ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.1 DBMS 튜닝

(1) DBMS 튜닝 목표
(2) 튜닝 도구

# DBMS 튜닝

- **DBMS 튜닝이란?**
  - DBMS의 성능 향상을 목적으로 진행하는 일련의 작업
  - 어플리케이션 튜닝과 서버 튜닝으로 나뉨
- **DBMS 튜닝의 목표**
  - DBMS call을 최소화
  - Prepare를 최소화
  - Disk I/O를 최소화
  - CPU 사용률을 최소화
- **관점에 따른 튜닝의 종류**
  - Design 관점
  - Application 관점
  - DBMS 서버 관점
  - OS 관점

# DBMS call 최소화

- **DBMS call을 최소화**
  - 어플리케이션에서 DBMS로 호출하는 횟수가 많은가?
  - DBMS call을 최소화하여 어플리케이션의 성능을 향상
- **DBMS call을 최소화하기 위한 방법**
  - array processing으로 처리 : Array 단위 Fetch, Bulk Insert / Update / Delete
  - Fetch call을 최소화 : 부분범위처리, ArraySize 조정
  - APRE*C/C++

```
struct
  {
    char   gno[3][10+1];
    char   gname[3][20+1];
    char   goods_location[3][9+1];
    int    stock[3];
    double  price[3];
  } a_goods2;
…
EXEC SQL INSERT INTO GOODS VALUES (:a_goods2);
```

# DBMS call 최소화

- Java

```
1. array processing
        for(…){
            …
            pstmt.addBatch();
        }
        pstmt.executeBatch();

2. Fetch call 최소화
        …
        stmt.setFetchSize(100);
         …
```

# Prepare 최소화

- **Prepare 비용**
  - 단순 SQL  처리시 약 70%정도가 Prepare 비용
  - DBMS call을 최소화하여 어플리케이션의 성능을 향상
- **Prepare를 최소화하기 위한 방법**
  - C/C++/APRE* 프로그램 작성 시 bind 변수
  - Java 프로그램 작성 시 PreparedStatement를 이용
  - SQL Plan Cache에 적재되어있는 Execution Plan을 재 사용
- **실행계획을 공유하지 못해 Prepare를 다시 수행하는 경우**
  - 공백이 다르거나 줄바꿈이 다른 경우
  - 주석이 다른 경우
  - 힌트가 다른 경우
  - 조건절 비교 값이 다른 경우
  - 대소문자가 다른 경우

# Disk I/O 최소화

## Disk DBMS의 I/O

- Disk DBMS는 한건의 레코드만 읽어도 page 단위로 I/O가 일어남

## Memory I/O vs. Disk I/O

| Memory I/O | Disk I/O |
|---|---|
| 전기적인 신호에 의한 입출력 | 액세스 Arm이 움직이면서 헤드를 통해 입출력 |
| 속도가 빠르다 | 속도가 느리다 |

## Disk I/O를 최소화하기 위한 방법

- 자주 access되는 table은 memory table로 구성한다.
- 필요한 최소 page만 읽도록 SQL을 작성한다.
- Buffer hit율을 높인다.
- Random access를 줄인다.
- Sequential access를 늘인다.
- Single I/O page read와 multi I/O page read를 고려하여 SQL이 index scan/ full table scan 중 유리한 쪽으로 수행되도록

# CPU 사용률 최소화

- **CPU 사용률의 최소화**
  - Memory DBMS는 I/O 보다는 CPU사용률이 튜닝 factor임
  - 질의 1개 수행 동안 CPU 1EA(100%)를 사용
- **CPU 사용률 최소화하기 위한 방법**
  - Memory table은 전체 건수를 다 질의하는 경우라도 full table scan 보다는 index scan이 빠름
  - index scan을 하도록 SQL을 작성

# ALTIBASE HDB 튜닝 factor

- **튜닝을 위해 다음의 사항을 점검하라.**
  - Application 관점
    - 트랜잭션 증가
    - 매번 connect - disconnect를 반복
    - SQL 실행 시 빈번한 prepare 수행
    - full table scan 등의 오래 수행되는 질의
    - lock을 잡고 있는 질의
  - ALTIBASE 관점
    - checkpoint I/O
    - logfile writing
    - service thread 병목
    - memory ager
    - buffer
    - SQL plan cache
  - OS 관점
    - OS 설정
    - disk I/O 성능

# 튜닝 도구

- **ALTIBASE HDB 튜닝 도구**
  - Explain Plan
  - Profiling
  - Meta table & Performance view
  - OS 유틸리티 or 명령어(ps, top, nmon, glance)

# Explain Plan

## ❖ explain plan

- SQL의 실행 계획을 확인하고자 할 때 설정
- iSQL에서 EXPLAIN PLAN을 설정 후 확인 가능

(1) 질의 수행후 출력하며, plan tree와 access 횟수 등을 출력

```
iSQL> ALTER SESSION SET explain plan =on;
```

(2) Plan tree 출력하지 않음

```
iSQL> ALTER SESSION SET explain plan =off;
```

(3) 질의 수행하지 않고 Plan tree 만 출력

```
iSQL> ALTER SESSION SET explain plan =only;
```

# Explain Plan

## ❖explain plan 예제

### ➢ EXPLAIN PLAN = ON 설정

```
iSQL> ALTER SESSION SET explain plan =on;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno=1;
ENO         ENAME                    SALARY
------------------------------------------------
1           EJJUNG
1 row selected.
----------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 47 )
 SCAN ( TABLE: EMPLOYEE, INDEX: SYS_C0011065, ACCESS: 1, SELF_ID: 2 )
----------------------------------------------------------------
```

### ➢ EXPLAIN PLAN = OFF 설정

```
iSQL> ALTER SESSION SET explain plan =off;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno=1;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno = 1;
ENO         ENAME                    SALARY
------------------------------------------------
1           EJJUNG
1 row selected.
```

# Explain Plan

## ❖explain plan 예제

- EXPLAIN PLAN = ONLY 설정

```
iSQL> ALTER SESSION SET explain plan =only;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno=1;
ENO          ENAME                    SALARY
------------------------------------------------
No rows selected.
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 47 )
 SCAN ( TABLE: EMPLOYEE, INDEX: SYS_C0011065, ACCESS: ??, SELF_ID: 2 )
------------------------------------------------------------
```

# Profiling

## Profiling

- 사용자가 수행한 SQL 실행 정보를 수집하는 기능
- system level에서 property로 설정
- Profiling 된 파일은 $ALTIBASE_HOME/trc 에 저장됨

## Profiling으로 저장되는 정보

- 질의를 수행한 클라이언트 정보
- 질의 처리의 통계정보
  1. 실행시각
  2. 질의문
  3. 수행시간
  4. 인덱스정보
  5. 버퍼/디스크 접근비용
  6. 실행계획

# Profiling

## ❖ 실행방법

```
iSQL> ALTER SYSTEM SET QUERY_PROF_FLAG = 1 ;
ALTER  SUCCESS.
iSQL> - -  (Execute an SQL query here.)
iSQL> ALTER SYSTEM SET QUERY_PROF_FLAG = 0 ;
ALTER  SUCCESS.
$ cd  $ALTIBASE _HOME / trc
$ altiProfile  alti-xxxxxxx-0.prof  > result.log
```

➢ QUERY_PROF_FLAG 값 설명

| 값 | 이름 | 설명 |
|---|---|---|
| 0 | | 기록하지 않음 |
| 1 | [STATEMENT] | SQL문이 실행될때마다 실행된 SQL문, 실행시간, 색인 및 디스크 접근 정보 출력 |
| 2 | [BIND] | SQL문이 실행될때마다 BIND 파라미터 출력 |
| 4 | [PLAN] | SQL문이 실행될때마다 실행계획 출력 |
| 8 | [SESSION STAT] | 3초마다 세션 정보 출력 (V$SESSTAT 출력) |
| 16 | [SYSTEM STAT] | 3초마다 시스템 정보 출력 (V$SYSSTAT 출력) |
| 32 | [MEMORY STAT] | 3초마다 메모리 정보 출력 (V$MEMSTAT 출력) |

➢ 프로파일링 파일이 커져 디스크 Full 발생될수 있으므로 주의

# Profiling

## ❖ 프로파일 로그 예 ( QUERY_PROF_FLAG = 1 )

```
[STATEMENT] 2013/10/11 15:23:13 (5/327680/71172)
 SQL
   => [SELECT g.gname, g.price, o.qty, o.order_date FROM orders o, goods g WHERE g.gno = o.gno]   → 쿼리문
 User Info
   User    ID     = 2
   Client PID     = 32427
   Client Type    = [CLI-64LE]
   Client AppInfo = [isql]

 Elapsed Time                 ----------------------------------------→ 실행시간
   Total =          0 sec       0 usec
   SoftP =          0 sec       0 usec
   Parse =          0 sec       0 usec
   Valid =          0 sec       0 usec
   Optim =          0 sec       0 usec
   Execu =          0 sec       0 usec
   Fetch =          0 sec       0 usec

 Query Execute Info           ----------------------------------------→ 쿼리수행정보
   EXECUTE Result =              4 (0:failure, 1:rebuild, 2:retry, 3:queue empty, 4:success)
   Optimizer Mode =              0
   Cost      Mode =            163
   Used Memory    =              0
   SUCCESS   SUM  =              4
   FAILURE   SUM  =              0
   PROCESSED ROW  =              0
```

# Profiling

## ❖ 프로파일 로그 예

```
XA Info
  XA Flag       =              0 (0:Non-XA, 1:XA)

Result Set Info
  FETCH Result  =              1 (0:failure, 1:success, 2:no result set)

Index Access Info              ----------------------------------→ 인덱스 접근정보
  Memory Full  Scan Count  =             4
  Memory Index Scan Count  =            30
  Disk   Full  Scan Count  =           0
  Disk   Index Scan Count  =            0

Disk Access Info               ----------------------------------→ 디스크 접근정보
  READ   DATA PAGE  =            0
  WRITE  DATA PAGE  =            0
  GET    DATA PAGE  =           0
  CREATE DATA PAGE  =            0
  READ   UNDO PAGE  =            0
  WRITE  UNDO PAGE  =            0
  GET    UNDO PAGE  =           0
  CREATE UNDO PAGE  =            0
```

# Meta table & Performance view

- **ALTIBASE HDB의 상태를 모니터링하기 위한 도구**
  - Meta table
  - Performance view
- **Meta table**
  - 데이터베이스 객체에 관한 모든 정보를 기록하기 위한 시스템 정의 테이블
  - 소유자는 SYSTEM_
  - DDL 수행 시 시스템에 의해 변경
- **Performance view**
  - 인스턴스와 성능 관련정보를 저장
  - 시스템 메모리, 프로세스 상태, 세션, 버퍼 등의 최신 정보를 제공
  - read only

# OS 명령어

❖OS 및 ALTIBASE HDB를 점검하기 위해 OS의 명령어 및 유틸리티 이용

| | AIX | HPUX | LINUX | Solaris |
|---|---|---|---|---|
| Performance monitor | top<br>topas<br>nmon | top<br>glance | top | top |
| System activity reporter | sar | sar | sar | sar |
| Virtual Memory statistics | vmstat | vmstat | vmstat | vmstat |
| I/O statistics | iostat | iostat | iostat | iostat |
| Error log | errpt | dmesg | dmesg | emesg<br>/var/adm/syslog |

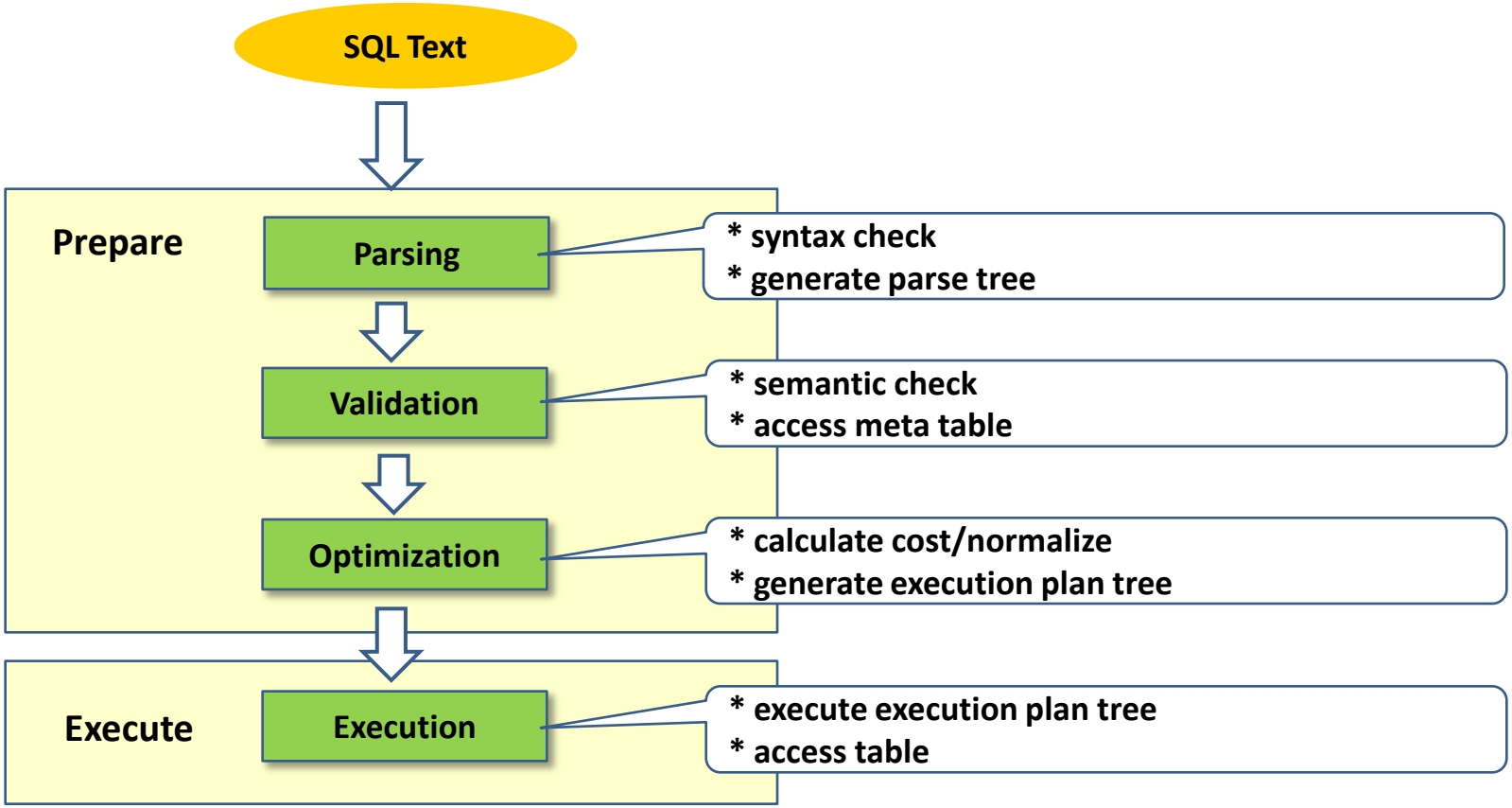
ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.2 SQL 튜닝

(1) SQL 처리절차
(2) 통계정보
(3) SQL 튜닝절차 및 방법

# SQL 처리 절차

## ❖SQL 질의 처리 절차



# Memory table vs. Disk table

## ❖Disk table과 Memory table의 질의 처리 방법의 차이

| Item | Memory table | Disk table |
|---|---|---|
| Object identifier | Pointer | OID(RID) |
| Buffer management | N/A | Limited buffer |
| Join methods | One-pass algorithms | Multi-pass algorithms |
| Main cost | CPU | Disk |
| Index selection | Minimize record access | Minimize disk I/O |
| Cost factor | T(R), V(R.a), etc | + B(R), M |

T(R) : 테이블의 레코드 수
V(R.a) : 컬럼의 Value Cardinality
B(R) : 테이블의 page 수
M : Memory Buffer 수

- One- pass algorithms : 가용메모리 버퍼에 중간 결과를 모드 적재할 수 있을 때 사용
- Multi-pass algorithms : 중간 결과를 메모리상에 모두 적재할 수 없을 때 버퍼교체를 최소화 하기 위해 사용

# 인덱스

## ❖Index

- 검색 시 성능 향상을 위해 테이블과는 별도로 저장되는 객체
- Index 대상 컬럼 값을 sorting하여 저장

| Memory Index | Disk Index |
|---|---|
| DB가 구동될 때마다 Memory에 생성 | 인덱스 생성 시점에 Disk에 생성 |
| 실제 테이블의 데이터에 대한 포인터만 저장 (16 bytes) | 컬럼의 값과 테이블의 레코드 주소 값이 저장 |
| DML 시 Logging을 하지 않음 | DML 시 Logging을 함 |

## ❖ALTIBASE HDB Index 특징

- UNIQUE, PRIMARY KEY로 지정한 컬럼은 내부적으로 Unique Index가 생성
- 테이블과 별도의 디스크에 분리해서 저장하는 것을 권장
- Btree Index와 Rtree Index 만 지원
  - Reverse, Bitmap, Global partitioned Index 등은 지원하지 않음

# Sequential vs. Random 엑세스

- **Sequential 엑세스**
  - 레코드 간 물리적인 순서를 따라 차례대로 읽어 나가는 방식
  - Full table scan 시 레코드를 읽는 경우
  - 인덱스 리프 노드에 위치한 데이터들을 읽는 경우
- **Random 엑세스**
  - 레코드 간 순서를 따르지 않고 한 건의 레코드를 읽기 위해 한 page 씩 접근하여 읽는 방식
  - 인덱스에 저장되어 있는 물리적인 레코드의 주소를 참조하여 테이블의 레코드를 찾아가는 경우
- **튜닝 방법**
  - Sequential 엑세스에 의한 선택 비중 ↑
  - Random 엑세스 발생량 ↓

# 질의 최적화 과정

## ❖Optimizer의 질의 최적화 과정

1. 조건절의 분류
2. access 방법 결정
3. Join 순서 결정
4. Join 방법 결정
5. Group/Aggregate 연산의 수행 방법 결정
6. DISTINCT절의 수행 방법 결정
7. SET 절의 수행 방법 결정
8. ORDER BY 절의 수행 방법 결정
9. Projection의 수행 방법 결정

# SQL 튜닝 절차

## ❖SQL 튜닝 절차

1. 오래 수행되는 질의문 확인
2. explain plan 설정
3. set timing on 설정
4. 실행계획 확인
5. SQL 문 변경, 인덱스 설정, 힌트 등을 사용하여 SQL 튜닝

# SQL 튜닝 방법

## ❖SQL 튜닝 방법

- 가능한 한번만 Prepare하라.
- 효율적인 인덱스를 사용하는지 확인하라.
- Driving table이 적절한지 확인하라.
- 인덱스가 필요하다면 추가하라.
- hint를 활용하라.
- 가능하다면, limit을 이용한 부분범위 처리를 해라.

ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.3 SQL 실행계획

(1) 실행계획 확인
(2) 실행계획 노드 종류
(3) 조건절 처리유형
(4) 인덱스 스캔
(5) JOIN 방법

# 실행계획 확인

## ❖실행계획이란?

- SQL이 실행될 때 필요한 처리 절차.
- Optimizer에 의해 생성

## ❖실행계획 확인

- EXPLAIN PLAN 설정

**ALTER SESSION SET EXPLAIN PLAN = {ON|ONLY|OFF}**
- ON : 실행 계획 + 수행 결과
- ONLY : 실행 계획 만
- OFF : 수행 결과 만

- 예

```
iSQL> ALTER SESSION SET explain plan =on;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno=1;
ENO          ENAME                    SALARY
------------------------------------------------
1            EJJUNG
1 row selected.
----------------------------------------------------------
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 35 )
 SCAN ( TABLE: EMPLOYEE, INDEX: __SYS_IDX_ID_163, ACCESS: 1, SELF_ID: 2 )
----------------------------------------------------------
```

# 실행계획 확인

## ❖Predicate를 자세히 보고자 할 경우

- System level의 property 설정

```
ALTER SYSTEM SET TRCLOG_DETAIL_PREDICATE=1
```

- 예

```
iSQL> ALTER SYSTEM SET trclog_detail_predicate=1;
Alter success.
iSQL> ALTER SESSION SET explain plan=on;
Alter success.
iSQL> SELECT e.ename, d.dname
      2 FROM employee e, department d
      3 WHERE e.dno = d.dno
      4 AND e.ename='KSKIM';
ENAME                 DNAME
------------------------------------------------------
KSKIM                 CUSTOMER SUPPORT DEPT
1 row selected.
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 2, TUPLE_SIZE: 54 )
 JOIN
  SCAN ( TABLE: EMPLOYEE E, FULL SCAN, ACCESS: 20, SELF_ID: 2 )
  [ FILTER ]      <= predicate 정보
  E.ENAME = 'KSKIM'
 SCAN ( TABLE: DEPARTMENT D, INDEX: __SYS_IDX_ID_322, ACCESS: 1, SELF_ID: 3 )
  [ VARIABLE KEY ]     <= predicate 정보
   OR
    AND
     E.DNO = D.DNO
------------------------------------------------------------
```

# 조건절 처리 유형

## ❖WHERE절에서 조건절을 처리하는 유형

| 분류 | 처리순서 | 설명 | example |
|---|---|---|---|
| Key Range | 2 | 인덱스를 사용하여 범위 검색이 가능한 조건 | T1.i1 > 1 |
| Key Filter | 3 | 범위 검색은 안되나 인덱스의 키 값을 비교하여 검사가 가능한 조건 | T1.i2 = 1 |
| Constant Filter | 1 | 테이블의 데이터와 관계 없이 한 번의 검사만으로 판단이 가능한 조건 | ? = 1 |
| Filter | 4 | 데이터에 대한 직접적인 비교가 필요한 조건 | T1.i4 = abc |
| Subquery Filter | 5 | Subquery를 포함하고 있는 조건 | EXISTS () |

```
Index : T1(i1, i2, i3)
질의 : SELECT * FROM T1
        WHERE ? = 1 AND i1 > 1 AND i2 = 2 AND i4 = 'abc'
        AND EXISTS ( SELECT * FROM T2 WHERE T2.a1 = T1.i3 );
```

# 옵티마이저의 인덱스 선택

## ❖INDEX가 있더라도 사용할 수 없는 조건절

- 인덱스를 사용할 수 없는 연산자

```
WHERE enname LIKE '_K%';
WHERE enname NOT LIKE 'KIM%';
```

- INDEX 칼럼에 포함되었더라도 변형을 한 경우

```
WHERE TO_CHAR(c1) = ‘a’;
WHERE substr(ename,1,4) = 'alti';
WHERE salary * 12 < 30000000;
```

- 데이터 타입이 다른 경우(일부의 데이터 타입)

```
WHERE start_flag = 1; (start_flag 의 타입이 CHAR/VARCHAR인 경우)
```

- Composite Index일 경우 첫번째 컬럼의 Index를 탈수 있는 조건이 없는 경우

```
WHERE c2 = ‘a’; (c1, c2 순서로 결합인덱스를 생성한 경우)
```

- ALTIBASE Optimizer 가 Index SCAN COST 가 더 소요된다고 판단하였을 때

# JOIN 방법

- **ALTIBASE HDB에서 지원하는 조인 방법**
  - Nested loops 조인 계열
  - Sort-based 조인 계열
  - Hash-based 조인 계열
  - Merge 조인 계열

# Nested loops 조인

## ❖Nested loops 조인의 수행 방법

1. 선행 테이블에서 조건에 만족하는 레코드 검색
2. 선행 테이블의 조인 키 값으로 후행 테이블에서 조인을 수행
3. 선행 테이블의 조건에 만족하는 모든 레코드에 대해 1,2번을 반복

## ❖Nested loops 조인 관련 용어

- Driving table, Outer table
  - 먼저 Scan 하는 선행 테이블
- Lookup table, Inner table
  - 선행테이블에서 조인조건에 만족하는 레코드와 연결하는 후행 테이블
  - Random access로 찾아감
  - 조인 컬럼에 index가 있어야 효율적

## ❖Nested loops 조인의 특징

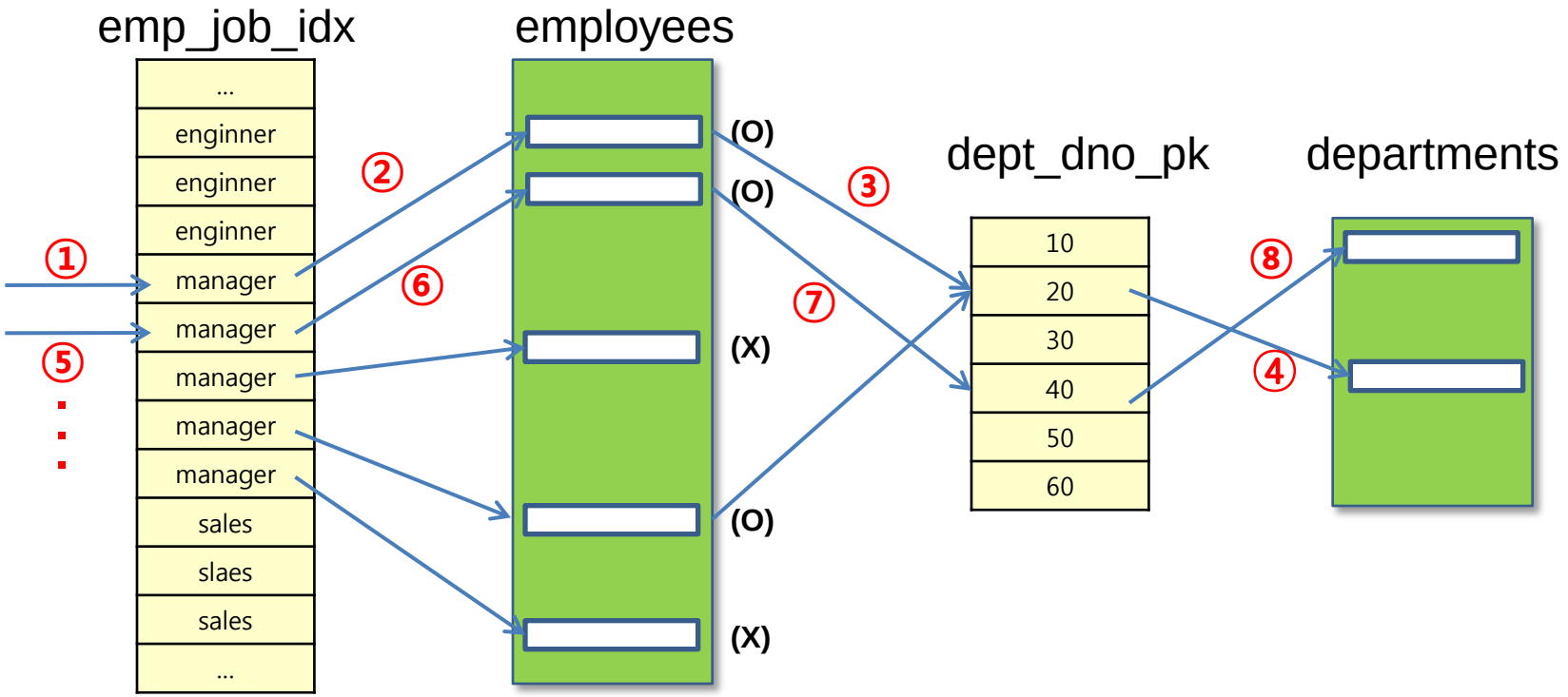
- Random access 위주의 조인 방식
- 선행 테이블의 조건에 만족하는 한 레코드 씩 순차적으로 진행
- OLTP 시스템에서 일차적으로 고려
- 부분범위처리시 유용

# Nested loops 조인

## ❖ Nested loops 조인

- ➢ 중첩 루프와 동일한 방식으로 동작

```
iSQL> SELECT * FROM employees e, departments d
      2 WHERE e.dno = d.dno
      3 AND e.emp_job = 'manager'
      4 AND e.salary > 2000000;
```



# Sort Merge 조인

## ❖Sort Merge 조인의 수행 방법

1. Outer 테이블의 조건에 만족하는 레코드를 조인 컬럼의 값으로 정렬
2. Inner 테이블의 조건에 만족하는 레코드를 조인 컬럼의 값으로 정렬
3. Outer 테이블의 조인 컬럼의 값이 같은 레코드를 Inner 테이블에서 찾아가고, 다른 값이 나오는 순간 조인을 멈춤
4. Outer 테이블의 다음 레코드의 조인 컬럼의 값이 같은 레코드를 Inner 테이블에서 찾아가며 3번 step을 반복

## ❖Sort Merge 조인의 특징

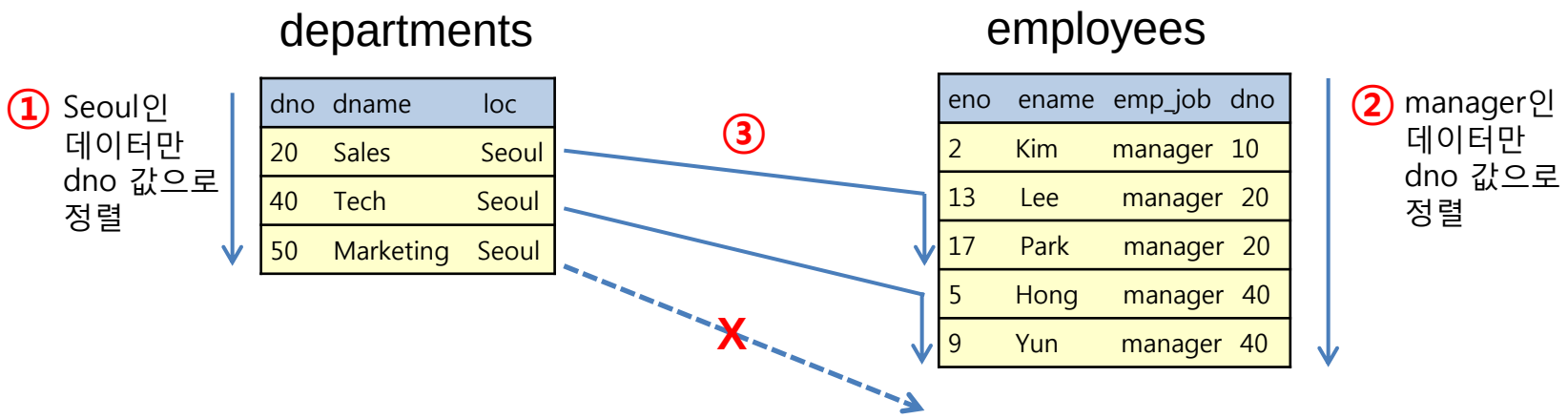
- 양쪽 테이블에 정렬이 수행된 후 조인을 수행
- index를 통해 미리 정렬이 되어있을 경우는 조인을 수행하지 않음
- Non equi 조인일 경우 Sort Merge 조인으로 수행됨

# Sort Merge 조인

## ❖Sort Merge 조인

- Sort 단계 : 양쪽 집합을 조인 컬럼으로 정렬
- Merge 단계 : 정렬된 양쪽 집합을 서로 머지

```
iSQL> SELECT * FROM employees e, departments d
      2 WHERE e.dno = d.dno
      3 AND e.emp_job = 'manager'
      4 AND d.loc='Seoul';
```



# Hash-based 조인

## ❖Hash- based 조인의 수행 방법

1. 선행 테이블(Build)의 조건에 맞는 레코드의 조인 컬럼에 Hash function을 수행한 후 결과로 Hash map을 생성
2. 후행 테이블(Probe)의 조건에 맞는 레코드의 조인 컬럼에 Hash function을 수행한 후 Hash map을 탐색하면서 조인을 수행

## ❖Hash-based 조인의 특징

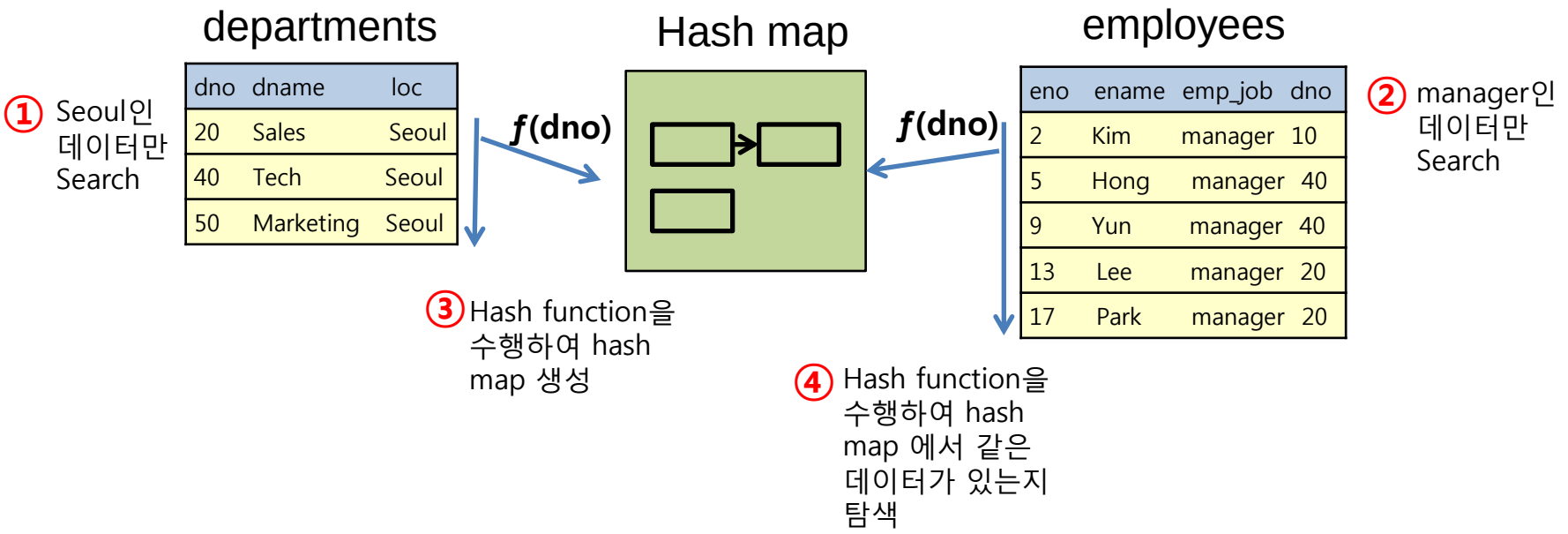
- NL 조인의 Random access와 SM 조인의 정렬 부담이 없음
- Hash function 수행 및 Hash map 생성 비용
- 조인 컬럼에 인덱스가 없을 경우 수행
- Equi join 에서만 사용
- 일반적으로 대용량의 테이블 조인 시 사용
- Build table이 작아야 효과적(Hash map이 disk에 생성될 경우 disk I/O 발생)

# Hash-based 조인

## ❖Hash-based 조인

- 작은 테이블(Build)을 먼저 읽어 Hash map을 생성하고 큰 테이블(Probe)을 읽어 Hash map을 탐색하면서 조인

```
iSQL> SELECT * FROM employees e, departments d
      2 WHERE e.dno = d.dno
      3 AND e.emp_job = 'manager'
      4 AND d.loc='Seoul';
```



ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.4 SQL 힌트

(1) 힌트 사용방법
(2) 힌트 종류

# 힌트 사용 방법

- **힌트란?**
  - 옵티마이저가 올바른 실행계획을 생성할 수 있도록 유도
- **힌트를 사용하는 SQL**
  - SELECT
  - UPDATE
  - DELTE
  - MOVE
  - INSERT(APPEND 힌트만 사용가능)
- **힌트를 사용하는 방법**

```
SELECT /*+ hint */ ~
UPDATE /*+ hint */ ~
DELETE /*+ hint */ ~
MOVE /*+ hint */ ~
INSERT /*+ hint */ ~
```

# 힌트 종류

| 힌트 | 설명 |
|---|---|
| PUSH_PRED( view_name ) | 외부의 WHERE절의 조건 중 뷰와 관계된 조인 조건을 뷰 내부로 이동하여 처리 |
| NO_MERGE( view ) | 뷰가 main query와 merge되는 것을 막을 때 사용 |
| COST | 비용을 고려한 실행 계획 생성 |
| RULE | 비용을 배제한 실행 계획 생성 |
| ORDERED | FROM절에 나열된 순서대로 조인 순서를 결정 |
| TEMP_TBS_MEMORY | 질의 처리 중에 생성되는 모든 중간 결과를 저장하기 위해 메모리 임시 테이블을 사용 |
| TEMP_TBS_DISK | 질의 처리 중에 생성되는 모든 중간 결과를 저장하기 위해 디스크 임시 테이블을 사용 |
| APPEND | Direct-Path INSERT가 수행 |
| NO_PLAN_CACHE | 생성된 플랜을 플랜 캐시에 저장하지 않음 |
| KEEP_PLAN | 한 번 생성된 플랜이 참조하는 테이블의 통계 정보가 변경되더라도 플랜이 재생성되는 것을 방지하고 그대로 사용하도록 함 |

# 힌트 종류

| 힌트 | 설명 |
|---|---|
| FULL SCAN ( table ) | 테이블에 이용 가능한 인덱스가 존재하더라도 인덱스를 사용하지 않고 테이블 전체를 스캔 |
| INDEX ASC ( table, index, index, ... ),<br>INDEX ASC ( table index index ... ) | 명시된 인덱스를 사용하여 해당 테이블에 대해서 인덱스 스캔을 수행, 오름 차순으로 탐색 |
| INDEX DESC ( table, index, index, ... ),<br>INDEX DESC ( table index index ... ) | 명시된 인덱스를 사용하여 해당 테이블에 대해서 인덱스 스캔을 수행, 내림 차순으로 탐색 |
| INDEX ( table, index, index, ... ),<br>INDEX ( table index index ... ) | 명시된 인덱스를 사용하여 해당 테이블에 대해서 인덱스 스캔을 수행 |
| NO INDEX ( table, index, index, ... ),<br>NO INDEX ( table index index ... ) | 명시된 인덱스를 사용해서 해당 테이블에 대한 인덱스 스캔을 수행하지 않도록 |
| USE_NL ( table, table ) | Nested loops 조인 계열의 조인 방법을 사용 |
| USE_HASH ( table, table ) | Hash-based 조인 계열의 조인 방법을 사용 |
| USE_SORT ( table, table ) | Sort-based 조인 계열의 조인 방법을 사용 |
| USE_MERGE ( table, table ) | Merge 조인 조인 계열의 조인 방법을 사용 |
| NO_PUSH_SELECT_VIEW ( table ) | 외부의 WHERE 절의 조건을 뷰 내부로 이동하여 처리하지 않음 |
| PUSH_SELECT_VIEW ( table ) | 외부의 WHERE절의 조건 중 가능한 것은 모두 뷰 내부로 이동하여 처리 |

# 힌트 - FULL SCAN

## ❖FULL SCAN(table)

➢ 이용 가능한 인덱스가 있더라도 테이블 데이터를 전체 스캔하여 검색

```
iSQL> SELECT /*+ full scan(tb02 ) */ DISTINCT col1 FROM tb02
    2 WHERE col3 ='testing'
    3 AND col1 NOT IN (1,2,3);
COL1
--------------
4
5
6
7
8
9
0
7 rows selected.
------------------------------------------------------------
PROJECT (COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 DISTINCT (ITEM_SIZE:16, ITEM_COUNT:7, DISK_PAGE_COUNT:3, ACCESS:7, SELF_ID:3, REF_ID: 2 )
  SCAN (TABLE: TB02, FULL SCAN, ACCESS: 1000000, DISK_PAGE_COUNT: 14272, SELF_ID: 2 )
   [ FILTER ]
   AND
    COL3 = 'testing'
    COL1 <>ALL (1, 2, 3)
------------------------------------------------------------
```

=> 디스크 테이블에 대해서는 거의 대부분의 row를 찾는 질의에 대해서는 full table scan이 multi block IO를 실행하므로 성능이 빠를 수 있다.

# 힌트 - INDEX ASC

- **INDEX ASC(table, index, index, ...), INDEX ASC(table index index ...), INDEX(table, index, index, ...), INDEX(table index index ...)**
  - 지정된 인덱스를 생성된 순서대로 검색

```
iSQL> SELECT /*+ index asc(tb01 tb01_idx01) */ col2 FROM tb01
    2 WHERE col1=1
    3 LIMIT 1;
COL2
--------------
1
1 row selected.
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 SCAN ( TABLE: TB01, INDEX: TB01_IDX01, ACCESS: 1, SELF_ID: 2 )
  [ FIXED KEY ]
  AND
   OR
    COL1 = 1
------------------------------------------------------------
```

=> 인덱스의 두번째 칼럼의 min max 값을 인덱스를 이용해 찾을 때 첫번째 조건에 해당하는 모든 값들에 대해 min max를 계산하므로, index asc/ index desc 힌트와 limit 1 절을 이용하면 필요한 한개의 레코드만 찾을 수 있으므로 성능이 향상된다.

# 힌트 - INDEX DESC

## ❖INDEX DESC(table, index, index, ...), INDEX DESC(table index index ...)

- ➢ 지정된 인덱스를 역순으로 검색

```
iSQL> SELECT /*+ index desc(tb01 tb01_idx01) */ col2 FROM tb01
    2 WHERE col1=1
    3 LIMIT 1;
COL2
--------------
999991
1 row selected.
-----------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 SCAN ( TABLE: TB01, INDEX: TB01_IDX01, ACCESS: 1, SELF_ID: 2 )
  [ FIXED KEY ]
  AND
   OR
    COL1 = 1
-----------------------------------------------------------------
```

=> 인덱스의 두번째 칼럼의 min max 값을 인덱스를 이용해 찾을 때 첫번째 조건에 해당하는 모든 값들에 대해 min max를 계산하므로, index asc/ index desc 힌트와 limit 1 절을 이용하면 필요한 한개의 레코드만 찾을 수 있으므로 성능이 향상된다.

# 힌트 - NO INDEX

## ❖ NO INDEX(table, index, index, ...), NO INDEX(table index index ...)

- ➢ 지정된 인덱스를 사용하지 않음

```
iSQL> SELECT /*+ no index(tb01 tb01_idx02) */ DISTINCT col3 FROM tb01
    2 WHERE col1=1
    3 AND col3 = 'testing';
COL3
---------------------------------------------------
testing
1 row selected.
-----------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 52 )
 DISTINCT ( ITEM_SIZE:24, ITEM_COUNT:1, BUCKET_COUNT:1024, ACCESS:1, SELF_ID:3, REF_ID:2 )
  SCAN ( TABLE: TB01, INDEX: TB01_IDX01, ACCESS: 100000, SELF_ID: 2 )
   [ FIXED KEY ]
   AND
    OR
     COL1 = 1
   [ FILTER ]
   COL3 = 'testing'
-----------------------------------------------------------------
```

# 힌트 - USE_NL

## ❖USE_NL(table, table)

- ➢ Nested loops 조인 계열의 조인 방법을 사용
- ➢ 첫 번째 테이블을 먼저 스캔

```
iSQL> SELECT /*+ use_nl(b, a) */ a.col1 FROM tb01 a , tb02 b
    2 WHERE a.col2 = b.col2
    3 AND a.col1 = b.col1
    4 AND a.col3 ='testing'
    5 AND b.col3 LIKE 'some%';
COL1
--------------
No rows selected.
-------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 JOIN
 SCAN ( TABLE: TB02 B, INDEX:TB02_IDX02, ACCESS: 200, DISK_PAGE_COUNT: 14272, SELF_ID: 3 )
  [ FIXED KEY ]
  AND
   OR
    B.COL3 LIKE 'some%'
 SCAN ( TABLE: TB01 A, INDEX: TB01_IDX01, ACCESS: 200, SELF_ID: 2 )
  [ VARIABLE KEY ]
  OR
   AND
    A.COL1 = B.COL1
    A.COL2 = B.COL2
  [ FILTER ]
  A.COL3 = 'testing'
-------------------------------------------------------------
```

# 힌트 - USE_HASH

- **USE_HASH(table, table)**
  - Hash-based 조인 방법을 사용
  - HASH 노드의 테이블을 먼저 스캔

```
iSQL> SELECT /*+ use_hash(b, a) */ a.col1 FROM tb01 a, tb02 b
    2 WHERE a.col2 = b.col2
    3 AND a.col1 = b.col1
    4 AND a.col3 ='testing' AND b.col3 LIKE 'some%';
COL1
--------------
No rows selected.
-------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 JOIN
  SCAN (TABLE: TB02 B, INDEX:TB02_IDX02, ACCESS:200, DISK_PAGE_COUNT:14272, SELF_ID:3 )
   [ FIXED KEY ]
   AND
    OR
     B.COL3 LIKE 'some%'
  HASH (ITEM_SIZE:24, ITEM_COUNT:999800, BUCKET_COUNT:1024, ACCESS:0, SELF_ID:4, REF_ID:2)
   [ FILTER ]
   AND
    A.COL1 = B.COL1
    A.COL2 = B.COL2
   SCAN ( TABLE: TB01 A, INDEX: TB01_IDX02, ACCESS: 999800, SELF_ID: 2 )
    [ FIXED KEY ]
    AND
     OR
      A.COL3 = 'testing'
-------------------------------------------------------------
```

# 힌트 - USE_MERGE

## ❖USE_MERGE(table, table)

- ➢ Sort Merge 조인 방법을 사용
- ➢ SORT 노드의 테이블을 먼저 스캔

```
iSQL> SELECT /*+ use_merge(b, a) */ DISTINCT a.col1 FROM tb01 a , tb02 b
    2 WHERE a.col2 > b.col2 AND a.col3 LIKE 'some AND b.col3 LIKE 'some%'
    3 AND b.col1=a.col1 AND b.col1=1;
col1
--------------
1
1 row selected.
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 1, TUPLE_SIZE: 4 )
 DISTINCT(ITEM_SIZE:16, ITEM_COUNT:1, DISK_PAGE_COUNT 3, ACCESS: 1, SELF_ID:6, REF_ID:2)
  MERGE-JOIN
   [ VARIABLE KEY ]
   B.COL1 = A.COL1
   [ FILTER ]
   A.COL2 > B.COL2
   SCAN (TABLE:TB02 B, INDEX:TB02_IDX01, ACCESS:100000, DISK_PAGE_COUNT:14272, SELF_ID:3)
    [ FIXED KEY ]
    AND
     OR
      B.COL1 = 1
    [ FILTER ]
    B.COL3 LIKE 'some%'
   SORT ( ITEM_SIZE: 16, ITEM_COUNT: 200, ACCESS: 40000, SELF_ID: 4, REF_ID: 2 )
    SCAN ( TABLE: TB01 A, INDEX: TB01_IDX02, ACCESS: 200, SELF_ID: 2 )
     [ FIXED KEY ]
     AND
      OR
       A.COL3 LIKE 'some%'
------------------------------------------------------------
```

# 힌트 - TEMP_TBS_MEMORY

## TEMP_TBS_MEMORY

- 질의 처리 중에 생성되는 모든 중간 결과를 저장하기 위해 메모리 임시 테이블을 사용
- 질의에 1개 이상의 디스크 테이블이 포함되어 있다면 중간 결과를 디스크에 저장
- 사용 시 메모리 증가

```
iSQL> SELECT * FROM tb01 ORDER BY col2;
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 4, TUPLE_SIZE: 64 )
 SORT (ITEM_SIZE:72, ITEM_COUNT:10000, DISK_PAGE_COUNT:920, ACCESS:10000, SELF_ID:2,
REF_ID:1)
  SCAN ( TABLE: TB01, FULL SCAN, ACCESS: 10000, DISK_PAGE_COUNT: 14272, SELF_ID: 1 )
------------------------------------------------------------

iSQL> SELECT /*+ temp_tbs_memory */ * FROM tb01 ORDER BY col2;
------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 4, TUPLE_SIZE: 64 )
 SORT ( ITEM_SIZE: 24, ITEM_COUNT: 10000, ACCESS: 10000, SELF_ID: 2, REF_ID: 1 )
  SCAN ( TABLE: TB01, FULL SCAN, ACCESS: 10000, DISK_PAGE_COUNT: 14272, SELF_ID: 1 )
------------------------------------------------------------
```

ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.5 ALTIBASE HDB 서버튜닝

(1) logfile writing
(2) Checkpoint
(3) Memory Ager
(4) SQL Plan Cache

# Logfile Writing

## ❖ logfile에 write하는 동안 트랜잭션이 대기하는가?

- prepare된 logfile에 다 쓴 후 다음 logfile을 생성할 때까지 트랜잭션이 대기하여 성능이 느려지고 CPU 사용률이 높아짐
- V$LFG의 lf_prepare_wait_count 값 확인

```
iSQL> SELECT lf_prepare_wait_count FROM v$lfg;

=>lf_prepare_wait_count 는 ALTIBASE가 구동 중에 logfile이 미처 생성되지 않아서 트랜잭션이 대기했던 누적 횟수를 나타냄
```

- DISK 속도 체크

```
Shell> time dd if=/dev/zero of=/altibase/altibase_home/logs/alti_test_file bs=8K count=1280
1280+0 records in
1280+0 records out
real    0m0.032s
user    0m0.000s
sys     0m0.033s
```

## ❖ 조치 사항

- PREPARE_LOG_FILE_COUNT 프로퍼티 값을 조절
  (PREPARE_LOG_FILE_COUNT 은 logfile manager 쓰레드가 지정한 값 만큼 미리 logfile을 쟁성)
- PREPARE_LOG_FILE_COUNT 값이 클 수록 메모리가 증가함

# Checkpoint

## ❖checkpoint 발생 시 disk I/O에 대한 부하로 성능 저하

- ➢ altibase_sm.log 파일을 통해 checkpoint 진행 과정 확인

1. checkpoint 시작

```
[CHECKPOINT-BEGIN]
```

2. checkpoint 시작 LSN 기록

```
 [CHECKPOINT-step2] Write BeginChkpt Log [0,89,6929497]
Active Tx Recovery LSN [0,3,3874568] :
Disk Buffer Oldest LSN [0,3,3874568] :
```

3. dirty page 대상 지정 및 트랜잭션 로그의 sync, , dirty page sync

```
[CHECKPOINT-step3] Flush Dirty Page(s)
[PRE-DirtyPageCount=0]
[NEW-DirtyPageCount=33036]
[DUP-DirtyPageCount=0]
Begin Sync For All-LFG - Request LSN [0,3,3875009]
Begin Bulk DB Sync (Prop.3200)
[FLU-DirtyPageCount=33036]
[REM-DirtyPageCount=0]
```

- PRE-DirtyPageCount: 이전에 썼던 Dirty Page 수
- NEW-DirtyPageCount: 새로 추가된 Dirty Page 수
- DUP-DirtyPageCount: New-DirtyPage중에서 PRE-DirtyPage와 중복되는 DirtyPage의 수
- Begin Sync For All-LFG - Request LSN: 트랜잭션로그를 Sync
- Begin Bulk DB Sync: dirty page sync
- FLU-DirtyPageCount: Flush 한 Dirty Page 수
- REM-DirtyPageCount: 2 벌의 데이터파일에 모두 기록되어 Dirty Page List에서 제거된 Page의 수

# Checkpoint

## ❖checkpoint 발생 시 disk I/O에 대한 부하로 성능 저하

- ➢ altibase_sm.log 파일을 통해 checkpoint 진행과정 확인

4. datafile sync

```
[CHECKPOINT-step4] sync Database File
```

5. checkpoint 종료 LSN 기록

```
[CHECKPOINT-step5] Write End_Chkpt Log [0,3,3974818]
```

6. checkpoint 종료에 대한 트랜잭션 로그를 sync

```
[CHECKPOINT-step6] Sync Log File
Begin Sync For All-LFG - Request LSN [0,89,6885511]
```

7. 이중화로 보낼 트랜잭션 로그가 어떤 SN인지 확인

```
[CHECKPOINT-step7] Check LogFiles That Is Not Needed
Replication MinSN368769
```

8. loganchor에 checkpoint 수행 정보 기록

```
[CHECKPOINT-step8] Update and Flush Log Anchor
```

9. 불필요한 logfile 삭제

```
[CHECKPOINT-step8] Update and Flush Log Anchor
```

# Checkpoint

## ❖checkpoint 발생 시 disk I/O에 대한 부하로 성능 저하

- sample

```
[2011/10/05 13:51:01] [Thread-182894216896] [Level-9]
[CHECKPOINT BY SYSTEM]

[2011/10/05 13:51:01] [Thread-182894216896] [Level-9]
[CHECKPOINT-BEGIN]

...

[2011/10/05 13:51:01] [Thread-182894216896] [Level-9]
[CHECKPOINT-summary] BeginChkptLSN=[0,153,3043510], EndChkptLSN=[0,153,3043951],
DiskRecLSN=[0,153,3043510]

[2011/10/05 13:51:01] [Thread-182894216896] [Level-9]
Minimum LSN = [0,153,3043510]

[2011/10/05 13:51:01] [Thread-182894216896] [Level-9]
[CHECKPOINT-END]
```

- checkpoint가 수행 중에 [CHECKPOINT-step3] Flush Dirty Page(s) 혹은 [CHECKPOINT-step4] sync Database File가 오래 수행 중이라면 disk I/O를 모니터링

# Checkpoint

- disk I/O
  - sar, iostat을 통해 disk I/O 병목 확인

```
Shell> sar 1 3
02:32:26 PM       CPU     %user     %nice   %system   %iowait     %idle
02:32:30 PM       all      0.25      0.00      2.87      1.87     95.01
02:32:31 PM       all      0.12      0.00      6.24      6.99     86.64
02:32:32 PM       all      0.25      0.00      8.61      3.75     87.39

Shell> iostat 1
avg-cpu:  %user   %nice    %sys %iowait   %idle
           0.13    0.00    8.76    3.88   87.23
Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sdb1           2821.78        63.37    608388.12         64     614472
```

# Checkpoint

## ❖조치 사항

- logfile과 datafile의 디스크 분리
- checkpoint 관련 프로퍼티 조절

| | |
|---|---|
| CHECKPOINT_BULK_WRITE_PAGE_COUNT | checkpoint 시 dirty page를 여러 번 나누어서 disk에 저장할 수 있는데 이 때 한번에 디스크에 기록하는 page 수 |
| CHECKPOINT_BULK_WRITE_SLEEP_SEC<br>CHECKPOINT_BULK_WRITE_SLEEP_USEC | checkpoint 시에 한번 디스크에 기록한 후 sleep 하는 시간 (두 값이 합산된 시간을 sleep) |
| CHECKPOINT_BULK_SYNC_PAGE_COUNT | checkpoint 시에 memory와 disk의 데이터를 일치시킬 page 단위 |

  - 기본 값보다 프로퍼티의 값을 줄여주면 disk I/O 분산 효과가 있지만 logfile이 증가할 수 있음
- 위의 사항을 적용했음에도 불구하고 성능향상이 없다면, disk 성능 한계치에 도달한 것으로 보임

# Disk Buffer

- **disk 테이블에 대한 buffer hit 율이 낮아서 disk I/O가 빈번하지는 않은가?**
  - disk table 페이지를 적재할 때 한정된 크기의 buffer pool을 사용하므로 버퍼 교체가 이루어지고, 버퍼 교체가 일어나면 disk I/O로 인해 성능이 저하
  - 버퍼 관련 performance view를 조회하여 버퍼 상황을 확인

```
iSQL> SELECT hit_ratio 'HIT_RATIO(%)' , victim_search_warp FROM v$buffpool_stat;
HIT_RATIO(%)            VICTIM_SEARCH_WARP
------------------------------------------------
99.8061076102763        0

=> VICTIM_SEARCH_WARP 값이 증가하고 있다면, flusher가 밀리고 있음
```

  - disk 테이블에 대한 SQL 중 많은 페이지를 access하는 쿼리 튜닝
  - BUFFER_AREA_SIZE 프로퍼티의 크기를 늘려줌

# Service Thread

## ❖ 모든 service thread가 busy하면 새로운 service thread를 생성해야하므로 시스템에 부하를 줌

- ➢ V$SERVICE_THREAD를 통해 service thread 관련 부하 확인

```
iSQL> SELECT rpad(type, 30), count(*) FROM v$service_thread GROUP BY type
      2 UNION ALL
      3 SELECT rpad(name, 30), value1 FROM v$property
      4 WHERE name LIKE 'MULTIPLEXING%_THREAD_COUNT';
RPAD(TYPE, 30)                 COUNT
-----------------------------------------------
SOCKET                         44
IPC                            10
MULTIPLEXING_THREAD_COUNT      8
MULTIPLEXING_MAX_THREAD_COUNT  1024
```

- SOCKET 항목의 수치가 MULTIPLEXING_THREAD_COUNT 항목의 수치보다 클 경우
  - 오래 수행되는 질의를 튜닝
  - MULTIPLEXING_THREAD_COUNT 프로퍼티 값을 늘려줌

# Memory Ager

## ❖memory ager가 aging을 잘하고 있는가?

- 메모리 테이블의 MVCC 기법으로 인해 생성되는 old versioning 정보를 memory ager가 삭제
- memory ager 종류

| | |
|---|---|
| MEM_LOGICAL_AGER | 인덱스 페이지에 대한 versioning 정보 삭제 |
| MEM_DELTHR | 데이터 페이지에 대한 versioning 정보 삭제 |

- memory ager가 밀린다면?
  - versioning 증가에 의한 메모리 증가
  - SQL 질의 시 versioning 정보들을 참조해야 하기 때문에 성능 저하
- V$MEMGC를 통해 memory ager가 잘 수행하고 있는지 확인
- memory ager 관련 프로퍼티
  - AGER_WAIT_MINIMUM, AGER_WAIT_MAXIMUN
    - ager가 해야할 Job이 없을 경우 대기하는 최소시간, 최대시간을 의미
    - ager는 작업이 없으면 MAX만큼 쉬고, 일이 있으면 MAX부터 1/2씩 쉬는 시간을 줄여 MIN까지 점차적으로 쉬는 시간을 줄여감.일이 없으면 다시 MAX까지 쉬는 시간을 늘려감

# Memory Ager

## ❖확인사항

- gc gap을 조회

```
iSQL> SELECT gc_name, add_oid_cnt, gc_oid_cnt , add_oid_cnt - gc_oid_cnt gcgap FROM v$memgc;
ADD_OID_CNT           GC_OID_CNT            GCGAP
------------------------------------------------------------------------
113                   113                   0
113                   113                   0
```

  - gc gap이 클수록 ager가 삭제해야 할 old version의 양이 많다는 것을 의미

- Ager가 대기하는 트랜잭션 정보

```
iSQL>  SELECT session_id, total_time, execute_time, tx_id, query
     2 FROM v$statement
     3 WHERE tx_id IN (SELECT id
     4                 FROM v$transaction
     5                 WHERE memory_view_scn = (SELECT minmemscnintxs FROM v$memgc LIMIT 1))
     6                 AND execute_flag = 1
     7 ORDER BY 2 DESC;
```

## ❖조치 사항

- GC의 대상 트랜잭션에서 수행하는 질의 튜닝
- AGER_WAIT_MINIMUM, AGER_WAIT_MAXIMUM 값을 줄여 ager가 수행을 자주할 수 있도록 함

# Summary

- **성능 향상을 위해 ALTIBASE 관점에서 확인해야 하는 사항**
  - logfile을 write하는 동안 대기하는가?
  - checkpoint 시 disk I/O에 대한 병목이 큰가?
  - buffer의 hit율은 좋은가?
  - memory ager가 잘 수행되는가?

ALTIBASE HDB ADVANCED ADMINISTRATION

# 1.6 트랜잭션 튜닝

(1) Long Term Query
(2) Lock
(3) Isolation Level
(4) Update Retry

# Prepare 부하

## ❖ 빈번한 prepare 수행

- Prepare-Validation-Optimization(PVO) 과정이 단순 DML 문장일 경우 전제 질의 처리과정의 60~70% 의 비중을 차지
- V$SYSSTAT에서 execute 대비 prepare 수치를 조회

```
iSQL> SELECT TO_CHAR(SYSDATE,'HH:MI:SS'), name, value FROM v$sysstat
    2 WHERE name IN ('execute success count','prepare success count');
TO_CHAR(SYSDATE,'HH:MI:SS')  NAME                          VALUE
-------------------------------------------------------------------
11:35:47                     execute success count         1225664
11:35:47                     prepare success count         234218
iSQL> /
TO_CHAR(SYSDATE,'HH:MI:SS')  NAME                          VALUE
-------------------------------------------------------------------
11:36:47                     execute success count         1272912
11:36:47                     prepare success count         273319

=> 47248(1272912-1225664) 문장 실행 시 39101(273319-234218) PREPARE를 실행(약 83%)
```

- prepare 비중이 많다면 바인딩 변수를 사용하는 구조로 변경

# Long Term Query

## 오래 수행되는 질의

- 질의를 처리하는 동안 해당 서비스 쓰레드가 CPU 1장(100%)을 사용
- CPU 부하의 가장 많은 원인
- V$STATEMENT를 통해 오래 수행되는 질의를 조회
- V$STATEMENT의 query는 16K까지만 보여주므로 전체 query를 조회하기 위해서는 V$SQLTEXT를 함께 질의

# Long Term Query

## V$SQLTEXT

- V$STATEMENT의 QUERY 칼럼은 16K까지만 보여주므로 때 전체 질의문을 확인할 때 조회

| 칼럼 | 설명 |
|---|---|
| SID | session ID |
| STMT_ID | statement ID |
| PIECE | 질의문의 조각 순서<br>(64byte단위로 나누어 저장되어 있음) |
| TEXT | 실제 질의문의 내용 |

- V$STATEMENT의 session_id, id 칼럼과 sid, stmt_id의 칼럼이 각각 같다는 조건을 이용하여 질의

```
iSQL> SELECT text FROM v$sqltext
      2 WHERE (sid,stmt_id) IN (SELECT session_id, id FROM v$statement WHERE id = 1123)
      3 ORDER BY piece;
```

# Long Term Query

## ❖V$STATEMENT에서 query와 time 정보 확인

```
iSQL> ALTER SYSTEM SET timed_statistics=1;
iSQL> set vertical on;
iSQL> SELECT query, execute_time, parse_time, total_time, optimize_time, validate_time,
    2         execute_success, mem_cursor_full_scan, mem_cursor_index_scan, disk_cursor_full_scan,
    3         disk_cursor_index_scan
    4 FROM v$statement
    5 ORDER BY execute_time DESC
    6 LIMIT 10;

QUERY                  : select * from employee where ename='KSKIM'
EXECUTE_TIME           : 9
PARSE_TIME             : 151
TOTAL_TIME             : 423
OPTIMIZE_TIME          : 76
VALIDATE_TIME          : 97
EXECUTE_SUCCESS        : 1
MEM_CURSOR_FULL_SCAN   : 1
MEM_CURSOR_INDEX_SCAN  : 0
DISK_CURSOR_FULL_SCAN  : 0
DISK_CURSOR_INDEX_SCAN : 0
```

=> full table scan 질의로 대부분의 시간이 prepare 하는데 소요

# Long Term Query

## ❖오래 수행되는 질의 튜닝

- 실행 계획을 확인하여 access 방식, join 방식, join 순서 등을 파악하여 튜닝
- 메모리 테이블은 full table scan 보다는 index scan이 더 유리

## ❖실행계획 확인하는 방법

- EXPLAIN PLAN 설정
  - ON : 실행결과 + 실행계획
  - ONLY : 실행계획만
  - OFF : 실행결과만

```
iSQL> ALTER SESSION SET explain plan =on;
iSQL> SELECT eno, ename, salary FROM employee WHERE eno=1;
ENO          ENAME                    SALARY
------------------------------------------------
1            EJJUNG
1 row selected.
----------------------------------------------------------------
PROJECT ( COLUMN_COUNT: 3, TUPLE_SIZE: 35 )
 SCAN ( TABLE: EMPLOYEE, INDEX: __SYS_IDX_ID_163, ACCESS: 1, SELF_ID: 2 )
----------------------------------------------------------------
```

# Lock

## ❖lock을 잡고 있는 질의

- lock 부하로 인해 다른 트랜잭션에 영향을 주어 시스템이 느려짐
- V$LOCK, V$LOCK_WAIT, V$TRANSACTION, V$STATEMENT 정보 확인
- 트랜잭션을 가급적 작은 단위로 나누어 lock 지속 시간을 줄임
- dead lock이 발생할 수 있는 상황을 방지

# Lock

## ❖lock을 잡고 있는 질의 확인

```
iSQL> SELECT tx.id tx_id, lw.wait_for_trans_id blocked_tx_id, l.lock_desc,
    2          DECODE(tx.first_update_time, 0, '0', to_char(to_date('1970010109','YYYYMMDDHH')
    3                     + tx.first_update_time / (60*60*24), 'MM/DD HH:MI:SS')) first_update_time,
    4          DECODE(tx.status, 0, 'BEGIN', 1, 'PRECOMMIT', 2, 'COMMIT_IN_MEMORY', 3, 'COMMIT',
    5                     4, 'ABORT', 5, 'BLOCKED', 6, 'END') status,
    6          st.query current_query
    7  FROM v$transaction tx, v$lock l
    8          LEFT OUTER JOIN v$lock_wait lw  ON l.trans_id = lw.trans_id
    9          LEFT OUTER JOIN (SELECT st.query,tx_id FROM v$statement st, v$session ss
   11                                WHERE ss.id = st.session_id ) st
   12               ON l.trans_id = st.tx_id
   13 WHERE tx.id = l.trans_id;

TX_ID              : 103489
BLOCKED_TX_ID      :
LOCK_DESC          : IX_LOCK
FIRST_UPDATE_TIME  : 09/02 14:42:35
STATUS             : BEGIN
CURRENT_QUERY      : update t1 set c1=1 where c1 between 1.11  and 1.112
TX_ID              : 4288
BLOCKED_TX_ID      : 103489
LOCK_DESC          : IX_LOCK
FIRST_UPDATE_TIME  : 0
STATUS             : BLOCKED
CURRENT_QUERY      : update t1 set c1=1 where c1 =1.11
```

# Lock

## ❖lock을 잡고 있는 세션 강제 종료

iSQL(sysdba)> **ALTER DATABASE mydb SESSION CLOSE 10;**

- mydb : DB이름
- 10 : session_id

➢ 강제로 세션 종료 시 세션이 종료될 때까지 다소 시간이 걸릴 수 있음 (rollback 처리 등)

# Durability

## Durability 설정방법

- altibase.properties 파일에서 관련 프로퍼티의 변경하여 설정
  - COMMIT_WRITE_WAIT_MODE, LOG_BUFFER_TYPE 프로퍼티로 설정
- DB 구동 시 설정된 Durability 로 구동
- DB 운영 중에 실시간으로 변경 불가능

## COMMIT_WRITE_WAIT_MODE =0

- OS Kernel 영역의 로그버퍼를 사용하기 때문에 ALTIBASE HDB 프로세스가 비정상 종료를 하더라도 트랜잭션이 commit한 로그는 운영체제의 의해 로그 파일에 반영 (성능지향 설정 방법)
- OS의 crash 상황만 아니라면 트랜잭션 durability를 완벽하게 지원

## COMMIT_WRITE_WAIT_MODE = 1

- 로그를 프로세스 영역의 로그버퍼에 기록하고, 물리적인 로그파일에 기록하는 것을 보장하기 때문에 ALTIBASE HDB의 장애 시에도 durability를 보장함
- 모든 로그가 로그파일에 반영됨을 보장하기 때문에 어떠한 시스템 장애 상황에서도 완벽하게 트랜잭션 durability를 보장하나 durability level중 성능이 가장 느림

# Durability

## ❖Durability 설정방법

| 목적 | 설명 및 설정 항목 | |
|---|---|---|
| 성능 위주의 설정 (Durability = 3) | 운영체제가 트랜잭션 로그파일의 디스크 Sync를 보장하는 설정으로 운영체제의 Crash와 같은 장애를 제외한 모든 장애의 경우에 대해 보장함 | |
| | LOG_BUFFER_TYPE | 0 |
| | COMMIT_WRITE_WAIT_MODE | 0 |
| 안정성 위주의 설정 (Durability = 5) | ALTIBASE가 직접 디스크 Sync를 보장하는 설정으로 물리적인 디스크 장애 외에는 어떠한 장애라도 보장함 | |
| | LOG_BUFFER_TYPE | 1 |
| | COMMIT_WRITE_WAIT_MODE | 1 |

- COMMIT_WRITE_WAIT_MODE
  - ALTER SESSION 명령어로 변경 가능
- LOG_BUFFER_TYPE
  - 변경하려면 DB restart가 필요함

# Summary

- **성능 향상을 위해 application 관점에서 확인해야 하는 사항**
  - 트랜잭션 양이 많지는 않은가?
  - connect - disconnect를 반복하는 구조인가?
  - prepare를 자주 하는가?
  - long term query가 수행되지는 않는가?
  - lock 경합이 빈번히 발생하는가?
  - update retry가 발생하는가?
  - 통신 방식은 무엇을 사용하는가?

# 실습 문제

ALTIBASE HDB ADVANCED ADMINISTRATION

# 2.1 모니터링

(1) Trace File

(2) Performance View

(3) OS 항목

# 모니터링

## ❖평상시 모니터링 항목

- Altibase Trace Files
  - altibase_boot.log
  - altibase_sm.log
  - altibase_qp.log
  - altibase_rp.log
- ALTIBASE HDB performance view
- OS 항목
  - ALTIBASE  HDB 프로세스 상태
  - CPU 사용률
  - 메모리 사용량
  - 디스크 사용량

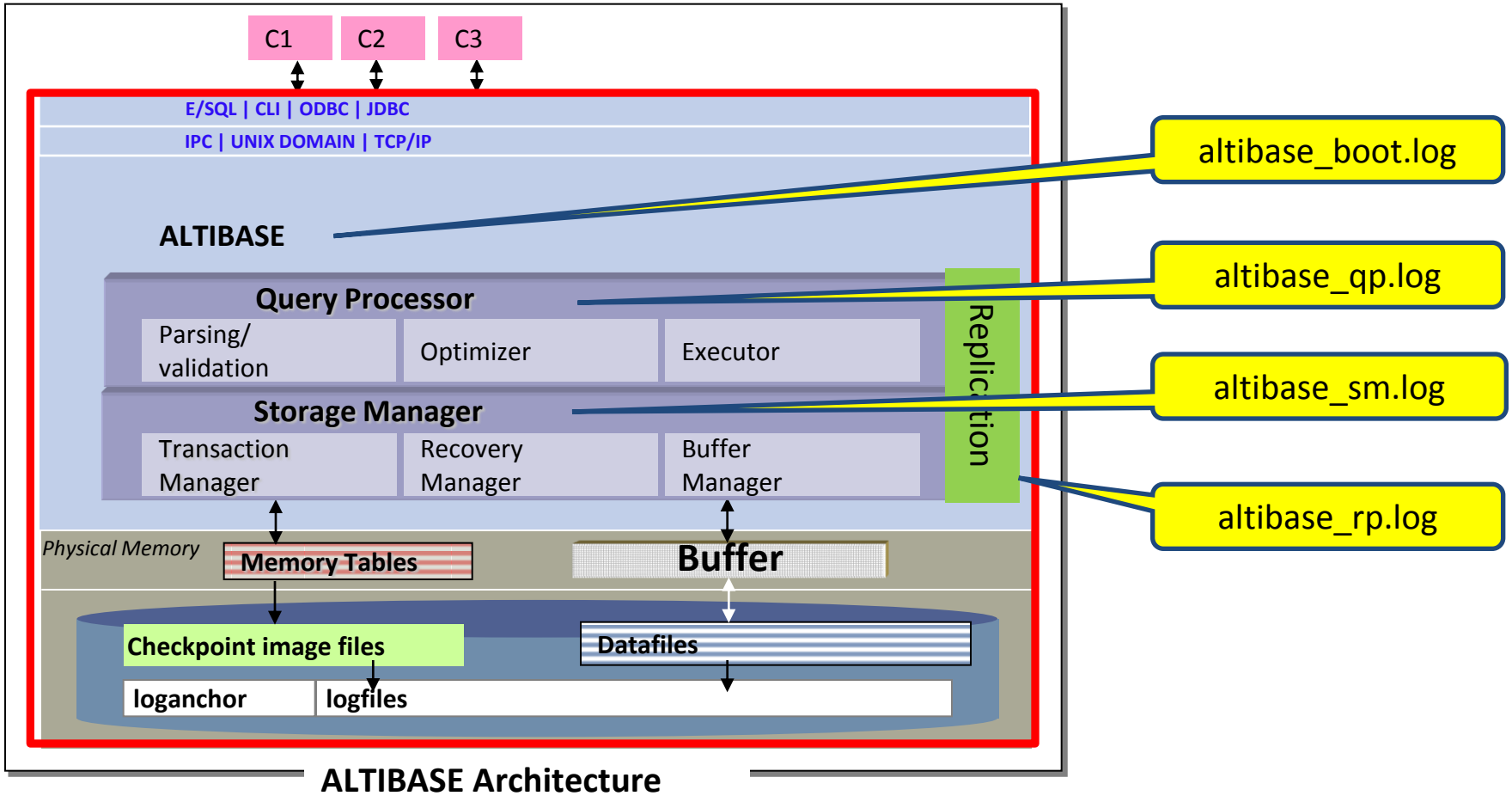
## ❖모니터링 관련 유틸리티

- iSQL
- Orange
- altimon

# Trace files

## ALTIBASE HDB Trace file

- $ALTIBASE_HOME/trc에 위치




ALTIBASE Architecture

# Trace files

## ❖Trace file 종류

| 파일 이름 | 주요 내용 |
|---|---|
| altibase_boot.log | ALTIBASE HDB가 동작된 상태 정보<br>구동 및 종료에 관련된 각종 step 정보<br>구동 중 디스크 부족으로 인한 에러 등 여러 가지 에러 상황<br>비정상 종료가 발생할 경우 Call-Stack 정보 |
| altibase_rp.log | Replication 모듈에서 발생하는 경고 메시지나 트레이스 메시지<br>이중화 상태, 이중화 conflict 등의 메시지 |
| altibase_sm.log | Storage Manager 모듈에서 발생하는 경고 메시지나 트레이스 메시지<br>checkpoint, 백업 등의 메시지 |
| altibase_qp.log | Query Processor 모듈에서 발생하는 경고 메시지나 트레이스 메시지<br>DDL 문의 성공/실패 메시지 |

# Trace file - altibase_qp.log

- **altibase_qp.log에 기록되는 내용**
  - 사용자가 수행한 DDL문에 관한 정보
  - ALTER SYSTEM 명령어로 property를 변경한 정보
- **sample**
  - Tablespace DDL

```
[2012/10/27 13:32:48] [Thread-258] [Level-2]
[EXEC_DDL_BEGIN : create tablespace DISK_TBS DATAFILE 'disk_tbs001.dbf' SIZE 100M autoextend off]
[2012/10/27 13:32:50] [Thread-258] [Level-2]
[EXEC_DDL_END : SUCCESS]
```

  - Table DDL

```
[2012/10/27 14:11:53] [Thread-515] [Level-2]
[EXEC_DDL_BEGIN : truncate table TEST12]
[2012/10/27 14:11:53] [Thread-515] [Level-2]
[EXEC_DDL_END : SUCCESS]
```

  - Property 변경

```
[2012/10/31 11:36:59] [Thread-515] [Level-0]
[SET-PROP] TIMED_STATISTICS=[1]
```

# Trace file - altibase_sm.log

- **altibase_sm.log에 기록되는 내용**
  - 메모리 체크포인트 진행 및 수행 결과
  - 디스크 체크포인트 진행 및 수행 결과
  - 인덱스 생성에 대한 진행 및 수행 결과
  - 백업에 대한 진행 및 수행 결과
  - archive thread의 오류 발생 메시지
  - 기타 디스크 관련 오류 메시지
- **sample**
  - 체크포인트

```
-- CHECKPOINT가 발생한 이유에 대해 기록
   CHECKPOINT BY LOGFILE SWITCH COUNT, CHECKPOINT BY USER, CHECKPOINT BY TIME
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[CHECKPOINT BY TIME(6000 sec)]

-- CHECKPOINT가 시작
[201310/25 19:44:06] [Thread-15986] [Level-9]
[CHECKPOINT-BEGIN]

-- BEGIN CHECKPOINT log를 logfile에 기록
   메모리 DB의 recoveryLSN정보가 기록되고 차후 Recovery를 할 때 이 정보를 이용
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[CHECKPOINT-step2] Write BeginChkpt Log [0,10003,7546027]
```

# Trace file - altibase_sm.log

```
-- 체크포인트가 발생한 시점에 가장 오래된 트랜잭션이 기록한 첫 번째 BeginLog
Active Tx Recovery LSN [0,10003,7546027]

-- 체크포인트가 발생한 시점에 버퍼에 접근한 가장 오래된 트랜잭션이 기록한 첫 번째 BeginLog
Disk Buffer Oldest LSN [0,10003,7546027]

-- 변경된 메모리DB의 DirtyPage들을 내림
   메모리에 대한 트랜잭션로그를 디스크로 쓰고, 변경된 데이터페이지를 Checkpoint Image File에 저장
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[CHECKPOINT-step3] Flush Dirty Page(s)

-- Ping-Pong 체크포인트를 수행하기 때문에 이전에 내렸던 DirtyPage의 개수
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[PRE-DirtyPageCount=2201]

-- 현재 체크포인트 시점에 새롭게 추가된 DirtyPage의 개수
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[NEW-DirtyPageCount=8]

-- 이전 체크포인트와 현재 체크포인트 시점에 중복되는 DirtyPage의 개수
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[DUP-DirtyPageCount=0]

-- DirtyPage를 내리기 전, 트랜잭션 로그를 먼저 디스크로 sync
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
    + Begin Sync For All-LFG - Request LSN [0,10003,7546468]

[2013/10/25 19:44:06] [Thread-15986] [Level-9]
    + End Sync For All-LFG
```

# Trace file - altibase_sm.log

```
-- flush한 DirtyPage 수
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[FLU-DirtyPageCount=2209]

-- 2벌의 데이터파일에 모두 기록되어 DirtyPageList에서 제거된 Page의 수
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[REM-DirtyPageCount=2201]

-- Checkpoint Image 파일들을 모두 sync
[2013/10/25 19:44:06] [Thread-15986] [Level-9]
[CHECKPOINT-step4] sync Database File

-- DISK IO에 대한 통계정보
============================================================
SM IO STAT - Checkpoint
DB SIZE : 1526693888 Byte ( 46591 Page)
LOG SIZE : 4315735241 Byte
TOTAL TIME : 5 s 5402996 us
LOG SYNC TIME: 0 s 56013 us
DB FLUSH TIME: 5 s 5346983us
SYNC TIME : 3 s 3742363 us
WAIT TIME : 0 s 0 us
WRITE TIME: 1 s 1604620 us
LOG IO PERF : 259.682890775062 MB/sec
DB IO PERF : 272.297246877351 MB/sec
============================================================
```

# Trace file - altibase_sm.log

```
-- Checkpoint가 완료됐음을 메모리에 로깅
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
[CHECKPOINT-step5] Write End_Chkpt Log [0,10003,7553341]

-- 체크포인트 과정에서 변경된 페이지에 접근한 트랜잭션이 존재할 수 있고
   체크포인트가 완료된 로그도 디스크로 Sync해야 하기 때문에 다시 한번 메모리상의 로그를 sync
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
[CHECKPOINT-step6] Sync Log File

[2013/10/25 19:44:07] [Thread-15986] [Level-9]
    + Begin Sync For All-LFG - Request LSN [0,10003,7553382]

[2013/10/25 19:44:07] [Thread-15986] [Level-9]
    + End Sync For All-LFG

-- 현재는 의미 없는 로그임으로 무시
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
[CHECKPOINT-step7] Check LogFiles That Is Not Needed

-- sender가 어떤 오류가 발생했을 때 재전송을 시작해야 할 SN 정보를 기록
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
Replication MinSN710440983

-- RecoveryLSN에 대해 디스크 부분에 대한 RecoveryLSN정보를 loganchor 파일에 기록
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
[CHECKPOINT-step8] Update and Flush Log Anchor

-- 메모리상의 로그와 DirtyPage들을 디스크로 기록했음으로 더 이상 복구에 필요하지 않은 로그파일을 삭제
[2013/10/25 19:44:07] [Thread-15986] [Level-9]
Remove Online Log File at LFG [0]: File[9992 ~ 10002]
```

# Trace file - altibase_sm.log

-- 모든 Checkpoint Image 파일 및 디스크 데이터파일에 RedoLSN정보를 기록
[2013/10/25 19:44:08] [Thread-15986] [Level-6]
[CHECK DATABASE SID=0, PPID=0, FID=0]

[2013/10/25 19:44:08] [Thread-15986] [Level-6]
LogAnchor SpaceID=0, SmVersion=84148225, LFGCount=1
DBFileHdr SpaceID=0, SmVersion=84148225, LFGCount=1

[2013/10/25 19:44:08] [Thread-15986] [Level-6]
RedoLSN=control[0,10003,7546027], [0,10003,7546027]

[2013/10/25 19:44:08] [Thread-15986] [Level-6]
CreateLSN=control[0,0,468], [0,0,468]
...

-- 체크포인트의 Begin/End시점의 LSN정보를 로그로 남기고 디스크의 RedoLSN정보를 로그로 남김
[2013/10/25 19:44:08] [Thread-15986] [Level-9]
[CHECKPOINT-summary] BeginChkptLSN=[0,10003,7546027], EndChkptLSN=[0,10003,7553341],
DiskRecLSN=[0,10003,7546027]

-- 현재 체크포인트가 진행되는 시점에서 가장 오래된 트랜잭션이 기록한 첫 번째 트랜잭션로그의 시작위치를 기록
[2013/10/25 19:44:08] [Thread-15986] [Level-9]
Minimum LSN = [0,10003,7546027]

-- 체크포인트가 완료되었음 기록
[2013/10/25 19:44:08] [Thread-15986] [Level-9]
[CHECKPOINT-END]

# Trace file - altibase_sm.log

```
-- 다음 체크포인트까지 체크포인트 쓰레드가 휴식에 들어감
[2013/10/25 19:44:08] [Thread-15986] [Level-9]
Sleep checkpoint thread ( next time : 2011-10-25 21:24:8 )
```

# Trace file - altibase_sm.log

- 백업

```
[2013/11/16 16:06:04] [Thread-1094719840] [Level-9]
MEMORY TABLESPACE DATAFILE /home/altibase/altibase-HDB-server-6.1.1/dbs/SYS_TBS_MEM_DIC-0-0
BACKUP TO /home/altibase/backup/SYS_TBS_MEM_DIC-0-0

[2013/11/16 16:06:08] [Thread-1094719840] [Level-9]
MEMORY TABLESPACE DATAFILE /home/altibase/altibase-HDB-server-6.1.1/dbs/SYS_TBS_MEM_DATA-0-0
BACKUP TO /home/altibase/backup/SYS_TBS_MEM_DATA-0-0

[2013/11/16 16:06:10] [Thread-1094719840] [Level-9]
Starting LogAnchor Backup

[2013/11/16 16:06:10] [Thread-1094719840] [Level-9]
/home/altibase/altibase-HDB-server-6.1.1/logs/loganchor0 BACKUP TO /home/altibase/backup/loganchor0

...

[2013/11/16 16:06:36] [Thread-1094719840] [Level-9]
Waiting logfile7196 to archive

[2013/11/16 16:06:41] [Thread-1094719840] [Level-9]
Database-Level Backup Completed [SUCCESS]
```

# Trace file - altibase_rp.log

- **altibase_rp.log에 기록되는 내용**
  - 이중화 시작/중지 및 장애 상황에 대한 정보
  - conflict가 발생한 경우 해당 질의문과 유형에 대한 정보
  - REPLICATION_LOCK_TIMEOUT에 의한 오류 정보
  - REPLICATION_RECEIVE_TIMEOUT에 의한 오류 정보
- **Sample**
  - 이중화 시작
    - Sender

```
[2013/11/30 14:24:22] [Thread-6516] [Level-0]
[Sender] Replication REP1:0 Start... at [722896995]
```

    - Receiver

```
[2013/11/30 14:24:21] [Thread-6469] [Level-0]
[Receiver] Replication REP1 Started ...
```

# Trace file - altibase_rp.log

- 이중화 중지
  - Sender

```
[2013/11/30 14:27:06] [Thread-6516] [Level-0]
SEND Stop Message!

[2013/11/30 14:27:06] [Thread-6516] [Level-0]
SEND Stop Message SUCCESS!!!

[2013/11/30 14:27:06] [Thread-6516] [Level-0]
[Sender] Stop sender thread REP1:0 at [722897008], Restart SN[722897008]

=> Restart SN은 다음 이중화를 재시작할 때 해당 값이 기록된 로그부터 재전송을 하겠다는 의미
```

  - Receiver

```
[2013/11/30 14:27:06] [Thread-6469] [Level-0]
RECEIVER:REPLICATION STOP MSG arrived!

[2013/11/30 14:27:06] [Thread-6469] [Level-0]
[ReceiverApply] REPLICATION STOP XLog arrived!
[2013/11/30 14:27:06] [Thread-6469] [Level-0]
Normal Stop!

[2013/11/30 14:27:06] [Thread-6469] [Level-0]
[Receiver] Replication REP1 Stopped ...
```

# Trace file - altibase_rp.log

- 이중화 SYNC
  - Sender

```
[2013/11/30 14:31:05] [Thread-7289] [Level-0]
[PJChild] [0] PJ_RUN
[2013/11/30 14:31:05] [Thread-7289] [Level-0]
[PJChild] [0] TABLE: T1, FST: 0, TUPLE: 2                <= sync 할 대상 테이블 및 레코드건수
[2013/11/30 14:31:05] [Thread-7289] [Level-0]
[PJ Child] SEND Stop Message!

[2013/11/30 14:31:05] [Thread-7289] [Level-0]            <= sync 완료 후 sync 중지
[PJ Child] SEND Stop Message SUCCESS!!!

[2013/11/30 14:31:05] [Thread-7289] [Level-0]
[PJChild] [0] PJ_EXIT
[2013/11/30 14:31:07] [Thread-6518] [Level-0]
[Sender] Replication REP1:0 Start... at [722897032]   <=== sender 자동 재 구동
```

  - Receiver

```
[2013/11/30 14:31:04] [Thread-6470] [Level-0]        <= receiver 자동 재 구동
[Receiver] Replication REP1 Started ...
[2013/11/30 14:31:05] [Thread-6727] [Level-0]        <= sync 용 receiver 구동
[Receiver] Replication REP1 Started ...
[2013/11/30 14:31:05] [Thread-6727] [Level-0]
RECEIVER:REPLICATION STOP MSG arrived!

[2013/11/30 14:31:05] [Thread-6727] [Level-0]        <= sync 용 receiver 종료
[ReceiverApply] REPLICATION STOP XLog arrived!
```

# Trace file - altibase_rp.log

- Sender 측 비정상 종료
  Sender 측은 비정상 종료되었으므로 로그를 남길 수 없음
  - Receiver

```
[2013/11/30 14:38:49] [Thread-6470] [Level-0]            <= 단절 감지
ERR-7101a(errno=0) Connection closed

[2013/11/30 14:38:49] [Thread-6470] [Level-0]
ERR-7101a(errno=0) Connection closed

[2013/11/30 14:38:49] [Thread-6470] [Level-0]
ERR-61047(errno=0) [Receiver] REP1 receiver has error in recvXlog()

[2013/11/30 14:38:49] [Thread-6470] [Level-0]
ERR-61048(errno=0) [Receiver] REP1 receiver has recvXLog error in run()

[2013/11/30 14:38:49] [Thread-6470] [Level-0]
ERR-6104b(errno=0) [Receiver] REP1 receiver is ended (by thr_exit)

[2013/11/30 14:38:49] [Thread-6470] [Level-0]
 Error Stop!

[2013/11/30 14:38:49] [Thread-6470] [Level-0]          <= Receiver 중단
[Receiver] Replication REP1 Stopped ...
```

# Trace file - altibase_rp.log

- Sender 복구 후
  - Sender 측

```
[2013/11/30 14:42:13] [Thread-6518] [Level-0]
[Sender] Replication REP1:0 Start... at [722897032]
```

  - Receiver 측

```
[2013/11/30 14:42:12] [Thread-6472] [Level-0]
[Receiver] Replication REP1 Started ...
```

# Trace file - altibase_rp.log

- Receiver 측 비정상 종료
  Receiver 측은 비정상 종료되었으므로 로그를 남길 수 없음
  - Sender

```
[2013/11/30 14:43:46] [Thread-6775] [Level-0]
ERR-7101a(errno=0) Connection closed                    <= 연결 단절을 감지

[2013/11/30 14:43:46] [Thread-6775] [Level-0]
ERR-7101a(errno=0) Connection closed

[2013/11/30 14:43:46] [Thread-6518] [Level-0]          <= Sender를 중단시킴
ERR-620f0(errno=16) [Sender] Stop sender thread REP1:0 at [722899417], Restart SN[722899323]

[2013/11/30 14:43:46] [Thread-6518] [Level-0]          <= 이중화 백업 라인 체크
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2013/11/30 14:43:46] [Thread-6518] [Level-0] ERR-71017(errno=79) Failed to invoke the connect()
system function

[2013/11/30 14:43:46] [Thread-6518] [Level-0] ERR-61012(errno=79) [Sender] Failed to connect to the
peer server

[2013/11/30 14:43:46] [Thread-6518] [Level-0]          <= 다음 재 접속 시도까지
                                                       (REPLICATION_SENDER_SLEEP_TIMEOUT=60초) 만큼 Sleep
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2012/10/31 14:43:46] [Thread-6518] [Level-0]
ERR-61022(errno=79) [Sender] Sender Sleep : 60 seconds
```

# Trace file - altibase_rp.log

- 60초 후에도 Receiver가 복구되지 않는 경우
  장애시점의 로그와 동일하게 Connection 오류가 발생하고 이중화 백업라인에 대해 시도해보고 안되면 다시 Sleep에 들어간다

```
[2011/10/31 14:46:46] [Thread-6518] [Level-0]
ERR-71017(errno=79) Failed to invoke the connect() system function

[2011/10/31 14:46:46] [Thread-6518] [Level-0]
ERR-61012(errno=79) [Sender] Failed to connect to the peer server

[2011/10/31 14:46:46] [Thread-6518] [Level-0]
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2011/10/31 14:46:46] [Thread-6518] [Level-0]
ERR-61022(errno=79) [Sender] Sender Sleep : 60 seconds

[2011/10/31 14:47:46] [Thread-6518] [Level-0]
ERR-71017(errno=79) Failed to invoke the connect() system function

[2011/10/31 14:47:46] [Thread-6518] [Level-0]
ERR-61012(errno=79) [Sender] Failed to connect to the peer server

[2011/10/31 14:47:46] [Thread-6518] [Level-0]
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2011/10/31 14:47:46] [Thread-6518]
[Level-0] ERR-61022(errno=79) [Sender] Sender Sleep : 60 seconds
```

- Receiver 측이 복구된 이후의 로그는 Sender측 복구된 로그와 동일

# Trace file - altibase_rp.log

- Conflict
  - Insert Duplicate 오류

    Insert를 상대편에서 수행하였으나 수신 측에서는 해당하는 PK를 가진 레코드가 이미 존재할 경우 오류를 기록

```
[2013/11/30 15:10:24] [Thread-6482] [Level-2]
ERR-11058(errno=0) The row already exists in a unique index.

[2013/11/30 15:10:24] [Thread-6482] [Level-3]
INSERT INTO SYS.T1 VALUES ( 1, 1 );
```

  - Update Not Found 오류

    Update를 상대편에서 수행하였으나 수신 측에서는 해당하는 PK를 가진 레코드가 존재하지 않을 경우 오류를 기록

```
[2013/11/30 15:08:22] [Thread-7038] [Level-2]
ERR-610f7(errno=0) [Receiver] Unable to find record in executeUpdate() function

[2013/11/30 15:08:22] [Thread-7038] [Level-3]
UPDATE SYS.T1 SET C2 = 101 WHERE C1 = 100;
```

# Trace file - altibase_rp.log

- Update Conflict 오류

  상대편에서 update를 수행했던 시점의 변경대상 컬럼이 가지는 변경 전값이 수신 측의 변경대상 컬럼이 가지는 현재의 값과 다를 경우 오류를 기록

```
[2013/11/30 15:15:41] [Thread-6482] [Level-2]
ERR-61035(errno=0) [Receiver] An update conflict encountered.

[2013/11/30 15:15:41] [Thread-6482] [Level-2]
ERR-61001(errno=0) A conflict has been occurred while executing the received statement.

[2013/11/30 15:15:41] [Thread-6482] [Level-3]
UPDATE SYS.T1 SET C2 = 2 WHERE C1 = 1;
```

  Sender측에서는 Update문에 대해서는 변경대상 컬럼의 (변경 전 값, 변경 후 값)을 함께 보내어 비교한후 Update가 수행될수 있도록 한다

- Delete Not Found 오류

  Delete를 상대편에서 수행하였으나 수신 측에서는 해당하는 PK를 가진 레코드가 존재하지 않을 경우 오류를 기록

```
[2013/11/30 15:09:33] [Thread-7038] [Level-2]
ERR-610f7(errno=0) [Receiver] Unable to find record in executeDelete() function

[2013/11/30 15:09:33] [Thread-7038] [Level-3]
DELETE FROM SYS.T1 WHERE C1 = 100;
```

# Trace file - altibase_rp.log

- REPLICATION_LOCK_TIMEOUT 이 발생할 경우
  어떤 이유로 수신 받은 트랜잭션을 반영하려는 과정에서 레코드에 Lock을 획득하려고 대기하는 시간이 “REPLICATION_LOCK_TIMEOUT”에 정의된 시간(초)동안 Lock을 획득하지 못할경우 아래와 같은 오류 발생

```
[2013/11/30 15:36:01] [Thread-6486] [Level-2]
 ERR-11075(errno=0) The transaction exceeds lock timeout specified by user.

[2013/11/30 15:36:01] [Thread-6486] [Level-3]
UPDATE SYS.T1 SET C2 = -1 WHERE C1 = 1;
```

# Trace file - altibase_rp.log

- REPICLICATION_RECEIVE_TIMEOUT이 발생할 경우
  - sender

```
-- Timeout이 발생한 기록
[2013/11/30 16:13:03] [Thread-6786] [Level-0]
ERR-61075(errno=16) Timeout exceed.

-- 반복적으로 재 접속이 발생
[2013/11/30 16:36:02] [Thread-6529] [Level-0]
ERR-620f0(errno=16) [Sender] Stop sender thread REP1:0 at [785419687], Restart SN[783217166]

[2013/11/30 16:36:07] [Thread-6529] [Level-0]
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2013/11/30 16:36:07] [Thread-6529] [Level-0]
ERR-7101a(errno=0) Connection closed

[2013/11/30 16:36:07] [Thread-6529] [Level-0]
ERR-61003(errno=0) Unable to read from a socket

[2013/11/30 16:36:07] [Thread-6529] [Level-0]
[Sender] getNextLastUsedHostNo: from 192.168.1.131:37585 to 192.168.1.131:37585
[2013/11/30 16:36:07] [Thread-6529] [Level-0]
ERR-61022(errno=0) [Sender] Sender Sleep : 60 seconds

[2013/11/30 16:37:07] [Thread-6529] [Level-0]
ERR-7101a(errno=0) Connection closed

[2013/11/30 16:37:07] [Thread-6529] [Level-0]
ERR-61003(errno=0) Unable to read from a socket
 ...
```

# Trace file - altibase_rp.log

- receiver
  - receiver 측의 lock 대기로 timeout이 발생할 경우

```
[2013/11/30 16:38:07] [Thread-5956] [Level-0]
ERR-61030(errno=11) [Receiver] Failed to build meta information  ...
```

  - sender에 의해 재구동이 되는 형태

```
[2013/11/30 16:45:18] [Thread-6486] [Level-0]
ERR-7101a(errno=32) Connection closed

[2013/11/30 16:45:18] [Thread-6486] [Level-0]
ERR-61048(errno=32) [Receiver] REP1 receiver has recvXLog error in run()

[2013/11/30 16:45:39] [Thread-6486] [Level-0]
ERR-71032(errno=76) Unable to shutdown the socket

[2013/11/30 16:45:39] [Thread-6486] [Level-0]
ERR-61087(errno=76) [Network] Shutdown link operation is failed

[2013/11/30 16:45:39] [Thread-6486] [Level-0]
ERR-6104b(errno=76) [Receiver] REP1 receiver is ended (by thr_exit)

[2013/11/30 16:45:39] [Thread-6486] [Level-0]
Error Stop!

[2013/11/30 16:45:39] [Thread-6486] [Level-0]
[Receiver] Replication REP1 Stopped ...
```

ALTIBASE

ALTIBASE HDB ADVANCED ADMINISTRATION

# 2.1 모니터링

(1) Trace File

**(2) Performance View**

(3) OS 항목

# Performance view

- **ALTIBASE HDB performance view**
  - 시스템 통계 정보
  - session
  - statement
  - lock
  - service thread
  - memory ager
  - buffer
  - logfile
  - tablespace usage
  - table usage
  - memory usage
  - replication

# 시스템 통계 정보

## V$SYSSTAT

- 시스템의 상태정보를 나타냄.
- 3초마다 갱신
- 적절한 주기를 두고 전, 후의 값을 비교하여 각 항목이 얼마만큼 증가했는지 추이를 관찰
- 시스템의 어느 부분에 부하를 많이 줬는지 확인할 때 사용
- 칼럼 정보

| 칼럼 | 내용 |
|---|---|
| NAME | 시스템 각 상태의 이름 |
| VALUE | 각 상태 값의 누적 치 |

- sample 질의

```
iSQL> SELECT TO_CHAR(SYSDATE,'HH:MI:SS'), name, value FROM v$sysstat
    2  WHERE name IN ('logon cumulative', 'execute success count', 'prepare success count',
    3                 'memory table cursor full scan count', 'memory table cursor index scan count',
    4                 'disk table cursor full scan count', 'disk table cursor index scan count',
    5                 'update retry count', 'lock row retry count'
    6                 );
```

# 세션 통계 정보

## V$SESSTAT

- 접속한 세션의 상태정보를 나타냄.
- 접속이 끊겨진 세션의 정보는 사라짐
- 칼럼 정보

| 칼럼 | 내용 |
|---|---|
| SID | 세션 ID |
| NAME | 시스템 각 상태의 이름 |
| VALUE | 각 상태 값의 누적치 |

- sample 질의

```
iSQL> SELECT sid, TO_CHAR(SYSDATE,'HH:MI:SS'), name, value FROM v$sesstat
    2 WHERE name IN ('logon cumulative', 'execute success count', 'prepare success count',
    3                'memory table cursor full scan count', 'memory table cursor index scan count',
    4                'disk table cursor full scan count', 'disk table cursor index scan count',
    5                'update retry count', 'lock row retry count'
    6                );
```

# Session 정보

## V$SESSION

- 현재 접속되어 있는 세션의 정보
- 칼럼 정보

| 칼럼 | 내용 |
|---|---|
| ID | 세션 ID |
| TRANS_ID | 현재 사용하고 있는 트랜잭션 ID |
| TASK_STATE | WATING, READY, EXECUTING, QUEUE WAIT, QUEUE READY , UNKNOWN |
| COMM_NAME | 클라이언트의 접속정보 |
| OPENED_STMT_COUNT | statement의 개수 |
| CLIENT_PID | 클라이언트의 프로세스 ID |
| AUTOCOMMIT_FLAG | autocommit 모드 인지 (0/1) |
| SESSION_STATE | INIT, AUTH, SERVICE READY, SERVICE, END, ROLLBACK, UNKNOWN |
| CURRENT_STMT_ID | 현재 수행중인 statement ID. recursive SQL은 제외 |
| LOGIN_TIME | 클라이언트가 접속한 시간<br>TO_CHAR(TO_DATE('1970010109','yyyymmddhh24') + login_time/60/60/24, 'yyyymmdd hh:mi:ss') |

# Session 정보

## V$SESSIONMGR

- 비정상 종료된 세션의 누적 통계를 확인

| 칼럼 | 내용 |
|---|---|
| IDLE_TIMEOUT_COUNT | 발생된 idle timeout의 횟수 |
| QUERY_TIMEOUT_COUNT | 발생된 query timeout의 횟수 |
| DDL_TIMEOUT_COUNT | 발생된 DDL timeout의 횟수 |
| FETCH_TIMEOUT_COUNT | 발생된 fetch timeout의 횟수 |
| UTRANS_TIMEOUT_COUNT | 발생된 utrans timeout의 횟수 |
| SESSION_TERMINATE_COUNT | sysdba에 의해 강제로 연결이 끊긴 횟수 |

# Session 정보

- sample 질의
  - 접속 유형 별 세션의 수

```
iSQL> SELECT  SUBSTR(comm_name, 1,4) con_type, COUNT(*) cnt
      2 FROM   v$session
      3 GROUP BY SUBSTR(comm_name, 1,4);
CON_TYPE                                                 CNT
-------------------------------------------------------------
TCP                                                      105
UNIX                                                     1
IPC                                                      100
```

  - 비정상 종료된 세션의 누적치

```
iSQL> SELECT  login_timeout_count,  idle_timeout_count, query_timeout_count,
      2           fetch_timeout_count, utrans_timeout_count, session_terminate_count
      3 FROM  v$sessionmgr;

LOGIN_TIMEOUT_COUNT      : 0
IDLE_TIMEOUT_COUNT       : 0
QUERY_TIMEOUT_COUNT      : 10
FETCH_TIMEOUT_COUNT      : 0
UTRANS_TIMEOUT_COUNT     : 0
SESSION_TERMINATE_COUNT  : 0
```

# Statement 정보

## V$STATEMENT

- 접속이 유효한 세션에서 실행한 질의 정보
- 어떤 SQL문이 오래 수행되는지 확인할 때 사용
- time 정보를 수집하기 위해서는 TIMED_STATISTICS=1로 설정
- time 정보는 microsec(10-6 초) 단위
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| ID | statement ID |
| SESSION_ID | 세션 ID |
| TX_ID | 해당 질의가 속해있는 트랜잭션 ID |
| QUERY | 실제 질의 문장 (16Kbyte까지만 표현) |
| TOTAL_TIME | 질의가 최종 수행된 시점의 전체 소요시간 |
| EXECUTE_TIME | 질의가 DBMS 내부에서 처리된 순수 소요시간<br>(SELECT의 경우 첫번째 FETCH가 일어나지 전까지 수행된 시간) |
| FETCH_TIME | 최종 질의에 대해 DB와 프로그램과 발생한 통신의 누적 소요시간 |
| PARSE_TIME | parse 하는데 소요된 시간 |
| VALIDATE_TIME | validation 체크하는데 소요된 시간 |

# Statement 정보

| 칼럼 | 설명 |
|---|---|
| EXECUTE_SUCCESS | 동일한 질의가 성공적으로 수행된 누적횟수(동일 세션에서 누적치) |
| PROCESS_ROW | INSERT/UPDATE/DELETE/MOVE 문으로 처리된 레코드 수 |
| EVENT | 대기 이벤트 명 |
| WAIT_TIME | 대기로 인해 소모된 시간을 의미하며 millisecond 단위 |
| SECOND_IN_TIME | 대기로 인해 소모된 시간을 의미하며 second단위 |

# Statement 정보

- sample 질의
  - 전체 statement 개수

```
iSQL> SELECT count(*) stmt_cnt FROM v$statement;
```

  - 쿼리 정보

```
iSQL> ALTER SYSTEM SET timed_statistics=1;
iSQL> SELECT session_id, id stmt_id, tx_id,
    2           (parse_time+validate_time+optimize_time) prepare_time, fetch_time, execute_time,
    3           total_time, execute_flag,
    4           DECODE(last_query_start_time, 0, '-', TO_CHAR(TO_DATE('1970010109', 'yyyymmddhh')
    5          + last_query_start_time / (24*60*60), 'mm/dd hh:mi:ss'))
    6           last_start_time, NVL(LTRIM(query), 'NONE') query
    7 FROM v$statement
    8 ORDER BY execute_time DESC ;
```

# Lock 정보

## V$LOCK

- lock에 관한 정보
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| LOC_ITEM_TYPE | LOCK이 획득 된 대상의 종류 (TBL, TBS, DBF) |
| TBS_ID | LOCK이 획득 된 대상이 속한 테이블스페이스 ID |
| TABLE_OID | LOCK이 획득 된 테이블의 OID |
| TRANS_ID | 트랜잭션 ID |
| LOCK_DESC | LOCK 유형 (X_LOCK, IX_LOCK, IS_LOCK) |
| IS_GRANT | 테이블 level의 LOCK을 획득했는지 여부 (1: 획득 0: 대기) |

# Lock 정보

- **V$LOCK_WAIT**
  - 트랜잭션 간의 lock 대기 정보
  - 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| TRANS_ID | 트랜잭션의 고유번호 |
| WAIT_FOR_TRANS_ID | TRANS_ID가 대기하고 하는 트랜잭션의 고유번호 |

# Lock 정보

## V$TRANSACTION

- 현재 수행중인 모든 트랜잭션에 대한 정보
- 주로 V$LOCK, V$MEMGC등 다른 뷰와 조인해서 사용
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| ID | 트랜잭션 ID |
| SESSION_ID | 세션 ID |
| STATUS | 0: BEGIN, 1: PRECOMMIT, 2: COMMIT_IN_MEMORY, 3: COMMIT, 4: ABORT, 5: BLOCKED, 6: END |
| LOG_TYPE | 0: 일반, 1: 이중화 관련 |
| FIRST_UPDATE_TIME | 최초 변경이 일어난 시각. TO_CHAR(TO_DATE('1970010109','YYYYMMDDHH') + first_update_time / (60*60*24), 'MM/DD HH:MI:SS')) |
| DDL_FLAG | 0: non-DDL, 1: DDL |
| FIRST_UNDO_NEXT_LSN_FILENO | 트랜잭션이 처음 기록한 로그의 파일 번호 |
| UPDATE_SIZE | UPDATE 시 작성된 로그의 크기. LOCK_ESCALATION_MEMORY_SIZE 값보다 커지면 in-place MVCC로 동작 |
| MEMORY_VIEW_SCN | 메모리 테이블에 대해 열려있는 커서의 view SCN 중 가장 작은 값 |

# Lock 정보

- sample 질의
  - lock을 잡고 있는 질의 확인

```
iSQL> SELECT tx.id TX_ID, lw.wait_for_trans_id BLOCKED_TX_ID, l.lock_desc,
    2         DECODE(tx.log_type, 0, st.db_username, 'REPLICATION') USER_NAME,
    3         DECODE(tx.first_update_time, 0, '0', to_char(to_date('1970010109','YYYYMMDDHH')
    4         + tx.first_update_time / (60*60*24), 'MM/DD HH:MI:SS')) FIRST_UPDATE_TIME,
    5         DECODE(tx.status, 0, 'BEGIN', 1, 'PRECOMMIT', 2, 'COMMIT_IN_MEMORY',3, 'COMMIT',
    6          4, 'ABORT', 5, 'BLOCKED', 6, 'END') STATUS, st.query current_query
    7  FROM v$transaction tx, v$lock l  LEFT OUTER JOIN v$lock_wait lw ON l.trans_id = lw.trans_id
    8         LEFT OUTER JOIN (SELECT st.query,tx_id, ss.db_username
    9                          FROM v$statement st, v$session ss
   10                          WHERE ss.id = st.session_id ) st
   11         ON l.trans_id = st.tx_id
   12 WHERE tx.id = l.trans_id;

TX_ID              : 103489
BLOCKED_TX_ID      :
LOCK_DESC          : IX_LOCK
USER_NAME          : SYS
FIRST_UPDATE_TIME  : 09/02 14:42:35
STATUS             : BEGIN
CURRENT_QUERY      : update t1 set c1=1 where c1 between 1.11  and 1.112
TX_ID              : 4288
BLOCKED_TX_ID      : 103489
LOCK_DESC          : IX_LOCK
FIRST_UPDATE_TIME  : 0
STATUS             : BLOCKED
CURRENT_QUERY      : update t1 set c1=1 where c1 =1.11
```

# Service Thread 정보

## V$SERVICE_THREAD

- service thread에 관한 정보
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| TYPE | service thread의 접속 방법<br>SOCKET: TCP 혹은 Unix Domain 방식<br>IPC: IPC 방식 |
| STATE | service thread의 현재 상태<br>NONE: service thread가 초기화된 상태<br>POLL: service thread가 이벤트를 기다리고 있는 상태<br>QUEUE-WAIT: service thread가 queue를 대기하는 상태<br>EXECUTE: service thread가 statement를 수행중인 상태 |
| RUN_MODE | service thread의 운영모드<br>SHARED: TCP로 연결된 작업을 처리, DEDICATED: IPC로 연결된 작업을 처리 |
| SESSION_ID | service thread가 수행중인 session ID |
| STATEMEMT_ID | service thread가 수행중인 statement ID |
| TASK_COUNT | service thread에 할당된 session의 개수 |

# Service Thread 정보

- sample 질의
  - service thread의 상태 확인

```
iSQL> SELECT rpad(type, 30), state, count(*)
      2 FROM v$service_thread
      3 GROUP BY type, state;
RPAD(TYPE, 30)                 STATE        COUNT
------------------------------------------------------------------------
SOCKET                         EXECUTE      1
SOCKET                         POLL         7
IPC                            EXECUTE      1
IPC                            POLL         9
```

# Memory Ager 정보

## V$MEMGC

- memory garbage collection 정보를 확인
- aging 할 대상이 증가하는지 확인할 때 사용
- V$TRANSACTION과 V$STATEMENT와 조인하여 GC가 대기하는 트랜잭션이 수행중인 질의문을 찾을 때 사용
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| GC_NAME | GC 이름<br>MEM_LOGICAL_AGER는 인덱스의 old version을 삭제하는 GC<br>MEM_DELTHR는 테이블 레코드의 old version을 삭제하는 GC |
| MINMEMSCNINITX | 메모리 관련 트랜잭션 중 가장 작은 view SCN<br>ALTIBASE 가장 오래된 old version의 번호 |
| ADD_OID_CNT | aging을 위해 추가된 OID list 개수 |
| GC_OID_CNT | aging 된 OID list 개수 |

※**GC GAP(ADD_OID_CNT - GC_OID_CNT) 값이 증가되고 있다면, aging 할 대상이 증가**

# Memory Ager 정보

- sample 질의
  - memory ager의 gap 증가 확인

```
iSQL> SELECT gc_name, add_oid_cnt, gc_oid_cnt , add_oid_cnt - gc_oid_cnt gcgap
    2 FROM v$memgc;
GC_NAME                 ADD_OID_CNT           GC_OID_CNT            GCGAP
------------------------------------------------------------------------
MEM_LOGICAL_AGER        1275                  1275                  0
MEM_DELTHR              1275                  1275                  0
```

# Logfile 정보

- **V$LFG**
  - 로그파일이 미처 생성되지 못해 트랜잭션이 로그파일이 생성될 때까지 기다렸는지 확인할 때 사용
  - 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| CUR_WRITE_LF_NO | 기록 로그 파일번호 |
| LF_PREPARE_COUNT | 미리 생성한 로그파일의 수 |
| LF_PREPARE_WAIT_COUNT | 로그 스위치 시 대기한 횟수 |
| END_LSN_FILE_NO | restart redo가 시작될 LSN의 파일번호 |
| END_LSN_OFFSET | restart redo가 시작될 LSN의 오프셋 |

# Logfile 정보

- sample 질의
  - logfile switch 시 트랜잭션이 얼마나 대기했는지 확인

```
iSQL> SELECT lf_prepare_wait_count
    2  FROM v$lfg;
LF_PREPARE_WAIT_COUNT
------------------------
0
```

# Tablespace 정보

- **V$MEM_TABLESPACES**
  - 메모리 테이블스페이스에 대한 정보
  - 메모리 테이블스페이스의 사용량을 확인하고 싶을 때 사용
  - 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| SPACE_ID | 테이블스페이스 ID |
| SPACE_NAME | 테이블스페이스 이름 |
| MAXSIZE | 테이블스페이스 최대 크기<br>DECODE(MAXSIZE, 0, ALLOC_PAGE_COUNT*PAGE_SIZE, MAXSIZE) |
| ALLOC_PAGE_COUNT | 테이블스페이스의 전체 페이지 개수 |
| FREE_PAGE_COUNT | 테이블스페이스의 free 페이지 개수 |

# Tablespace 정보

- sample 질의
  - memory tablespace의 사용률을 조회

```
iSQL> SELECT space_id, space_name, autoextend_mode,
    2           DECODE(maxsize, 140737488322560, 'UNDEFINED',
    3                       0, alloc_page_count*32/1024,
    4                       maxsize/1024/1024)  'MAX(M)',
    5          alloc_page_count * 32/1024  'TOTAL(M)',
    6          (alloc_page_count-free_page_count)*32/1024 'ALLOC(M)',
    7          ((alloc_page_count-free_page_count)*32/1024)
    8         /DECODE(maxsize, 0, alloc_page_count*32/1024, maxsize/1024/1024)  'USAGE(%)'
    9  FROM v$mem_tablespaces;
```

- MAX(M) - 메모리 테이블스페이스 maxsize. 테이블스페이스 생성 시 지정
- TOTAL(M) - 메모리 테이블스페이스가 현재까지 할당 받은 크기
- ALLOC(M) - 메모리 테이블 스페이스가 현재까지 할당 받은 페이지 중 빈 페이지를 제외한 사용 공간
- USAGE(%) - 메모리 테이블스페이스가 최대로 할당할 수 있는 크기 대비 사용중인 공간에 대한 백분율

# Tablespace 정보

## V$TABLESPACES

- 디스크 테이블스페이스에 대한 정보
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| ID | 테이블스페이스 ID |
| NAME | 테이블스페이스 이름 |
| TYPE | 테이블스페이스 타입 |
| STATE | 테이블스페이스 상태 |
| SEGMENT_MANAGEMENT | 테이블스페이스에서 세그먼트를 생성할 때 어떤 타입으로 생성할 것인지 나타냄(AUTO, BITMAP, CIRCULAR) |
| TOTAL_PAGE_COUNT | 테이블스페이스의 전체 페이지 개수 |
| ALLOCATE_PAGE_COUNT | 테이블스페이스에 할당된 페이지 개수 |
| PAGE_SIZE | 테이블스페이스의 페이지 크기 |
| EXTENT_PAGE_COUNT | extent의 페이지 개수 |

- 테이블스페이스의 사용률

  V$TABLESPACE의 ALLOCATE_PAGE_COUNT / V$DATAFILE의 SUM(DECODE(MAXSIZE, 0, CURRSIZE, MAXSIZE))

# Datafile 정보

## V$DATAFILES

- 디스크 테이블스페이스의 데이터파일에 대한 정보
- 주로 데이터파일의 사용량을 확인하고 싶을 때 사용
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| ID | 데이터파일 ID |
| NAME | 데이터파일 경로와 이름 |
| SPACEID | 데이터파일이 속한 테이블스페이스 ID |
| MAXSIZE | 데이터파일 생성 시 지정한 MAXSIZE.<br>AUTOEXTEND OFF 이면 0 |
| INITSIZE | 데이터파일 생성 시 지정한 SIZE |
| NEXTSIZE | 데이터파일 생성 시 지정한 NEXT<br>AUTOEXTEND OFF 이면 0 |
| CURRSIZE | 데이터파일의 현재 크기<br>AUTOEXTEND OFF이면 INITSIZE와 같음 |
| AUTOEXTEND | 0: OFF, 1: ON |
| STATE | 1: 오프라인, 2: 온라인, 4: 백업시작, 8: 백업종료, 128: 삭제(dropped) |

# Datafile 정보

- sample 질의
  - disk tablespace의 데이터파일 별 사용량 확인

```
iSQL> SELECT b.name tbs_name, a.id 'FILE#', a.name datafile_name,
    2           currsize*8/1024 'ALLOC(M)',
    3           ROUND(CASE2(a.maxsize=0, currsize, a.maxsize)*8/1024) 'MAX(M)',
    4           DECODE(autoextend, 0, 'OFF', 'ON') 'AUTOEXTEND'
    5 FROM v$datafiles a,
    6         v$tablespaces b
    7 WHERE b.id = a.spaceid
    8 ORDER BY b.name, a.id;
```

- ALLOC(M) - 데이터파일의 현재 크기
- MAX(M) - 데이터파일이 AUTOEXTEND ON으로 생성되었을 경우에 확장될 수 있는 최대 크기
  AUTOEXTEND OFF 로 생성되었다면 생성 시 크기

# Memory Table 정보

- **V$MEMTBL_INFO**
  - 메모리 테이블 정보
  - 어느 테이블이 메모리를 많이 사용하는지 확인하고 싶을 때 사용
  - 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| TABLESPACE_ID | 테이블스페이스 ID |
| TABLE_OID | 테이블 식별자. system_.sys_tables_와 조인하여 table_name 확인 |
| FIXED_ALLOC_MEM | 테이블에서 할당한 고정 영역의 메모리 크기 |
| FIXED_USED_MEM | 테이블에서 실제 사용하고 있는 고정 영역의 메모리 크기 |
| VAR_ALLOC_MEM | 테이블에서 할당한 가변 영역의 메모리 크기 |
| VAR_USED_MEM | 테이블에서 실제 사용하고 있는 가변 영역 메모리 크기 |
| UNIQUE_VIOLATION_COUNT | UNIQUE 제약조건이 위반된 횟수 |
| UPDATE_RETRY_COUNT | UPDATE 시 재시도한 횟수 |
| DELETE_RETRY_COUNT | DELETE 시 재시도한 횟수 |

- 할당공간은 (FIXED_ALLOC_MEM+VAR_ALLOC_MEM),
  사용공간은 (FIXED_USED_MEM+VAR_USED_MEM)을 질의

# Memory Table 정보

- sample 질의
  - 휘발성 테이블스페이스와 메모리 테이블스페이스에 속한 테이블의 실제 사용 메모리를 확인

```
iSQL> SELECT a.user_name, b.table_name, d.name tablespace_name,
    2           (c.fixed_alloc_mem + c.var_alloc_mem)/(1024*1024) 'ALLOC(M)',
    3           (c.fixed_used_mem + c.var_used_mem)/(1024*1024) 'USED(M)',
    4           (c.fixed_used_mem + c.var_used_mem)/(c.fixed_alloc_mem + c.var_alloc_mem)*100
    5                'EFFICIENCY(%)'
    6 FROM system_.sys_users_ a,  system_.sys_tables_ b, v$memtbl_info c, v$tablespaces d
    7 WHERE a.user_name <> 'SYSTEM_'
    8 AND b.table_type = 'T'
    9 AND a.user_id = b.user_id
   10 AND b.table_oid = c.table_oid
   11 AND b.tbs_id = d.id
   12 ORDER BY 4 DESC ;
```

- ALLOC(M) - 테이블이 할당 받은 메모리 합계
- USED(M) - 테이블이 할당 받은 페이지중에서 “실제로 데이터가 적재된 페이지”의 메모리 합계
  예를 들어, ALLOC이 100M 크기인 메모리 테이블에 전체 DELELE를 수행하면 ALLOC은 100M로 변함없으나 USED는 0에 가깝게 됨
- EFFICIENCY(%) - 테이블이 소유한 페이지중에서 “실제로 데이터가 적재된 페이지”에 대한 백분율로 공간 효율성을 나타냄

# Disk Table 정보

## V$DISKTBL_INFO

- 디스크 테이블 정보
- 어느 테이블이 디스크를 많이 사용하는지 확인하고 싶을 때 사용
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| TABLESPACE_ID | 테이블스페이스 ID |
| TABLE_OID | 테이블 식별자. system_.sys_tables_와 조인하여 table_name 확인 |
| DISK_PAGE_CNT | 테이블에서 데이터를 갖고 있는 페이지 개수 |

- sample 질의

```
iSQL> SELECT user_name, a.table_name, d.name tbs_name,
    2           ROUND((b.disk_page_cnt*8)/1024) 'ALLOC(M)'
    3 FROM system_.sys_tables_ a, v$disktbl_info b, system_.sys_users_ c, v$tablespaces d
    4 WHERE a.table_oid = b.table_oid
    5 AND a.user_id = c.user_id
    6 AND a.tbs_id=d.id
    7 AND c.user_name <> 'SYSTEM_'
```

# Replication 정보

## V$REPSENDER

- 이중화 sender 정보
- 이중화 sender가 동작 중일 때만 확인 가능
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| REP_NAME | 이중화 이름 |
| STATUS | sender의 현재 상태<br>0: STOP, 1: RUN, 2: RETRY |
| NET_ERROR_FLAG | 네트워크 오류 여부<br>0: OK 1: ERROR |
| REPL_MODE | 이중화가 설정된 모드<br>EAGER, LAZY |
| XSN | remote로 전송한 로그의 일련번호 |
| START_FLAG | 이중화 start 시 지정한 옵션<br>NORMAL: 0, QUICK: 1, SYNC: 2, SYNC_ONLY: 3, SYNC RUN : 4,<br>SYNC END: 5, RECOVERY from Replication : 6, OFFLINE: 7 |

# Replication 정보

## V$REPRECEIVER

- 이중화 receiver 정보
- remote 서버의 sender가 start되면 확인 가능
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| REP_NAME | 이중화 이름 |
| APPLY_XSN | Receiver가 적용중인 로그의 SN |
| INSERT_SUCCESS_COUNT | Receiver가 성공적으로 적용한 INSERT 로그레코드의 수 |
| INSERT_FAILURE_COUNT | Receiver가 적용에 실패한 INSERT 로그레코드의 수 |
| UPDATE_SUCCESS_COUNT | Receiver가 성공적으로 적용한 UPDATE 로그레코드의 수 |
| UPDATE_FAILURE_COUNT | Receiver가 적용에 실패한 UPDATE 로그레코드의 수 |
| DELETE_SUCCESS_COUNT | Receiver가 성공적으로 적용한 DELETE 로그레코드의 수 |
| DELETE_FAILURE_COUNT | Receiver가 적용에 실패한 DELETE 로그레코드의 수 |

- SUCCESS_COUNT와 FAILURE_COUNT는 TIMED_STATISTICS=1 로 설정되어있는 경우만 확인 가능

# Replication 정보

## V$REPGAP

- 이중화 sender의 작업 로그와 local 서버에 생성된 최근 로그 파일간의 차이
- 이중화가 정상적으로 수행되고 있는지, 밀리지는 않는지 확인할 때 사용
- 이중화 sender가 동작 중일 때만 확인 가능
- 칼럼 정보

| 칼럼 | 설명 |
|---|---|
| REP_NAME | 이중화 이름 |
| REP_SN | remote로 전송한 로그의 일련번호 |
| REP_LAST_SN | local 서버에서 발생한 트랜잭션에 의한 가장 최근의 로그 일련번호 |
| REP_GAP | 얼마나 더 remote로 보내야 하는가?<br>REP_LAST_SN-REP_SN |
| READ_FILE_NO | 현재 읽고 있는 로그 파일 번호 |
| START_FLAG | 이중화 start 시 지정한 옵션<br>NORMAL: 0, QUICK: 1, SYNC: 2, SYNC_ONLY: 3, SYNC RUN : 4,<br>SYNC END: 5, RECOVERY from Replication : 6, OFFLINE: 7 |

# Replication 정보

- sample 질의
  - sender 확인

```
iSQL> SELECT rep_name, status FROM v$repsender;
REP_NAME                                          STATUS
-------------------------------------------------------------
REP1                                              1
```

**=> status가 1이 아닐 경우 이중화는 정상적이 않음**

  - receiver 확인

```
iSQL> SELECT rep_name FROM v$repreceiver;
REP_NAME
---------------------------------
REP1
```

**=> 데이터가 조회되지 않을 경우 receiver가 정상적이지 않음**

  - 이중화 갭 확인

```
iSQL> SELECT rep_name, rep_last_sn, rep_sn,  rep_last_sn-rep_sn FROM v$repgap;
REP_NAME      REP_LAST_SN         REP_SN              REP_LAST_SN-REP_SN
-------------------------------------------------------------------------
REP           12638584            -1                  12638585
```

**=> rep_sn 값이 -1 이면 원격 서버에 이중화를 생성하지 않음**
**rep_last_sn-rep_sn 값이 증가하는데 rep_sn 값이 변하지 않으면 이중화 장애**

ALTIBASE HDB ADVANCED ADMINISTRATION

# 2.1 모니터링

(1) Trace File
(2) Performance View
**(3) OS 항목**

# OS 항목

## ALTIBASE 프로세스 체크

```
Shell> ps -ef | grep "altibase -p" | grep -v grep
altibase   11300     1  0 Sep22 ?       00:00:10 /altibase/altibase_home/bin/altibase -p boot from admin
```

## CPU 사용률

```
Shell> ps -o pcpu -p 11300
%CPU
 10.1
```

## 메모리 사용률

```
Shell> ps -o vsz -p 11300
 VSZ
8447864
```

## 디스크 사용률

```
Shell> df -k
 Filesystem         1K-blocks      Used    Available Use% Mounted on
/dev/sda1           20641788    1625008   17968140   9% /
none                8211648     3924128    4287520  48% /dev/shm
/dev/sdb1         1663084532  1395923292  182681432  89% /home
/dev/sdb2         1701178400   462146088 1152617476  29% /alti_data
/dev/sda3          386418928   101322240  265467668  28% /alti_logs
/dev/sda5           20641788     669688   18923460   4% /opt
…
```

# OS 항목

## ❖ 시스템 로그

➢ 시스템 로그를 통해 ALTIBASE HDB가 OS에 의해 종료되었는지 확인

| 운영제제 | 확인할 시스템 로그 |
|---|---|
| SUN | /var/adm/message 파일 |
| HP | /var/adm/syslog/syslog.log 파일 |
| AIX | errpt -a |
| LINUX | /var/log/message 파일 |

ALTIBASE HDB ADVANCED ADMINISTRATION

# 2.2 ALTIBASE 접속장애

(1) 신규 연결 불가
(2) 기존 연결 단절
(3) ALTIBASE 비정상 종료시
대응절차

# ALTIBASE 접속 장애

## ❖현상

- ➢ 새로운 연결 불가
  - 최대 연결 개수 제한
  - ALTIBASE process가 내려감
  - network 장애
- ➢ 기존 연결의 단절
  - timeout 설정에 의한 연결 단절
  - 방화벽, L4, 네트웍 설정에 의한 연결 단절

## ❖확인사항

- ➢ 새로운 연결 불가
  - altibase_boot.log에서 에러메시지 확인
  - 현재 세션의 수 확인
  - IPC로 연결된 세션 수 확인
  - OS 설정 확인
- ➢ 기존 연결단절
  - altibase_boot.log에 timeout 관련 에러메시지 확인
  - 네트웍 및 L4 설정 확인

# 신규 연결 불가 - 프로퍼티 값에 도달

## ❖MAX_CLIENT, IPC_CHANNEL_COUNT 프로퍼티에 도달하여 접속 불가

- MAX_CLIENT
  - 동시에 접속할 수 있는 최대 클라이언트의 개수
  - 이 값을 초과할 경우 altibase_boot.log에 다음의 메시지가 기록

```
ERR-41059(errno=16) Task pool overflow. Check properties.
Dispatcher failed callback
```

  - netstat 명령어로 ALTIBASE의 연결된 클라이언트 수 확인

```
Shell> netstat -an | grerp 20300
```

- IPC_CHANNEL_COUNT
  - 서버와 동시에 연결될 최대 IPC 연결 개수
  - IPC로 연결 시 이 값을 초과할 때 altibase_boot.log에 다음의 메시지가 기록

```
ERR-7108b(errno=16) No more IPC channel (MAX=512, USED=512, BUFSIZE=32768)
Dispatcher failed callback
```

  - IPC 연결 개수 확인

```
iSQL> SELECT COUNT(*) FROM v$session
      2 WHERE comm_name LIKE 'IPC%';
```

# 기존 연결의 단절

## ❖TIMEOUT 설정 값을 초과했을 경우 연결이 단절되는 경우가 발생

➢ TIMEOUT의 종류

| 프로퍼티 | 설명 | 기본값 |
|---|---|---|
| UTRANS_TIMEOUT | DML 수행 후 COMMIT/ROLLBACK이 수행될 때까지의 트랜잭션 시간에 대한 timeout | 3600 |
| QUERY_TIMEOUT | 질의를 수행하는 전체 시간에 대한 timeout | 600 |
| FETCH_TIMEOUT | SELECT 수행 시 서버로부터 fetch 한 후 다음 fetch할 때까지 사이에 대한 timeout | 60 |
| IDLE_TIMEOUT | idle 상태에 대한 timeout | 0(무제한) |
| TIMEOUT | DB에 접속을 시도하는 시간에 대한 timeout | 3 |
| CONNECTION_TIMEOUT | DB에 접속된 상태에서 네트웍 상의 송.수신간에 블로킹 된 상태의 timeout, client에서 server로 SQL 수행 요청을 보냈는데, server 측으로부터 응답을 대기하는 time | 0(무제한) |

# 기존 연결의 단절

- timeout 시 altibase_boot.log에 남는 메시지
  - UTRANS_TIMEOUT, FETCH_TIMEOUT, QUERY_TIMEOUT, IDLE_TIMEOUT

```
[2011/10/06 13:10:35] [Thread-182894171744] [Level-1]
[Notify : UTrans Timeout] Session Closed by Server : Session ID = 53
CLIENT_INFO => TCP 127.0.0.1:3992(PID : 13645)
Time Limit => 3
Running Time => 5
Last Query => insert into t1 select * from t1 limit 1
Caused by Transaction => 82368
```

- UTRANS_TIMEOUT, FETCH_TIMEOUT, IDEL_TIMEOUT 발생 시 기존 연결이 단절되므로 쿼리를 튜닝하거나 해당 프로퍼티를 변경해줌

# 기존 연결의 단절

## ❖그 외 확인사항

- 방화벽 설정
- L4 설정
- WAS의 Connection 관련 설정
- OS의 네트워크 설정

# Thank you!

education@altibase.com

Copyright©2013 Altibase.corp All rights reserved

ALTIBASE