Altibase에서의 Date 연산 개선 방법+

정동인O, 김상우, 이상원

성균관대학교 정보통신공학부
{furybird, spun, swlee}@skku.edu

How to improve date calculation in Altibase

Jeong Dong-inO, Kim Sang-woo, Lee Sang-won
School of Information and Communication Engineering, Sungkyunkwan University

요 약

DBMS에서는 날짜를 저장할 수 있는 자료형을 일반적인 자료형과 별도로 구현하고 있다. 하지만 복잡한 역법과 다른 자료형에 비한 상대적으로 적은 빈도의 사용으로 대부분의 DBMS에서 소홀히 하고 있는 실정이다. 시스템 호출을 이용하여 간단히 구현하거나 정확한 역법에 따라 구현하지 못하여 기능적, 성능적으로 다른 자료형에 비해 많이 뒤쳐져 있다. 이에 본고는 정확한 역법을 집어보고 상용화되어 널리 쓰이고 있는 Hybrid DBMS인 Altibase에서의 Date 연산을 개선한 내용에 대해 다루고 있다. 이로 인하여 기능적으로 연산 범위가 확장되었으며, 성능적으로 약 8배의 개선을 이루었다.

1. 서 론

1.1 배 경

Altibase DBMS(이하 알티베이스)는 Hybrid DBMS로서 단일 DBMS 내에서 메인 메모리 DBMS(MMDBMS)와 디스크 DBMS를 제공, 하나의 데이터베이스를 여러 개의 저장매체에 나누어 저장하여 관리할 수 있도록 설계되어 있다. 이로써 데이터의 성능 가중치에 따라 자주 그리고 빠르게 접근하는 데이터는 메모리에, 그렇지 않은 데이터는 디스크에 저장하여 관리할 수 있어 10배 이상 빠른 트랜잭션 속도를 제공하는 MMDBMS의 장점과 대용량 데이터 처리를 지원하는 디스크 DBMS의 장점을 동시에 이용할 수 있는 새로운 개념의 DBMS이다[1].

알티베이스에서 많은 사용이 되고 있는 날짜형 자료를 위해 Date 타입과 연산을 별도로 구현하여 날짜 저장 및 연산을 쉽고 효율적으로 할 수 있도록 하였다. 구현 시 라이브러리 자료형과 시스템 호출을 이용하였으며, 사용된 것들은 달력형 날짜를 저장하는 struct tm, 시스템 시간을 저장하는 time_t와 그것들을 상호변환 해주는 mktime(), localtime() 함수이다. 이 중 time_t형은 32bit의 초단위로서 -2147483648초 ~ 2147483647초까지 표현할 수 있다. 년으로 환산하면 약 -68년 ~ +68년으로서 UTC 1970-01-01 00:00:00을 0초로 기준하여 카운트를 하고, Unix 및 Linux 시스템에서 시스템 시간을 계산하는데 쓰이고 있다. 이 범위제약 때문에 2038-01-19 03:14:07까지 밖에 표현을 하지 못하고, time_t형을 사용하는 시스템은 2038년이 오기 전에 범위제약을 개선해야 하는 문제점을 가지고 있다. 알티베이스 역시 time_t형을 이용하여 Date연산을 하기 때문에 1970년부터 2038년까지의 약 68년 정도만 연산 할 수 있는 기능적인 한계가 있었다. 또한 성능이 중시되는 MMDBMS를 포함한 Hybrid DBMS 임에도 불구하고 Date 연산은 뛰어날 정도의 성능은 내지 못하는 한계도 있었다. 이에 본고는 알티베이스를 기능적, 성능적으로 개선하고자 라이브러리 자료형과 시스템 호출로 구현되어 있는 Date 연산을 알티베이스 내부 자료형과 함수로 구현하는 방법을 기술하고 있다. 이로 인하여 기능적으로 연산 범위가 확장되었으며, 성능적으로 약 8배의 개선을 이루었다.

본 논문의 구성은 다음과 같다. 2장에서는 정확한 Date형 구현을 위한 역법 등의 관련 연구를 기술하였고, 3장에서는 개선 방법을 구현한 내용에 대해 다뤘다. 4장은 결론 및 한계점을 기술하였다.

2. 관련 연구

2.1 현 상

2.1.1 예제테이블

```
iSQL> select * from timetbl;

TIMETBL.T1   TIMETBL.ETC
-----------------------------
09-JAN-2008  Start
18-JAN-1985  jisung
15-MAR-1895  gain
3 rows selected.
```

+ 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2008-(C1090-0801-0046))

2.1.2 덧 셈

```
iSQL> select t1+10 from timetbl order by i1;

T1+10
---------------
19-JAN-2008
28-JAN-1985
[ERR-21002 : Not applicable]
2 rows selected.
```

예제테이블에서 10일을 더하는 쿼리를 실행하였다. 두 번째 튜플까지를 정상적으로 10일이 더해졌다. 하지만 세 번째 튜플은 time_t형의 범위를 벗어나는 연산을 하여 계산이 되지 않고 에러메시지를 띄우고 있다. 정상적인 결과는 다음과 같이 나와야 한다. 체크한 부분이 정상적으로 나온 결과이다.

```
iSQL> select t1+10 from timetbl;

T1+10
---------------
19-JAN-2008
28-JAN-1985
25-MAR-1895  √
3 rows selected.
```

2.1.3 뺄 셈

```
iSQL> select t1-10 from timetbl;

T1-10
---------------
30-DEC-2007
08-JAN-1985
[ERR-21002 : Not applicable]
2 rows selected.
```

덧셈을 뺄셈으로 바꾸어 10일을 빼는 쿼리를 실행하였다. 이 역시 두 번째 튜플까지를 정상적으로 10일이 계산되어졌고, 세 번째 튜플은 에러메시지를 띄우고 있다. 정상적인 결과는 다음과 같이 나와야 한다.

```
iSQL> select t1-10 from timetbl order by i1;

T1-10
---------------
30-DEC-2007
08-JAN-1985
05-MAR-1895  √
3 rows selected.
```

2.2 라이브러리 자료형 및 시스템 호출

시스템 호출이란 입출력 제어 등 운영체제의 기능을 요청하기 위해 응용 프로그램에서 사용되는 프로그래밍 인터페이스이고, 라이브러리는 컴퓨터에서 독립적으로 실행되지 않고, 다른 프로그램의 실행을 도와주는 프로그램들을 가리킨다. 아래의 자료형들은 C 표준 라이브러리의 자료형들이다.

2.2.1 struct tm

```
int  tm_sec;     /* seconds after the minute [0, 61] */
int  tm_min;     /* minutes after the hour [0, 59] */
int  tm_hour;    /* hour since midnight [0, 23] */
int  tm_mday;    /* day of the month [1, 31] */
int  tm_mon;     /* months since January [0, 11] */
int  tm_year;    /* years since 1900 */
int  tm_wday;    /* days since Sunday [0, 6] */
int  tm_yday;    /* days since January 1 [0, 365] */
int  tm_isdst;   /* flag for daylight savings time */
```

비선형으로 시간의 달력 표현을 저장하는 구조체.

2.2.2 time_t

시스템 시간을 저장하는 32bit의 정수형 자료형. UTC 1970-1-1 0:00:00 이후 경과한 초를 담고 있다.

2.2.3 mktime()

mktime 함수는 달력 표현으로 된 tm 시간을 time_t의 시스템 시간으로 변환시키기 위해서 사용된다. 그것은 다른 날짜에 기초하여 연도, 주의 날수와 시간 요소들을 채워서 tm 구조체의 내용을 일반화한다.

2.2.4 localtime()

localtime 함수는 인자로 받은 시스템 시간을 사용자가 정한 시간대와 맞추어서, 시간의 달력표현으로 변환하여 사용자가 지정한 tm 구조체에 저장한다.

2.3 Date 자료형의 표현 범위

2.3.1 역법

2.3.1.1 그레고리력(Gregorian Calendar)

오늘날 거의 모든 나라에서 사용하는 세계 공통력이라고 할 수 있다. 교황 그레고리우스 13세의 초기시대에는 율리우스력(曆)을 쓰고 있었는데, 율리우스력에서는 오랫동안 누적된 역법상의 오차로 원래는 3월 21일이어야 할 춘분이 달력에서는 3월 11일로 옮겨져 있었다. 춘분은 부활절을 정할 때 기준이 되는 중요한 날이었기 때문에 기독교는 이 오차를 받아들이기 힘들었다.

결국, 교황은 각 교회와 의논한 끝에 1582년 10월 5일부터 14일까지를 건너뛰고, 즉 10월 4일 다음날을 10월 15일로 한다는 새 역법을 공포하였다. 이것이 현재까지 사용하는 그레고리력이다. 아래와 같은 윤년 계산법을 갖는다[2].

- 4로 나누어 떨어지는 해는 윤년이다.
- 그 중에서 100으로 나누어 떨어지는 해는 윤년이 아니다.
- 그 중에서 400으로 나누어 떨어지는 해는 윤년이다.

2.3.1.2 율리우스력(Julian Calendar)

로마의 율리우스 카이사르가 개정한 역법으로 평년을 365일로 하고 4년마다 한번 씩 윤년을 두어 366일로 하였다. 이것이 BC 46년 1월 1일 실시된 태양력의 시초인 율리우스력이다.

당시에 1, 3, 5, 7, 9, 11월은 31일로 하고 나머지 달은 30일로 하고 2월은 평년 29일, 윤년에는 2월이 30일이었는데 다음 황제인 아우구스투스가 황제로 등극하여 율리우스의 달인 7월이 31일이었는데 자기의 생일달인 8월이 30일까지로 작으므로 이를 31일까지로 고치고 9월과 11월은 30일, 10월과 12월은 31일로 하고 2월은 평년 28일, 윤년이들 때는 29일로 만든 것이 지금 우리가 사용하는 각 달의 크기이다[3].

2.3.1.3 율리우스적일제(Julian Day)

율리우스력의 BC 4713년 1월 1일 정오부터 흘러간 날짜수로 표시하는 날짜체계이다. 정오 이외의 시점을 표현하기 위해 소수점 이하의 표기를 허용한다. 천문학이나 연대기에서 어떤 시점을 혼동 없이 표시하기 위해 1582년 그레고리력과 동시에 도입되었다. 예를 들자면 2000년 1월 1일 세계시 0시는 율리우스적일로 2451544.5가 된다[4].

2.3.2 역 전환일

그레고리력을 선포한 날은 1582년 10월이다. 하지만 대부분의 나라들이 이 역법을 100년 후 쯤에나 채택하였다. 그 중 그 당시 가장 강력했던 나라 중 하나인 영국이 1752년 9월에 적용하였는데 그 때가 세계 달력에서 그레고리력으로 전환된 날이 되었다. 실제로 리눅스의 cal 명령어로 달력을 출력하면 아래와 같은 결과를 얻는다.

```
# cal 9 1752

      9월 1752
일 월 화 수 목 금 토
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

몇몇 율리우스력 미지원 DBMS들이 이 비어있는 날짜에 대한 연산을 피하기 위해 1753년 이후 연산만 지원하는 것도 있다.

2.3.3 Oracle DBMS의 경우

Oracle DBMS(이하 오라클)의 경우 여러 가지의 역법을 사용 할 수가 있는데 기본적으로 율리우스력과 그레고리력 두가지 역법을 혼용하는 방식을 택하고 있다. 두가지 역법을 사용하면서 생기는 역 전환일은 Unix/Linux 시스템의 역 전환일을 사용하지 않고 실제 역사상의 역 전환일인 1582년 10월을 사용하였다. 1582년 10월 4일 이전은 율리우스력을 사용했기 때문에 오라클도 이를 따라서 실제 역법 전환일인 1582년 10월 15일 이후로만 그레고리력을 따르고, 그 이전은 율리우스력을 따른다. 따라서 이 역법을 사용한다면 1582년 10월 5일부터 10월 14일까지의 달력은 오라클에서는 존재하지 않는다.

다음은 1582/10/05 부터 1582/10/14까지가 비어있는 것을 확인할 수 있는 예제 쿼리들이다.

```
SQL> select to_date('1582-10-01', 'yyyy-mm-dd') -
to_date('1582-10-16', 'yyyy-mm-dd') diff from timetbl;

      DIFF
----------
-5

SQL> select to_date('1582-10-01', 'yyyy-mm-dd') + 3 from timetbl;

TO_DATE(
--------
82/10/04

SQL> select to_date('1582-10-01', 'yyyy-mm-dd') + 4 from timetbl;

TO_DATE(
--------
82/10/15
```

1582/10/05 ~ 1582/10/14 사이의 날짜를 입력 받을 때 그것에 대한 처리는 유효한 날짜까지 무조건 올림하여 저장한다. 다음은 그것을 확인할 수 있는 예제 쿼리들이다.

```
SQL> insert into timetbl values ( 1, '1582-10-08');
SQL> insert into timetbl values ( 2, '1582-10-04');
SQL> insert into timetbl values ( 3, '1582-10-05');
SQL> insert into timetbl values ( 4, '1582-10-14');
SQL> insert into timetbl values ( 5, '1582-10-15');

SQL> select * from timetbl;
```

```
I1         T1
---------- --------
1          82/10/15
2          82/10/04
3          82/10/15
4          82/10/15
5          82/10/15
5 개의 행이 선택되었습니다.
```

2.3.4 오라클의 한계

2.3.4.1 Date 자료형의 표현 범위

율리우스적일제의 시작년부터 9999년까지 표현 가능하다.

표 1. 오라클에서 시간 관련 자료형의 표현 범위[5]

자료형	최소값 (0년 제외)	최대값 (0년 제외)
DATE	- 4712-01-01.00.00.00	9999-12-31
TIME STAMP	- 4712-01-01.00.00.00. 000000000	9999-12-31.23.59.59. 999999999

2.3.4.2 0년 버그

기원후 1년 1월 1일에서 기원전 1년 12월 31일을 뺀 결과는 1도 아니고 366도 아닌 367이 나온다. 이것은 Date 연산 시 내부적으로 0년이 존재한다는 것을 의미한다. 또한 0년이 윤년으로 적용되어 결과가 367일로 나왔다. 따라서 오라클에서는 윤년의 주기인 4년 주기로 계산을 해보면 BC 1년이 윤년이 아니라 BC 4년이 윤년이 된다[6].

```
SQL> select to_date('AD 0001-01-01', 'BC YYYY-MM-DD') -
to_date('BC 0001-12-31', 'BC YYYY-MM-DD') diff from dual;

      DIFF
----------
       367
```

2.3.4.3 BC 4712년의 버그

오라클 문서에서는 BC 4712년이 율리우스적일제의 시작년 이라고 하였지만[2] 실제로는 BC 4713년이 율리우스적일제의 시작년이다. 게다가 4년 단위 윤년이기 때문에 BC 4년이 윤년 이었으므로 BC 4712년도 윤년이여야 하지만 실제로는 윤년으로 적용되지 않는 버그도 있다. BC 4708, BC 4704는 윤년으로 적용 된다[6].

```
SQL> select to_date('BC 4712-02-29', 'BC YYYY-MM-DD') dt
from dual;
select to_date('BC 4712-02-29', 'BC YYYY-MM-DD') dt from
dual
                *
ERROR at line 1:
ORA-01839: date not valid for month specified

SQL> select to_date('BC 4708-02-29', 'BC YYYY-MM-DD') dt
from dual;

DT
------------------
29-FEB-08
```

2.3.4.4 Austronomical year numbering

Austronomical year numbering은 AD/CE year numbering에 기초하고 있다. 하지만 AD/CE 방식에 비해 일반적인 정수 numbering을 더 엄격하게 따른다. 따라서 0년이 존재하고, 0년 이전은 - 기호로 표기한다. 종종 연도의 연산을 할 때 숫자 0이 필요하기 때문에 천문학에서 자주 쓰인다[7].

표 2. 두가지 연도 표기법

Common year numbering	Astronomical year numbering
2 BC	-1
1 BC	0
1 AD	+1 (+는 생략할 수 있다)

좌측은 일반 BC/AD 표기이고 우측은 Astronomical year numbering이다. 0년이 존재하며 기원전 표기는 - 부호를 붙여서 표현한다. 이것에 따르면 율리우스적일제 시작년은 -4712(BC 4713)년이 된다.

2.3.4.5 요 약

4년마다 한번인 윤년이 BC에서는 1, 5, 9,…이여야 한다. 하지만 오라클에서는 4, 8, 12,…을 윤년으로 삼았고, Date 자료형의 표현범위가 BC 4712년부터인 것을 보면 Astronomical year numbering과 혼동을 한 것으로 추측된다. 그렇기 때문에 내부적으로 0년이 존재한다고 생각된다. 하지만 Astronomical year numbering을 따르는 것을 의도했다면 0년도 유효해야 하지만 오라클에서는 0년을 제외하는 실수를 두었다.

3. 구 현

3.1 자료형

3.1.1 mtdIntervalType

알티베이스 내부 자료형인 Interval 형은 64bit의 Big int 형으로 두 Date 간의 간격을 초로 저장하는데 쓰이는

자료형이다. 64bit로 나타낼 수 있는 초의 범위는 약 584,942,417,355년이다. 광범위한 범위이기 때문에 현재 연산 알고리즘에서 별도의 수정 없이 time_t를 이용하여 연산하는 부분을 Interval 형으로 대체하여 연산한다면 기능적으로 개선할 수 있다.

3.1.2 mtdDateType

알티베이스에서 Date 형을 저장하는데 사용되는 알티베이스에 최적화 된 달력표현 구조체이다. 년, 월, 일, 시, 분, 초, 마이크로초의 멤버 변수들을 갖는다.

3.2 변환 함수

3.2.1 기존 변환 과정

알티베이스에서 Date 연산 시 자료형의 변환 및 연산 과정은 다음과 같다.

mtdDateType → struct tm → time_t →

struct tm → mtdDateType

각 연산과 함수는 time_t 형과 struct tm 형의 라이브러리 자료형을 사용하고 있다. 최종적으로 피연산자들을 선형 자료형인 time_t형으로 통일시켜 연산을 수행한다. 이 자료형들 때문에 시스템 호출이 필요하게 되는 것인데, 이 자료형들을 새로운 자료형으로 대체하고 시스템 호출에 대응되는 함수를 새로 만든다.

3.2.2 새로운 변환 과정

Date 형을 Interval 형으로, Interval 형을 Date 형으로 변환하는 두 가지 인터페이스 함수를 만들어 피연산자들의 자료형을 통일시키는 방법이다. 인터페이스 함수들은 각각 달력표현을 초단위로, 초단위를 달력표현으로 변환하여준다. 여러 번의 변환 과정을 단축시킴으로써 성능의 향상도 꾀할 수 있다.

mtdDateType → mtdIntervalType → mtdDateType

3.3.3 수행 효과

3.3.3.1 기능적 효과

날짜 연산이 가능한 범위가 1970 ~ 2038년에서 1 ~ 9999년까지로 확대되었다.

3.3.3.2 성능적 효과

릴리즈 버전에서 실행한 성능 측정 결과이다. 변환 작업 수행으로 인하여 전보다 약 8배의 성능 향상을 가져왔다. 테스트 환경은 다음과 같다. 노트북, Intel(R) Core(TM)2 Duo CPU T7250 @ 2.00GHz, 1.9GB Memory, OS는 Ubuntu 7.10 gutsy 이다. 500,000건에 대한 연산을 수행하여 분석한 결과이다.

▲ 성능 측정시 사용하였던 SQL문

500,000 Dates Insert	drop table t1;
	create table t1(i1 date);
	
	create or replace procedure proc1
	as
	begin
	for i in 1 .. 500000 loop
	insert into t1 values (to_date('0001-01-01', 'YYYY-MM-DD') + mod(random(0), 8999*12*30));
	end loop;
	end; /
	
	set timing on;
	exec proc1;

실제 성능 측정 쿼리	select count(*) from (select i1 + 1 from t1);

MMDBMS가 포함된 Hybrid DBMS답게 500,000건의 Date들을 저장하는데 디스크 기반 DBMS보다 몇 배 빠른 시간이 소요되었다.

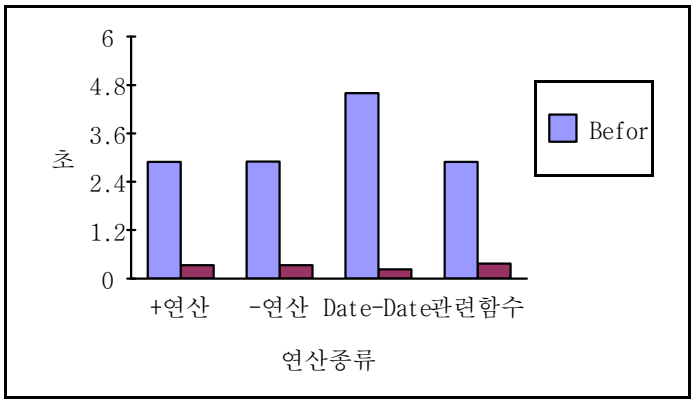

그림 1. 작업 수행 전과 후의 연산 속도 비교

일반적인 연산의 경우 평균 8 ~ 9배정도 차이가 났다. Date - Date 형태의 연산은 두 개의 피연산자가 비선형에서 선형으로 두 번의 변환과정을 갖기 때문에 한 개의 피연산자만 변환되는 다른 연산에 비해 2배정도 더 차이가 났다.

4. 결 론

라이브러리 자료형과 시스템 호출을 직접 구현한 내부

자료형과 함수로 대체하였더니 Date 연산의 범위 확장뿐만 아니라 성능도 대폭 향상되었다. 성능이 이처럼 대폭 향상된 것은 시스템 호출이 변환작업을 할 때 DBMS에서 필요로 하지 않는 다른 많은 연산들을 수행하기 때문에 느린 것으로 파악된다. 성능이 중요시 되는 프로그램이라면 시스템 호출을 사용할 것이 아니라 프로그램 성격에 따라 필요한 기능만 직접 구현하여 사용해야 할 것이다.

이 변환 작업은 기능적, 성능적으로 몇 가지 한계점을 남기고 종료하였다.

첫째로 기원전의 미지원이다. 1년 이후로는 선형 연산이기 때문에 구현이 쉽지만 1년 이전, 기원전까지 지원을 하려면 구현 시 0년에 대한 비선형 처리를 해야 한다. 본고에서는 비선형 처리까지는 다루지 못하였다.

둘째로 여러 역법의 미지원이다. 역법 변환을 지원하려면 비어있는 날짜들 때문에 역시 비선형 연산이 필요하다.

셋째로 9999년 이후의 미지원이다. 이는 DBMS에서 현재 4자리의 연도만 지원하기 때문인데 연도의 자리 제한이 5자리로 늘어난다면 9999년 이후의 지원은 별도의 작업 없이 충분히 가능하다.

그리고 성능상 아직 더 개선의 여지가 남아있다는 점이다. 본고는 변환 했을 시 효과에 대해서만 다루고 있다. 즉, 본고의 테스트 결과는 최적의 알고리즘과 코드 최적화를 거치지 않은 테스트 알고리즘이다. 그렇기 때문에 최적화를 수행한다면 2배 이상의 성능을 보일 것으로 예상한다.

참고 문헌

[1] Altibase, Altibase history - Hybrid DBMS, "http://www.altibase.com/sub02/sub_020105.jsp"

[2] Wikipedia, Gregorian calendar, "http://en.wikipedia.org/wiki/Gregorian_calendar"

[3] Wikipedia, Julian calendar, "http://en.wikipedia.org/wiki/Julian_calendar"

[4] Wikipedia, Julian day, "http://en.wikipedia.org/wiki/Julian_day"

[5] Shibu Kalluvila Raj, Dates in DB2 and Oracle: Same Data Type, Different Behavior, "http://www.devx.com/dbzone/Article/28713/1954?pf=true"

[6] Peter Gulutzan and Trudy Pelzer, The Oracle Calendar, 2003, "http://www.orafaq.com/papers/dates_o.doc"

[7] Wikipedia, Astronomical year numbering, "http://en.wikipedia.org/wiki/Astronomical_year_numbering"