УДК 681.3.06

ОПТИМІЗАЦІЯ ЗАПИТІВ ПРИ КОНВЕРТАЦІЇ DL/1(IMS) В SQL

А.В. Анісімов, О.П. Кулябко, П.П. Кулябко,О.О. Марченко

Київський національний університет імені Тараса Шевченка, факультет кібернетики

Мова програмування DL/1 (СУБД IMS) має суттєво нижчий рівень порівняно з мовою SQL, але для багатьох технологій при міграції кожному DL/1-оператору ставиться у відповідність точно один оператор мови SQL. Разом з тим досить часто один SQL-оператор може замінити (тобто є функціонально еквівалентним) цілу низку DLI-операторів. Така заміна може суттєво оптимізувати код порівняно з заміною один на один.

The programming language DL/1 (DBMS IMS) has significantly lower level than SQL-language, but many technologies consider that in migration process each DL/1-statement is changed with exactly one SQL-statement. However sometimes one SQL-statement can be put instead of sequence of DLI-statements. Last case may mean the sufficient optimization of the source code in comparison with the previous one.

Вступ

СУБД IMS все ще досить широко використовується в наш час, хоча була запропонована IBM близько 40 років тому [1]. Враховуючи ієрархічну нереляційну структуру бази даних, для IMS стає все важче інтегруватися із сучасними технологіями та новими мовами. Наприклад, технологія реляційної СУБД Oracle (або DB2) є цікавою альтернативою для користувачів IMS, навіть IBM рекомендує перехід до технологій реляційних баз даних як спосіб досягнення більш гнучких систем, хоча з можливою втратою ефективності.

Складність міграції програм з використанням IMS на сучасні платформи змусила деякі організації відкласти ці перетворення на пізніший час, але аргументи на їх користь невідпорні:

- складність інтеграції даних, що надходять з бази даних IMS;
- особливо важко здійснюється доступ до IMS даних із сучасних web-орієнтованих мов;
- наявні ресурси IMS є недостатніми та можуть бути дуже дорогими;
- IMS не підтримує багато з тих можливостей, які пропонуються новими технологіями;
- використання нових технологій може суттєво покращити продуктивність розроблюваних програм.

Однією з найбільш актуальних задач сьогодення є задача міграції програмного забезпечення із застарілих програмних платформ у більш сучасне програмне середовище. Оскільки об'єми великі, то процес міграції (чи конвертації) бажано автоматизувати.

Уточнення завдання

Мова програмування DL/1 має суттєво нижчий рівень порівняно з мовою SQL, але для багатьох технологій [2–9] при міграції кожному оператору EXEC DLI (або ж виклик процедури взаємодії з СУБД IMS) ставиться у відповідність точно один оператор EXEC SQL. Разом з тим досить часто один SQL-оператор може замінити (тобто він є функціонально еквівалентним) цілу низку DLI-операторів. Така заміна може суттєво оптимізувати код порівняно з заміною «один в один», оскільки значна частина функціональності програми (написаної, наприклад, мовою JAVA) перекладається на плечі реляційної СУБД.

Розпізнання конструкцій мови DL/1 разом з конструкціями програми на базовій мові програмування доцільно здійснити за допомогою синтаксичного аналізу, внаслідок якого для програми буде побудоване абстрактне синтаксичне дерево (AST). Власне фрагменти такого дерева і є об'єктами процесу конвертації в об'єкти, які відповідають SQL-операторам. Але виникає проблема побудови правил заміни однієї конструкції (DL/1) на іншу (SQL). Далі розглянемо кілька прикладів виконання такої заміни як основи для побудови відповідних правил.

Оскільки мова SQL має вищий рівень у порівнянні з мовою DL/1 і є класифікація для SQL-запитів, то варто почати з SQL, тобто для кожного типу SQL-запиту у відповідності з класифікацією побудувати фрагмент програми (а точніше – AST) з EXEC DLI + PL/1(COBOL або інша мова програмування). Таких піддерев для кожного SQL-запиту, взагалі кажучи, може бути кілька. Таким чином, побудуємо набір зразків AST, які можуть бути розпізнані у тексті програми за допомогою синтаксичного аналізу, а потім замінені на відповідний SQL-оператор.

Класифікація запитів

У відповідності з реляційним підходом класифікація запитів [10, 11] базується на реляційному численні Кодда, яке є аналогом числення предикатів першого порядку. Кожен запит представляється за допомогою деякої формули реляційного числення у вигляді пренексної нормальної форми. Саме склад цієї формули і служить основою класифікації:

- *простий запит* – не має кванторів у формулі;
- *складний запит 1-го типу* має у формулі тільки квантори існування і не має кванторів узагальнення;
- *складний запит 2-го типу* має у формулі хоча б один квантор узагальнення.

Хоча ця класифікація базується на реляційному численні, вона досить добре узгоджується з ефективністю виконання запитів.

SQL як мова програмування має суттєво нижчий рівень у порівнянні з реляційним численням, тому можна деталізувати деякі з вищенаведених пунктів, беручи до уваги особливості мови SQL.

*Простий запит пр*ацює тільки з однією таблицею (точніше з одним її входженням);

- запит по ключовому полю;
- запит по не ключовому полю;

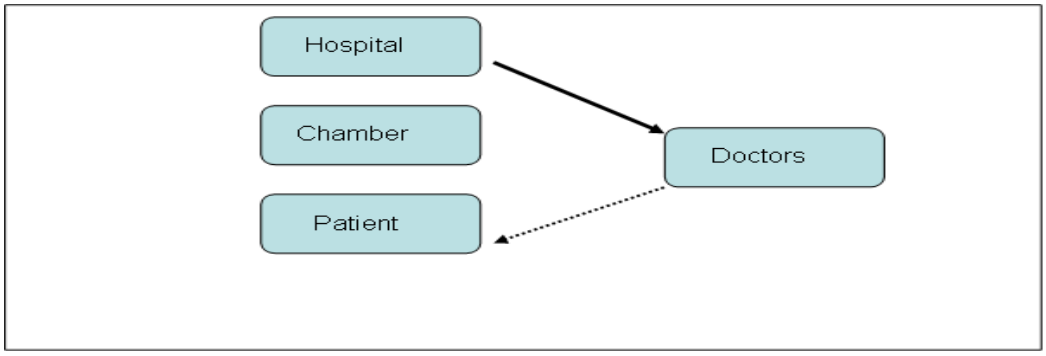
Складний запит 1-го типу працює з кількома таблицями (або з однією, але з кількома входженнями), але всі умови в ньому мають вигляд порівняння між одиничними значеннями (які представлені іменами стовпчиків, змінними з базової програми чи константами), або перевіркою того, чи належить деяке одиничне значення певній множині, яка зокрема може бути представлена оператором SELECT.

Складний запит по методу від супротивного у відповідності з попередньою класифікацією має бути віднесений до третього пункту, але згідно з особливостями виконання операторів мови SQL, його слід віднести до 2-го пункту, бо першу фазу виконання такого запиту, зважаючи на складність її виконання, слід саме до другого пункту; а друга фаза такого запиту – це SQL-операція EXCEPT(MINUS). Принагідно зауважимо, що запити з операціями UNION та INTERSECT відповідно до їх складності виконання слід також віднести до цього ж класу.

Складний запит 2-го типу також працює з кількома таблицями, але має принаймні одну умову типу множинного порівняння. Звичайно, агрегатні та інші SQL-функції створюють певний вплив на складність виконання, а відтак і на місце відповідного запиту у класифікації і це слід враховувати.

Приклад бази даних

Нехай за допомогою DBD у нас задана наступна структура бази даних «Лікарня»



Нехай ім’я DBD, зображеного на рисунку буде **Hosp_inf і**з сегментами Hospital, Chamber, Patient та Doctors. Склад сегментів подано у наступній таблиці.

Segment name	Field list
Hospital	(Numb, Name, Region, City)
Chamber	(Numb, Type, Q-beds, Floor)
Patient	(Numb, Name, Age, Temper)
Doctors	(Numb, Name, Spec, Age)

Детальний опис полів сегментів пропустимо, бо це не є важливим для нашого подальшого розгляду. Зазначимо лише, що підкреслені поля означають унікальність їх значень, тобто вони мають опис (…,SEQ,U). Для використання у прикладних програмах будемо вважати, що у нас є два PCB: PCB1 та PCB2. PCB1 має у своєму складі сегменти (Hospital, Chamber, Patient), а PCB2 – сегменти (Hospital, Doctors, Patient). Після перетворення сегментів отримаємо наступні реляційні таблиці.

Table name	Column list
Hosp_inf_Hospital	(Numb, Name, Address)
Hosp_inf_Chamber	(Numb, *Hosp_Numb*,Type, Q-beds, Floor)
Hosp_inf_Patient	(Numb, *Chamb_Numb*, *Doct_Numb*, Name, Age, Temper)
Hosp_inf_Doctors	(Numb, *Hosp_Numb*, Name, Spec, Age)

Підкреслені імена полів є ключами відповідних таблиць, а імена подані курсивом – «чужими» ключами. Поля «Numb» для сегментів та реляційних таблиць мають, взагалі кажучи, різні значення, бо ключове поле реляційної таблиці має унікальне значення для всієї таблиці, а поле впорядкування сегмента має унікальне значення тільки в межах дії примірника батьківського сегмента.

Загальні зауваження до запису запитів

Мова DL/1має кілька варіантів синтаксису, і для кожного з них характерна значна кількість деталей, які не мають прямого відношення до теми нашої роботи, тому у прикладах будемо використовувати спрощений варіант синтаксису. Наприклад, запит: знайти сегмент палати № 6 у лікарні №1 буде виглядати так:

IO_AREA = GU(PCB1,(HOSPITAL,NUMB=1),(CHAMBER,NUMB=6)).

Для прикладів з використанням мови теж будемо застосовувати деякі спрощення SQL, зокрема будемо пропускати початкові та прикінцеві ключові слова EXEC SQL та END-EXEC.

Для аналізу успішності виконання запиту у системі IMS використовується маска PCB (спеціальна структурна змінна з базової програми), точніше деякі її поля. Для скорочення запису замість згаданої маски будемо використовувати бульові змінні виду FOUND_HOSP чи FOUND_CHAMB, значення яких будуть сигналізувати про успішність (чи неуспішність) пошуку потрібного примірника сегмента HOSPITAL чи CHAMBER.

Приклади запитів

Простий запит по ключовому полю.

1.	Знайти назву лікарні з номером 1
SQL	SELECT Name INTO :T-HOSP-NAME FROM Hosp_inf_Hospital WHERE NUMB = 1
DLI	IO_AREA = GU(PCB1, (HOSPITAL,NUMB=1))
Rem	Структурна змінна з прикладної програми IO-AREA повинна мати у своєму складі поле T-HOSP-NAME на відповідному місці

Простий запит по не ключовому полю.

2.	Знайти всі назви лікарень у місті “N”
SQL	SELECT Name FROM Hosp_inf_Hospital WHERE ADDRESS = “N”

DL/1	IO_AREA = GU(PCB1,(HOSPITAL,ADDRESS = “N”))
	/* other statements */
	WHILE FOUND_HOSP DO
	IO_AREA = GN(PCB1,(HOSPITAL,ADDRESS = “N”))
	/* other statements */
	END;

Rem	Зазначимо, що оскільки такий запит передбачає множинний результат, то для його коректної обробки засобами мови SQL потрібно оголосити курсор та застосувати оператор FETCH всередині циклічного оператора, але у цьому прикладі наводимо тільки SELECT-оператор

Складний запит першого типу по ключових полях кількох таблиць.

3	Знайти прізвище пацієнта з номером 5 з палати номер 6, лікарні номер 1
SQL	SELECT Hosp_inf_Patient.Name INTO :T-PAT-NAME FROM Hosp_inf_Hospital,Hosp_inf_ Chamber, Hosp_inf_Patient WHERE Hosp_inf_Hospital. NUMB = Hosp_inf_ Chamber.Hosp_Numb AND Hosp_inf_ Chamber. NUMB = Hosp_inf_ Patient.Chamb_Numb AND Hosp_inf_Hospital. NUMB = 1 AND Hosp_inf_ Chamber. NUMB = 6 AND Hosp_inf_ Patient.Numb = 5
DLI	IO_AREA_PAT = GU(PCB1, (HOSPITAL,NUMB=1), (CHAMBER,NUMB=6), (PATIENT,NUMB=5))
Rem	

Складний запит першого типу з агрегатною функцією COUNT.

4.	Знайти кількість палат в лікарні номер 1.
SQL	SELECT COUNT (Hosp_inf_ Chamber. NUMB) INTO :T-CNT-CHAMB FROM Hosp_inf_Hospital, Hosp_inf_ Chamber WHERE Hosp_inf_Hospital. NUMB = 1 AND Hosp_inf_Hospital. NUMB = Hosp_inf_ Chamber.Hosp_Numb

DLI	IO_AREA = GU(PCB1, (HOSPITAL,NUMB=1));
	/* other statements */
	COUNTER = 0;
	WHILE FOUND_CHAMB DO
	IO_AREA_CHAMB = GNP(PCB1, (CHAMBER));
	/* other statements */
	COUNTER = COUNTER + 1;
	END;

Rem	У змінній COUNTER накопичується число успішних GNP викликів, що відповідає числу палат у лікарні 1

Складний запит першого типу по методу від супротивного.

5.	Знайти назви всіх лікарень, у яких **не** працює жодного онколога
SQL	SELECT NAME FROM Hosp_inf_Hospital WHERE NUMB NOT IN (SELECT HOSP_NUMB FROM Hosp_inf_DOCTORS WHERE SPEC = 'ONKOLOGY')

DLI	IO_AREA = GU(PCB1, (HOSPITAL));
	/* other statements */
	WHILE FOUND_HOSP DO
	IO_AREA_DOCT = GNP(PCB2,(DOCTORS,SPEC='ONKOLOGY'));
	IF NOT FOUND_DOC THEN PROCESS(IO_AREA);
	/* other statements */
	IO_AREA = GN(PCB1, (HOSPITAL));
	END;

Rem	Процедура PROCESS проводить вихідну обробку даних з сегмента HOSPITAL. Зазначимо, сегмент HOSPITAL попадає на вихідну обробку тільки тоді, коли у ньому не знайдеться жодного лікаря зі спеціальністю онкологія.

Складний запит 2-го типу (з множинним порівнянням).

6.	Знайти імена лікарів, які «ведуть» тільки тих пацієнтів, що лежать на 2-му поверсі (пацієнта може вести тільки один лікар, але один лікар може «вести» багато пацієнтів)
SQL	SELECT NAME FROM HOSP_INF_DOCTORS WHERE NOT EXISTS (SELECT HOSP_INF_PATIENT.NUMB FROM HOSP_INF_PATIENT WHERE HOSP_INF_PATIENT.DOCT_NUMB = HOSP_INF_DOCTORS.NUMB) EXCEPT (SELECT HOSP_INF_PATIENT.NUMB FROM HOSP_INF_PATIENT, HOSP_INF_CHAMBER WHERE HOSP_INF_CHAMBER.NUMB = HOSP_INF_PATIENT.CHAMB_NUMB AND HOSP_INF_CHAMBER.FLOOR =2)

DLI	IO_AREA_CHAMB = GU(PCB1,(HOSPITAL),(CHAMBER,FLOOR=2));
	WHILE FOUND_PAT DO
	IO_AREA_PAT = GNP(PCB1,(PATIENT));
	Q_PAT_ARRAY = POPULATE(PAT_ARRAY,IO_AREA_PAT);
	END;
	/* find out patients from the 2 floor */
	WHILE FOUND_DOCT DO
	IO_AREA_DOCT = GN(PCB2,(HOSPITAL),(DOCTORS));
	FIN = 1; COUNTER = 0;
	WHILE FOUND_PAT & FIN DO
	IO_AREA_PAT = GNP(PCB2,(PAT));
	COUNTER = COUNTER + 1;
	IF NOT_IN(PAT_ARRAY,IO_AREA_PAT) THEN FIN = 0;
	END;
	IF (COUNTER > 0) & FIN THEN PRINT_SEGM(IO_AREA_DOCT);
	END;

Rem	Функція POPULATE зберігає знайдені сегменти PATIENT (чи їх ключові поля) у спеціальній послідовності PAT_ARRAY. Ми свідомо не уточнюємо тип цієї послідовності. Це може бути масив, список чи файл або база даних. Потім послідовно у циклі, фіксуючи як поточний, сегменти DOCTORS, перевіряємо: (за допомогою функції NOT_IN), чи належать раніше утвореній послідовності PAT_ARRAY сегменти PATIENT, які є підлеглими поточного сегмента DOCTORS. Якщо серед пацієнтів «поточного» лікаря трапиться хоча б один, який не належить PAT_ARRAY, то відповідний лікар не потрапить у вихідний список (формується процедурою PRINT_SEGM)

Складний запит 2-го типу (множинне порівняння частково зведене до кількісного)

7.	Знайти ім’я лікаря, що веде принаймні всіх пацієнтів, які старші за 50 років
SQL	SELECT NAME FROM HOSP_INF_DOCTORS WHERE NOT EXISTS (SELECT NUMB FROM HOSP_INF_PATIENT WHERE AGE > 50) EXCEPT (SELECT NUMB FROM HOSP_INF_PATIENT WHERE HOSP_INF_PATIENT.DOCT_NUMB = HOSP_INF_DOCTORS.NUMB)

DLI	WHILE FOUND_PAT DO
	IO_AREA_PAT = GN(PCB2,(HOSPITAL),(DOCTORS),(PATIENT,AGE>50));
	Q_PAT_ARRAY = POPULATE(PAT_ARRAY,IO_AREA_PAT);
	/* QUANTITY OF THE SELECTED PATIENTS */
	WHILE FOUND_DOCT DO
	IO_AREA_DOCT = GN(PCB2,(HOSPITAL),(DOCTORS));
	QP = 0;
	WHILE FOUND_PAT DO
	IO_AREA_PAT = GNP(PCB2,(PATIENT));
	IF IN(PAT_ARRAY,IO_AREA_PAT) THEN QP = QP + 1;
	END;
	IF QP >= Q_PAT_ARRAY THEN PRINT_SEGM(IO_AREA_DOCT);
	END;

Rem	Останній запит у певному сенсі є зворотним стосовно попереднього, тобто множинне порівняння здійснюється зворотним способом. Тому тут застосований підрахунок сегментів, які задовольняють умові, з наступним порівнянням по кількості. Функція IN є протилежною функції NOT_IN.

Висновки

З наведених прикладів досить добре проглядається складність процесу синтаксичного аналізу для різних типів запитів. Найменшу складність для синтаксичного аналізу мають 1 та 3 запити, але оптимізації тут практично немає, бо конвертація здійснюється «один в один».

Середню позицію по синтаксичній складності займають запити 2, 4 та 5; а оскільки конвертація в цих випадках відповідає формулі «багато в один», то й оптимізація може бути значною.

Найбільш складними для синтаксичного аналізу є 6 та 7 запити, але й оптимізаційний ефект тут очікується найбільшим. Зазначимо також, що запити з множинними порівняннями в реальних програмних комплексах зустрічаються досить рідко.

Слід зауважити, що певна частина проблем технологічного плану залишилась за межами розгляду цієї роботи. Наприклад, для SQL-операторів характерно явне задання імен таблиць чи стовпчиків, а DL/1-виклики досить часто використовують змінні базової програми, значеннями яких є імена сегментів та їх полів. Віднайти ці значення за допомогою синтаксичного аналізу програми є досить складним завданням.

1. http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?
2. http://www.migrationpros.com/dbre-IMStoSQL.html
3. http://www.ateras.com/Converting_IMS_Databases.aspx
4. http://findarticles.com/p/articles/mi_hb5570/is_200610/ai_n23570939
5. http://www.migrationware.com/pm/dbims.html
6. http://forums.mysql.com/read.php?63,52523,52523
7. http://www.clerity.com/resources/transition/datasheets/IMS_Datasheet.pdf
8. http://www.move2open.com/ims-to-rdbms.html
9. http://www.zjournal.com/index.cfm?section=article&aid=366
10. *Codd E.F.* Relational Completeness of Data Base Sublanguage. Data Base Systems // Courant Computer Science Symposium 6 (May 24–25, 1971). – S. 1. – Prentice-Hall, 1972. – P. 65–98.
11. *Дейт К.* Введение в системы баз данных. – М.: Вильямс. – 2005. – 1328 с.