УДК 004.658.3

С.А. Глухова, С.Л. Зіноватна
Одеський національний політехнічний університет, пр. Шевченка, 1, Одеса, 65044

ФОРМАЛІЗАЦІЯ ЗАДАЧІ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ НАДЛИШКОВИХ ДАНИХ ПРИ ПРОВЕДЕННІ ДЕНОРМАЛІЗАЦІЇ БАЗИ ДАНИХ

Представлено правила та алгоритми для реалізації засобів підтримки цілісності даних при використанні висхідної та нисхідної денормалізації. Забезпечено автоматичне створення тригерів для відношень бази даних, для яких виконана денормалізація, що підвищує якість реструктуризації бази даних.
Ключові слова: *База даних – Цілісність даних – Денормалізація – Тригер.*

Rules and algorithms for realization of data integrity support facilities at the use of ascending and descending denormalization are presented. Automatic trigger creation is provided for database relations, for which denormalization that promotes quality of database restructurization is executed
Keywords: *Data base – data integrity – Denormalization – Trigger.*

I. ВСТУП

Майже в будь-якій інформаційній системі актуальним є питання прискорення виконання запитів до бази даних (БД). Тому, щоб збільшити швидкість вибірки даних і тим самим підвищити продуктивність системи, в окремих випадках доводиться проводити вибіркову денормалізацію схеми реляційної бази даних.

Денормалізація – це процес досягнення компромісів у нормалізованих таблицях за допомогою навмисного введення надмірності з метою збільшення продуктивності [1]. Найбільш поширеними видами денормалізації є низхідна і висхідна.

Спосіб денормалізації, який передбачає дублювання неключового атрибута з батьківської сутності в дочірню, називається низхідною денормалізацією [2]. Перевагою даного варіанту є виключення операції з'єднання при виконанні запитів.

Висхідною денормалізацією називається збереження інформації з дочірньої сутності в батьківській у формі підсумку [2].

Реструктуризація БД дозволяє усунути операцію з'єднання в запитах і/або зменшити розмір відношень, які беруть участь в операціях, що може підвищити продуктивність роботи інформаційної системи у цілому [1].

Однак у випадку застосування денормалізації виникає необхідність в уведенні додаткових засобів підтримки цілісності й несуперечності даних, щоб забезпечити користувача актуальними даними в будь-який момент часу. Так, при використанні висхідної денормалізації потрібен перерахунок значення введеного атрибута при кожній зміні кортежів дочірнього відношення для підтримки актуальності агрегованого значення [3].

Кожний спосіб денормалізації передбачає виконання відповідного набору дій для забезпечення цілісності даних. При цьому набір дій може бути різним для різних операцій модифікації даних [1].

Цей набір дій може бути реалізований у вигляді тригерів або окремих програмних фрагментів у застосуванні.

Існують програмні продукти, за допомогою яких можна моделювати бази даних і оптимізувати інформаційні системи і, зокрема, які надають можливість денормалізації БД. Але в них відсутнє автоматичне створення засобів підтримки цілісності надлишкових даних, які повинні виконувати необхідні оновлення в каскадному режимі. Наприклад, ERWin не створює автоматично тригери забезпечення цілісності даних і, відповідно, вимагає додаткового їх написання, що ускладнює і уповільнює роботу адміністратора БД і підвищує ризик виникнення помилок [3].

Відсутній формальний опис алгоритмів програмного коду, який гарантовано забезпечує несуперечність у реструктурованій БД, для різних способів денормалізації.

Таким чином, актуальним є забезпечення автоматичного створення засобів підтримки цілісності надлишкових даних, що підвищує якість реструктуризації БД з використанням низхідної і висхідної денормалізації.

II. ОСНОВНА ЧАСТИНА

Для досягнення мети розроблена програма, яка має наступну загальну схему (рисунок 1).

Рисунок 1 *– Загальна схема програми*

Робота програми передбачає виконання таких послідовних етапів:

1) завдання вхідних даних: вибір виду денормалізації та сукупності таких даних, які залежать від обраного виду денормалізації, і визначають учасників реструктуризації;

2) автоматична зміна структури БД;

3) автоматичне заповнення доданих атрибутів даними;

4) автоматичне створення засобів підтримки цілісності (тригерів).

Загальне представлення вхідних даних має наступний вид:

$$DN = <TD, S, sCol, D, PathTrans, RTrans>,$$

де TD – тип денормалізації, $TD=N$ у випадку низхідної денормалізації, $TD=V$ у випадку висхідної денормалізації ;

S – батьківське відношення;

$sCol$ – множина атрибутів, які необхідно продублювати;

D – дочірнє відношення;

$RTrans$ – список відношень, які потрібно з'єднати для переходу від S до D (рисунок 2);

$PathTrans$ – умова з'єднання відношень S та D (рисунок 2).

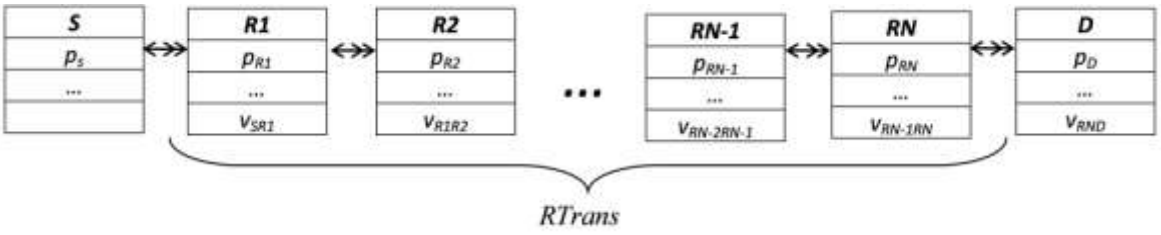

Рисунок 2 *– Схема транзитних зв'язків між відношеннями*

$PathTrans$= *"*$p_{R1}= v_{R1R2}$ *and…and* $p_{RN-2}= v_{RN-2N-1}$ *and* $p_{RN-1}= v_{RN-1RN}$.

У окремому випадку $PathTrans$= " " та $RTrans$=" ", якщо S та D пов'язані безпосередньо між собою.

Кожний атрибут з множини $sCol$ для низхідної денормалізації описується параметрами

$$<col\_name, col\_type>,$$

де col_name – ім'я атрибута;

col_type – тип даних атрибута.

Кожний атрибут з множини $sCol$ для висхідної денормалізації описується параметрами

$$< agrF, fCol, col\_type>,$$

де $agrF$ – вид агрегатної функції, $agrF \in \{MAX, MIN, SUM, COUNT, AVG\}$;

$fCol$ – вираз аргументу функції $agrF$.

Зміна структури БД полягає в генерації запита ALTER TABLE

$$alter\ table\ R\ \{add\ newCol\ col\_type\}_{|sCol|},$$

де $R=D$, якщо $TD=N$, та $R=S$, якщо $TD=V$;

$|sCol|$ – кількість атрибутів у множини $sCol$.

При додаванні атрибуту необхідно дотримуватися правил унікальності імен атрибутів всередині відношення. Пропонується використовувати наступні правила при формуванні імені атрибуту, який додається:

- у разі низхідної денормалізації ім'я $newCol$ збігається з ім'ям атрибута, що копіюється з S, якщо в D відсутній атрибут з таким ім'ям; інакше ім'я може бути сформоване з використанням настроювань користувача, наприклад, формуватися з імені відношення та імені продубльованого атрибуту;

- у разі висхідної денормалізації ім'я $newCol$ утворюється з $agrF$ та індексу; інакше ім'я може бути сформоване з використанням настроювань користувача.

За результатами реструктуризації формується таблиця відповідності $\{col\_name \rightarrow newCol\}_{|sCol|}$. Заповнення БД полягає в генерації запиту UPDATE.

При низхідній денормалізації відбувається копіювання даних з батьківського відношення у дочірнє з урахуванням значень зовнішніх ключів:

for i=1 to $|S|$

for j=1 to $|sCol|$

UPDATE D SET $newCol_j = col\_name_{ij}$

WHERE v_{RND} IN

(SELECT p_{RN} FROM $RTrans$

WHERE $PathTrans$ AND $v_{SR1}=a_i$),

де $|S|$ – кількість рядків у відношенні S;

col_name_{ij} – значення j-го атрибуту з множини $sCol$ у i-му рядку;

a_i – значення первинного ключа p_S відношення S у i- рядку відношення S.

При висхідній денормалізації виконується розрахунок значень функції і заповнення ними доданих атрибутів батьківського відношення: генерується запит UPDATE для відповідних агрегатних функцій $agrF$ з множини $sCol$:

for i=1 to $|S|$

for j=1 to $|sCol|$

UPDATE S SET $newCol_j$ =

(SELECT $agrF_j(FCol_j)$

FROM $RTrans$,D

WHERE $PathTrans$ AND $p_{RN}=v_{RND}$

AND $v_{SR1}=a_i$)

WHERE $p_S=a_i$

Для низхідної денормалізації генеруються тригери, які спрацьовують при оновленні даних (UPDATE) у S і при оновленні або внесенні даних (UPDATE або INSERT) у D. Для висхідної – тригери, які спрацьовують при всіх операціях модифікації відношення D.

Загальне представлення тригеру:

$Tr = <R, OpM, BTg>$,

де R – відношення, для якого створений тригер;

OpM – перелік операцій модифікації, для яких спрацьовує тригер;

BTg – тіло тригера, яке визначає алгоритм отримання цілісних даних.

Для низхідної денормалізації правила перетворення $DN=>Tr$ виглядають наступним чином.

1) Тригер для батьківського відношення

$R=S$

OpM = {UPDATE}

BTg = {for each $col\_name_j \in sCol$

{IF (NEW.col_name_j != OLD.col_name_j)

UPDATE D SET $newCol_j$ = NEW.col_name_j

WHERE v_{RND} IN

(SELECT p_{RN} FROM *RTrans*

WHERE *PathTrans* AND v_{SR1} = NEW.p_S)

}}

2) Тригер для дочірнього відношення

$R=D$

OpM = {UPDATE, INSERT }

BTg = {if (NEW.v_{RND}!= OLD.v_{RND})

for each $col\_name_j \in sCol$

{New.$newCol_j$ =

(SELECT col_name_j FROM S WHERE p_S IN

(SELECT v_{SR1} FROM *RTrans*

WHERE *PathTrans* AND p_{RN}=NEW.v_{RND}))

}}

Для висхідної денормалізації правила перетворення $DN=>Tr$ виглядають наступним чином:

$R=D$

C_{UNEW}= "p_S IN (SELECT v_{SR1} FROM *RTrans* WHERE *PathTrans* AND p_{RN}=NEW.v_{RND})"

C_{UOLD}=" p_S IN (SELECT v_{SR1} FROM *RTrans* WHERE *PathTrans* AND p_{RN}=OLD.v_{RND})"

а) OpM = {INSERT}

BTg = {for each $col\_name_j \in sCol$

case *agrF*

{***SUM***:

UPDATE S

SET $newCol_j$= $newCol_j$ +NEW.$fCol_j$

WHERE C_{UNEW}

COUNT:

UPDATE S SET $newCol_j$= $newCol_j$ +1

WHERE C_{UNEW}

MIN:

if (NEW.$FCol_j$< (SELECT $newCol_j$ FROM S

WHERE p_S IN

(SELECT v_{SR1} FROM *RTrans*

WHERE *PathTrans* AND

p_{RN}=NEW.v_{RND}))

UPDATE S SET $newCol_j$= NEW.$FCol_j$

WHERE C_{UNEW}

MAX:

if (NEW.$FCol_j$> (SELECT $newCol_j$ FROM S

WHERE p_S IN

(SELECT v_{SR1} FROM *RTrans*

WHERE *PathTrans* AND

p_{RN}=NEW.v_{RND}))

UPDATE S SET $newCol_j$= NEW.$FCol_j$

WHERE C_{UNEW}

AVG:

UPDATE S SET $newCol_j$ =

(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*

WHERE *PathTrans* AND p_{RN}=v_{RND}

AND v_{RND} = NEW.v_{RND})

WHERE C_{UNEW}

}}

б) OpM = {UPDATE}

BTg = {

for each $col\_name_j \in sCol$

case *agrF*

{***SUM***:

if (NEW.v_{RND} = OLD.v_{RND})

UPDATE S

SET $newCol_j$= $newCol_j$ +NEW.$fCol_j$–OLD.$fCol_j$

WHERE C_{UNEW}

if (NEW.v_{RND} != OLD.v_{RND})

{UPDATE S

SET $newCol_j$= $newCol_j$ +NEW.$fCol_j$

WHERE C_{UNEW}

UPDATE S

SET $newCol_j$= $newCol_j$ –OLD.$fCol_j$

WHERE C_{UOLD}}

COUNT:

if (NEW.v_{RND} != OLD.v_{RND})

{UPDATE S SET $newCol_j$= $newCol_j$ +1

WHERE C_{UNEW}

UPDATE S SET $newCol_j$= $newCol_j$ –1

WHERE C_{UOLD}}

MIN:

if (NEW.v_{RND} = OLD.v_{RND})

{if (NEW.$FCol_j$<

(SELECT $newCol_j$ FROM S

WHERE p_S IN

(SELECT v_{SR1} FROM *RTrans*

WHERE *PathTrans*

AND p_{RN}=NEW.v_{RND}))

UPDATE S SET $newCol_j$= NEW.$FCol_j$

WHERE C_{UNEW}

if (NEW.$FCol_j$>

(SELECT $newCol_j$ FROM S

WHERE p_S IN

(SELECT v_{SR1} FROM *RTrans*

WHERE *PathTrans*

AND p_{RN}=NEW.v_{RND}))

UPDATE S SET $newCol_j$ =

(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*

WHERE *PathTrans* AND

p_{RN}=v_{RND} AND v_{RND} = NEW.v_{RND})

WHERE C_{UNEW}}

if (NEW.v_{RND} != OLD.v_{RND})

{UPDATE S SET $newCol_j$ =

(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*

WHERE *PathTrans* AND p_{RN}=v_{RND}

AND v_{RND} = NEW.v_{RND})

WHERE C_{UNEW}

UPDATE S SET $newCol_j$ =

(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*

WHERE *PathTrans* AND p_{RN}=v_{RND}

AND v_{RND} = OLD.v_{RND})

WHERE C_{UOLD}}

MAX:

if (NEW.v_{RND} = OLD.v_{RND})
if (NEW.$FCol_j$>(SELECT $newCol_j$ FROM S
WHERE p_S IN
(SELECT v_{SR1} FROM *RTrans*
WHERE *PathTrans* AND p_{RN}=NEW.v_{RND}))
UPDATE S SET $newCol_j$= NEW.$FCol_j$
WHERE C_{UNEW}
if (NEW.$FCol_j$<
(SELECT $S.newCol_j$
WHERE p_S IN
(SELECT v_{SR1} FROM *RTrans*
WHERE *PathTrans* AND p_{RN}=NEW.v_{RND}))
UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = NEW.v_{RND})
WHERE C_{UNEW} }
if (NEW.v_{RND} != OLD.v_{RND})
{UPDATE S SET $newCol_j$ =(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND} AND v_{RND} = NEW.v_{RND}) WHERE C_{UNEW}
UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = OLD.v_{RND})
WHERE C_{UOLD}}

AVG:

UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = NEW.v_{RND})
WHERE C_{UNEW}
if (NEW.v_{RND} != OLD.v_{RND})
UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = OLD.v_{RND})
WHERE C_{UOLD} }}

в) OpM = {DELETE}
BTg = {
for each $col\_name_j \in sCol$
case *agrF*
{*SUM:*
UPDATE S SET $newCol_j$= $newCol_j$ –OLD.$fCol_j$
WHERE C_{UNEW}
COUNT:
UPDATE S SET $newCol_j$= $newCol_j$ –1
WHERE C_{UOLD}
MIN, MAX:
if (NEW.$FCol_j$=
(SELECT $newCol_j$ FROM S WHERE p_S IN
(SELECT v_{SR1} FROM *RTrans*
WHERE *PathTrans* AND
p_{RN}=NEW.v_{RND}))
UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = NEW.v_{RND})
WHERE C_{UOLD}
AVG:
UPDATE S SET $newCol_j$ =
(SELECT $agrF_j$ ($FCol_j$) FROM *RTrans, D*
WHERE *PathTrans* AND p_{RN}=v_{RND}
AND v_{RND} = NEW.v_{RND})
WHERE C_{UOLD}
}}

Розроблена програма, яка реалізує наведені правила. Головне вікно програми містить меню, елементи якого реалізують окремі функції, наприклад, елемент «База даних» призначений для вказання параметрів з'єднання з БД.

Після того, як зв'язок встановлено, можна переглянути всі відношення, які знаходяться в БД, в основному вікні програми (рисунок 3).

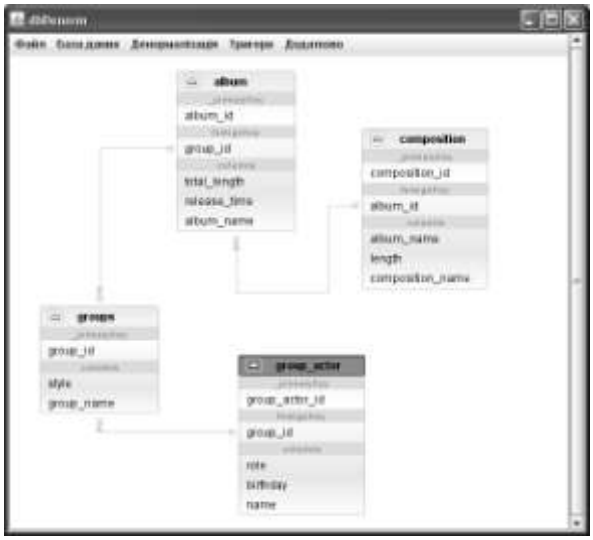

Рисунок 3 – *Основне вікно програми*

Для проведення висхідної денормалізації необхідно скористатися пунктом меню «Денормалізація/Висхідна» и визначити відповідні параметри (рисунок 4).

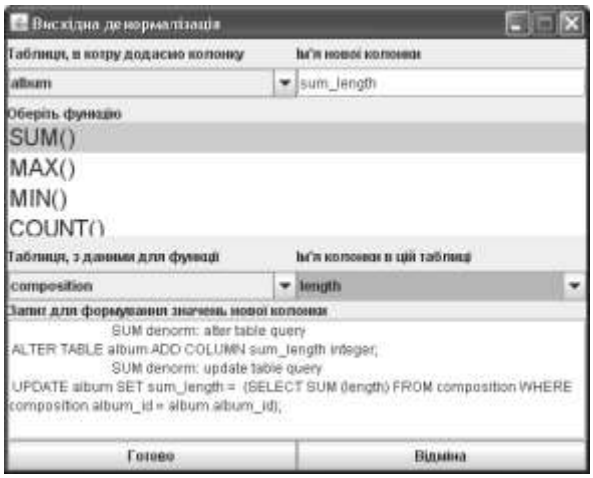

Рисунок 4 – *Висхідна денормалізація БД*

Якщо всі параметри вказано та існують зв'язки між вхідними відношеннями, система

представляє всі згенеровані запити для зміни структури БД та код тригерів (рисунок 5).

Рисунок 5 *– Згенеровані SQL-запити*

ІІІ. ВИСНОВКИ

Надані формалізовані представлення всіх необхідних даних для тригерів, що підтримують цілісність надлишкових даних. При формуванні команд створення тригерів враховується синтаксис конкретно використаної системи керування базами даних.

За результатами дослідження розроблена програма, яка підвищує якість проведення денормалізації БД, а саме:

– підвищує швидкість проведення денормалізації завдяки візуальному відображенню даного процесу у графічному інтерфейсі користувача;

– дозволяє уникнути помилок, які можуть виникнути при ручному проведенні денормалізації, завдяки автоматичній генерації тригерів;

– дозволяє проводити денормалізацію не тільки на етапі проектування БД, а і для БД з заповненими відношеннями;

– дозволяє одночасну роботу з декількома атрибутами відношень БД;

– дозволяє реструктурувати не тільки безпосередньо пов'язані через зовнішній ключ сусідні відношення, а і такі відношення, між якими існує зв'язок через зовнішні ключі інших відношень.

ЛІТЕРАТУРА

1. **Кунгурцев А.Б., Зиноватная С.Л.** Модель реструктуризации реляционной базы данных путем денормализации схемы // Тр. Одесск. политехн. ун-та. – Одесса, 2006. – 2(26). – С. 105-111.

2. Принятие решения о денормализации [электронный ресурс]. — Режим доступа: http://lavsoft.webhost.ru/oracle/book/dezdb/04/04.html.

3. **Горин С.В.** Применение CASE-средства ERwin для информационного моделирования в системах обработки данных / С.В. Горин [электронный ресурс]. – Режим доступа: http://citforum.ru/database/kbd96/65.shtml.

Получена в редакции 15.01.2013, принята к печати 17.01.2013