오라클의 Job 을 이용한 Geodatabase 성능개선

1. 개요.

ArcGIS 에서 버저닝을 통한 Geodatabase 편집 시 그 변경 사항들은 원본 테이블에 직접 반영되는 것이 아닌 트랜젝션 형태로 저장, 관리된다. 이 때 변경 사항들은 ArcSDE 의 버저닝 관련 테이블(Version, States, State_lineages, Mvtables_modified)과 각 레이어의 실제 변경사항을 저장하는 델타 테이블(Add, Delete 테이블)에 저장된다. 이 델타 테이블은 편집이 진행됨에 따라 레코드가 급격하게 증가하는 특징이 있으며, Version query, Reconcile 과 같은 작업이 진행되면 버저닝 테이블들이 변경되게 된다. 만일 버전 관련 테이블에 대한 통계정보가 최신의 것으로 갱신되지 않는다면 이들 테이블에 대한 액세스 성능은 떨어지게 된다. 따라서 이 테이블들에 대한 액세스 성능을 보장하기 위해서는 통계정보가 최신의 상태로 유지되어야 하며 특히 다중버전을 이용하는 경우 Reconcile 작업을 수행할 때 반드시 통계정보가 최신의 것으로 유지되어야 최적의 성능을 발휘할 수 있다.

이 문서에서는 최신의 통계정보 유지를 위한 처리 방안으로 버전 관련 테이블과 델타테이블 및 인덱스에 대해서 일정한 시간 간격으로 자동화된 통계 갱신을 통해 Versioned geodatabase 에 대해 액세스 성능을 향상시키고자 한다. (테이블에 대한 통계 갱신 시 에는 테이블 인덱스에 대한 통계갱신이 함께 이루어 짐)

2. 버저닝 관련 테이블

- ArcSDE 시스템 테이블
  - State_Lineages – 각 state 의 편집 리니지를 저장
  - States – state 의 메타데이터 저장 (state ID, 소유자, 생성/종료 시간, 부모 stateID)
  - Mvtables_Modified – 변경된 테이블의 state 와 table ID 저장
- 사용자 테이블 및 인덱스
  - A[Registration_id] – 추가되는 편집 내용 저장
  - D[Registration_id] – 삭제되는 편집 내용 저장.

    * Registragion_id 는 Table_Registry 테이블에서 확인 가능, 각 테이블에 대한 인덱스 포함

    * 공간 데이터 레이어 및 일반 테이블 (Business, Spatail Index)들은 Compress 수행 시에 갱신이 이뤄지므로 Compress 후에 통계정보를 갱신한다.

편집이 진행되는 동안에는 변경사항들이 버전 관련 테이블에만 반영되기 때문에 이들 테이블에 대해서만 통계정보를 갱신하도록 한다.

3. 오라클의 job 를 이용하기 위한 기본 사항.(개념 부분이므로 SKIP 해도 상관 없습니다.)

1) 프로시저 (Procedure)

프로시저란 데이터베이스에 보관되며, 파라메터를 받을 수 있고, 애플리케이션이나 PL/SQL 루틴에서 호출 할 수 있는 이름을 가진 PL/SQL 블록이다.

A. 프로시저의 특징

- PL/SQL 블럭들로 구성된다.
- 함수, 프로시저 등을 호출할 수 있다.
- 리턴값을 가질 수 없다.
- Query 문에서 독립적으로 사용될 수 없다.
- 데이터에 변형을 가할 수 있다.
- 프로시저 내의 PL/SQL 블록은 어떤 DML 문장이든 포함 될 수 있으나 DDL 문장은 사용할 수 없다.

B. 프로시저 생성과 실행

프로시저의 생성과 실행을 위해서는 권한이 필요하다. 생성을 위한 권한은 CREATE PROCEDURE 권한이 필요하며, 실행을 위해서는 EXECUTE 권한이 필요하다. 또한 프로시저는 소유자의 이름을 지정함으로써 다른 유저의 프로시저에 대해서도 실행할 수 있으며, 데이터베이스 링크를 이용해서도 다른 데이터베이스에 있는 프로시저를 호출 할 수 있다.

2) 패키지 (Package)

패키지는 관련된 객체들을 함께 저장할 수 있게 하는 PL/SQL 블록들로 구성되어 있다. 패키지는 두 개의 분리된 부분(명세, 주요부)으로 구성되어 있다. 각각은 데이터베이스에 따로 저장되지만 하나라도 없으면 실행 할 수 없다. 패키지 내에서는 함수나 프로시저는 오버로드(overload)될 수 있다. 즉 패키지는 프로시저나 함수들의 집합인 것이다.

A. 프로시저 보다 패키지를 권장하는 이유

패키지와 프로시저의 정의를 살펴보면 그다지 큰 차이점을 발견할 수 없다. 그렇다면 조금이라도 코드가 짧은 프로시저를 사용하지 않고 패키지를 사용하는 이유는 무엇일까? 답은 패키지의 향상된 의존성 관리 때문이다. 프로시저를 이용하게 될 경우 일어날 수 있는 심각한 시스템 오류 상황을 패키지를 이용하게 되면 간단한 컴파일 만으로 시스템 오류를 막을 수 있기 때문이다.

3) 오라클 Job

DBMS_JOB 패키지는 오라클에서 주기적으로 수행되는 백업작업이나 쿼리나 프로시저등의 Job 을 시간단위, 일단위, 월단위 등 주기적인 예약 작업으로 등록하여 동작할 수 있도록 하는 스케줄러이다.

유닉스에서 Cron 에 등록하고 사용하는 것과 유사한데 차이점이라면 Cron Job 은 OS 가 직접 관리하고 실행하지만, DBMS_JOB 에 등록된 job 는 오라클이 관리한다.

* 단 oracle parameter 의 JOB_QUEUE_PROCESSES 가 0 보다 커야 JOB 스케줄링이 작동된다.

DBMS_JOB 패키지의 프로시저

```
DBMS_JOB.submit : job 등록
DBMS_JOB.remove : job 제거
DBMS_JOB.change : job 변경
DBMS_JOB.next_date : job 의 다음 수행시간 변경
DBMS_JOB.interval : job 의 다음 실행 cycle 지정
DBMS_JOB.what : job 의 수행으로 등록된 object 를 변경
DBMS_JOB.run : job 을 수동으로 실행
```

4) 추가 사항

이러한 패키지, 프로시저, 펑션들은 작성한 상태에서 바로 사용할 수 있는 것은 아니다.
PL/SQL 블록으로 작성되어 있는 구문을 컴파일이라는 과정을 거쳐야 실행할 수 있는 상태가 되며, 다음과 같이 실행하면 된다.

```
프로시저 컴파일
Alter procedure TEST_PROCEDURE compile;
함수(function) 컴파일
Alter function TEST_FUNCTION compile;
패키지 선언부와 BODY 를 컴파일
Alter package [사용자.]TEST_PACKAGE compile [PACKAGE | BODY];
PACKAGE : 패키지 선언부와 BODY 를 컴파일
BODY : 패키지의 BODY 만 컴파일
```

4. 자동화된 통계갱신을 위한 작업절차

(별첨 되어 있는 Update_stats.sql 을 sqlplus 에서 실행하여 프로시저를 생성한다.)

빈번한 편집이 발생하는 테이블에 대해서 최신의 통계정보를 유지해야 하며 이를 자동으로 처리하기 위해 DBMS_JOB 을 이용해서 처리하는 방법을 제시한다.

만일 DBMS_JOB 를 사용하고자 할 때에는 초기화 파라메터 중 JOB_QUEUE_PROCESSES 의 값을 충분하게 설정해주어야 한다. (예 JOB_QUEUE_PROCESSES=10)

● Oracle 8i 의 경우

$ORACLE_BASE\admin\\pfile\init.ora 파일을 편집하여 다음 파라메터를 변경 JOB_QUEUE_PROCESSES=10

● Oracle 9i 의 경우

C:\sqlplus "/ as sysdba"

SQL> alter system

Set job_queue_processes=10

Scope=both;

● Oracle 10g, 11g 의 경우 (변경할 필요는 없으나 확인 필요)

Default 설치 시 각각 10(10g), 1000(11g)로 설정되어 있음

C:\sqlplus "/ as sysdba"

SQL> show parameter job_queue_prosses;

다음은 DBMS_JOB 패키지를 이용하는 새로운 JOB 을 생성하는 과정이다.

1) JOB 을 생성하기 위한 권한 설정

JOB 을 생성하기 위해서는 DBMS_JOB 패키지에 대한 실행 권한이 있어야 한다. JOB 은 SDE 계정의 사용자 JOB 으로 생성할 것이기 때문에 SDE 계정에 대해 DBMS_JOB 패키지를 실행할 수 있는 권한을 부여해야 한다. 사용자에 대한 권한 여부는 sysdba 권한으로 수행해야 한다.

```
C:\ sqlplus "/ as sysdba"
SQL> GRANT EXECUTE ON SYS.DBMS_JOB TO SDE;
SQL> GRANT ANALYZE ANY TO SDE;
SQL> GRANT DELETE ANY TABLE TO SDE;
```

2) 통계 갱신을 위한 프로시저 생성

별첨에 첨부된 내용의 파일에 있는 sql 문을 실행한다.

```
SQL> connect sde/sde
SQL> @Update_states.sql
```

3) JOB 생성 및 스케줄링

JOB 을 생성하기 위해 SYS.DBMS_JOB 패키지를 사용한다. 다음과 같이 DBMS_JOB 을 생성하며, 이 때 일정한 시간 간격으로 자동실행 되도록 설정할 수 있다.

아래의 예는 1 시간 간격으로 실행하는 예이다. 변경이 필요할 때에는 next_date, interval 부분을 수정한다. (예 부분을 "12"로 치환)

```
Begin
DBMS_JOB.ISUBMIT
(job → 12,
what → 'update_stats;',
next_date → trunk(sysdate +1),
Interval → 'sysdate + 1,
No_parse→ FALSE);
Commit;
End;
/
```

SQL 문장으로 실행 (위의 내용을 수동으로 변경 하거나 다음 내용 중 한가지만 선택)

```
DECLARE
X NUMBER;
BEGIN
SYS.DBMS_JOB.SUBMIT
(
job => X
,what => 'SYS.DBMS_JOB.ISUBMIT
(12 /* BINARY_INTEGER */ ,
''update_stats;'' /* VARCHAR2 */ ,
TRUNC(SYSDATE+1) /* DATE */ ,
''SYSDATE+1'' /* VARCHAR2 */ ,
FALSE /* PL/SQL BOOLEAN */ );'
,next_date => to_date('08-04-2010 14:32:21','mm/dd/yyyy hh24:mi:ss')
```

```
,interval => 'TRUNC(SYSDATE+1)'
,no_parse => TRUE
);
:JobNumber := to_char(X);
END;
```

※ JOB 상세

기타 명령어

```
JOB 실행 유무 확인
SQL> SELECT * FROM USER_JOBS;
JOB 의 삭제
SQL> EXEC DBMS_JOB.REMOVE(JOBNO);
JOB 의 DISABLE
SQL> EXEC DBMS_JOB.BROKEN(JOBNO, TRUE);
JOB 의 강제 실행
SQL> EXEC DBMS_JOB.RUN(JOBNO);
USER_JOB view 에서 등록된 JOB 에 관한 상세한 정보 및 처리예상 시간 확인
SQL> SELECT job, next_date, interval, failures, broken FROM user_jobs;
SQL> SELECT job,
To_char(LAST_DATE, 'DD-MON-YYYY, HH24:MI')LAST_DATE,
To_char(THIS_DATE, 'DD-MON-YYYY, HH24:MI')THIS_DATE,
To_char(NEXT_DATE, 'DD-MON-YYYY, HH24:MI')NEXT_DATE
From user_jobs;
Interval 시간 지정 예제
1 분에 한번
SYSDATE + 1/24/60
매일 새벽 2 시
TRUNC(SYSDATE)+1+2/24 -> 다음날 새벽 2 시를 지정
매일 밤 11 시
TRUNC(SYSDATE)+23/24 -> 오늘밤 11 시를 지정
등록된 JOB 삭제
```

```
DELETE FROM USER_JOBS;
WHERE JOB IN(12);
COMMIT;
```

별첨 : 프로시저 생성을 위한 Update_stats.sql 내용

```
CREATE OR REPLACE PROCEDURE UPDATE_STATS IS
str varchar2(200);
recno_a NUMBER;
recno_d NUMBER;
CURSOR C1 IS
SELECT B.OWNER || '.A' ||A.REGISTRATION_ID as tabA, B.OWNER || '.D'
||A.REGISTRATION_ID as tabD
FROM (SELECT REGISTRATION_ID FROM SDE.MVTABLES_MODIFIED GROUP BY
REGISTRATION_ID ) A,SDE.TABLE_REGISTRY B
WHERE A.REGISTRATION_ID = B.REGISTRATION_ID;
BEGIN
FOR tlist IN C1 LOOP
str := 'select count(*) from ' || tlist.tabA;
EXECUTE IMMEDIATE str into recno_a;
str := 'select count(*) from ' || tlist.tabD;
EXECUTE IMMEDIATE str into recno_d;
IF recno_a <> 0 AND recno_d <> 0 THEN
str := 'ANALYZE TABLE ' ||tlist.tabA ||' COMPUTE STATISTICS';
EXECUTE immediate str;
str := 'ANALYZE TABLE ' ||tlist.tabD ||' COMPUTE STATISTICS';
EXECUTE immediate str;
ELSIF recno_a = 0 AND recno_d <> 0 THEN
str := 'ANALYZE TABLE ' ||tlist.tabD ||' COMPUTE STATISTICS';
EXECUTE immediate str;
ELSIF recno_a <> 0 AND recno_d = 0 THEN
str := 'ANALYZE TABLE ' ||tlist.tabA ||' COMPUTE STATISTICS';
```

```
EXECUTE immediate str;
END IF;
END LOOP;
EXECUTE IMMEDIATE 'ANALYZE TABLE sde.state_lineages COMPUTE STATISTICS';
EXECUTE IMMEDIATE 'ANALYZE TABLE sde.states COMPUTE STATISTICS';
EXECUTE IMMEDIATE 'ANALYZE TABLE sde.mvtables_modified COMPUTE
STATISTICS';
END;
/
```