тригерів у системі логування можливе, але виключно для систем, де зміна таблиць відбувається лише одним чином, і де немає потреби логування юзерів і функціоналу.

Можливості доповнення логування сторонніми продуктами

Також на сьогодення існує багато додаткових продуктів системного забезпечення для створення ETL моделей. Так вже існує багато інтерфейсів для створення блок схем для початкового моделювання систем. Так на початку роботи я вказував на draw.io, також є багато додатків, таких як Talend, Star UML, Apache Hive, Apache Pig і MapReduce та інші.

Тим паче, що Star UML дозволяє створювати програмний код згідно створених систем, тобто ви можете створити візуально класи та процедури, а програма створить програмний код, згідно вашого малюнку. Так може бути створений код .NET, JAVA чи PHP.

В свою чергу програмне забезпечення для сховищ даних Apache Hive спрощує виконання запитів і управління великими наборами даних, розміщеними в розподіленої середовищі зберігання. Hive - це потужний інструмент для ETL, організації сховищ даних для Hadoop і управління базами даних для Hadoop. Однак він є відносно повільним, в порівнянні з традиційними

засобами управління базами даних. Він не пропонує всі функції SQL, або навіть функції управління базами даних, доступні в традиційних СУБД. Але він підтримує SQL, він функціонує як база даних, і він надає доступ до Hadoop більшій кількості користувачів (навіть не програмістам). Він пропонує можливість перетворювати неструктуровані і напівструктуровані дані в придатні для використання дані на базі схеми. Хочете створити систему управління основними даними? Ви можете зробити це, використовуючи Hive. Хочете створити сховище даних? Ви можете зробити і це з використанням Hive, але вам буде потрібно вивчити прийоми, що дозволяють зробити Hive потужним ETL-інструментом.

На відміну від Apache Pig і MapReduce, Hive дозволяє розробникам традиційних реляційних СУБД і інших фахівців, які знають SQL, простіше отримувати доступ до даних в Hadoop і перетворювати їх. Однак Pig не такий простий у розумінні, і потрібно чимало додаткове навчання для тих, хто не є розробником програмного забезпечення. MapReduce – це технологія, яку програмісти на Java, C ++ і Python можуть освоїти відносно швидко. Але без знання таких технологій, як Java, майже неможливо вивчити MapReduce. Тому, якщо ви знаєте SQL, то Hive може бути відносно простий у вивченні і використанні. [15] І у Hive і у інших додатків можливо задіяти систему логування. В Hive – це вкладена функція та не потребує додаткового програмування та створення коду, у MapReduce та Pi – це можливо за рахунок створення нових класів та об'єктів які будуть створювати логи.

Але всі ці додатки та програми є не безкоштовними, тому для фірм та компаній користування цими програмами буде невигідно.

2.3. Реалізація цілісної системи логування на базі SSIS і безпосередньо бази даних

Як вже було зазначено, на сьогодні є багато можливих систем логування процесів, багато можливостей реалізації та підходів до створення таких систем. Але з моєї точки зору найбільш функціональною є система, створена на стороні бази даних, з можливістю виклику процедур чи функцій додавання записів у таблиці логів.

Збереження логів на стороні бази даних надає перевагу у збереженні даних, навіть, якщо, основна програма буде примусово завершена чи буде помилка, чи дані будуть стерті, то дані на стороні бази даних залишаться для подальшого аналізу. Також перевагою є можливість створення звітів та отримання даних по процесам швидко та з допомогою великого функціоналу фільтрів, без додаткових витрат операційної пам'яті та продуктивності. Навіть для порівняння є запит від користувача на отримання сету даних вивести дані по операціям користувача за окремий період і по окремих операціях. Для файлової системи, незалежно від реалізації, потрібно завантажувати декілька файлів чи один дуже великий, після чого перетворювати це на перерахування об'єктів, створення linq to sql чи запросу на стороні програми, запит на дані, створення нового перерахування об'єктів, що знову займає операційну пам'ять і виведення на екран чи передача користувачу у потрібному форматі. Для бази даних - це простий запит з використання індексів, бистрий пошук та виведення дата сету на сторону додатку, програма в свою чергу виводить інформацію користувачу і потрібному форматі.

Та запису даних – при використанні SSIS пакету, як було раніше вказано, можливо використовувати евенти(events) на виконання чи на завершення та на помилки. Кожен з цих евентів може викликати просту процедуру чи функцію, чи взагалі код додавання запису у таблицю. А на стороні бази даних, при виклику складних процедур чи функцій буде додано в кожний крок додавання

рядків в потрібну таблицю з інформацією про операцію. А при помилці на стороні SSIS буде спрацьовувати евент на помилку (On error event) який у свою чергу буде обновляти запис у таблиці та проставляти статус помилки з додаванням коду та опису помилки, а на стороні бази даних - в блоці CATCH додається оновлення також цієї таблиці і проставлення відповідного статусу. Так пропонується запис у таблицю логів при початку операції записувати зі статусом “в роботі” і після завершення обновляти статус на “опрацьовано успішно” чи при помилці на статус “помилка” та додавати коментар з описом помилки.

РОЗДІЛ 3
ПРОЕКТУВАННЯ, РОЗРОБКА ТА ДОСЛІДЖЕННЯ МОДЕЛІ ЛОГУВАННЯ В ETL СИСТЕМАХ

3.1. Функціональне призначення системи

На сьогодні майже всі процеси в світі логуються, фіксуються, та десь міститься інформація про той чи інший процес. Це можна порівняти з життям людини. Так, при народжені – надається свідоцтво про народження, потім заводиться медична картка, в якій фіксуються щеплення, потім, коли людина йде до школи – ведеться щоденник з фіксацією розкладу занять та оцінок, багато журналів, на кожний предмет. Потім видається паспорт людині, потім університет – де все також фіксується, і результати будь-якої діяльності також фіксуються, надаються різноманітні сертифікати, дипломи, атестати. І наприкінці життя у людини зберігається купа різноманітних документів, фото і в електронному вигляді багато інформації. І так людина може пригадати свої будь-які кроки, досягнення, чи дії. Так і в світі комп’ютерних процесів треба вести логування, фіксацію всіх процесів, результатів, вхідних та вихідних параметрів і так далі. Сама найрозповсюджена система такого логування – це банківська система ведення рахунку. Якщо у Вас э банківський рахунок – ви можете отримати інформацію починаючи з першого дня користування про всі операції, всі деталі. Тобто, якщо не було б логування – то Ви б мали змогу лише спостерігати теперішній стан Вашого рахунку, і навіть не мали б змоги відслідкувати зміни балансу.

На сьогодні майже всі процеси та технології мають так чи інакше хоч якусь систему логування. Тому і виникає питання як впровадити таку систему, яка буде працювати з будь-якими додатками та у різних інстансах. Ця система можлива для імплементації в будь-яку технологію чи у вже працюючий

функціонал. Головною умовою є наявність бази даних, і головне – будь-якої реляційної бази даних. Тобто, цю систему можуть використовувати користувачі як з Oracle, так з MS SQL, PostgreSQL, Sybase, Informix чи іншими.

Ця система буде зберігати інформацію про процеси, етапи та результати виконання цих процесів. Головною її перевагою є те, що процеси можуть бути розрізнені по різноманітним системам але результати будуть доступні в одній базі даних, результати будуть доступні, прозорі та мати повну інформацію.

Дана система може бути розширена розробниками для більшого розгалуження та для збільшення різноманітних сутностей.

Ця система спростить роботу користувача, та буде відповідати сучасним тенденціям. Всі дані користувача будуть збережені та кінцеві юзери зможуть користуватись даними у будь-який час, система не буде потребувати зберігання даних у оперативній пам’яті та не буде потребувати додаткового устаткування.

3.2. Опис використаних технологій

Проаналізувавши поставлені задачі, вирішено реалізувати задану систему на базі Microsoft SQL Server Database та за допомогою інструментарію розробки Microsoft SQL Server Management Studio. Додатково використовувалась мова C# .NET а допомогою інструментарію розробки Microsoft Visual Studio.

Основними технологіями в цій роботі є використання реляційної бази даних – як сховища та середи зберігання даних, які необхідні для параметризації процесу. Основною технологією в цій роботі є використання SQL синтаксису, це може бути як T-SQL так і PL/SQL чи PSQL чи PL/pgSQL чи будь-який інший процедурний язик, який підтримує стандарти ANSI SQL-92. Також в роботі буде використано частково для реалізації проектування додаток, розроблений за допомогою SSIS (.NET 4.5 C# платформа)

Через те, що тип реляційної бази даних не впливає на розробку чи на результат, розробник може обрати реляційну базу даних згідно свої особистих переконань чи потреб проекту в цілому. В нашій роботі було обрано середу Microsoft SQL Server Database для зберігання логів через те, що можливо також реалізація деякого додаткового функціоналу з використанням SSIS.

Для побудови додатку SQL Server Integration Services (SSIS) також потрібно мати базу даних і додаток буде розроблений на базі C# .NET .

Основним інструментарієм для побудови коду буде Microsoft SQL Server Management Studio для створення об’єктів на стороні серверу бази даних та додатково використовувалась мова C# .NET та допомогою інструментарію розробки Microsoft Visual Studio 2017.

3.3. Опис структури системи та алгоритмів функціонування

Система розроблена на базі реляційної бази даних то основними об’єктами системи будуть таблиці бази даних, та код який створює функціонал. Цей код може бути представлений у вигляді збережених процедур чи функцій бази даних. На стороні додатків – основними об’єктами виступають запити к базі даних та процеси, які мають бути логовані в рамках системи запису логів (логуванні). Функціоналом є передача даних та запис їх у базу даних, та, дуже важливим пунктом є виведення інформації по запиту, тобто, коли потрібно отримати дані по процесу чи історії процесу чи його параметрів чи характеристик.

Головними об’єктами бази даних будуть виступати 3 основні таблиці з даними по процесу, також наданий код процедури для завантаження даних в таблицю процесу. Для тестування розроблено додатково на стороні базі даних 2 простих таблиці та один тригер.

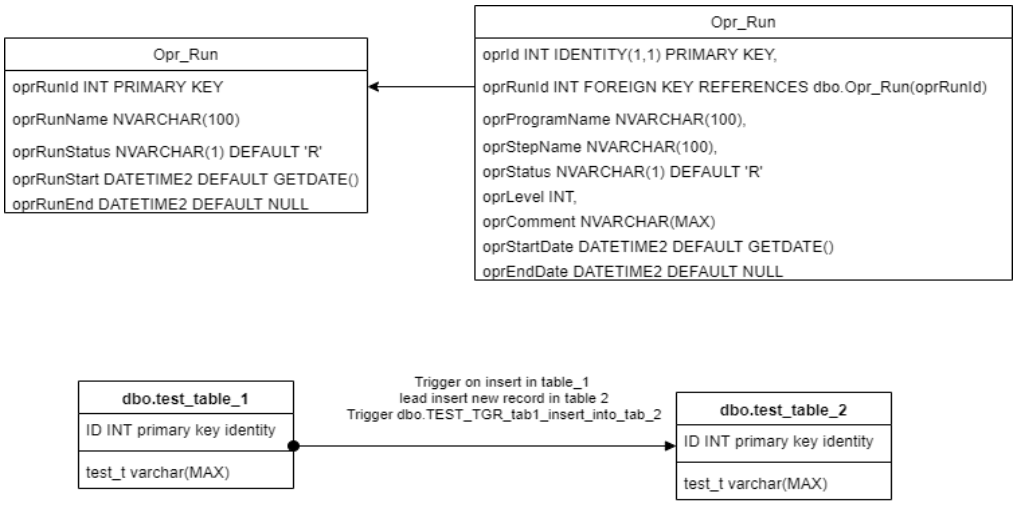

Рис. 3.1. Схема таблиць бази даних

Також розроблений код для додавання інформації в таблиці логів. Він не є об’єктом та може зберігатись в будь-якому місці та середовищі. Так код додавання інформації може бути присутній на стороні додатків та у функціях, як код – який передається до бази даних.

Зовнішніми об’єктами є підключення к базі даних, та евенти (події) на стороні додатку і код, який буде вносити дані в таблиці логів. Зовнішніми об’єктами можуть бути будь-які процеси, в яких є можливість підключення к відкритій базі дані та користувач має доступи на додавання та зміну інформації в цих таблицях. Для реалізації системи логування ми використали процес SQL Server Integration Services (SSIS) та події, які будуть вносити дані в наші таблиці тим самим пишучи лог по процесу.

Система працює як процес логування (запису) всіх даних по процесам. Тому в роботі розроблено систему ETL, яка завантажує дані з двох джерел та вивантажує дані у базу даних, запускає 3 процедури та вивантажує результат у 2 різних вихідних файли. Призначенням системи є не логічність чи цілісність даних, а показати можливості розробленої моделі для логування будь-яких процесів, збереження вхідних даних, параметрів та результатів виконання кроків системи.

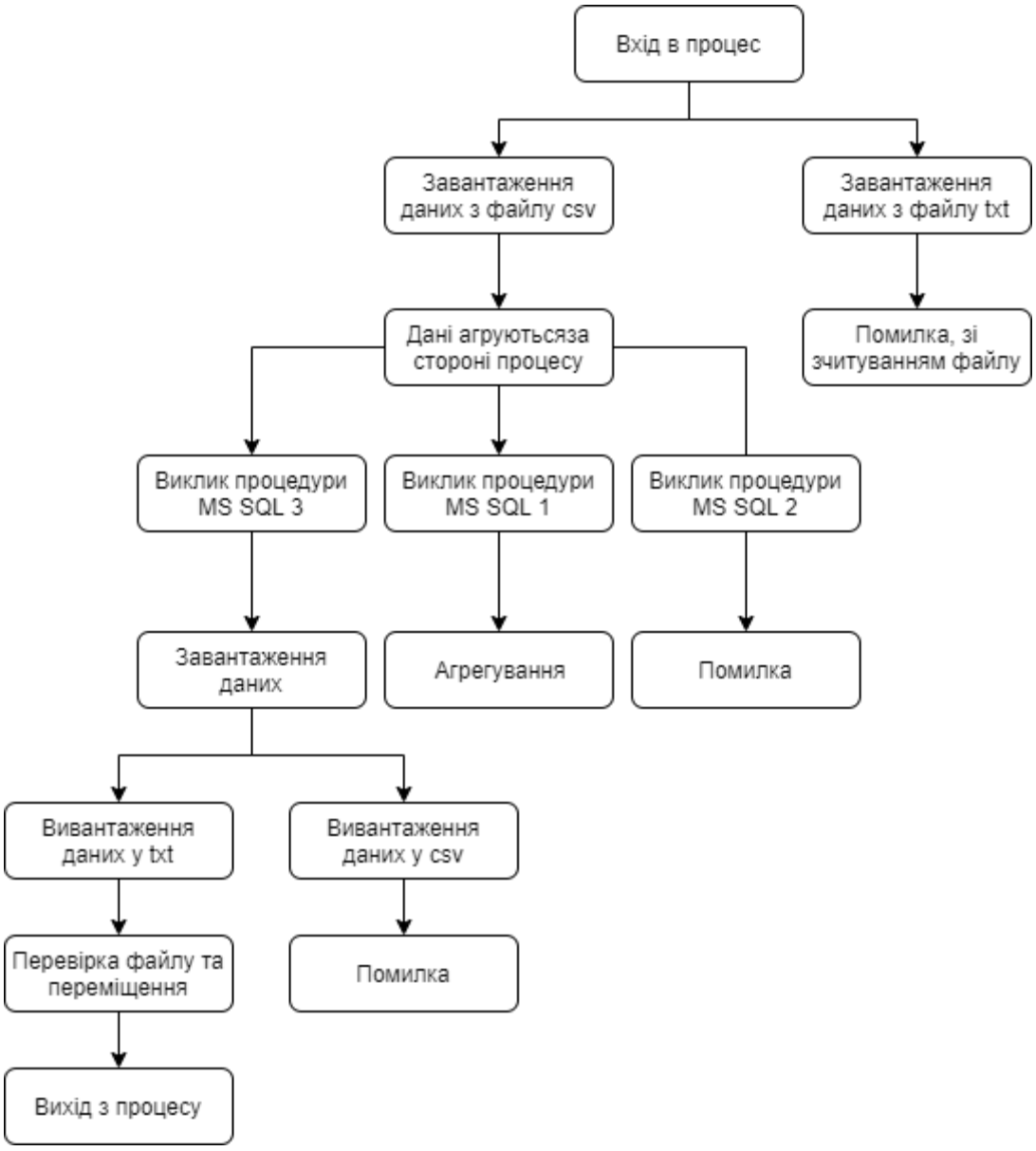

Рис. 3.2. Схема ETL процесу для логування

Розроблена наступна схема роботи вигаданої системи ETL: декілька процесів завантаження даних, один із яких буде вести до помилки, інший до успішного результату, так само буде викликатись 3 збережені процедури, одна буде завантажувати дані з джерела до бази, інша агрегувати та щось робити за даними та завантажувати в іншу таблицю, і третя – буде мати декілька кроків, і один з яких буде вести до помилки. І третім етапом буде вивантаження даних у файли csv, де не буде знайдений шлях до папки і вести до помилки, та інший процес, який буде писати дані у файл txt.

Так кожний крок системи має записувати дані у базу даних, по кожному кроку, зі всіма характеристиками та параметрами. Навіть якщо якісь процеси ведуть до помилок.

Розберемо докладніше кожний крок системи, щоб розуміти, які дані повинні бути логовані та записані у базу. Процес буде запускатись декілька разів, щоб отримати декілька результатів та різні вхідні параметри, можливо також різні результати виконання для того, щоб показати різні можливості системи для логування.

Вся ETL система буде розроблена за допомогою SSIS пакетів та Microsoft Visual Studio 2017 як візуальної компоненти. Детальніше охарактеризуємо елементи та кроки системи:

1. Вхід у процес (Enter to process) –це виключно логічний блок, який отримує поточний код процесу та записує в таблицю виконання дані про поточний запуск (поточне виконання) .

2. Завантаження файлу csv (Load_scv) – це процес Data Flow Task в SSIS пакеті, де використовується Flat File Source та Connction manager Connction_to_load_csv. Параметри якого – file name буде змінною, і буде визначатись, як @file_path + @file_name . Результатом ми програмуємо помилку

3. Завантаження файлу txt (Load_txt) – це Script Task написана на Microsoft Visual C# 2017, реалізована як клас в проекті, який буде імплементувати логіку розробника. Вхідними параметрами так само будуть @file_path + @file_name. У середині коду буде визначений шлях до файлу, використана бібліотека System.IO та завантажені дані у програму. Результатом буде успіх.

4. Агрегація (Aggregation) – це будь-який код/будь-який блок, за для додаткового навантаження на логіку, виключно є частиною, що відображає якусь логіку. В нашому випадку змінює параметр @file_name.

5. Виконання збережених процедур (Execute SQL proc 1) – це 3 запити в базу даних. Першим запитом ми викликаємо збережену процедуру dbo.sp_test_1, яка в середині має декілька блоків логіки: перший – перевірка даних, другий – агрегація, третій – оновлення даних. Другим кроком ми викликаємо процедуру, яка на зовні буде видавати помилку. Та третім кроком – буде процедура, яка в середині має блок try/catch і буде помилка, але оброблена на стороні бази даних. Лістинг всіх трьох процедур приведений у додатку 2 до диплому [Додаток 2]. Кожна з процедур має різноманітні вхідні параметри та результати виконання. Але основним для кожної процедури є вхідний параметр який відповідає за унікальний ідентифікатор процесу. Він потрібен для того, щоб по ньому додавати запис в таблицю логів.

6. Вивантаження даних у табличний плоский файл (Export CSV) – Це Data Flow Task, яка включає в себе вивантаження даних у файл формату csv у вказану директорію та з заданим ім’ям файлу. Цей крок повинен вести до помилки.

7. Вивантаження даних у текстовий файл (Export txt) – це виконання bat файлу, який в свою чергу парсить csv файл та перетворює його на декілька менших файлів.

8. Перевірка вивантажених файлів (Check file) – цей крок – це File System Task, який через можливість встановлення атрибутів перевіряє наявність файлів, це робиться, щоб не міняти файл, чи не переміщати.

9. Останній блок – вихід із процесу ETL (Exit from process).Цей останній крок виконує лише роль фіксатора закінчення процесу, записує дані в таблицю запусків.

Рис. 3.3. SSIS ETL процес

Ця ETL система реалізована з використанням SSIS. Як було показано, розроблена ETL система має в собі дуже багато різноманітних компонентів, таких, як виконання збережених процедур, виконання BAT файлу, виконання .NET C# коду, різні блоки та вбудовані функції в SSIS такі як робота з файлами, агрегація, та Data Flow (процеси з даними).

Головним в цій системі є використання подій, та першочергове отримання коду процесу. Так, початковий блок «Enter to process» має у собі подію на отримання поточного коду виконання.

```
SET NOCOUNT ON;
DECLARE @runId INT = 0

IF NOT EXISTS (SELECT 1 FROM dbo.P1_Account_Updater_Opr_Run)
	BEGIN
		INSERT INTO dbo.P1_Account_Updater_Opr_Run(
			oprRunId,
			oprRunName
		)
		VALUES(1,'SSIS: AccountUpdater')
		SET @runId = 1
	END
ELSE
	BEGIN
		SELECT @runId = (SELECT MAX(oprRunId) +1 FROM
dbo.P1_Account_Updater_Opr_Run)
		INSERT INTO dbo.P1_Account_Updater_Opr_Run(
			oprRunId,
			oprRunName
		)
		VALUES(@runId,'SSIS: AccountUpdater')
	END

INSERT INTO dbo.P1_Account_Updater_Opr_Log(
	oprRunId,
	oprProgramName,
	oprStepName,
	oprLevel
)
VALUES(@runId,'SSIS','Package start',0)

      SELECT @runId as runId
```

Рис. 3.4. Лістинг отримання ідентифікатору процесу

Так, цим кодом ми отримаємо наступний код виконання, тобто, кожний наступний запуск виконання ETL процесу буде мати свій унікальний ідентифікатор, більший на одиницю за попередній. І всі дії, всі кроки у рамках поточного виконання будуть ідентифіковані цим номером запуску. І в цьому ж блоці ми записуємо наш перший запис у таблицю логів. З інформацією, що процес почався.

Головним чинником коректного запису є код, який використовується у подіях пакету. Так, навіть в інших системах створення ETL є свої системи подій, але вони є, чи їх можливо налаштувати для використання.

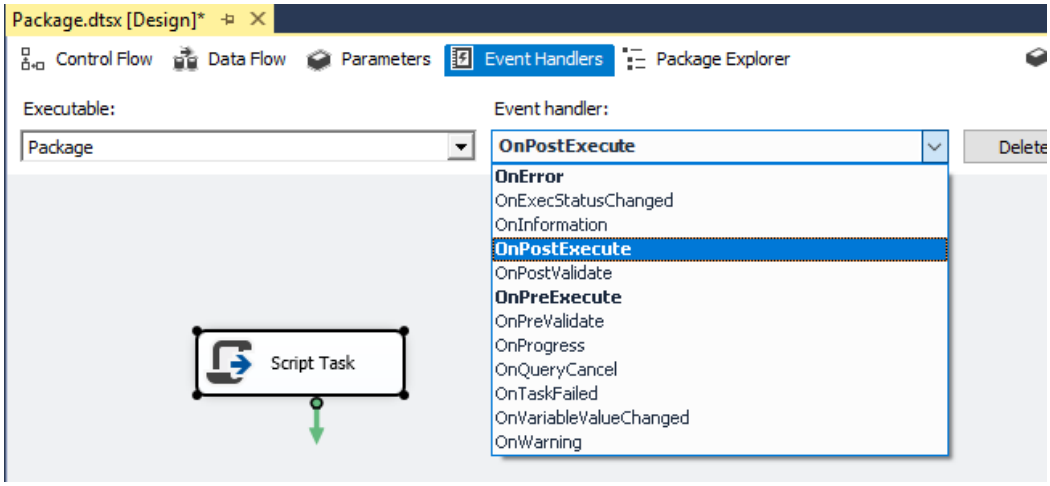

Рис. 3.5. Додавання обробників на події

Так на кожний крок, у випадку SSIS, створено події на попереднє виконання (PreExeucte) – тобто цей код виконується перед виконанням самого блоку. Також запрограмований код на подію пост виконання (post execute) – це блок коду, який обов'язково виконуються після того, як основний код відпрацював. І третя подія – на помилку. Тобто, якщо будь-який блок призведе до помилки – то виконається цей блок.

3.4. Логіка даних на стороні ETL процесу

Так, при запуску ETL ми отримуємо унікальний ідентифікатор поточного процесу, який зберігається до кінця виконання та передається на кожний блок подій кожного кроку. Як згадувалось раніше, кожний крок має 3 блоки обробки подій.

```
      OnPreExecute of each block
      INSERT INTO dbo.Opr_Log (
       oprRunId,
       oprProgramName,
       oprStepName,
       oprLevel
      )
      VALUES(?,'SSIS','Package      retriving      configuration      from
database.Getting archive folder',1)

      SELECT CAST(SCOPE_IDENTITY() AS INT) oprId
      ------------------------------------------------------
      OnError
      Declare @msg_er varchar(500);

      update dbo.Opr_Log
      SET oprStatus = 'F',
      oprEndDate = GETDATE(),
      oprComment = ?
      where oprId = ?
      ------------------------------------------------------
      OnPostExecute
      IF(? = 0)
      UPDATE dbo.Opr_Log
       SET oprStatus = 'C',
        oprEndDate = GETDATE(),
        oprComment = 'path varArchivedFolder =' + ?
      WHERE oprId = ?

      SELECT 0 as error_in_task
```

Рис. 3.5. Лістинг «Код в кожному обробнику подій»

Як бачимо, перед виконанням блоку – ми записуємо дані в таблицю логів з вказанням поточного коду процесу, отримаємо при цьому унікальний ідентифікатор процесу. І за підсумком виконання процесу (кроку) ми або записуємо статус як помилка, чи навпаки як успішний і в цих блоках ми використаємо oprId який отримали перед початком виконання. Через те, що це є унікальним первинним ключем у таблиці dbo.Opr_Log, то ми гарантовано будемо чи зачиняти чи навпаки проставляти статус помилки по саме тому процесу, який проходив.

Як бачимо – ми робимо все за наступною системою:

Таблиця 3.1.
Логіка процесу запису даних у таблицю логів

1	При запуску ETL системи отримаємо унікальний ідентифікатор запуску і записуємо у таблицю dbo.Opr_Run	Записуємо у runId дані – унікальний ідентифікатор oprRunId
2	Для кожного блоку перед виконанням записуємо дані у dbo.Opr_Log з oprRunId із першого кроку, та інші дані.	Отримаємо дані – унікальний ідентифікатор отримуємо oprId
3	Якщо виникає помилка, то система в таблиці dbo.Opr_Log оновлює дані по oprId з попереднього пункту та записує помилку.	
4	Після завершення процесу – оновлюємо таблицю dbo.Opr_Log зі статусом – «коректно» по полю oprId – унікальному ідентифікатору	
5	Після завершення всієї системи ETL йде оновлення dbo.Opr_Run по раніше генерованому oprRunId	

Ще однією перевагою системи є рівень вкладеності і рівень виконання, в таблиці dbo.Opr_Run це поле oprLevel. Так, наприклад вся система логується під рівнем 1, наступні всі кроки – будуть рівня 2, вкладені – під рівнем 3. В тілі програми є виклик процедур - це буде 3й рівень виконання, в свою чергу в самій процедурі є логічні блоки, які можуть бути обернуто в блоки try\catch і які є великими окремими частинами, і вони можуть бути залоговні під окремими записами, але вже буде 4й рівень і так далі.

3.5. Записи даних та додаткове логування

Як було показано в попередніх пунктах, кожен процес логується, тобто йде запису у базу даних. В таблиці зупусків є унікальний ідентифікатор, за яким можна однозначно визначити запуск, та в таблиці логів є унікальний ідентифікатор для кожного підпроцесу, навіть для найменшої складової

частини буде свій унікальний код, та дані по цьому процесу (кроку). Приведемо приклад логування декількох блоків розробленої ETL системи.

Таблиця 3.2.

Приклади записів у таблиці логів

Ім’я процесу	Опис та система	Параметри	Запис у таблицю логів
Load_scv	Це стандартний блок SSIS, завантажує дані з плоского файлу у систему	Вхідними параметрами є шлях до файлу, та може бути ім’я файлу	Записується дата та час старту процесу, записується шлях до файлу, ім’я файлу, і результат, якщо буде помилка, то записує опис помилки, рівень логування - 3
Execute SQL proc 1	Запуск сриктпу SQL, вбудований блок у SSIS, в нашому випадку – логіка реалізована на стороні процедури. В процедурі – 3 окремих важливих блоки, які потребують логування	В процедурі 7 вхідних параметрів. Також в середині процедури є окремі кроки, які обернуті блоками try/catch, і навіть якщо буде помилка, то процедура в цілому буде закінчена без помилок.	В таблиці буде декілька записів. 1 запис про вхід у процедуру, буде вищий рівень логування – 3, для кожного з кроків буде окремий лог, з результатом по кожному з них. А результат загальної процедури буде відображати лише закінчення її в цілому.

Так після завершення однієї ітерації ETL системи таблиця запусків (dbo.Opr_Run) буде мати лише один запис з інформацією коли була запущена наша система, який фінальний результат, скільки часу працювала система та додаткові можливі характеристики. Ця таблиця може включати не тільки дані про одну якусь систему, так, наприклад, на підприємстві може функціонувати декілька систем, і кожна з них буде записувати дані в цю таблицю.

У таблиці логів, в свою чергу, після завершення роботи системи буде десь 15 записів. Будуть записи по всім крокам нашого SSIS пакету і не тількі, будуть додатково записи по кожній збереженій процедурі і по кожному блоку, який розробник виділив як важливий блок, та той, що має буде залогований. Тобто розробник, чи адміністратор системи може включити будь-який блок коду, чи будь-яку операцію у логування, запис до таблиці. Навіть, якщо, є бізнес-потреба в логуванні найменших частин процесу, вони можуть бути записані у таблицю логів.

Як бачимо приведеного коду у лістингу з додатку А в процедурі dbo.sp_test_1 є додатковий код запису даних по процесу з використанням сторонньої процедури [dbo].[Create_Redundant_SPID]. В додатку А до диплому приведений код реалізації цієї процедури. Головною ідеєю цієї процедури є запис даних по поточному процесу на стороні бази даних, це дуже допомагає для аналізу процесів по користувачеві, через те, що, видаляється останній запис по процесу і записується поточний, це робиться для аналізу блокераторів та всіх процесів бази даних. Таку систему можливо використовувати при великій кількості процесів на стороні бази даних.

3.6. Результати системи, аналіз та порівняння з існуючими системами

Реалізована система дозволяє дуже детально аналізувати результати та по крокам отримати дані. Коли система ETL дуже насичена різноманітними процесами та кроками, дуже важливо отримувати дані по кожному кроку та розуміти, які кроки як відпрацювали, які були помилки, чи які параметри використовувались. Система була запущена 2 рази, тому у таблиці запусків буде лише 2 записи, а от у таблиці логів буде дуже багато записів.

Для виведення результатів з таблиці запусків – то можливо отримувати простим кодом:

```
SELECT * FROM dbo.opr_Run
WHERE oprRunName = {your etl name}
OR oprRunId = {Your run}
OR oprRunStart = {your date}
OR oprRunEnd IS NULL
-----
{Your run} = (SELECT MAX(oprRunId) FROM dbo.opr_Run) – Останній запуск
oprRunEnd IS NULL – не закінчений запуск, ще в процесі роботи
DATEDIFF (second, oprRunStart , oprRunEnd) – визначається як довго йшов
процес
```

Рис. 3.6. Лістинг «Отримання даних по запускам»

Як бачимо, така структура таблиці дозволяє будувати різноманітні звіти, та отримувати дані по потрібному запуску. Кожний аналітик, чи адміністратор системи сам обирає які параметри шукати та як сортувати дані.

Дані по логам найкраще отримувати наступним чином.

```
SELECT SPACE(t1.oprLevel * 3) + t1.oprProgramName AS Module,
       SPACE(t1.oprLevel * 3) + (t1.oprStepName) AS Step,
       oprStatus AS [Status],
       oprComment AS Comment,
       oprStartDate AS StartDate,
       oprEndDate AS EndDate,
       DATEDIFF(ss, oprStartDate, oprEndDate) AS DurationSec
FROM dbo.Opr_Log t1
WHERE oprRunId = (SELECT MAX(oprRunId)
                      FROM dbo.Opr_Run
                )
      ORDER BY oprId;
```

Рис. 3.7. Лістинг «Отримання даних з логів»

В цьому скрипту oprRunId визначається як останній запуск, але адміністратор, чи аналітик може самостійно визначити, який запуск аналізувати та по якому отримувати дані.

В цій роботі було запущено цю систему 2 рази. Припущено, що така система на підприємстві буде запускатись за визначеним графіком.

Як було згадано раніше, на сьогодні є багато інших варіантів логування, 2 з низ – це за допомогою тригерів, та з використанням внутрішньої системи SSIS логування.

Є декілька проблем використання тригерів:

- Неможливість відслідковувати всі операції системи, тобто в нашому випадку ми будемо бачити лише 3 операції, які пов’язані зі змінами таблиць даних
- Трнзакційність. Тобто, якщо по всьому скрипту піде відкат, то всі зміни будуть не дійсні і ми жодним чином не зможемо відслідкувати що було запущено, з якими параметрами і чому з’явилась помилка.

В додатку А показано, що є, на приклад 2 прості таблиці і тригер, які записує інформацію в таблицю dbo.test_table_2 інформацію про те, що трапилось з таблицею dbo.test_table_1. Так, якщо операція успішна, то ми матимемо інформацію про зміну dbo.test_table_1, але, якщо транзакція закінчилась ролбеком чи помилкою, то н буде ніяких записів, всі дії тригеру також відкатяться, згідно з теорією транзакцій та цілісності даних. Можливо записати лише, якщо обернути спробу у блок try\catch та додати у catch запис даних у якусь таблицю помилок. Але, якщо допрацювати таку вже систему – то буде наша запропонована раніше.

Та в разі використання вбудованої функції логування у SSIS ми жодним чином не можемо спостерігати всі процеси, які проходять на стороні бази даниих, ми можемо лише спостерігати все, що відбувається на стороні SSIS і може бути записано через подію. Але внутрішні процеси .Net коду чи процедур

ми жодним чином не можемо прослідкувати і не можемо знати причини помилок чи параметри, чи найдрібніші деталі.

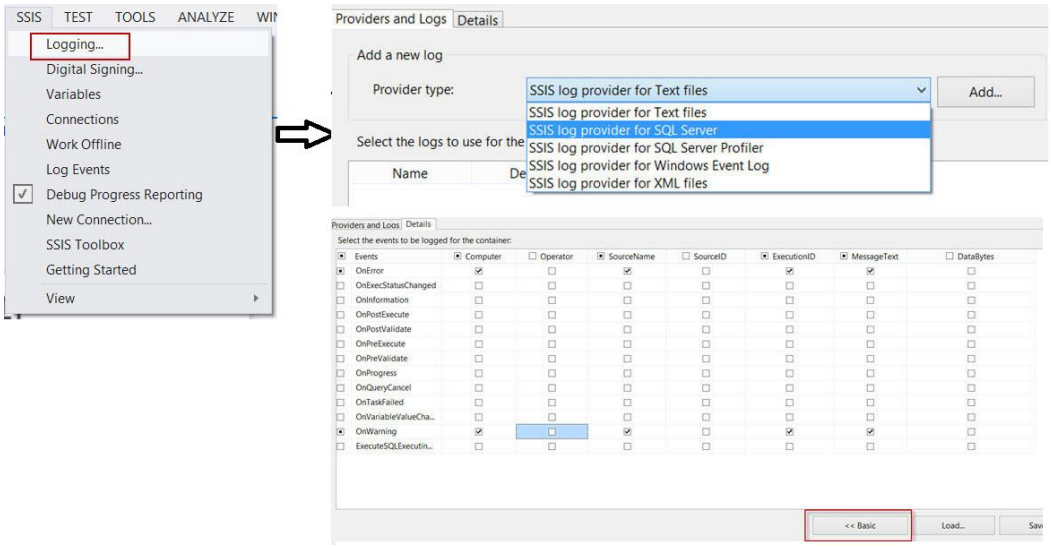

Рис. 3.8. Використання SSIS логування

На відміну від вже існуючих моделей ми показали як легко можна логувати дані по зовнішнім процесам, таким як блоки коду .NET чи збережені процедури на стороні бази даних. Вже існуючи системи не можуть одразу охопити і процеси основної системи ETL і кожного блоку цієї системи, кожного процесу. Так, наприклад, якщо в системі навіть Talend буде використання збережених процедур, чи класів і методів Java –то не буде можливосте й логування внутрішніх блоків. Розроблена система показала як легко можна логувати верхній рівень – саме пакет SSIS та його блоки, та зовнішні блоки – такі як збережені процедури, і всередині процедури, якщо є потреба – можливо вбудувати запис. Так було показано, що процедура виконує декілька важливих блоків, і кожний з них логується окремо, і якщо блок обернутий у try\catch – на зовні помилки не буде, але всередині блок може впасти та не виконати логіку, і коли адміністратору потрібно відслідкувати такі можливі помилки чи логи – то звичайні системи не дадуть такої можливості, лише розроблена система може адаптуватись під потреби користувача та записувати всю необхідну інформацію.

РОЗДІЛ 4
ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Розрахунок трудомісткості і вартості розробки програмного продукту

Вхідні дані:

- ✓ передбачуване число операторів – 2800
- ✓ коефіцієнт складності програми – 1,6
- ✓ коефіцієнт корекції програми в ході її розробки – 0,07
- ✓ часовая заработная плата программиста, грн/г – 120,0

У процесі створення ПЗ нормування праці ускладнено в силу творчого характеру праці програміста. Тому, трудомісткість розробки ПЗ розраховується на основі системи моделей з різною точністю оцінки.

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ чол.-г} \qquad (4.1)$$

де t_u – затрати праці на дослідження алгоритму розв'язання задачі, чол.-г;

t_a – затрати праці на розробку блок-схеми алгоритму, чол.-г;

t_n – затрати праці на програмування по готовій блок–схемі, чол.-г;

$t_{отл}$ – затрати праці на налагодження програми на ЕОМ, чол.-г;

$t_д$ – затрати праці на підготовку документації по завданню, чол.-г.

Ці затрати праці визначаються через умовне число операторів при розробці ПЗ, в число яких входять ті оператори , які необхідно написати в процесі роботи над програмою з урахуванням можливих уточнень у постановці завдання і вдосконалення алгоритму .

Умовне число операторів в програмі обчислюється за формулою:

$$Q = qC(1 + p), \qquad (4.2)$$

де q - передбачуване число операторів q = 2100 ;

c – коефіцієнт складності програми c = 1,6;

p − коефіцієнт кореляції програми в ході ее розробки p = 0,07.

Q = 2800 * 1,6 (1+ 0,07) = 3595

Затрати праці на вивчення опису завдання $t_и$ визначається з урахуванням уточнення опису та кваліфікації програміста.

$$t_u = \frac{QB}{(75..85)K} = \frac{3595*1,3}{77*1,2} = 50,58 \quad \text{чол.-год.,} \qquad (4.3)$$

де В - коефіцієнт збільшення затрат праці внаслідок недостатнього опису завдання:

$$B = 1,2. . . . 1,5;$$

К – коефіцієнт кваліфікації програміста, який визначається залежно від стажу роботи за даною спеціальністю. Він становить при стажі роботи, роки:

до 2 – 0,8;

від 2 до 3 – 1,0;

від 3 до 5 - 1,1 . . . 1,2;

від 5 до 7 - 1,3 . . . 1,4;

вище 7 – 1,5 . . . 1,6.

Затрати праці на розробку алгоритма для рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25)K} = \frac{3595}{22*1,2} = 136,17 \quad \text{чол.-год.} \qquad (4.4)$$

Витрати на складання програми по готовій блок -схемі:

$$t_n = \frac{Q}{(20..25)K} = \frac{3595}{22*1,2} = 136,17 \quad \text{чол.-год.} \qquad (4.5)$$

Затрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4..5)K} = \frac{3595}{4*1,2} = 748,96 \text{ чол.-год.} \qquad (4.6)$$

$$t_{отл}^{K} = 1,5 t_{отл} = 1,5*748,96 = 1123,44 \text{ чол.-год.} \qquad (4.7)$$

Витрати на підготовку документації:

$$t_д = t_{др} + t_{до} \quad \text{чол.-год.,} \qquad (4.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів:

$$t_{др} = \frac{Q}{(15 \dots 20)K} = \frac{3595}{17 * 1{,}2} = 206{,}61 \text{ чол.-год.} \quad (4.9)$$

$t_{до}$ – трудомісткість редагування, друку та оформлення документації:

$$t_{до} = 0{,}75 t_{др} = 0{,}75 * 206{,}61 = 154{,}96 \text{ чол.-год.} \quad (4.10)$$

$t_{д} = t_{др} + t_{до} = 206{,}61 + 154{,}96 = 361{,}57$ чол.-год.

У підсумку отримуємо, що трудомісткість розробки ПЗ становить:

t = 50,58+136,17+136,17+1123,44+361,57 = 1807,93 чол.-год.

4.2 Затрати на створення програмного забезпечення

Витрати на створення ПО (Кпо) включають витрати на заробітну плату виконавців програми (Зз/п), визначену множенням сумарної трудомісткості розробки ПО (t) на середню зарплату з нарахуваннями програміста і вартості машинного часу, необхідного для відладки програми на ЕОМ (Змв), визначеною виходячи з вартості 1-го машинного години конкретного типу ЕОМ, і витрат машинного часу на налагодження.

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ , грн.} \quad (4.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * C_{ПР} = 1807{,}93 * 120{,}0 = 216951{,}6 \text{ грн.,} \quad (4.12)$$

де t - загальна трудомісткість, чол.-г.;

$C_{пр}$ - середня годинна заробітна плата програміста, грн./год;

$C_{пр}$ =50,0 грн./год.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * C_{мч} = 1123{,}44 * 1{,}2 = 1348{,}13 \text{ грн.,} \quad (4.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.;

$C_{мч}$ – вартість машинного часу ЕОМ, грн./год.

Кпо = 216951,6 + 1348,13 = 218299,73 грн. (4.14)

Визначені таким чином витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУТП . Очікуваний період розробки ПЗ:

$$T = \frac{t}{B_k \bullet F_p} \text{ міс.,} \qquad (4.15)$$

де B_k – кількість виконавців;

F_p – місячний фонд робочого часу (при 40-ка годинному робочому тиждні F_p=176 годин).

$$T = \frac{1807,93}{1 * 176} = 10 \text{ місяців.}$$

Таким чином, період розробки програми складе приблизно 10 місяців.

методу.

4.3 Маркетингові дослідження ринку

На відміну від вже існуючих моделей показали як легко можна логувати дані по зовнішнім процесам, таким як блоки коду .NET чи збережені процедури на стороні бази даних. Вже існуючи системи не можуть одразу охопити і процеси основної системи ETL і кожного блоку цієї системи, кожного процесу. Так, наприклад, якщо в системі навіть Talend буде використання збережених процедур, чи класів і методів Java то не буде можливосте й логування внутрішніх блоків. А коли внутрішні блоки можуть бути дуже важливі та необхідні для логування то їх запис необхідний. Наша система показала свою ефективність та можливість запису навіть найменших блоків кожного процесу великої системи. Розроблена система показала як легко можна логувати верхній рівень – саме пакет SSIS та його блоки, та зовнішні блоки – такі як збережені процедури, і всередині процедури, якщо є потреба – можливо вбудувати запис. Так було показано, що процедура виконує декілька важливих блоків, і кожний з них логується окремо, і якщо блок обернутий у try\catch – на зовні помилки не

буде, але всередині блок може впасти та не виконати логіку, і коли адміністратору потрібно відслідкувати такі можливі помилки чи логи то звичайні системи не дадуть такої можливості, лише розроблена система може адаптуватись під потреби користувача та записувати всю необхідну інформацію.

Розроблена система буде задовольняти будь-які потреби кінцевого користувача та можлива для налаштування у будь-якій середі. І на відміну від існуючих систем – більш гнучка та більш інформативна. Головною цінністю на сьогодні є інформація, і тому, детальні дані про кожний процес дуже часто необхідні. Так, наприклад для фармацевтичних компаній, чи банківських установ, така система буде логувати всі дії, навіть найдрібніші та надавати можливість відслідковувати всі процеси та кроки. Тому можемо зробити висновки, що система на сьогодні є актуальною для використання, гнучкою та може використовуватись на багатьох підприємствах. І використання та розгортання – не буде дуже коштовним, тому що, майже всі компанії, які використовують ETL-системи мають реляційні бази даних, а головною необхідністю є наявність реляційної бази даних.

ВИСНОВКИ

Було розроблено модель логування запису даних по всім процесам ETL-системи. Модель показує свою ефективність на відміну від вже знаних систем, таких як використання тригерів чи використання вбудованих в систему розробки моделей. В системі SSIS є можливість вбудувати систему логування та запису процесів. Як було показано, більшість систем ETL можуть включати в себе додаткові стороні процесі. Так в розробленій системі ETL є такі блоки, як використання коду .NET C#, виклик складних навантажених збережених процедур на стороні бази даних, використання BAT файлів та інші вбудовані блоки. Так і в будь-якій іншій системі можуть бути використані зовнішні блоки, так в системах Jenkins, Informatica чи Talend можуть бути використані збережені процедури бази даних, чи BigData, чи використані блоки коду Java, Pyton, JavaScript чи .NET в залежності від системи. Тому використання представленої моделі задовольнить вимогу запису (логування) даних у будь-якій системі та з використанням будь-якої реляційної бази даних. Навіть можливо записувати дані в базу NoSql, але користування таких баз даних для процесів логування не є найкращим вибором.

Розроблена система показала свою ефективність та глибину можливого логування, що дозволило отримувати логи (детальні записи) по процесам навіть найменших частин, які потрібні адміністратору чи програмісту, згідно з бізнес-потреб. На відміну від вже існуючих моделей ми показали як легко можна логувати дані по зовнішнім процесам, таким як блоки коду .NET чи збережені процедури на стороні бази даних. Вже існуючи системи не можуть одразу охопити і процеси основної системи ETL і кожного блоку цієї системи, кожного процесу. Система показала свою ефективність та можливість запису навіть найменших блоків кожного процесу великої системи. Розроблена система показала як легко можна логувати верхній рівень – саме пакет SSIS та його блоки, та зовнішні блоки – такі як збережені процедури, і всередині процедури,

якщо є потреба – можливо вбудувати запис. Так було показано, що процедура виконує декілька важливих блоків, і кожний з них логується окремо, і якщо блок обернутий у try\catch – на зовні помилки не буде, але всередині блок може впасти та не виконати логіку, і коли адміністратору потрібно відслідкувати такі можливі помилки чи логи то звичайні системи не дадуть такої можливості, лише розроблена система може адаптуватись під потреби користувача та записувати всю необхідну інформацію.

Було показано, як розроблена система буде обробляти кожний крок системи ETL кожний процес та записувати всі дані, можливі помилки та характеристики, також є дані про час та дату старту та час та дату закінчення, тому для аналітиків чи адміністраторів системи буде корисна ця інформація для аналізу виконання кроків чи блоків, та, якщо виконання закінчилось помилкою чи довго виконувалось, адміністратор чи розробник однозначно може визначити блок та частину, яку потрібно модернізувати чи простежити.

Можемо зробити висновок, що розроблена система буде задовольняти будь-які потреби кінцевого користувача та можлива для налаштування у будь-якій середі. І на відміну від існуючих систем – більш гнучка та більш інформативна. Так, наприклад для фармацевтичних компаній, чи банківських установ, така система буде логувати всі дії, навіть найдрібніші та надавати можливість відслідковувати всі процеси та кроки. Тому можемо зробити висновки, що система на сьогодні є актуальною для використання, гнучкою та може використовуватись на багатьох підприємствах. І використання та розгортання – не буде дуже коштовним, тому що, майже всі компанії, які використовують ETL системи мають реляційні бази даних, а головною необхідністю є наявність реляційної бази даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jos van Dongen, Matt Casters, Roland Bouman (2010). Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration. John Wiley & Sons, 674
2. Betsy Burton Minding Your Own Business: Architecting the Information Assets // Gartner, Inc. (NYSE: IT) Retrieved from http://gartner.com/.
3. Ralph Kimball & Joe Caserta, (2004) The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. — Wiley Publishing, Inc., 491.
4. Oracle Data Integrator Enterprise Edition // ORACLE. Retrieved from http://www.oracle.com/us/products/middleware/data-integration/odi-ee-11g-ds-168065.pdf
5. Peter Jamack Hive as a tool for ETL or ELT, Extract, transform, and load OR extract, load, and then transform // IBM Developer - Published January 28, 2014
6. Alkis Simitsis, (2017) Modeling and managing ETL processes // Alkis Simitsis // National Technical University of Athens, Dept. of Electrical and Computer Eng., Computer Science Division // Computer news in Athens - 2017 - № 3
7. Fern X. Z. Solving cluster ensemble problems by bipartite graph partitioning / X. Z. Fern, C. E. Brodley // Proceedings of the 21 st International Conference on Machine Learning, Canada, 2004. – P. 47.
8. Iris Dataset Set. URL: https://archive.ics.uci.edu/ml/datasets/iris (дата звернення: 4.12.2019)
9. Seeds Data Set. URL: https://archive.ics.uci.edu/ml/datasets/seeds (дата звернення: 4.10.2021)
10. Glass Identification Data Set. URL: https://archive.ics.uci.edu/ml/datasets/glass+identification (дата звернення: 4.10.2021)

11. Wine Data Set. URL: https://archive.ics.uci.edu/ml/datasets/wine (дата звернення: 4.10.2021)
12. M. Guillaumin, T. Mensink, J. Verbeek, C. Schmid, TagProp: Discriminative metric learning in nearest neighbor models for image auto-annotation, Kyoto, Japan, 2009. – 6c.
13. W. Zhou, H. Li, Q. Tian, Recent Advance in Content-based Image Retrieval: A Literature Survey, Fellow, 2017. – 31c.
14. A. Makadia, V. Pavlovic, S. Kumar, New Baseline for Image Annotation, Piscataway, 2015. – 10c.
15. P. Kingma, J. Lei Ba, Adam: a method for stochastic optimization, Toronto, 2015. – 3c.
16. J. Fu, Y. Rui, Advances in deep learning approaches for image tagging, Cambridge, 2017. – 1c.
17. Y. Gong, Y. Jia, T. K. Leung, Deep Convolutional Ranking for Multilabel Image Annotation, Fellow , 2014. – 31c.
18. Frahim J. Securing the Internet of Things: A Proposed Framework / J. Frahim // Cisco White Paper.- 2015.
19. Aujla GS Data Offloading in 5G-Enabled Software-Defined Vehicular Networks: A Stackelberg Game-Based Approach / GS Aujla // IEEE Commun. Mag.- vol. 55, no. 7.- July 2017.
20. Peng M. Energy-Efficient Resource Assignment and Power Allocation in Heterogeneous Cloud Radio Access Networks / M. Peng // IEEE Transactions on Vehicular Technology.- vol. 64, no. 11.- Nov. 2015.- P. 5275-5287.
21. Gonzales D. Cloud-trust - A Security Assessment Model for Infrastructure as a Service (Iaas) Clouds / D. Gonzales // IEEE Transactions on Cloud Compu-ting.- vol. 5, no. 3.- July-Sept. 1, 2017.- P. 523-536.
22. John W. Research Directions in Network Service Chaining / W. John // 2013 IEEE SDN for Future Networks and Services.- Nov. 2013.- p. 1-7.

23. Automatic speech recognition and speech variability: A review / M. Benzeghiba, R. De Mori, O. Deroo, S. Dupont, T. Erbes, D. Jouvet, L. Fissore, P. Laface, A. Mertins, C. Ris, R. Rose, V. Tyagi, C. Wellekens // Speech Communication, Elsevier. – 2007. – №49. – P. 763-786.
24. V.V.R. Vegesna. Prosody modification for speech recognition in emotionally mismatched conditions / V.V.R. Vegesna, K. Gurugubelli, A.K. Vuppala // International Journal of Speech Technology, 3. – 2018. – P. 521-532.
25. Harpreet Kaur. Prosody Modification of its Output Speech Signal / Harpreet Kaur, Parminder Singh // International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE). – 2014. – №5. – P. 1056–1059.
26. Krothapalli Sreenivasa Rao. Real Time Prosody Modification / Krothapalli Sreenivasa Rao // Journal of Signal and Information Processing. – 2010. – №1. – P. 50-62.
27. Fast Prosody Modification using Instants of Significant Excitation / S. R. M. Prasanna, D. Govind, K. S. Rao, B. Yegnanarayana // Speech Prosody. – 2010.
28. Synthesis of Emotional Speech by Prosody Modification of Vowel Segments of Neutral Speech / Md Shah Fahad, Shreya Singh, Shruti Gupta, Akshay Deepak and Abhinav // Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE). – 2019. – P. 49-54
29. D. Joshi. Speech Emotion Recognition: A Review / D. Joshi, M. B. Zalte // IOSR Journal of Electronics and Communication Engineering (IOSR – JECE). – 2013. – №4. – P. 34–37.

Додаток А

Лістинг програми

Створення об’єктів на стороні бази даних

```
IF NOT EXISTS
(
   SELECT 1
   FROM sys.sysobjects
   WHERE id = OBJECT_ID(N'[dbo].[Opr_Run]')
       AND OBJECTPROPERTY(id, N'IsTable') = 1
)
  CREATE TABLE dbo.Opr_Run(
            oprRunId                INT PRIMARY KEY,
            oprRunName              NVARCHAR(100),
            oprRunStatus      NVARCHAR(1) DEFAULT 'R', -- 'R,C,F' R- running C -completed, F - failed
            oprRunStart             DATETIME2 DEFAULT GETDATE(),
            oprRunEnd               DATETIME2 DEFAULT NULL
  )


IF NOT EXISTS
(
   SELECT 1
   FROM sys.sysobjects
   WHERE id = OBJECT_ID(N'[dbo].[Opr_Log]')
       AND OBJECTPROPERTY(id, N'IsTable') = 1
)
BEGIN
  CREATE TABLE dbo.Opr_Log(
            oprId                   INT IDENTITY(1,1) PRIMARY KEY,
            oprRunId                INT FOREIGN KEY REFERENCES dbo.Opr_Run(oprRunId),
            oprProgramNameNVARCHAR(100), -- SSIS, AU parser, AU SP, etc.
            oprStepName             NVARCHAR(100), -- parsing parameters, update table, delete, etc.
            oprStatus               NVARCHAR(1) DEFAULT 'R', -- 'R,C,F' R- running C -completed, F -
failed
            oprLevel                INT,
            oprComment              NVARCHAR(MAX),
            oprStartDate      DATETIME2 DEFAULT GETDATE(),
            oprEndDate              DATETIME2 DEFAULT NULL
  )
  CREATE NONCLUSTERED INDEX ncl_indx_oprRunId_OprStepName ON
dbo.Opr_Log(oprRunId,oprStepName) include(oprStatus,oprEndDate);
  CREATE NONCLUSTERED INDEX ncl_indx_runId ON dbo.Opr_Log(oprRunId)
INCLUDE(oprProgramName,oprStepName,oprLevel,oprStatus,oprComment,oprStartDate,oprEndDate)
END


IF NOT EXISTS
(
   SELECT 1
   FROM sys.sysobjects
   WHERE id = OBJECT_ID(N'[dbo].[process_info]')
       AND OBJECTPROPERTY(id, N'IsTable') = 1
)
CREATE TABLE [dbo].[process_info](
  [spid] [int] NOT NULL,
  [window_id] [char](15) NOT NULL,
  [window_start_date] [smalldatetime] NOT NULL,
  [user_no] [int] NOT NULL,
  [qspid] [int] NULL,
```

```
  [message1] [varchar](255) NULL,
  [message2] [varchar](255) NULL,
  [message3] [varchar](255) NULL,
  [change_no] [int] NULL,
  [contact_no] [int] NULL,
  [incident_no] [int] NULL,
  [issue_seqno] [smallint] NULL,
  [cust_no] [int] NULL,
  [site_no] [int] NULL,
  [archive_date] [datetime] NULL,
  [lang_id] [char](3) NULL,
  [message4] [varchar](255) NULL,
 CONSTRAINT [XPKprocess_info] PRIMARY KEY CLUSTERED
(
  [spid] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, FILLFACTOR = 90) ON [PRIMARY]
) ON [PRIMARY]
GO



GO

IF(OBJECT_ID('Create_Redundant_SPID') <> 0 ) drop PROCEDURE[dbo].[Create_Redundant_SPID];
GO
CREATE PROCEDURE [dbo].[Create_Redundant_SPID]
   @spid                        INT,
   @Windowid                    CHAR(15),
   @window_start_date       SMALLDATETIME,
   @user_no                     INT,
   @archive_date            DATETIME,
  @err_msg                      VARCHAR(500) OUTPUT
WITH EXECUTE AS CALLER
AS
  BEGIN
        SET NOCOUNT ON;
        SET XACT_ABORT ON;

        SET @err_msg = 'OK'

        IF (@spid IS NULL OR @Windowid IS NULL OR @window_start_date IS NULL OR @user_no
IS NULL)
                BEGIN
                        SET @err_msg = 'Error: Mandatory parameters (@spid, @Windowid,
@window_start_date, @user_no) are not correctly set';
                        RETURN
                END

        BEGIN TRAN

        BEGIN TRY
                DELETE FROM dbo.process_info
                WHERE spid = @spid

                INSERT INTO dbo.process_info
                (
                        spid,
                        window_id,
```

```
                        window_start_date,
                        user_no,
                        archive_date
                )
                VALUES
                (@spid, @Windowid, @window_start_date, @user_no, @archive_date)

                COMMIT TRAN
        END TRY
        BEGIN CATCH
                SET
                        @err_msg ='ERROR: '+ LEFT(ERROR_MESSAGE() + ' at line '+
CAST(ERROR_LINE()  AS VARCHAR),500)

                ROLLBACK TRAN
        END CATCH
  END
```

Тестові процедури

```
--EXEC dbo.sp_test_1 @startDate = '20190101'


IF(OBJECT_ID('sp_test_1') <> 0 ) drop PROCEDURE [dbo].[sp_test_1];
GO
Create procedure dbo.sp_test_1
                                                @startDate DATETIME = NULL,
                                                @endtDate DATETIME = NULL,
                                                @parameter1 varchar(3200) = '',
                                                @parameter2 INT = 0,
                                                @opr_run_id       INT = NULL,
                                                @user_id          INT = 0,
                                                @result VARCHAR(MAX) = ''

AS
BEGIN

SET NOCOUNT ON
SET XACT_ABORT ON
SET ANSI_NULLS ON

DECLARE @err_msg_spid VARCHAR(3200) = 'OK';
DECLARE @curDate DATETIME = GETDATE();
DECLARE @Run_id_sp INT = 0;
DECLARE @cur_run INT =0;

Declare @comment VARCHAR(MAX) = 'Parameters: @startDate'  + convert(varchar,@startDate,102) +
                                                ' @endtDate : '+
convert(varchar,@endtDate,102) +
                                                ' @parameter1: ' + @parameter1 +
                                                ' @parameter2: ' + CAST(@parameter2 as
VARCHAR(MAX)) +
                                                ' @user_id: ' + CAST(@user_id as
VARCHAR(MAX));

INSERT INTO dbo.Opr_Log(
 oprRunId,
 oprProgramName,
```

```
 oprStepName,
 oprComment,
 oprLevel
)
VALUES(@opr_run_id,'Test ETL system', 'MS SQL: SQL proc 1',@comment, 3)

SELECT @Run_id_sp = CAST(SCOPE_IDENTITY() AS INT)

BEGIN TRY -- Create_Redundant_SPID

   EXEC [dbo].[Create_Redundant_SPID] @@SPID, 'WCF-Suid', @curDate, @user_id, @curDate,
@err_msg_spid OUTPUT


END TRY
BEGIN CATCH
   SET @err_msg_spid ='ERROR in ap_set_next_expected: '+ LEFT(ERROR_MESSAGE() + ' at line '+
CAST(ERROR_LINE()  AS VARCHAR),493)
END CATCH

IF(@err_msg_spid<> 'Ok')
BEGIN
   update  dbo.Opr_Log
   SET oprStatus = 'F',
   oprEndDate = GETDATE(),
   oprComment += @err_msg_spid
   where oprId = @Run_id_sp

   RETURN 0;
END


-------------------------------------------------------------------
---Some check
-------------------------------------------------------------------
DECLARE @some_check BIT = 0;

BEGIN

SET @comment = 'Checking some data @result: ' +  @result;

   INSERT INTO dbo.Opr_Log(

oprRunId,

oprProgramName,

oprStepName,

oprComment,
                                                                                                    oprLevel
                                                                                                   )
   VALUES(@opr_run_id,'Test ETL system', 'MS SQL sp_test_1: Check some data',@comment, 4)

   SELECT @cur_run = CAST(SCOPE_IDENTITY() AS INT)

   BEGIN TRY

            SELECT @some_check = 1
```

```
        FROM [dbo].[Employees]
        where [Name] = @result;

  END TRY
  BEGIN CATCH
        SELECT @err_msg_spid = ERROR_MESSAGE();
  END CATCH


IF(@err_msg_spid<> 'Ok')
  BEGIN
        update  dbo.Opr_Log
        SET oprStatus = 'F',
        oprEndDate = GETDATE(),
        oprComment += ' ERROR:' + @err_msg_spid
        where oprId = @Run_id_sp;

        update  dbo.Opr_Log
        SET oprStatus = 'F',
        oprEndDate = GETDATE(),
        oprComment +=' ERROR:' + @err_msg_spid
        where oprId = @cur_run;

        RETURN 0;
  END
ELSE
  BEGIN
        update  dbo.Opr_Log
        SET oprStatus = 'C',
        oprEndDate = GETDATE(),
        oprComment +=' - All ok'
        where oprId = @cur_run;
  END


END

-----------------------------------------------------------------
---Some aggregation
-----------------------------------------------------------------

BEGIN
  INSERT INTO dbo.Opr_Log(

oprRunId,

oprProgramName,

oprStepName,
                                                                                    oprLevel
                                                                                  )
  VALUES(@opr_run_id,'MS SQL sp_test_1: Agragation','Agragation', 4)

  SELECT @cur_run = CAST(SCOPE_IDENTITY() AS INT)

  BEGIN TRY
        IF OBJECT_ID('tempdb..#rez1') IS NOT NULL DROP TABLE #rez1

        SELECT [Employee_ID], SUM(Qty) as Qty, SUM([Sum_Sale]) as [Sum_Sale]
```

```
            INTO #rez1
            FROM dbo.Sales
            WHERE [Date_Sale] between @startDate and @endtDate
            AND ([Employee_ID] = @parameter2 OR @parameter2 = 0)
            GROUP BY [Employee_ID]

  END TRY
  BEGIN CATCH
            SELECT @err_msg_spid = ERROR_MESSAGE();
  END CATCH


IF(@err_msg_spid<> 'Ok')
  BEGIN
            update  dbo.Opr_Log
            SET oprStatus = 'F',
            oprEndDate = GETDATE(),
            oprComment += ' ERROR:' + @err_msg_spid
            where oprId = @Run_id_sp;

            update  dbo.Opr_Log
            SET oprStatus = 'F',
            oprEndDate = GETDATE(),
            oprComment +=' ERROR:' + @err_msg_spid
            where oprId = @cur_run;

            RETURN 0;
  END
ELSE
  BEGIN
            update  dbo.Opr_Log
            SET oprStatus = 'C',
            oprEndDate = GETDATE(),
            oprComment +=' - All ok'
            where oprId = @cur_run
  END

END

-----------------------------------------------------------------
---Some update
-----------------------------------------------------------------

BEGIN
  INSERT INTO dbo.Opr_Log(

oprRunId,

oprProgramName,

oprStepName,
                                                                                oprLevel
                                                                               )
  VALUES(@opr_run_id,'Test ETL system', 'MS SQL sp_test_1: Update some data', 4)

  SELECT @cur_run = CAST(SCOPE_IDENTITY() AS INT)

  BEGIN TRY
            Update dbo.Sales
```

```
            Set Tax = 0.23

  END TRY
  BEGIN CATCH
            SELECT @err_msg_spid = ERROR_MESSAGE();
  END CATCH


IF(@err_msg_spid<> 'Ok')
  BEGIN
            update  dbo.Opr_Log
            SET oprStatus = 'F',
            oprEndDate = GETDATE(),
            oprComment += ' ERROR:' + @err_msg_spid
            where oprId = @Run_id_sp;

            update  dbo.Opr_Log
            SET oprStatus = 'F',
            oprEndDate = GETDATE(),
            oprComment +=' ERROR:' + @err_msg_spid
            where oprId = @cur_run;

            RETURN 0;
  END
ELSE
  BEGIN
            update  dbo.Opr_Log
            SET oprStatus = 'C',
            oprEndDate = GETDATE(),
            oprComment +=' All ok, record updated'
            where oprId = @cur_run
  END

  RETURN 1;
END


END
GO

IF(OBJECT_ID('sp_test_2') <> 0 ) drop PROCEDURE[dbo].[sp_test_2];
GO

Create procedure dbo.sp_test_2
                                                @startDate DATETIME = NULL,
                                                @endtDate DATETIME = NULL,
                                                @opr_run_id       INT = NULL,
                                                @user_id          INT = 0


AS
BEGIN

SET NOCOUNT ON
SET XACT_ABORT ON
SET ANSI_NULLS ON

SELECT 1;

SELECT cast('kjhasdash' as int)
```

```
END

GO

IF(OBJECT_ID('sp_test_3') <> 0 ) drop PROCEDURE [dbo].[sp_test_3];
GO

Create procedure dbo.sp_test_3
                                                    @startDate DATETIME = NULL,
                                                    @endtDate DATETIME = NULL,
                                                    @opr_run_id      INT = NULL,
                                                    @user_id         INT = 0

AS
BEGIN

SET NOCOUNT ON
SET XACT_ABORT ON
SET ANSI_NULLS ON


DECLARE @err_msg_spid VARCHAR(3200) = 'OK';
DECLARE @curDate DATETIME = GETDATE();
DECLARE @Run_id_sp INT = 0;
DECLARE @cur_run INT =0;

Declare @comment VARCHAR(MAX) = 'Parameters: @startDate'  + convert(varchar,@startDate,102) +
                                                    ' @endtDate : '+
convert(varchar,@endtDate,102) +
                                                    ' @user_id: ' + CAST(@user_id as
VARCHAR(MAX));

INSERT INTO dbo.Opr_Log(
 oprRunId,
 oprProgramName,
 oprStepName,
 oprComment,
 oprLevel
)
VALUES(@opr_run_id,'Test ETL system','MS SQL: SQL proc 3',@comment, 3)

SELECT @Run_id_sp = CAST(SCOPE_IDENTITY() AS INT)

BEGIN TRY -- Create_Redundant_SPID

   EXEC [dbo].[Create_Redundant_SPID] @@SPID, 'WCF-Suid', @curDate, @user_id, @curDate,
@err_msg_spid OUTPUT


END TRY
BEGIN CATCH
   SET @err_msg_spid ='ERROR in ap_set_next_expected: '+ LEFT(ERROR_MESSAGE() + ' at line '+
CAST(ERROR_LINE()  AS VARCHAR),493)
END CATCH

IF(@err_msg_spid<> 'Ok')
BEGIN
   update  dbo.Opr_Log
```

```
  SET oprStatus = 'F',
  oprEndDate = GETDATE(),
  oprComment += @err_msg_spid
  where oprId = @Run_id_sp

  RETURN 0;
END


BEGIN
  INSERT INTO dbo.Opr_Log(

oprRunId,

oprProgramName,

oprStepName,
                                                                                                    oprLevel
                                                                                                   )
  VALUES(@opr_run_id,'Test ETL system','MS SQL: SQL proc 3', 4)

  SELECT @cur_run = CAST(SCOPE_IDENTITY() AS INT)

  BEGIN TRY
        SELECT cast('kjhasdash' as int);
  END TRY
  BEGIN CATCH
        SELECT @err_msg_spid = ERROR_MESSAGE();
  END CATCH


IF(@err_msg_spid<> 'Ok')
  BEGIN
        update  dbo.Opr_Log
        SET oprStatus = 'F',
        oprEndDate = GETDATE(),
        oprComment += ' ERROR:' + @err_msg_spid
        where oprId = @Run_id_sp;

        update  dbo.Opr_Log
        SET oprStatus = 'F',
        oprEndDate = GETDATE(),
        oprComment +=' ERROR:' + @err_msg_spid
        where oprId = @cur_run;

        RETURN 0;
  END
ELSE
  BEGIN
        update  dbo.Opr_Log
        SET oprStatus = 'C',
        oprEndDate = GETDATE(),
        oprComment +=' All ok, record updated'
        where oprId = @cur_run
  END
  RETURN 1;

END
```

```
END
```

Використання системи тригерів

```
create table dbo.test_table_1 (ID INT primary key identity,
                                                    test_t varchar(MAX));

create table dbo.test_table_2 (ID INT primary key identity,
                                                    test_t varchar(MAX));

Create trigger dbo.TEST_TGR_tab1_insert_into_tab_2 on dbo.test_table_1
AFTER INSERT
AS
declare @te VARCHAR(MAX);

SELECT @te = (i.test_t + 'table 2 need to hase updated')
FROM inserted i;

BEGIN
INSERT INTO  dbo.test_table_2 (test_t)
VALUES (@te);
END

GO

INSERT INTO dbo.test_table_1 (test_t)
VALUES ('First insert ')
GO

SELECT * FROM dbo.test_table_1
SELECT * FROM dbo.test_table_2



SELECT * FROM dbo.test_table_1
SELECT * FROM dbo.test_table_2

DECLARE @sql_text VARCHAR(MAX) = 'INSERT INTO dbo.test_table_1       VALUES (2, "SECOND
insert ")';

BEGIN TRY
EXEC sp_executesql @sql_text
END TRY
BEGIN CATCH
  SELECt ERROR_MESSAGE()
END CATCH

SELECT * FROM dbo.test_table_1
SELECT * FROM dbo.test_table_2
```

Додаток Б

ВІДГУК

керівника економічного розділу

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Керівника економічної частини Професора Вагонової О.Г.
(прізвище, ім'я, по батькові, вчене звання)

на магістерську роботу

Студентки • ІІ курсу групи 121м-20-1 Білецької Анастасії Дмитрівни
(прізвище, ім'я, по батькові)

На тему: Розробка та дослідження моделі фіксації етапів в Extract Transform Load системах.

.

«___»_____2022 р. _____
(підпис)

Додаток Д

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
БІЛЕЦЬКА.doc	Пояснювальна записка кваліфікаційної роботи. Документ Word.
БІЛЕЦЬКА.pdf	Пояснювальна записка кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація.ppt	Презентація роботи