УДК 004.6

В.М. Томашевський, А.Ю. Яцишин
Національний технічний університет України "Київський політехнічний інститут"
проспект Перемоги 37, корпус 18

ОСОБЛИВОСТІ ПРОЕКТУВАННЯ ГІБРИДНИХ СХОВИЩ ДАНИХ З ВРАХУВАННЯМ ДЖЕРЕЛ ДАНИХ

Описано проектування гібридних сховищ даних з врахуванням джерел даних. Наведена архітектура розширеного гібридного сховища даних

Ключові слова: розширене гібридне сховище даних, джерело даних, проектування сховище даних.

This paper descrbes building of hybrid data warehouses considering data sources. Extended hybrid data warehouse architecture is presented in this article.

Key words: extended hybrid data warehouse, data source, data warehouse building.

Вступ. Постановка проблеми

Сховища даних мають важливе значення у створенні інформаційних систем. Від сховища даних істотно залежить швидкодія та стійкість цих систем. Тому проектуванню сховищ даних приділяється підвищена увага. Сьогодні існують різні види баз даних і сховищ даних. Основними підходами до проектування сховищ даних є підхід Б.Інмона (реляційне сховище даних), підхід Р. Кімболла (багатовимірне сховище даних) та підхід Д. Хекні (гібридне сховище даних). Крім того, цікавим є запропонована у [13] концепція узагальненого гібридного сховища даних. Вона полягає у наявності в одному сховищі реляційної та багатовимірної баз даних. Це дозволяє поєднувати в одному сховищі швидкодію (оперативністю) реляційної бази даних з гнучкістю (аналітичністю) багатовимірної бази даних. Однак, враховуючи той факт, що дані для сховища даних отримуються з наявних джерел даних, то доцільним є автоматизоване проектування сховища даних з їх врахуванням. Це дасть можливість оптимізувати швидкодію виконання запитів до сховища даних відповідно до конкретних джерел даних. У статті розглядаються відомі рішення, що стосуються проектування сховищ даних з врахуванням джерел даних. Особлива увага приділяється аспектам, що стосуються змін джерел даних та пов'язаним з цим змінам сховища.

Аналіз останніх досліджень та публікацій

Розглянемо основні рішення, що стосуються проектування сховищ даних з врахуванням джерел даних. Одним таких рішень, є проект Стенфордського університету з сховищ даних (Stanford Data Warehousing Project) [1–8]. Метою цього проекту є створення сховища даних, яке автоматично проектується з наявних джерел даних. Узагальнимо відомості про це рішення.

Сховище будується з представлень, у разі доступу до яких дані отримуються з відповідних джерел. За такого підходу можна отримати уніфікований доступ до даних. Це рішення, на відміну від комерційних рішень (Redbrick, Sybase, Arbor), які роблять акцент на складовій виконання запитів до сховища та аналізу інформації сховища, наголошує на складовій інтеграції даних у сховище.

Було запропоновано систему WHIPS (WareHouse Information Prototype at Stanford). Під час проектування цієї системи ставилися такі взаємозв'язані цілі:

Модульність. Система повинна працювати не тільки з певним набором джерел чи управляти поданнями певним способом. Інтеграційний компонент повинен складатися з певних модулів, кожний з яких надає певну функціональність. Наприклад, модуль "оболонка" відповідає за збереження інформації у сховище, яка може бути будь-якою системою баз даних. Якщо сховище змінює свою систему баз даних, потрібно лише змінити модуль "оболонка" на відповідний для нової бази даних.

Масштабовність. Інтеграційний компонент повинен працювати з великими обсягами даних. По мірі зростання навантаження, система повинна працювати без перерв в обслуговуванні, розподіляючи це навантаження на додаткове обладнання і додаткові модулі. Наприклад, в архітектурі WHIPS кожним матеріалізованим поданням керує певний модель. По мірі збільшення кількості подань, кожен модуль може виконуватися на одній машині. У подібний спосіб система повинна підтримувати високі ступені паралельності, за яких велику кількість оновлень можна обробляти одночасно.

Доступність 24 х 7. Система повинна працювати без перерв на обслуговування. Це спричинено потребою роботи в багатьох часових поясах і, як наслідок, відсутністю часу на обслуговування системи.

Стійкість даних. Коли дані збираються з автономних джерел, отримані матеріалізовані подання можуть бути нестійкими, тобто перебувати у неактуальному стані по відношенню до відповідних джерел даних. Тому завжди повинна бути можливість задати бажаний рівень стійкості, а система повинна підтримувати необхідні алгоритми для досягнення цих рівнів.

Визначення та ініціалізація подань. Подання визначаються модулем “вказівник подань” системним адміністратором. “вказівник подань” перевіряє на тип визначення кожного подання відповідно до сховища метаданих та передає це визначення модулю “інтегратор”, який генерує відповідний менеджер подання.

Обслуговування подань. Кожен монітор відношення ***R*** знаходить зміни, які відбуваються у джерелі даних і передає ці зміни модулю “інтегратор”. Кожний модуль “менеджер подань” використовує один з Strobe-алгоритмів для підтримки стійкості подання, яким він керує.

Менеджери подань. Для кожного подання використовується один модуль “менеджер подань”, який відповідає за керування ним. Ведення здійснюється за допомогою Strobe-алгоритмів, який вказано у визначенні подання для забезпечення стійкості подання. Різні Strobe-алгоритми дають різні рівні стійкості в залежності від частоти та кластеризації модифікацій :всі алгоритми вимагають відстеження послідовності змін і компенсуючих результатів запитів для змін, які могли бути пропущені. Є дві переваги у використанні одного менеджера на подання. По-перше, обслуговування кожного подання можна виконувати паралельно на різних машинах. По-друге, кожен менеджер подань може використовувати інший Strobe-алгоритм для забезпечення різного рівня узгодженості для його подання. Загальна архітектура сховища WHIPS наведена на рис. 1.

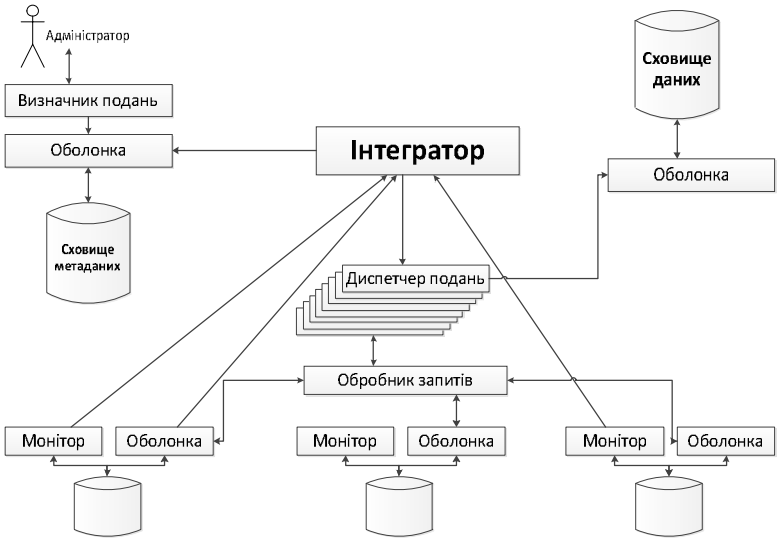

Рис. 1. Архітектура сховища WHIPS

У поточному прототипі (рис. 1) автори використовують реляційну модель для представлення даних сховища: подання визначені в реляційній моделі та відношення сховища даних. Вихідні дані перетворюються в реляційну модель за допомогою модулів джерела “монітор” та “оболонка” перед відправленням на будь-який інший модуль. Кожне джерело інкапсулюється модулями “монітор” та “оболонка”. Монітор відповідає за виявлення змін у джерелі даних і повідомлення модуля “інтегратор” про них. Автори реалізували модулі “монітор” на основі тригерів для реляційних джерел, а знімкові модулі “монітор” для джерел плоских файлів, які лише забезпечують регулярне створення моментальних знімків джерела даних.

Модулі “оболонка” транслюють запити з внутрішнього реляційного представлення у запити відповідною мовою джерела. Наприклад, модуль “оболонка” реляційної бази даних транслює вираз реляційної алгебри в SQL. Модуль “оболонка” для плоских файлів Unix може транслювати вирази алгебри в UNIX grep проекції, а потім перетворити результат у відношення. Як зазначалося вище, за допомогою однієї обгортки для кожного джерела приховуються деталі запитів конкретних джерел від обробника запитів і всіх інших модулів: всі модулі “обгортка” підтримуваних той самий метод інтерфейсу, хоча їх внутрішній код залежить від джерела.

Модуль “оболонка” для сховища отримує визначення подань та для одної умови вибірки, виконувати пост-обробки для застосовування додаткових умови відбору і зміни до даних подань у канонічній (внутрішньому) форматі, і транслює їх у конкретний синтаксис бази даних сховища. Модуль “оболонка” таким чином захищає всі інших модулі у системі WHIPS від особливостей сховища, що дозволяє використовувати будь-яку базі даних в якості сховища. Всі зміни, отримані модулем “оболонка” для сховища, в одному повідомленні застосовуються до сховища в одній транзакції, як необхідно для Strobe-алгоритмів стійкості подань.

Подання визначаються в підмножині SQL, що включає подання select-project-join та агреговані подання над всіма даними джерела без вкладеності. За необхідності, визначення подання може також вказувати, який алгоритм використовується для стійкості подання. Коли подання визначено, визначник подань аналізує його, додає відповідну інформацію з сховища метаданих (наприклад, ключові атрибути, типи атрибутів), і відсилає його на модуль “інтегратор”.

Модуль “інтегратор” координує запуск, зокрема додавання нових джерел, а також ініціалізує подання. Проте, головною роллю інтегратора є здійснення обслуговування подань за допомогою визначення, які змін джерела повинні бути поширені на які подання. Для цього інтегратор використовує набір правил, які визначають, яким менеджери подань необхідні певні модифікації. Ці правила створюються автоматично з дерев подань , коли визначається кожне подання. У найпростішому випадку, правилом є те, що зміни у відносини, над певним поданням направляються відповідному менеджеру подання.

Крім того, є один модуль “менеджер подань”, що відповідає за ведення кожного подання, використовуючи один з Strobe-алгоритмів (як зазначено у визначенні подання) для підтримки стійкості подання. Різні Strobe-алгоритми дають різні рівні послідовності залежно від частоти модифікацій. Всі алгоритми вимагають відстеження послідовності змін і компенсування результатів запитів для змін, які могли бути пропущені.

Обробник запитів отримує глобальні запити з точки зору менеджерів та подає відповідні запити до модулів “оболонка” джерел даних. Потім він передає загальну відповідь назад до менеджерів подань. Загалом, обробник запитів обробляє розподілені запити, з використанням стандартних методів, таких як передавання додаткової інформації фільтрації умов відбору, для скорочення запитів, які він надає модулю “оболонка”. Також він відстежує стан кожного глобального запиту при очікуванні локальних результатів запитів від модуля обгортки.

У роботах [1–8], що описують Stanford Data Warehousing Project, представлена система WHIPS, яка грунтується на модульній структурі сховища даних. Стосовно цього рішення було опубліковано декілька десятків статей, що описують різні аспекти функціонування сховища даних.

Особлива увага приділяється управлінню змінами, що проявляється в механізмах відстежування та застосування оновлень до сховища даних. Однак у цих роботах не враховується характер баз даних, що входять у сховище.

Іншим рішенням, що стосується проектування сховищ даних з врахуванням структур джерел даних, є “maintaining data warehouses over changing information sources” [9]. Ця стаття описує обслуговування сховищ даних за змінних джерел даних.

Ця стаття характеризує як типи динамічності (такі як оновлення даних, зміни схеми та модифікації обмежень), так і їхнє явне та неявне виникнення. Ця стаття також визначає особливості систем сховищ даних у разі врахування можливості змін джерел даних. Ці особливості включають пристосування модулів “оболонка” і визначень представлень до змін джерел даних та пристосування вмісту даних до сховища. Крім того, пропонуються можливі рішення для вирішення деяких з цих проблем, особливо в контексті системи Evolvable View Environment (EVE).

Система EVE функціонує так. Коли джерело долучається до системи, воно сповіщає EVE про його вміст, його здатності та можливі взаємозв’язки з іншими джерелами. Прикладом такої інформації є перекриття інформаційного вмісту, що надається двома джерелами (базами даних). Відомості про опис джерел зберігаються в МКВ (база метаданих). Основним засобом EVE є Evolvable-SQL (ESQL), розширення SQL, що дозволяє визначнику подань виражати переваги до розвитку подань. Використовуючи E-SQL, користувач під час визначення подання може вказати, яка інформація є незамінною, яка інформація може бути заміненою інформацією з інших інформаційних систем та чи припустима зміна подання, загалом. Це здійснюється шляхом призначення переваг елементам запиту до подання. Ця ж інформація є ключовою у розвитку подання за допомогою переписування його визначення в нееквівалентне інше, що також зберігає семантику, задану користувачем.

Як тільки сховище даних виявить зміну джерела, що впливає на подання, модуль синхронізації подань досліджує альтернативні для переписування запитів до подань з ціллю пристосування визначенням сховища даних в спосіб, прийнятний для користувача. Алгоритми синхронізації подань розвивають визначення подання шляхом знаходження відповідних замін для компонентів подань, базуючись на існуючій інформації про метадані, та видалення неістотних компонент подань, якщо вони помічені такими за допомогою E-SQL. Відповідно, компонент розвитку МКВ розвиватиме інформацію про метадані для відповідності з зміненим станом інформаційного простору.

Така гнучка технологія сховищ даних дозволить більшій кількості користувачів використовувати розподілену інформацію з мереж та збільшити продуктивність користувачів і системних адміністраторів шляхом обслуговування налаштовуваних інтерфейсів отримання інформації, що можуть автоматично обслуговуватися при змінах вихідних систем.

У роботі [9] наведено рішення, що пропонує розширення мови SQL для розвитку подань сховища даних. Однак у цьому рішенні також не враховується характер джерел даних.

У роботі “Data warehouse scenarios for model management” [10] описані сценарії сховищ даних для керування моделі даних. Автори оцінили застосування загального підходу управління моделями для двох сценаріїв сховищ даних. Ці сценарії використовують реляційні джерела, схеми типу “зірка” та SQL як мову виразів для прив’язок. Було розроблено декілька альтернатив для вирішення типових проблем прив’язування в загальний спосіб: інтеграції нового джерела даних та додавання нової вітрини. Рішення використовують значною мірою вже існуючі прив’язки та поєднують декілька альтернатив. Може знадобитися втручання користувача для надання інформації про семантичну еквівалентність та визначення нових вимог до прив’язок, які не можуть бути отримані з наявних моделей (прив’язок та схем). Робота Data warehouse scenarios for model management написана в контексті управління і також не враховує характер носія сховища даних.

У статті “Efficient View Maintenance at Data Warehouses” [11] описується обслуговування подань у сховищі даних. У ній представлено алгоритми інкрементального управління поданнями для сховища даних, що походить з багатьох розподілених автономних джерел даних. У роботі також описана деталізована основа для аналізу алгоритмів обслуговування подань з паралельними оновленнями.

Попередні підходи до обслуговування подань за наявності паралельних оновлень звичайно потребували двох типів повідомлень: першого – для розрахунку зміни подань внаслідок першого оновлення, та другого – для компенсації зміни подань внаслідок взаємовтручання паралельних оновлень. Алгоритми, наведені у статті [12], натомість, виконують локальну компенсацію шляхом використання даних, що вже є у сховищі. Перший алгоритм, “SWEEP”, забезпечує повну стійкість подань в сховищі даних за наявності паралельних оновлень. Попередні алгоритми для інкрементального обслуговування подань потребували або статичного стану сховища даних або експоненційної кількості повідомлень в розумінні сховища даних. Цей алгоритм не потребує статичного стану сховища даних для об’єднання нових подань, а складність повідомлень є лінійною в більшості джерел даних. Інший алгоритм, “Nested SWEEP”, намагається розрахувати композитну зміну подань для багатьох паралельних оновлень за підтримки сильної стійкості.

У цій роботі спочатку описується алгоритм, який забезпечує повну стійкість подань. Потім ця умова полегшується для досягнення сильної стійкості. Комерційні рішення для сховищ даних, такі як системи Red Brick, лише забезпечують конвергенцію.

Сумарна характеристика алгоритмів забезпечення стійкості подань наведена в таблиці.

Порівняльна характеристика алгоритмів забезпечення стійкості подань

Алгоритм	Архітектура	Стійкість	Вартість повідомлення на 1 оновлення
ECA	Централізована	Сильна	$o(1)$
Strobe	Розподілена	Сильна	$o(n)$
C-strobe	Розподілена	Повна	$o(n!)$
SWEEP	Розподілена	Повна	$o(n)$
Nested SWEEP	Розподілена	Сильна	$o(n)$

У роботі [12] розглядаються питання оновлень подань сховища даних, однак характер носія даних також не розглядається.

Розглянувши ці рішення, можна прийти до висновку, що всі розглянуті рішення не аналізують характер носія даних у сховищі.

Формулювання цілей статті

Розглянути існуючі рішення, що стосуються проектування сховищ даних з врахуванням структури джерел даних, та запропонувати рішення, що враховувало як характер носія сховища даних, так і джерела даних.

Виклад основного матеріалу

Під час функціонування сховища даних можливі три схеми функціонування сховища даних – автономне сховище даних, спадковане сховище даних та змішане сховище даних .

При першій схемі спочатку відбувається ініціалізація сховища даних за допомогою метаданих, отриманих з джерел даних або введених вручну. Після цього у системі створюються звичайні чи матеріалізовані представлення даних, які отримують дані безпосередньо з сховища даних. Зміни джерел даних не впливають на сховище даних, за необхідності дані синхронізуються вручну.

Основною перевагою цієї схеми роботи сховища даних є захищеність сховища даних від потенційно небажаних змін джерел даних у випадку, коли джерела даних є публічними або оновлюються дуже часто. Таку схема роботи сховища даних доцільно застосовувати тоді, коли

необхідно гарантувати, що сховище даних буде ізольованим від можливих негативних впливів джерел даних. У разі другої схеми сховище даних є синхронізованим з джерелами даних. Внаслідок цього як початкова ініціалізація, так і функціонування сховища даних здійснюється з урахуванням метаданих та даних джерел. Під час здійснення оновлень до джерел даних вони автоматично застосовуються до сховища даних відповідно до визначених параметрів оновлення. Цю схему варто використовувати тоді, коли джерела та сховище є захищеними чи коли оновлюваність сховища є критичною.

Змішана схема працює аналогічно до другої, однак для кожної таблиці та кожного подання можна задати, чи будуть синхронізовані зміни чи ні. Така схема рекомендується в загальному випадку, бо можна розмежувати синхронізацію між окремими елементами сховища.

Вибрана стратегія оновлення тісно пов'язана з поняттям стійкості сховища даних. Згідно з роботою [11] розглядаються три види стійкості сховища даних:

Конвергенція, за якої оновлення в остаточному результаті інтегруються в матеріалізоване подання.

Сильна стійкість, за якої порядок перетворень станів подань сховища даних у сховищі даних відповідає порядку перетворень станів у сховищах даних.

Повна стійкість, за якої кожен з станів сховищ даних відображається як окремий стан сховища даних та обмеження упорядкування в перетвореннях станів джерел даних зберігається у сховищі даних.

Як видно з аналізу існуючих рішень, наявні підходи не враховують характеру носія даних у сховищі даних. Тому доцільно використовувати рішення, яке б враховувало і зміни даних у джерелах даних, і характер носія даних у сховищі даних. Автор опублікував роботи [12–14], в яких розглядаються питання проектування гібридних сховищ даних. Враховуючи результати, запропоновані в відомих рішеннях, розширимо концепцію узагальненого гібридного сховища даних для підтримки джерел даних. Наведемо архітектуру розширеного гібридного сховища даних (рис. 2.)

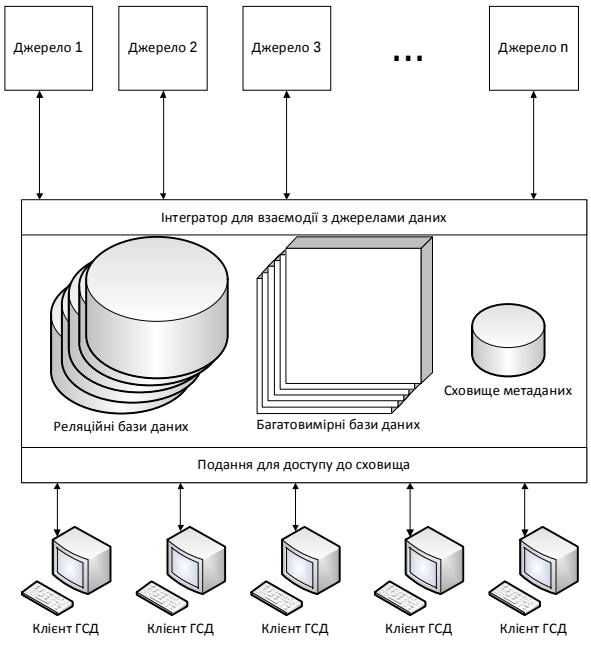

Рис. 2. Архітектура розширеного гібридного сховища даних

У цій архітектурі гібридне сховище даних зв'язане через інтегратор з джерелами даних та через подання доступне клієнтам ГСД.

Істотна відмінність цієї архітектури від попередньої [14] полягає в тому, що вона враховує наявність джерел даних та проектується відповідно до структури джерел даних.

Розширене гібридне сховище даних включає такі компоненти:

- реляційні та багатовимірні бази даних , які зберігають дані сховища даних;
- сховище метаданих, яке є реляційною базою даних та зберігає метадані сховища даних;
- інтегратор джерел даних, який виконує початкову ініціалізацію сховища та здійснює взаємодію зі джерелами даних. Взаємодія з джерелами даних полягає у відслідковуванні змін даних та метаданих, що відбуваються у джерелах, та застосуванні цих змін відповідно до налаштувань сховища даних;
- подання для доступу до сховища, які надають уніфікований доступ до сховища даних. Уніфікованість дає змогу користувачам єдиним звертатися до даних, що зберігаються у сховищі незалежно від їх фізичного та логічного розташування.

Алгоритм проектування узагальненого гібридного сховища даних розміщує дані у реляційній (РБД) чи багатовимірній (БВБД) базі даних для оптимізації часу виконання запитів. Цей алгоритм потрібно розширити для врахування джерел даних. Для розширення введемо такі позначення :

DS_i - *i*-те джерело даних,

$ST_j^i, i = \overline{1,n},\ j = \overline{1,m_n}$ – *j*-та таблиця *i*-го джерела даних,

$SV_j^i,\ i = \overline{1,n},\ j = \overline{1,q_n}$ – *j*-те подання *i*-го джерела даних,

$WT_j^d,\ d \in \{0,1\},\ j = \overline{1,n_d}$ - *j*-та таблиця (зріз) сховища даних,де

d=0 відповідає таблицям РБД, *d*=1 - зрізам БВБД,

$TS_j^d,\ d \in \{0,1\},\ j = \overline{1,n_d}$ – ознака синхронізації *j*-ї таблиці (зрізу) сховища даних,

$CV_p,\ p = \overline{1,c}$ – *o*-те додаткове подання для сховища даних.

$WV_k,\ k = \overline{1,l}$ – *p*-те подання сховища даних

$VS_k,\ k = \overline{1,l}$ – ознака синхронізації *k*-го подання сховища даних.

Алгоритм 1. Загальна схема роботи сховища даних

1. Ініціалізація сховища даних (алгоритм 2).
2. Поки сховище даних не отримало повідомлення про завершення роботи, виконувати таке:
   2.1. Якщо сховище даних перебуває у стані «готовність», то:
      2.1.1. Приймати повідомлення від клієнтів ГСД.
      2.1.2. Якщо отримане на кроці 2.1.1 повідомлення є запитом від клієнта ГСД, то:
         2.1.2.1. Обробити запит, повернувши результат користувачеві:
         2.1.2.2. Запустити алгоритм перепроектування сховища даних, представлений в [14]. На час перепроектування встановити стан сховища «обслуговування».
         2.1.2.3. Повернутися до кроку 2.1.1.
      2.1.3. Якщо отримане на кроці 2.1.1 повідомлення є запитом від інтегратора, то:
         2.1.3.1. Змінити сховище даних відповідно до даних, що вказані в повідомленні. Зміну сховища даних виконувати для джерел та подань, для яких ознаки синхронізації $TS_j^d = 1, VS_k = 1$.На час зміни встановити стан сховища «обслуговування».
         2.1.3.2. Повернутися до кроку 2.1.1.
      2.1.4. Якщо отримане повідомлення є запитом завершення роботи, перейти до кроку 3.
   2.2. Перейти до кроку 2.
3. Завершити роботу сховища даних.

Алгоритм 2. Ініціалізація сховища даних

Вхідні дані: метадані та дані джерел даних, визначення подань

Вихідні дані: метадані сховища даних

- Прийняти i=1.
- Поки i<=n, виконувати таке:
  - Прийняти j=1.
  - Поки j<=m_n, виконувати наступне:
    - Розмістити таблицю SV_j^i в реляційній базі даних сховища: $WV_k = SV_j^i$. Цей вираз означає створення таблиці WT_j^0 у сховищі даних з тими даними і метаданими, які є в таблиці ST_j^i джерела даних DS_i.
    - j=j+1.
    - Перейти до кроку 2.2.
  - i=i+1.
  - Перейти до кроку 2.
- Прийняти i=1, k=1.
- Поки i<=n, виконувати таке:
  - Прийняти j=1.
  - Поки j<=q_n, виконувати наступне:
    - Розмістити подання SV_j^i в реляційній базі даних сховища: $WV_k = SV_j^i$. Цей вираз означає створення подання WV_k у сховищі даних з тими ж метаданими, які є в поданні SV_j^i джерела даних DS_i.
    - Прийняти j=j+1,k=k+1.
    - Перейти до кроку 4.2.
  - Прийняти i=i+1.
  - Перейти до кроку 4.
- Прийняти p=1.
- Поки p<=c, виконувати наступне:
  - Розмістити подання CV_p в реляційній базі даних сховища: $WV_k = CV_p$. Цей вираз означає створення подання WV_k у сховищі даних з тими ж метаданими, які є в поданні CV_p, заданому у сховищі.
  - Прийняти p=p+1,k=k+1.
- Встановити сховище у стан «готовність».
- Кінець алгоритму.

Висновки

У роботі наведено архітектуру та алгоритми роботи розширеного гібридного сховища даних. Розширене гібридне сховище даних враховує джерела даних, що дозволяє отримувати уніфікований доступ до інформації джерел даних з автоматичним проектуванням сховища даних для забезпечення оптимальної швидкодії запитів до сховища даних.

У подальших дослідженнях планується проведення експериментів, пов'язаних з випробуваннями алгоритму на різних структурах даних та запитах до них. Крім того, варто звернути увагу на управління змінами у сховищі даних та на питання, пов'язані з обробкою паралельних оновлень.

1. Garcia-Molina H., Labio W. J., Wiener J. L., Zhuge Y. "Distributed and Parallel Computing Issues in Data Warehousing" In Proceedings of ACM Principles of Distributed Computing Conference, 1999. Invited Talk. 2. Wiener J. L. "What is data warehousing and what is Stanford doing about it?" An overview talk given in the Stanford DB Seminar series, Fall, 1997. 3. Labio W. J., Zhuge Y., Wiener J. L., Gupta H., Garcia-

Molina H., Widom J. "The WHIPS Prototype for Data Warehouse Creation and Maintenance." In Proceedings of the ACM SIGMOD Conference, Tuscon, Arizona, May, 1997. Demonstration Description. The demo was also given at the International Conference on Data Engineering, Binghamton, UK, April, 1997. 4. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom. "A System Prototype for Warehouse View Maintenance." In Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada, June 7, 1996, pp. 26-33. 5. J. Widom, "Research Problems in Data Warehousing." In Proceedings of the 4th Int'l Conference on Information and Knowledge Management (CIKM), November 1995. 6. J. Hammer, H. Garcia-Molina, J. Widom, W. J. Labio, Y. Zhuge. "The Stanford Data Warehousing Project" IEEE Data Engineering Bulletin, June 1995. 7. H. Gupta. "Selection of Views to Materialize in a Data Warehouse." In Proceedings of the International Conference on Database Theory, Athens, Greece, January 1997. 8. V. Harinarayan, A. Rajaraman, J. Ullman. "Implementing Data Cubes Efficiently." In Proceedings of ACM SIGMOD Conference, Montreal, Canada, June 1996. 9. Elke A. Rundensteiner.Maintaining data warehouses over changing information sources[Текст]. Communications of the ACM, Volume 43 Issue 6, June 2000 / Elke A. Rundensteiner , Andreas Koeller , Xin Zhang – 2000 10. Philip A. Bernstein. Data warehouse scenarios for model management [Текст]. ER'00 Proceedings of the 19th international conference on Conceptual modeling. / Philip A. Bernstein , Erhard Rahm - Springer-Verlag Berlin, Heidelberg – 2000. 11. A. El Abbadi. Efficient View Maintenance at Data Warehouses [Текст]. Proceedings of the 1997 ACM SIGMOD international conference on Management of data / A. El Abbadi, A. Singh, D. Agrawal, T. Yurek – New York, NY, USA – 1997. 12. Томашевський В.М.Математична модель задачі проектування гібридних сховищ даних з врахуванням структур джерел даних [Текст]. Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка: Зб. наук. пр. /Томашевський В.М.,Яцишин А.Ю. – К.:Век+, – 2011. – № 53. – 211 с. 13. Яцишин А.Ю. Застосування генетичного алгоритму для проектування гібридних сховищ даних [Текст]. Вісник Нац. ун-ту "Львівська політехніка", секція "Інформаційні систе-ми та мережі",/ Яцишин А.Ю. – м.Львів – 2011 14. Яцишин А.Ю. Підходи та алгоритми проектування гібридних сховищ даних [Текст]. Вісник Нац. ун-ту "Львівська політехніка", секція "Інформаційні системи та мережі" / Яцишин А.Ю – Львів, 2010.