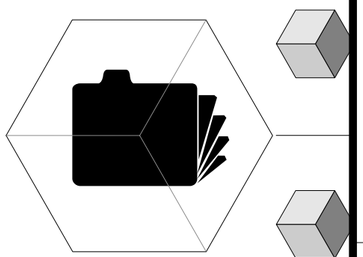


성능 뷰와 딕셔너리를 이용한 성능 저하 요인 찾기

지난 3회 동안 주로 애플리케이션 프로그래밍 측면, 즉 SQL문의 성능 저하 요인을 알아보았다. 프로젝트에서 일어나는 문제는 문법이 복잡다단하게 어려운 SQL문이 아니라 데이터 건수가 많은 것에 대한 경험 부족인 경우가 많다. 그러므로 지난 호에 기술한 10가지 정도의 유형에 맞춰 스스로 사례 분석을 해보기 바란다. 이번 호에서는 오라클에서 제공하는 성능 뷰와 딕셔너리를 이용해 문제를 일으키는 요인을 찾는 연습을 하자.

개발자들은 일반적으로 오라클의 딕셔너리 구조에 대해 친숙하지 않다. 오라클은 v$로 시작하는 성능 뷰와 DBA_, ALL_, USERS_로 시작하는 딕셔너리를 가지고 있다. 여기에 대한 정보를 파악하고 어떻게 이용하느냐에 따라 여러분의 작업 환경은 많이 개선될 것이다.

중요 딕셔너리

〈표 1〉은 개발자들도 꼭 알아두었으면 하는 성능 뷰와 자주 참조하는 딕셔너리 목록이다. DICTIONARY, DICT_COLUMNS를 조회하면 용도가 자세히 나온다. 〈표 1〉에 나오는 내용을 외울 필요는 없다. naming rule이 눈에 들어오면 자연스럽게 외워진다. 수시로 SQLPLUS상에서 DESC하여 해당 컬럼 구성을 눈에 익히고 오늘 기술할 성능 관련 SQL 스크립트에서 사용되는 컬럼은 레퍼런스 책을 통해 의미를 꼭 알아두기 바란다.

세션별로 문제 원인 찾기

지난 시간에도 간략히 언급한 바 있는데, 자신이 작성한 프로그램을 수행시켰을 때 예상 외로 시간이 많이 걸리고 있는데도 원인 찾기를 포기한 채 DBA에게 의뢰하거나 끝날 때까지 기다리는 개발자를 많이 보아왔다. 내 프로그램이 잘 수행되고 있는지 알고 싶을 때, 내 프로그램이 어느 SQL문을 수행하고 있는지 알고 싶을 때, 내 프로그램이 수행하고 있는 세션을 중단하고 싶을 때 등 일일이 DBA의 도움을 청하기도 미안하다. 자신이 작성하고 수행하는 프로그램에 대해서는 다음과 같은 절차로 원인 분석을 해 보길 바란다.

박성우 | dont4get@chol.com

현엉씨스템즈와 LG-CNS를 거쳐 현재는 수로텍 연구소 팀장으로 일하고 있다. 대형양 화메뉴류 개발, KT-ICS 요금관리시스템 프로젝트 DBA, KT-차세대 빌링 프로젝트 DBA, 통합 하수관거 유지관리 SW 개발 팀장으로 활동했다.

OS 상의 점검

1 top, glance, sar 명령을 통한 CPU 및 메모리 점유율 확인

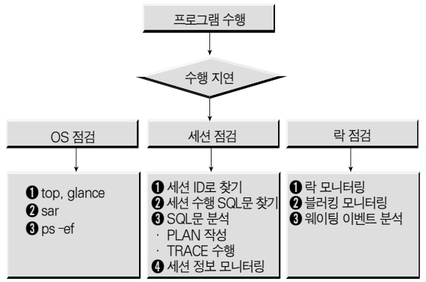
일단 프로그램의 수행이 느린 경우 OS나 DBMS에 대한 부하나 이상으로 인해 느린지 아니면 해당 애플리케이션 때문에 느린지를 빨리 판별하는 것이 중요하다. 간단한 명령으로 top, glance, sar 명령 등을 통해 서버의 부하가 많은지를 판별할 수 있다.

2 ps -ef|grep을 이용

오라클 관련 프로그램은 반드시 대응되는 오라클 서버 프로세스가 있다. MTS(Multi Thread Server) 환경으로 설정되어 있지 않다면 프로그램과 오라클 서버 프로세스는 대부분 1:1이다. 만약 리스너(Listener)를 통하지 않았다면 프로그램 ID를 가지고 오라클 서버 프로세스를 간단히 찾을 수 있다. SQLPLUS를 실행시킨 다음 SQLPLUS에서 ! 명령으로 OS로 빠져와서 〈화면 2〉와 같이 실행시켜 보자.

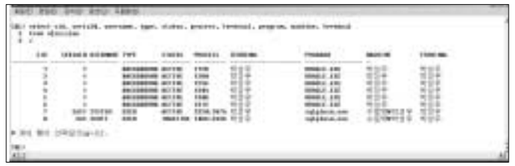
〈화면 2〉의 ⓐ와 같이 SQLPLUS의 프로세스 ID 24463을 가지고 ps -ef 명령을 수행하면 24463을 모 프로세스로 하는 오라클 서버 프

<그림 1> 세션별 점검 절차



<화면 1> OS 명령어를 통한 서버 현황 모니터링 화면

<화면 2> ps -ef|grep을 이용한 오라클 서버 프로세스 조회 화면



<화면 3> V$SESSION을 통해 세션정보 조회 화면

<화면 4> V$PROCESS를 통해 오라클 서버 프로세스 찾는 화면

로세스가 24466으로 나타난다. 이 세션에서 DB와 관련된 작업이 이루어진다면 ⓒ부분에 CPU가 증가되는 것이 보인다. 만약 이 CPU 변화량이 낮다면 시스템 자원을 제대로 활용을 못하거나 락에 의해 수행되지 못한다는 것이고 CPU 변화량이 많음에도 불구하고 프로그램이 느리다면 이는 잘못된 액세스 경로로 SQL문이 수행되고 있을 소지가 많다. 만약 SQLPLUS를 많은 유저에서 수행하였다면 OS 유저나 ⓑ에서처럼 터미널 번호를 가지고 해당 프로그램을 찾으면 된다.

기억할 것이 또 하나 있다. ⓒ부분을 보면 오라클 서버 프로세스에 LOCAL=YES라고 되어 있는 부분이 있다. 이는 해당 로컬 서버에서 리스너를 통하지 않고 직접 연결된 것이다. 만약 사용자 PC나 다른 서버에서 접속하거나 '@서비스 명'을 사용한다면 LOCAL=NO라고 나타나며 상위 프로세스 번호가 1로 보일 것이다. 이런 경우에는 오라클 딕셔너리 정보를 찾아 오라클 서버 프로세스 번호를 찾아내야 한다. 그리고 LOCAL=YES인데 상위 프로세스 번호가 1이라면 이는 비정상적으로 연결이 끊어진 프로세스이다. 이런 프로세스가 락을 잡고 다른 프로세스가 수행하는 것을 방해할 수 있으니 상태 확인 후 OS 명령 'kill'을 사용해서 삭제해야 한다.

세션 점검

❶ 세션 ID 찾기

여러분이 만약 세션에 대한 정보를 찾고자 한다면 세션에 관련된 ID, 즉 SID(System IDentifier), SERIAL#를 먼저 알아야 관련된 정보를 찾기 편하다. 세션의 기본 데이터는 V$SESSION이라는 성능 뷰를 통해 알 수 있다. V$SESSION에 대한 세부 내역은 오라클의 USER's REFERENCE 또는 DICT_COLS를 통해 확인하기 바란다. 중요 컬럼은 〈화면 3〉과 같다. 권한 문제로 해당 예제가 실행되지 않을 수 있다. 해당 DBA에게 SELECT ANY TABLE 권한을 임시로 부여해달라고 요청하기 바란다. 되도록이면 PC라도 좋으니 자기가 맘대로 환경을 구현하는 환경이면 좋겠다. 이런 조그마한 SQL문들 때문에 시스템이나 DB가 멈추진 않지만 괜히 의심받기 때문이다.

〈화면 3〉에서 프로세스 번호, 프로그램명, 실행 서버, 터미널, 접속 사용자 등의 정보를 통해 해당 세션의 SID를 구할 수 있을 것이다. 만약 프로그램이 강제 킬(kill)되어 롤백중이라면 Serial#의 번호가 계속 바뀔 것이다. 그 다음 오라클 서버 프로세스 번호를 알고 싶다면 V$PROCESS를 이용하면 된다. 데이터베이스 프로그램이나 오라클 서버 프로세스를 OS 차원에서 강제로 킬시키는 일은 되도록 자제해야 한다. 〈화면 3〉에서처럼 SID, SERIAL#을 구한 다음 ALTER SYSTEM KILL SESSION 'SID, SERIAL#'; 명령어로 정상적으로 종료시키는 것이 좋다.

주의할 것은 V$SESSION과 V$PROCESS의 연결 고리는 V$SESSION의 PADDR과 V$PROCESS의 ADDR이라는 것이다. 〈화면 4〉에서 왼쪽의 PROCESS는 수행한 프로그램 자체의 프로세스 번호를 뜻하며, SPID는 이와 연결된 오라클 서버 프로세스의 번호이다. 만약 TKPROF를 수행하기 위해 SQL_TRACE 옵션을 수행하였다면 udump 디렉토리에 trace 파일이 쌓이는데 이 오라클 서버

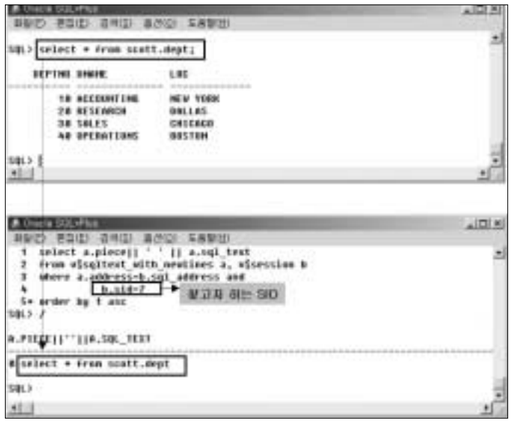
<표 1> 성능 뷰 및 딕셔너리 분류

분류	성능 뷰/딕셔너리	딕셔너리
세션과 관련된 정보	V$SESSION	세션에 대한 전반적인 정보를 보여준다.
	V$SESSSTAT	세션의 현황에 대한 통계정보를 보여준다.
	V$SESSION_WAIT	세션의 waiting 통계정보를 보여준다.
	V$SESSION_EVENT	세션의 현재 waiting 이벤트를 보여준다.
	V$SESS_IO	세션의 IO 현황을 보여준다.
	V$STATNAME	SESSSTAT의 STATUS의 이름을 보여준다.
성능 관련 정보	V$SYSTAT	시스템 전반의 성능 통계정보를 보여준다.
	V$SYSTEM_EVENT	시스템의 waiting 이벤트별 통계정보를 보여준다.
	V$LIBRARYCACHE	라이브러리 캐시 사용정보를 보여준다.
	V$ROWCACHE	데이터 딕셔너리의 사용정보를 보여준다.
	V$LATCH	LATCH에 대한 정보를 보여준다.
	V$LOCK	락에 대한 정보를 보여준다.
	V$LOCKED_OBJECT	락이 걸린 오브젝트에 대한 정보를 보여준다.
	V$SQLAREA	SQLAREA에 대한 정보를 보여준다.
	V$WAITSTAT	시스템의 현재 waiting 현황을 보여준다.
SQL 관련	V$SQL	parse된 SQL 문장을 보여줌
	V$SQLTEXT	라인별로 SQL 문장을 보여줌
	V$SQLTEXT_WITH_NEWLINESS	Newline 을 포함하여 SQL 문장을 보여준다.
시스템 구성정보	V$SGA	SGA의 정보를 보여준다.
	V$PARAMETER	InitSID.ora 등에서 설정된 파라미터, 즉 데이터베이스가 구동되었을 때의 환경 파라미터 정보이다.
	V$CONTROLFILE	컨트롤 파일에 대한 정보를 보여준다.
	V$DATAFILE	데이터 파일에 대한 정보를 보여준다.
	VLOG, VLOGFILE	redo 로그에 대한 정보를 보여준다.
사용자 권한	DBA_USERS	데이터베이스 사용자에 대한 정보를 보여준다.
	DBA_ROLES	롤에 대한 정보를 보여준다.
	DBA_TAB_PRIVS	테이블에 대한 권한이 설정된 정보를 보여준다.
	DBA_SYS_PRIVS	시스템 권한이 설정된 정보를 보여준다.
	DBA_ROLE_PRIVS	롤에 대한 권한이 설정된 정보를 보여준다.
	DBA_COL_PRIVS	컬럼 단위로 권한이 설정된 정보를 보여준다.
세그먼트&오브젝트	DBA_SEGMENTS	세그먼트(저장 공간이 있는 오브젝트)에 대한 정보를 보여준다.
	DBA_OBJECTS	모든 오브젝트에 대한 정보를 보여준다.
테이블 스페이스	V$TABLESPACE	테이블스페이스에 대한 정보를 보여준다.
	DBA_TABLESPACES	테이블스페이스에 대한 정보를 보여준다.
	DBA_DATA_FILES	테이블스페이스를 구성하고 있는 데이터 파일에 대한 정보를 보여준다.
	DBA_FREE_SPACE	아직 사용되지 않은 영역에 대한 정보를 보여준다.
	DBA_EXTENTS	할당된 익스텐트의 정보를 보여준다.
	DBA_TS_QUOTAS	QUOTA가 설정된 정보를 보여준다.
테이블	DBA_TABLES	테이블에 대한 정보를 보여준다.
	DBA_TAB_COLUMNS	테이블을 구성하는 컬럼에 대한 정보를 보여준다.
	DBA_TAB_COMMENTS	테이블의 설명에 대한 정보를 보여준다.
	DBA_PART_TABLES	파티션 테이블에 대한 정보를 보여준다.
	DBA_PART_KEY_COLUMNS	파티션을 구성하는 기준 컬럼에 대한 정보를 보여준다.
	DBA_COL_COMMENTS	컬럼의 설명에 대한 정보를 보여준다.
인덱스	DBA_INDEXES	인덱스에 대한 정보를 보여준다.
	DBA_PART_INDEXES	파티션된 인덱스에 대한 정보를 보여준다.
	DBA_IND_COLUMNS	인덱스를 구성하는 컬럼에 대한 정보를 보여준다.
CONSTRAINT	DBA_CONSTRAINTS	테이블에 걸려있는 제약조건을 보여준다.
	DBA_CONS_COLUMNS	제약 조건을 구성하는 컬럼에 대한 조건을 보여준다.
뷰	DBA_VIEWS	뷰를 정의한 정보를 보여준다.
시노님	DBA_SYNONYMS	시노님에 대한 정보를 보여준다.
시퀀스	DBA_SEQUENCES	시퀀스에 대한 정보를 보여준다.
DB LINK	DBA_DB_LINKS	DB 링크에 대한 정의를 보여준다.
트리거	DBA_TRIGGERS	트리거에 대한 정의를 보여준다.
	DBA_TRIGGER_COLS	컬럼 단위로 작성된 트리거에 대한 정의를 보여준다.
ROLLBACK	DBA_ROLLBACK_SEGS	롤백 세그먼트에 대한 정보를 보여준다.
FUNCTION, PROCEDURE, PACKAGE	DBA_SOURCE	FUNCTION, PROCEDURE,PACKAGE를 구성하는 PL/SQL 소스 코드를 보여준다.

프로세스의 번호가 이름에 포함된다. 이를 통해 tkprof를 수행시킬 대상 파일임을 알 수 있다.

2 세션에서 수행 중인 SQL문 찾기

프로그램이 느리다고 생각한 경우 어느 부분에서 느린지 몰라 답답해 하는 사람들이 많다. 물론 요즈음은 국산으로 개발된 DB 모니터링 툴이 많지만 개발자들에게까지 이 툴을 사용하게 할 만큼 충분한 라이선스를 구매해 주는 프로젝트는 없다. 1에서 SID를 구했다면 여러분은 간단한 SQL 문장을 통해 해당 세션에서 수행되고 있는 SQL문을 조회할 수 있다.



<화면 5> 세션에서 수행된 SQL문 찾는 화면

<화면 6> 세션 통계정보 조회 화면

〈화면 5〉와 같이 SQL 찾는 SQL문을 계속 반복해서 수행해 보면 현재 문제를 일으키는 SQL문을 금방 조회할 수 있다. 권한 문제로 실행을 못하는 독자는 DBA 계정을 가진 이에게 grant select on V_$SQLTEXT_WITH_NEWLINES to scott;와 같이 권한을 할당해 줄 것을 요청하기 바란다.

3 SQL문 분석

2번에서 문제가 될만한 SQL문을 찾았다면 EXPLAIN 문장을 써서 해당 SQL문을 PLAN 작성해 분석해야 한다. 플랜 작성은 지난 3회 동안 많은 지면을 할애해 설명했으니 다시 상기해 보자.

- SQLPLUS의 오토트레이스 기능 활용
- EXPLAIN 명령과 PLAN 테이블 조회
- tkprof 유틸리티 활용 방법

만약 아직도 이 방법에 대해 모른다면 튜닝은 할 생각도 말아야 한다. 기초 중의 기초임을 다시 한번 강조한다. tkprof를 활용해 플랜을 작성하고 싶으나 SQL_TRACE=TRUE 옵션없이 수행되는 프로그램에 대해서도 트레이스(TRACE)를 작성할 수 있는 방법이 있다.

```
EXPLAIN dbms_session.set_sql_trace_in_session(sid, serial#, true);
```

TRACE 파일이 어디 쌓이는지 모르는 독자가 있다면 USER_DUMP_DEST에 지정된 파일 경로를 찾는다(USER_DUMP_DEST는 V$PARAMETER를 조회하면 된다).

4 세션 모니터링

아마 SQL문으로 인한 잘못이었다면 3까지의 과정을 거쳐 지금 현재 어느 SQL문이 문제를 일으키는지를 파악할 수 있을 것이다. 그렇다면 이 SQL문이 CPU를 과도하게 수행하는지, I/O를 과도하게 일으키는지는 세션을 모니터링하는 정보를 가지고 파악할 수 있다. 이는 문제가 발생할 때마다 경우의 수가 많아 딱 집어 설명하기가 어렵다. 서버 튜닝에 대한 책자와 웹 사이트를 통해 자료를 스스로 찾아보기 바란다. 단지 이번 호에는 관련 스크립트를 보여줄테니 성능이 의심스러울 경우 수행하여 그때 나타나는 현상을 잘 정리해두기 바란다.

◆ 세션 통계정보

해당 세션의 통계정보를 보여주는 스크립트이다. 각 세션별로 CPU, 메모리 등의 전반적인 상태를 모니터링할 수 있다(〈화면 6〉).

```
col name for a50
select a.sid,b.name,a.value from v$sesstat a,v$statname b
where a.statistic#=b.statistic# and
a.sid = &SID
order by 1
```

◆ 세션 이벤트 통계정보

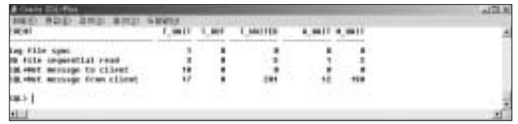
각 세션에 발생된 이벤트에 대한 통계치를 누적하여 보여준다

```
col event      format a30
col t_wait     format 999999
col t_out      format 99999
col t_waitd    format 99999
col m_wait     format 99999
select     event,
           total_waits t_wait,
           total_timeouts t_out,
           time_waited t_waited,
           average_wait a_wait,
           max_wait m_wait
from v$session_event
where sid = &SID
```

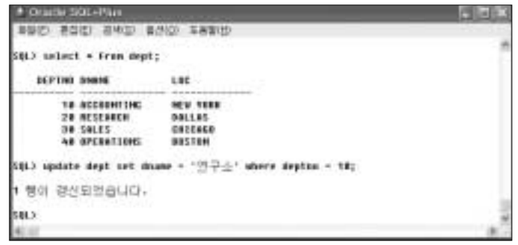
◆ 세션 이벤트 wait 정보

세션이 지금 waiting하고 있는 상태를 보여준다(〈화면 11〉).

```
col p1text format a10
col p2text format a10
col p3text format a10
col e_name format a20
col p3 format 999
select
   SID,
   substr(EVENT, 1, 20) e_name,
   P1,
   P1RAW,
   P3TEXT,
   P3
from v$session_wait
WHERE SID=&SID
```



<화면 7> 세션 이벤트 통계정보



<화면 8> 업데이트로 락거는 화면

<화면 9> 락걸린 오브젝트 조회 화면

락 모니터링

1 락걸린 테이블 찾기

데이터베이스는 데이터의 무결성을 보장하기 위해서 락이라는 개념을 사용한다. 락을 가장 일반적으로 설명하면 같은 데이터를 동시에 수정이 불가하다고 정의를 내리는 것이 이해하기가 편하리라 본다. 데이터를 생성, 수정, 삭제하는 프로그램이 CPU나 메모리 사용률이 증가하지 않고 I/O도 증가하지 않는다면 제일 먼저 락을 의심해 보는 것이 좋다. 다른 프로세스가 같은 데이터를 조작하고 있는 것이다. 개발자들은 막연히 어딘가 락이 걸려 있다고 의심해 보지만 모니터링할지 몰라 DBA를 찾는 경우가 많다. 그래서 큰 목소리로 '혹시 어느 테이블 수정하고 있는 사람 있나요?' 라고 다른 개발자들을 향해 소리치곤 한다. 자신의 PC에 다른 화면에서 커밋이나 롤백을 하지 않고 있음에도 불구하고.

또한 프로그램이 비정상적으로 종료하여 롤백하는 동안에도 락이 걸리기 때문에 ps -ef로 못 찾는 경우도 종종 있다. 다음의 예로 락을 찾아보자. 두 개의 화면을 열고 모니터링해 보도록 하자. 한 화면에서 업데이트를 해 본다.

그 다음 다른 화면에서 같은 SQL문을 수행해 보자. 여기서 필자는 첫 번째는 SCOTT 유저로 로긴해서 하였고 다음 다른 화면은 SYSTEM으로 로긴하여 UPDATE SCOTT.DEPT SET DNAME = '연구소' WHERE DEPTNO = 10; 을 수행했다.

<화면 10> 락으로 인한 블러킹 정보 조회 화면

〈화면 8〉에서 커밋이나 롤백을 하지 않았기 때문에 분명 두번째 화면은 실행되지 않을 것이다. 〈화면 9〉와 같이 실행시켜 락걸린 오브젝트를 찾아보자.

2 락으로 인해 수행되지 못하는 블러킹 세션 모니터링

〈화면 9〉의 SQL문은 락이 걸린 오브젝트를 쉽게 찾아 준다. 하지만 이 SQL문도 어느 세션이 다른 세션을 수행하지 못하도록 블러킹하고 있는지 판단을 내릴 수 있는 정보를 주지 않는다. 블러킹을 하는 세션이 어느 세션인지 알려면 〈화면 10〉의 SQL문을 실행해 주면 된다. 지난 호에도 설명했던 중요한 SQL문이다. 반드시 기억해두기 바란다.

<화면 11> 락으로인한 wait 현상 조회 화면

〈화면 10〉처럼 MODE_HELD가 익스클루시브 모드인 SID 8번의 프로세스가 다른 프로세스를 수행 못하도록 블러킹하고 있다고 보면 된다. 반대로 MODE_REQUEST가 익스클루시브인 SID 7번은 락이 풀리기를 기다리는 프로세스이다. 자 V$SESSION 을 이용하여 해당 정보를 모니터링해 보자.

앞에서 수행했던 SQL문을 응용하여 〈화면 11〉과 같이 수행하였다. 현재 SID 7번이 enqueue 이벤트를 일으키고 있으며 락이 풀리기를 기다리는 액티브한 상태이다. Enqueue 이벤트는 락을 기다리는 전형적인 이벤트이다. 락은 단지 트랜잭션간의 데이터 조작 뿐만 아니라 DDL문, 과도한 익스텐트 발생으로 인한 락 등 다양한 형태를 지니고 있다. 이런 다양한 형태의 락은 〈화면 10〉의 SQL문에서 LOCK_TYPE란을 조회해 본다면 원인이 무엇인지 쉽게 알 수 있다. 물론 용어가 아직 낯설 것이지만 TX, TM, ST, TS가 대부분 발생한다.

3 waiting 이벤트 분석

〈화면 10〉에서 수행했던 SQL문에서 보듯 v$session_event의 이벤트는 현재 작업이 발생되고 있는 행위를 나타내준다. 반드시 wait 이벤트가 일어난다고 해서 느린 것은 아니다. 하지만 원인 분석에 많은 도움을 준다. 이에 해당하는 이벤트를 모두 기술하자면 상당한 분량이 될 것이다. 〈표 2〉를 통해 개발자들이 알아두어야 할 이벤트에 대해 간단히 알아보자.

〈표 2〉 주요 waiting 이벤트

이벤트명	상태	대처 방안
SQL*Net message from client	Idle 상태를 의미하며 클라이언트로부터 작업 요구를 기다림을 의미한다.	
SQL*Net message to client	클라이언트로 작업 결과를 내보내고 있음을 의미	
db file scattered read	테이블의 인덱스 스캔시 나타남	
db file sequential read	테이블 풀 스캔시 나타남	
enqueue	트랜잭션간의 락킹 현상, 익스텐트 과도 발생 등으로 인하여 발생	프로세스의 간섭 현상 최소화, 적절한 익스텐트 할당 등
Latch free	주로 SGA에 대한 자원 경합이 발생할 경우 발생	SQL문의 표준화, BIND 변수 사용 등
Buffer cache waits	여러 세션이 동시에 같은 블럭의 데이터를 읽거나 변경하고자 할 때 주로 발생	· SQL문 튜닝을 통해 불필요 데이터를 읽지 않도록 함 · 블럭당 rows 수를 줄이도록 PCTFREE, PCTUSED 조정 · undo에 대한 경합은 롤백 세그먼트의 개수와 크기를 증가시킬 것
Free buffer waits, write complete waits	DBWR의 백그라운드 프로세스가 작성하는 대기 시간	Async-IO 또는 멀티 DBWR를 사용(Async-IO를 추천)
Redo buffer space	redo 로그를 할당하기 위해 대기중인 이벤트	로그 버퍼의 크기 증가
Redo file parallel write	LGWR이 redo 버퍼를 redo 로그 파일에 기록하도록 요청한 이벤트	redo 로그 발생량을 최대로 줄이고 디스크를 분산시킴

데이터베이스 전체 모니터링

지금까지 세션 단위로 모니터링하면서 문제점을 발견해 나가는 연습을 했다. 이제 내친김에 전체적으로 모니터링해 보자. 이것은 서버 튜닝의 맛배기로 생각하고 편안하게 즐기기 바란다. 그리고 서버 튜닝에 대한 자세한 내용은 책자를 읽으면서 각자 심도있게 공부하기 바란다. 사실 이 영역은 개발자의 영역을 벗어난다. 하지만 요즘 개발자가 프로그램만 코딩하고 설계하란 법이 있는가? 기왕 여기까지 온 이상 서버 튜닝 영역에도 도전해 보자. 서버 튜닝은 자신이 환경을 구축하고 마음대로 설정할 수 있는 환경이 필요하다. 인스톨부터 해 보는 것이 좋다.

다음은 서버 튜닝이라기보다 데이터베이스의 현황을 전반적으로 점검해 보는 스크립트를 살펴본다. 필자가 다니는 회사 홈 페이지(www.surotech.com) 자료실에 올려놓을 것이니 다운받아 필요할 때 사용하기 바란다. 사용목적에 따른 SQL문을 소개한다.

◆ 현재 클로즈 안 된 커서가 액세스하고 있는 오브젝트의 정보를 얻고자 하는 경우,
현재 사용되고 있는 오브젝트를 알고자 할 경우

```
select sid,
owner,
object,
type
from v$access
order by 1, 2, 3, 4;
```

◆ 데이터 딕셔너리의 상세 현황을 보고자 할 경우

```
select cache#,
type,
subordinate#,
parameter,
count,
usage,
fixed,
gets,
getmisses "Get Misses",
scans,
scanmisses "Scan Misses",
scancompletes "Scan Completes",
modifications,
flushes
from v$rowcache;
```

◆ 데이터 딕셔너리의 요약 현황을 보고자 할 경우

```
select sum(count) Count,
sum(usage) Usage,
sum(fixed) Fixed,
sum(gets) Gets,
sum(getmisses) "Get Misses",
sum(scans) Scans,
sum(scanmisses) "Scan Misses",
sum(scancompletes) "Scan Completes",
sum(modifications) Modifications,
sum(flushes) Flushes,
from v$rowcache;

select round(sum(gets)/(sum(gets)+sum(getmisses)) * 100, 2)
from v$rowcache;
```

◆ DB 블럭 버퍼에서 읽혀진 횟수를 보고자 할 경우

```
# Returns a count of gets in the db block buffer.
select sum(value)
from v$sysstat
where name in ('db block gets', 'consistent gets');
```

◆ DB 블럭 버퍼의 활용도를 보고자 하는 경우

```
select round((1-(pr.value/(bg.value+cg.value)))*100,2)
from v$sysstat pr, v$sysstat bg, v$sysstat cg
where pr.name = 'physical reads'
and bg.name = 'db block gets'
and cg.name = 'consistent gets';
```

◆ DB 블럭 버퍼의 전반적인 리포팅

```
column phys_read heading "Physical|Reads" format 99999999990
column block_get heading "Block|Gets" format 99999999990
column consi_get heading "Consistent|Gets" format 99999999990
column bchr heading "BCHR" format 999.90
select pr.value phys_read, bg.value block_get, cg.value consi_get,
(1 - ( pr.value/(bg.value+cg.value) ) ) * 100 bchr
from v$sysstat pr, v$sysstat bg, v$sysstat cg
where pr.name = 'physical reads'
and bg.name = 'db block gets'
and cg.name = 'consistent gets';
```

◆ DB 블럭의 사용 현황을 요약하고자 할 경우

```
select decode(state, 0, 'Free', 1, 'Read and Modified', 2, 'Read and Not Modified',
3, 'Currently Being Read', 'Other'), count(*)
```

```
from x$bh
group by decode(state, 0, 'Free', 1, 'Read and Modified', 2, 'Read and Not
Modified', 3, 'Currently Being Read', 'Other');
```

◆ 디스크로부터 가장 많이 읽혀지는 SQL 문장을 알고 싶은 경우

```
select sql_text
from v$sqlarea, v$session
where address = sql_address
and username is not null
and disk_reads/executions = (select max(disk_reads/executions)
from v$sqlarea, v$session
where address = sql_address
and username is not null
and executions > 0);
```

◆ 버퍼에서 가장 많이 읽혀지는 SQL 문장을 알고 싶은 경우

```
select sql_text
from v$sqlarea, v$session
where address = sql_address
and username is not null
and buffer_gets/executions = (select max(buffer_gets/executions)
from v$sqlarea, v$session
where address = sql_address
and username is not null);
```

◆ 익스텐트 현황을 알고 싶은 경우

```
select owner, segment_name, segment_type, count(*) numext,
round(sum(bytes)/1024/1024,1) MB
from sys.dba_extents
where owner not in ('SYS','SYSTEM')
group by segment_name, segment_type
order by segment_type, round(sum(bytes)/1024/1024,1) desc, segment_name;
```

◆ 익스텐트가 가장 많이 일어난 횟수

```
select max(extent_id) + 1
from sys.dba_extents
where owner not in ('SYS','SYSTEM');
```

◆ 익스텐트가 가장 많이 일어난 세그먼트

```
select owner, segment_name
from sys.dba_extents
where owner not in ('SYS','SYSTEM')
and extent_id = (select max(extent_id)
from sys.dba_extents
where owner not in ('SYS','SYSTEM'));
```

◆ 데이터 파일별 액세스 유형별 횟수

```
select name,
phyrds "Total Reads",
phywrts "Total Writes",
phyblkrd "Blocks Read",
phyblkwrt "Blocks Written"
from v$datafile d, v$filestat s
where d.file# = s.file#
order by d.file#;
```

◆ Free list wait 일어난 비율(낮을수록 좋음)

```
select round((sum(decode(w.class, 'free list',count, 0))
/ (sum(decode(name,'db block gets', value, 0))
+ sum(decode(name,'consistent gets', value, 0))))
* 100, 2)
from v$waitstat w, v$sysstat;
```

◆ 시스템 테이블스페이스 내 인덱스 생성 현황

```
select count(*)
from sys.dba_indexes i
where i.tablespace_name = 'SYSTEM'
and i.owner not in ('SYS','SYSTEM');
```

◆ 네트워크 부하(bytes)

```
select sum(value)
from v$sysstat
where name like 'bytes%SQL*Net%';
```

◆ 데이터 파일로부터 물리적 I/O 횟수

```
select sum(phyrds) + sum(phywrts) "Total I/O"
from v$filestat;
```

◆ I/O의 종합 현황

```
select sum(decode(name,'db block changes', value,0)) "Block Changes",
(sum(decode(name, 'db block gets', value,0))
+ sum(decode(name, 'consistent gets', value,0))) "Buffer Gets",
sum(decode(name,'physical reads', value, 0)) "Physical Reads",
(sum(decode(name, 'db block gets', value,0))
+ sum(decode(name, 'consistent gets', value,0)))
/ sum(decode(name,'physical reads', value, 0)) "Gets / Reads"
from v$sysstat;
```

◆ 래치(Latch)로 인한 경합률(0에 가까울수록 좋음)

```
select round(greatest(
(sum(decode(ln.name, 'cache buffers lru chain', misses,0))
/ greatest(sum(decode(ln.name, 'cache buffers lru chain', gets,0)),1)),
(sum(decode(ln.name, 'enqueues', misses,0))
/ greatest(sum(decode(ln.name, 'enqueues', gets,0)),1)),
(sum(decode(ln.name, 'redo allocation', misses,0))
/ greatest(sum(decode(ln.name, 'redo allocation', gets,0)),1)),
(sum(decode(ln.name, 'redo copy', misses,0))
/ greatest(sum(decode(ln.name, 'redo copy', gets,0)),1)))
* 100,2)
from v$latch l, v$latchname ln
where l.latch# = ln.latch#;
```

◆ 래치 상세 현황

```
select ln.name,
lh.pid,
l.immediate_gets,
l.immediate_misses,
l.gets,
l.misses,
l.sleeps
from v$latch l, v$latchholder lh, v$latchname ln
where l.latch# = ln.latch#
and l.addr = lh.laddr(+)
order by l.level#, l.latch#;
```

◆ 래치 효율성 평가(100에 가까울수록 좋음)

```
select round(((sum(l.immediate_gets) + sum(l.misses) + sum(l.gets))
/ (sum(l.immediate_gets) + sum(l.immediate_misses) + sum(l.gets) + sum(l.misses)))
* 100,2)
from v$latch l;
```

◆ 래치 종합 현황

```
select sum(l.immediate_gets),
sum(l.immediate_misses),
sum(l.gets),
sum(l.misses),
sum(l.sleeps)
from v$latch l, v$latchholder lh, v$latchname ln
where l.latch# = ln.latch#
and l.addr=lh.laddr(+);
```

◆ 라이브러리 캐시 효율성(100에 가까울수록 좋음)

```
select round(sum(pinhits)/sum(pins) * 100,2)
from v$librarycache;
```

◆ 라이브러리 캐시 상세 현황

```
select namespace name,
gets,
gethits,
round(gethitratio*100,2) "GetHit Percentage",
pins,
pinhits,
round(pinhitratio*100,2) "PinHit Percentage",
reloads,
invalidations
from v$librarycache
order by 1;
```

◆ 메모리 앨로케이트 현황

```
select sum(value)
from v$statname n, v$sesstat s
where n.statistic# = s.statistic#
and name = 'session uga memory';
```

```
select sum(value)
from v$statname n, v$sesstat s
where n.statistic# = s.statistic#
and name = 'session uga memory max';
```

◆ 오픈 트랜잭션 횟수

```
select sum(xacts) from v$rollstat;
```

◆ Parse 효율성

```
select round(sum(decode(name, 'opened cursors cumulative', value, 0))
/ sum(decode(name,'parse count', value,0)) * 100, 2)
from v$sysstat;
```

◆ Parse 현황

```
select ptc.value "Parse Time CPU",
pte.value "Parse Time Elapsed",
pc.value "Parse Count"
from v$sysstat ptc, v$sysstat pte, v$sysstat pc
where ptc.statistic#=96
and pte.statistic#=97
and pc.statistic#=98;
```

◆ 물리적 읽기 횟수

```
select sum(value)
from v$sysstat
where name = 'physical reads';
```

◆ 리커시브 콜 횟수

```
select value from v$sysstat where name = 'recursive calls';
```

◆ redo 로그 래치 경합

```
select round(greatest(
(sum(decode(ln.name, 'redo copy', misses,0))
/ greatest(sum(decode(ln.name, 'redo copy', gets,0)),1)),
(sum(decode(ln.name, 'redo allocation', misses,0))
/ greatest(sum(decode(ln.name, 'redo allocation', gets,0)),1)),
(sum(decode(ln.name, 'redo copy', immediate_misses,0))
/ greatest(sum(decode(ln.name, 'redo copy', immediate_gets,0))
+ sum(decode(ln.name, 'redo copy', immediate_misses,0)),1)),
(sum(decode(ln.name, 'redo allocation', immediate_misses,0))
/ greatest(sum(decode(ln.name, 'redo allocation', immediate_gets,0))
+ sum(decode(ln.name, 'redo allocation', immediate_misses,0)),1)))
* 100,2)
from v$latch l, v$latchname ln
where l.latch# = ln.latch#;
```

◆ redo 로그 정보

```
select value from v$sysstat where name = 'redo log space waittime';
select sum(decode(name,'redo blocks written', value,0)) "Block Writes",
sum(decode(name,'redo entries', value, 0)) "Entries",
```

```
sum(decode(name,'redo size', value, 0)) "Size",
sum(decode(name,'redo log space requests', value, 0)) "Space Requests",
sum(decode(name,'redo synch writes', value,0)) "Synch Writes",
sum(decode(name,'redo writes', value,0)) "Writes"
from v$sysstat;
```

◆ 라이브러리 캐시 활용도

```
select round((1 - (sum(reloads) / sum(pins))) * 100, 2)
from v$librarycache;
```

◆ 롤백 세그먼트 경합률

```
select round(sum(waits)/sum(gets),2) from v$rollstat;
```

◆ 롤백 세그먼트 현황

```
select n.usn,
n.name,
s.username Name,
s.osuser,
rs.extents,
rs.wraps,
rs.rssize "Size (Bytes)"
from v$rollname n, v$rollstat rs, v$session s, v$transaction t
where t.addr = s.taddr(+)
and rs.usn(+) = n.usn
and t.xidusn(+) = n.usn
and rs.status = 'ONLINE'
order by n.usn;
```

◆ SGA 프리 스페이스 현황

```
select sum(decode(name, 'free memory', bytes, 0))
from v$sgastat;
select round((sum(decode(name, 'free memory', bytes, 0))
/ sum(bytes)) * 100,0)
from v$sgastat;
```

◆ SGA 사이즈

```
select sum(value) from v$sga;
```

◆ 셰어드 풀의 리로드 횟수 및 비율

```
select sum(reloads)
from v$librarycache
where namespace in ('SQL AREA','TABLE/PROCEDURE', 'BODY','TRIGGER');

  select round(sum(reloads)
/ sum(pins) * 100,2)
from v$librarycache
where namespace in ('SQL AREA','TABLE/PROCEDURE','BODY','TRIGGER');
```

◆ Sort_Area 효율성

```
select round((sum(decode(name, 'sorts (memory)', value, 0))
/ (sum(decode(name, 'sorts (memory)', value, 0))
+ sum(decode(name, 'sorts (disk)', value, 0))))
* 100,2)
from v$sysstat;
```

◆ 소트 현황

```
select username Name,
osuser,
sd.value "Disk Sorts",
sm.value "Memory Sorts",
sr.value "Rows Sorted"
from v$session s, v$sesstat sd, v$sesstat sm, v$sesstat sr
where s.sid = sd.sid
and s.sid = sm.sid
and s.sid = sr.sid
and sd.statistic# = 101
and sm.statistic# =100
and sr.statistic# =102
and s.type != 'BACKGROUND';
```

◆ 소트 건수

```
select sum(value) from v$sysstat where statistic#=102);
```

◆ SQL AREA 활용 현황

```
select username,
sql_text,
sorts,
disk_reads Reads,
buffer_gets Gets
from v$sqlarea s,
sys.dba_users u
where s.parsing_user_id = u.user_id
and users_executing > 0
order by 1;
```

◆ V$SYSSTAT를 이용한 테이블 스캔 현황

```
select value
from v$sysstat
where name = 'table scans (long tables)';

select value
from v$sysstat
where name = 'table scans (short tables)';

select s.value + l.value
from v$sysstat s, v$sysstat l
where s.name = 'table scans (short tables)'
and l.name = 'table scans (long tables)';
```

◆ 시스템 테이블스페이스에 생성된 테이블 수

```
select count(*)
from sys.dba_tables t
where t.tablespace_name = 'SYSTEM'
and t.owner not in ('SYS','SYSTEM');
```

◆ Next 익스텐트가 프리 영역보다 큰 경우

```
select s.segment_name "Segment Name",
s.tablespace_name "Tablespace Name",
s.next_extent "Next Extent",
f.free_bytes "Free Bytes"
from dba_segments s,
(select tablespace_name,
sum(bytes) free_bytes
from dba_free_space
group by tablespace_name) f
where f.tablespace_name = s.tablespace_name
and s.next_extent > f.free_bytes;
```

◆ 테이블스페이스 프리 스페이스 현황

```
select tablespace_name Name,
sum(bytes) Bytes,
sum(blocks) Blocks,
count(*) "Number of Files"
from sys.dba_free_space
group by tablespace_name;
```

◆ 테이블스페이스 조각(Fragmentation) 현황

```
select f.tablespace_name "Tablespace Name",
file_name "File Name",
block_id, "Block Id"
f.blocks "Number of Blocks",
f.bytes "Number of Bytes"
from dba_free_space f, dba_data_files d
where f.file_id = d.file_id
order by f.tablespace_name ASC, file_name ASC, f.blocks DESC;
```

다음에는 유형별 튜닝 사례를

지금까지 총 4회에 걸쳐 부족하지만 개발자들이 꼭 알아야 할 튜닝 포인트를 기술했다. 고급 튜닝 기술자는 많은 어려움을 고민하고 극복한 끝에 탄생된다. 여러가지로 부족하지만 프로젝트를 수행하면서 어려운 용어와 두꺼운 책자에 기가 질려 첫 걸음도 떼지 못하는 개발자에게 조금이나마 도움을 주고자 용기를 내어 여기까지 이 강좌를 이끌어 왔다. 이론은 이번 호까지 마치고자 한다. 다음 5회에는 총정리를 하기 위해 실제로 일어났던 사례를 중심으로 마무리를 하고자 한다. 특히 지난 3회에 기고하였던 튜닝 유형을 중심으로 사례를 모아 기고할 것이니 좋은 사례가 있으면 메일을 통해 여러분의 참여를 기다린다. 끝