프로젝트 보고서_4주차

1. SELECT문

1.1. 수행

수행내역
전체 칼럼 조회
연산자 조회
JOIN문 조회 (JOIN사용 X)
JOIN문 조회 (JOIN 사용O)
집합연산자 조회
집계함수 조회
중복하지 않게 조회

1.2. 결과

1. 전체 칼럼 조회

```
SELECT * FROM DEPARTMENT;
SELECT * FROM EMPLOYEE;
```

#	DEPTNO	DNAME	LOC
1	10	ACCOUN...	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERAT...	BOSTON

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO
1	7,499	ALLEN	SALESMAN	7,839	0081-0...	1,600	30
2	7,521	WARD	SALESMAN	7,698	0081-0...	1,250	30
3	7,566	JONES	MANAGER	7,839	0081-0...	2,975	20
4	7,654	MARTIN	SALESMAN	7,698	0081-0...	1,250	30
5	7,698	BLAKE	MANAGER	7,839	0081-0...	2,850	30
6	7,782	CLARK	MANAGER	7,839	0081-0...	2,450	10
7	7,788	SCOTT	ANALTST	7,566	0082-1...	3,000	20
8	7,839	KING	PRESIDENT	<NULL>	0081-1...	5,000	10
9	7,844	TURNER	SALESMAN	7,698	0081-0...	1,500	30
10	7,876	ADAMS	CLERK	7,788	0083-0...	1,100	20
11	7,900	JAMES	CLERK	7,698	0081-1...	950	30
12	7,934	MILLER	CLERK	7,782	0082-0...	1,300	10

2. 연산자 조회

```
SELECT ENAME, SALARY*1.05 NEW_SALARY FROM EMPLOYEE WHERE DEPTNO=10;
```

#	ENAME	NEW_SALARY
1	CLARK	2572.5
2	KING	5,250
3	MILLER	1,365

3. JOIN문 조회 (JOIN사용 X)

```
SELECT ENAME, SALARY, DEPARTMENT.* FROM EMPLOYEE, DEPARTMENT WHERE EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
```

#	ENAME	SALARY	DEPTNO	DNAME	LOC
1	ALLEN	1,600	30	SALES	CHICAGO
2	WARD	1,250	30	SALES	CHICAGO
3	JONES	2,975	20	RESEARCH	DALLAS
4	MARTIN	1,250	30	SALES	CHICAGO
5	BLAKE	2,850	30	SALES	CHICAGO
6	CLARK	2,450	10	ACCOUN...	NEW YORK
7	SCOTT	3,000	20	RESEARCH	DALLAS
8	KING	5,000	10	ACCOUN...	NEW YORK
9	TURNER	1,500	30	SALES	CHICAGO
10	ADAMS	1,100	20	RESEARCH	DALLAS
11	JAMES	950	30	SALES	CHICAGO
12	MILLER	1,300	10	ACCOUN...	NEW YORK

4. JOIN문 조회 (JOIN 사용O)

```
SELECT ENAME, SALARY, LOC FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

#	ENAME	SALARY	LOC
1	ALLEN	1,600	CHICAGO
2	WARD	1,250	CHICAGO
3	JONES	2,975	DALLAS
4	MARTIN	1,250	CHICAGO
5	BLAKE	2,850	CHICAGO
6	CLARK	2,450	NEW YORK
7	SCOTT	3,000	DALLAS
8	KING	5,000	NEW YORK
9	TURNER	1,500	CHICAGO
10	ADAMS	1,100	DALLAS
11	JAMES	950	CHICAGO
12	MILLER	1,300	NEW YORK

→ 두 테이블 모두 있는 DEPTNO를 통해 JOIN

5. 집합연산자 조회

```
SELECT ENAME FROM EMPLOYEE WHERE DEPTNO=20
UNION
SELECT ENAME FROM EMPLOYEE WHERE DEPTNO=30;
```

#	ENAME
1	ADAMS
2	ALLEN
3	BLAKE
4	JAMES
5	JONES
6	MARTIN
7	SCOTT
8	TURNER
9	WARD

→ 중복을 없애기 위해 SORTING후 중복찾으므로 알파벳순으로 SORTING됨

6. 집계함수 조회

```
SELECT DEPTNO, MAX(SALARY) MAX_SALARY
FROM EMPLOYEE
GROUP BY DEPTNO
HAVING DEPTNO>=20;
```

#	DEPTNO	MAX_SALARY
1	20	3,000
2	30	2,850

7. 중복되지 않게 조회

```
SELECT DISTINCT DEPTNO FROM EMPLOYEE;
```

#	DEPTNO
1	10
2	20
3	30

2. SELECT~FOR UPDATE

2.1. 수행

수행내역
SESSION 1) 영화 잔여석 테이블 생성 및 조회
SESSION 1) SELECT문으로 테이블 조회 - FOR UPDATE
SESSION 2) SELECT문으로 테이블 조회 - 불가능
SESSION 1) UPDATE 실행 후 LOCK 해제
SESSION 2) SELECT및 UPDATE 가능

2.2. 결과

[SESSION 1]

1. 영화 잔여석 테이블 생성 및 조회

```
CREATE TABLE MOVIE
(MOVIE_NUMBER NUMBER,
SEAT_LEFT NUMBER,
NAME VARCHAR2(100));

INSERT INTO MOVIE VALUES (1,10,'OWL');
INSERT INTO MOVIE VALUES (2,5,'BLACK PANTHER');
INSERT INTO MOVIE VALUES (3, 20, 'SYMPATHY');
INSERT INTO MOVIE VALUES (4, 15, 'DECIBEL');

SELECT * FROM MOVIE;
```

```
23:39:42 SQL> COL NAME FOR A20
23:39:46 SQL> SELECT * FROM MOVIE;

MOVIE_NUMBER  SEAT_LEFT NAME
------------ ---------- --------------------
           1         10 OWL
           2          5 BLACK PANTHER
           3         20 SYMPATHY
           4         15 DECIBEL

4 rows selected.
```

2. SELECT문으로 테이블 조회 - FOR UPDATE

```
SELECT * FROM MOVIE WHERE MOVIE_NUMBER=4 FOR UPDATE;
```

```
23:39:49 SQL> SELECT * FROM MOVIE WHERE MOVIE_NUMBER=4 FOR UPDATE;

MOVIE_NUMBER  SEAT_LEFT NAME
------------ ---------- --------------------
           4         15 DECIBEL

1 row selected.
```

[SESSION 2]

3. SELECT문으로 테이블 조회 - HANG 걸림

```
SELECT * FROM MOVIE WHERE MOVIE_NUMBER=4 FOR UPDATE;
```

```
23:40:01 SQL> SELECT * FROM MOVIE WHERE MOVIE_NUMBER=4 FOR UPDATE;
```

[SESSION 1]

4. UPDATE 실행 후 LOCK 해제 - [SESSION2] HANG 풀림

```
UPDATE MOVIE SET SEAT_LEFT=14 WHERE MOVIE_NUMBER=4;
COMMIT;
```

```
23:40:38 SQL> UPDATE MOVIE SET SEAT_LEFT=14 WHERE MOVIE_NUMBER=4;

1 row updated.

23:41:26 SQL> COMMIT;

Commit completed.
```

[SESSION 2]

```
UPDATE MOVIE SET SEAT_LEFT=13 WHERE MOVIE_NUMBER=4;
SELECT * FROM MOVIE;
COMMIT;
```

```
23:41:30 SQL> UPDATE MOVIE SET SEAT_LEFT=13 WHERE MOVIE_NUMBER=4;

1 row updated.

23:41:51 SQL> SELECT * FROM MOVIE;

MOVIE_NUMBER  SEAT_LEFT NAME
------------ ---------- --------------------
           1         10 OWL
           2          5 BLACK PANTHER
           3         20 SYMPATHY
           4         13 DECIBEL

4 rows selected.

23:42:00 SQL> COMMIT;

Commit completed.
```

3. WITH문

- 같은 서브쿼리가 여러 번 사용된다면, with구문을 사용하는 것이 가독성이 좋다.
- with구문 여러 번 사용될 때 성능적으로 좋다.
- select 사용할 때 보기 편하다는 장점 → insert,update,delete 처럼 직접 데이터 변경이 있다면 비효율적이다.

3.1. 수행

수행내역
테이블 생성 및 데이터 입력
테이블 조회
WITH구문 - 1개만 있을 때
WITH구문 - 여러 개(2개 이상)
INSERT안에 WITH 사용법
일반 INSERT문과의 비교

3.2. 결과

1. 테이블 생성 및 데이터 입력

```
/*테이블 생성*/
CREATE TABLE BBS1 (SEQ NUMBER, TITLE VARCHAR2(20), CONTENTS VARCHAR2(20));
CREATE TABLE BBS2 (SEQ NUMBER, TITLE VARCHAR2(20), CONTENTS VARCHAR2(20));
CREATE TABLE BBS3 (SEQ NUMBER, TITLE VARCHAR2(20), CONTENTS VARCHAR2(20));
COMMIT;

/*데이터 입력*/
INSERT INTO BBS1 (SEQ, TITLE, CONTENTS) VALUES(1,'AAA','가');
INSERT INTO BBS1 (SEQ, TITLE, CONTENTS) VALUES(2,'BBB','나');
INSERT INTO BBS1 (SEQ, TITLE, CONTENTS) VALUES(3,'CCC','다');
INSERT INTO BBS2 (SEQ, TITLE, CONTENTS) VALUES(1,'DDD','라');
INSERT INTO BBS2 (SEQ, TITLE, CONTENTS) VALUES(2,'EEE','마');
INSERT INTO BBS2 (SEQ, TITLE, CONTENTS) VALUES(3,'FFF','바');
INSERT INTO BBS3 (SEQ, TITLE, CONTENTS) VALUES(1,'GGG','사');
INSERT INTO BBS3 (SEQ, TITLE, CONTENTS) VALUES(2,'HHH','아');
INSERT INTO BBS3 (SEQ, TITLE, CONTENTS) VALUES(3,'III','자');
COMMIT;
```

2. 테이블 조회

```
SELECT * FROM BBS1;
SELECT * FROM BBS2;
SELECT * FROM BBS3;
```

#	SEQ	TITLE	CONTENTS
1	1	AAA	가
2	2	BBB	나
3	3	CCC	다

#	SEQ	TITLE	CONTENTS
1	1	DDD	라
2	2	EEE	마
3	3	FFF	바

#	SEQ	TITLE	CONTENTS
1	1	GGG	사
2	2	HHH	아
3	3	III	자

3. WITH구문 한개만 있을 때

- 구성요소

  WITH [WITH절 명칭] AS (SELECT [컬럼명] FROM [테이블명])

  SELECT [컬럼명] FROM [WITH절 명칭]

```
WITH WITH_BBS1
AS (SELECT SEQ,TITLE
FROM BBS1
WHERE SEQ=1
)
SELECT C.SEQ AS BBS1_SEQ
,C.TITLE AS BBS1_TITLE
,A.TITLE AS BBS2_TITLE
,A.CONTENTS AS BBS2_CONTENTS
,B.TITLE AS BBS3_TITLE
,B.CONTENTS AS BBS3_CONTENTS
```

```
FROM BBS2 A, BBS3 B, WITH_BBS1 C
WHERE A.SEQ=B.SEQ;
```

#	BBS1_SEQ	BBS1_TITLE	BBS2_TITLE	BBS2_CONTENTS	BBS3_TITLE	BBS3_CONTENTS
1	1	AAA	DDD	라	GGG	사
2	1	AAA	EEE	마	HHH	아
3	1	AAA	FFF	바	III	자

→WITH_BBS1이라는 ALLIAS를 가진 서브쿼리 생성 후 메인쿼리에 FROM절에서 조회

3. WITH구문 여러 개 (2개 이상)

- 구성요소

  WITH [WITH절 명칭-1] AS (SELECT [컬럼명-1] FROM [테이블명-1]),

  [WITH절 명칭-2] AS (SELECT [컬럼명-2] FROM [테이블명-2])

  SELECT ALIAS명-1.[컬럼명-1]

  , ALLIAS명-2.[컬럼명-2]

  FROM [WITH절 명칭-1] ALIAS명-1

  , [WITH절 명칭-2] ALIAS명-2

```
WITH WITH_BBS1
AS (SELECT SEQ,TITLE, CONTENTS FROM BBS1)
, WITH_BBS2
AS (SELECT SEQ, TITLE, CONTENTS FROM BBS2)
, WITH_BBS3
AS (SELECT SEQ, TITLE, CONTENTS FROM BBS3)
SELECT A.SEQ BBS1_SEQ
,A.TITLE BBS1_TITLE
,A.CONTENTS BBS1_CONTENTS
,B.SEQ BBS2_SEQ
,B.TITLE BBS2_TITLE
,B.CONTENTS BBS2_CONTENTS
,C.SEQ BBS3_SEQ
,C.TITLE BBS3_SEQ
,C.CONTENTS BBS3_CONTENTS
FROM WITH_BBS1 A, WITH_BBS2 B, WITH_BBS3 C
WHERE A.SEQ=B.SEQ
AND A.SEQ=C.SEQ;
```

#	BBS1_SEQ	BBS1_TITLE	BBS1_CONTENTS	BBS2_SEQ	BBS2_TITLE	BBS2_CONTENTS	BBS3_SEQ	BBS3_SEQ	BBS3_CONTENTS
1	1	AAA	가	1	DDD	라	1	GGG	사
2	2	BBB	나	2	EEE	마	2	HHH	아
3	3	CCC	다	3	FFF	바	3	III	자

4. INSERT문안에 WITH구문

```
INSERT INTO BBS1
SELECT *
FROM (WITH WITH_TAB
AS (SELECT SEQ,TITLE,CONTENTS
FROM BBS2
WHERE SEQ=1)
SELECT * FROM WITH_TAB);

SELECT * FROM BBS1;
```

#	SEQ	TITLE	CONTENTS
1	1	AAA	가
2	2	BBB	나
3	3	CCC	다
4	1	DDD	라

5. INSERT문안에 WITH구문 vs 일반 INSERT문 (같은 결과)

```
SET TIMING ON
/*WITH구문*/
INSERT INTO BBS1
SELECT *
FROM (WITH WITH_TAB
AS (SELECT SEQ,TITLE,CONTENTS
FROM BBS2
WHERE SEQ=1)
SELECT * FROM WITH_TAB);

/*일반 INSERT문*/
INSERT INTO BBS1
SELECT *
FROM (SELECT SEQ, TITLE, CONTENTS
FROM BBS2
WHERE SEQ=1);
```

```
SQL> SET TIMING ON
SQL> INSERT INTO BBS1
SELECT *
FROM (WITH WITH_TAB
AS (SELECT SEQ,TITLE,CONTENTS
FROM BBS2
WHERE SEQ=1)
SELECT * FROM WITH_TAB);    2    3

1 row inserted.

Total elapsed time 00:00:00.003267
```

```
SQL> INSERT INTO BBS1
SELECT *
FROM (SELECT SEQ, TITLE, CONTENTS
FROM BBS2
WHERE SEQ=1);    2    3    4    5

1 row inserted.

Total elapsed time 00:00:00.002177
```

- 걸린 시간 : WITH구문 > 일반
- 쿼리 길이 : WITH구문 > 일반

→ 결과 : INSERT/UPDATE/DELETE문에서는 오히려 성능이 떨어진다.

4. ROW LEVEL FLASHBACK-1

FLASHBACK

- COMMIT이전이라면 ROLLBACK으로 수행한 작업 복구 가능
- 잘못 DELETE후 COMMIT을 했다면 특정한 시간 또는 특정 시점으로 되돌릴 수 있는 기능인 FLASHBACK으로 복구
- UNDO 기능을 사용하여 복구하는 방법
  1. UNDO_MANAGEMENT
     - 초기화 파라미터 파일에서 설정

- AUTO로 설정하면 데이터베이스 자동 UNDO관리 모드로 → UNDO 테이블스페이스가 필요
- AUTO로 설정되어 있어야 FLASHBACK사용 가능

2. UNDO_RETENTION (초 단위)
- UNDO 데이터 보유 기간 결정
- ALTER SYSTEM 명령을 사용하여 동적으로 수정 가능
- 기본값 900초(15분) → 데이터를 15분동안 보유

3. UNDO_TABLESPACE
- 사용할 특정 UNDO 테이블스페이스 지정
- 최소한 하나의 UNDO 테이블 스페이스 생성 필요
- ALTER SYSTEM 명령을 사용하여 동적으로 변경 가능
- 시스템이 관리하는 UNDO SEGMENT가 만들어질 TABLESPACE

4. UNDO_SEGMENT
- 지워진 데이터가 저장되는 곳
- SESSION1이 데이터를 지우고 COMMIT을 안했다면 → SESSION2는 UNDO SEGMENT에 있는 데이터 조회

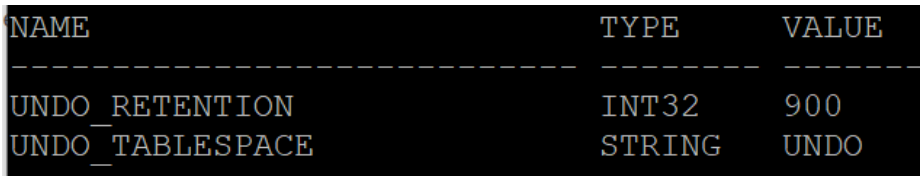
4.1. 수행

수행내역
FLASHBACK을 사용하기 위한 환경설정
테이블 생성 및 데이터 입력 후 COMMIT
데이터 삭제 후 COMMIT
AS OF TIMESTAMP 이용하여, 시점으로 확인
10분전 테이블 데이터 조회
20분전 테이블 데이터 조회
10분전 삭제된 데이터 다시 삽입

4.2. 결과

0. FLASHBACK을 사용하기 위한 환경설정

```
SHOW PARAM UNDO;
```



```
NAME                                 TYPE        VALUE
------------------------------------ ----------- -------
UNDO_RETENTION                       INT32       900
UNDO_TABLESPACE                      STRING      UNDO
```

1. 테이블 생성 및 데이터 입력 → COMMIT

```
CREATE TABLE TABLE_20M(
A NUMBER,
B NUMBER,
C NUMBER);

SET SERVEROUTPUT ON
DECLARE
BEGIN
```

```
FOR i IN 1..200 LOOP
INSERT INTO TABLE_20M VALUES(i,i,i);
END LOOP;
END;
/

COMMIT;
SELECT COUNT(*) FROM TABLE_20M;

CREATE TABLE TABLE_10M(
A NUMBER,
B NUMBER,
C NUMBER);

SET SERVEROUOTPUT ON
DECLARE
BEGIN
FOR i IN 1..100 LOOP
INSERT INTO TABLE_10M VALUES(i,i,i);
END LOOP;
END;
/
SELECT COUNT(*) FROM TABLE_10M;
COMMIT;
```

#	COUNT(*)
1	200

#	COUNT(*)
1	100

2. 데이터 잘못 삭제 → COMMIT

```
DELETE FROM TABLE_20M;
COMMIT;

SELECT COUNT(*) FROM TABLE_20M;
SELECT SYSTIMESTAMP FROM DUAL;

DELETE FROM TABLE_10M;
COMMIT;

SELECT SYSTIMESTAMP FROM DUAL;
```

#	COUNT(*)
1	0

#	SYSTIMESTAMP
1	2022/11/23 15:04:10.316292 Asia/Seoul

#	COUNT(*)
1	0

#	SYSTIMESTAMP
1	2022/11/23 15:14:32.061552 Asia/Seoul

3. AS OF TIMESTAMP 이용하여, 시점으로 확인

```
/*10분전 삭제한 테이블 조회*/
SELECT COUNT(*) FROM TABLE_10M AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL'10'MINUTE);
/*20분전 삭제한 테이블 조회*/
SELECT COUNT(*) FROM TABLE_20M AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL'20'MINUTE);
```

#	SYSTIMESTAMP
1	2022/11/23 15:23:03.403800 Asia/Seoul

#	COUNT(*)
1	100

[15:24:19.754]java.sql.SQLException: JDBC-10029:Unable to read table (or index) with old snapshot.

→ DEFAULT값은 15분이기때문에 20분전 데이터 조회 ERROR

4. 10분전 삭제된 데이터 다시 삽입

```
INSERT INTO TABLE_10M
(SELECT * FROM
TABLE_10M
AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL'10'MINUTE)
);

SELECT COUNT(*) FROM TABLE_10M;
```

#	COUNT(*)
1	100

5. 20분전 삭제된 데이터 다시 삽입

```
INSERT INTO TABLE_20M
(SELECT * FROM
TABLE_20M
AS OF TIMESTAMP(SYSTIMESTAMP-INTERVAL'20'MINUTE)
);

SELECT COUNT(*) FROM TABLE_20M;
```

#	COUNT(*)
1	0

5. ROW LEVEL FLASHBACK-2

5.1. 수행

SCN

- COMMIT이 발생되면 LGWR의 해당 트랜젝션은 고유한 번호를 부여받아 함께 관리한다.
- COMMIT할 때 생성되는 번호

수행내역
COMMIT 내용 검색 (OPT에서 D값 이전 ST_SCN번호 확인)
특정 시점 SCN번호를 통해 20분전 삭제된 데이터 조회
특정 시점 SCN번호를 통해 10분전 삭제된 데이터 조회

5.2. 결과

1. SCN 번호 확인

```
SELECT * FROM V$TSN_TIME WHERE TIME>='2022/11/23 15:04:10.316292'
AND TIME<='2022/11/23 15:14:32.061552';
```

#	TSN	TIME
1	540,636	2022/1...
2	540,636	2022/1...
3	540,637	2022/1...
4	540,637	2022/1...
5	540,637	2022/1...
6	540,638	2022/1...
7	540,638	2022/1...
8	540,638	2022/1...
9	540,639	2022/1...
10	540,639	2022/1...
11	540,639	2022/1...
12	540,640	2022/1...

554	540,885	2022/1...
555	540,886	2022/1...
556	540,886	2022/1...
557	540,886	2022/1...
558	540,887	2022/1...
559	540,887	2022/1...
560	540,887	2022/1...
561	540,888	2022/1...
562	540,888	2022/1...
563	540,889	2022/1...
564	540,889	2022/1...
565	540,889	2022/1...

2. 20분전 DELETE한 테이블 조회

```
SELECT COUNT(*) FROM TABLE_20M AS OF SCN 540636;
```

[15:29:07.017]java.sql.SQLException: JDBC-10029:Unable to read table (or index) with old snapshot.

3. 10분전 DELETE한 테이블 조회

```
SELECT COUNT(*) FROM TABLE_10M AS OF SCN 540880;
```

#	COUNT(*)
1	100

6. TABLE LEVEL FLASHBACK (DROP)

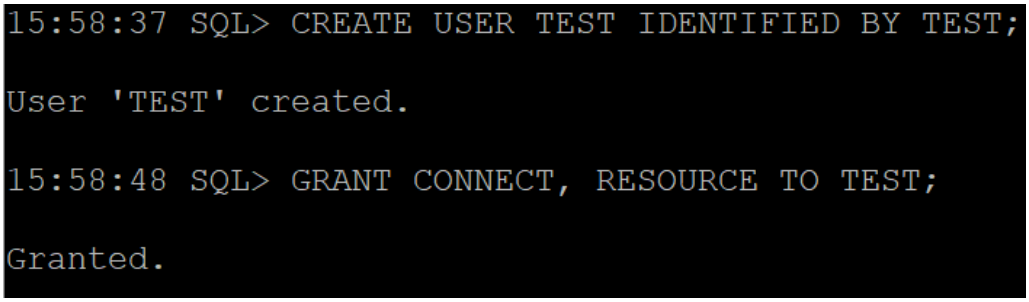
6.1. 수행

수행내역
TEST 사용자 생성 및 권한 부여
USE_RECYCLEBIN 활성화
테이블 생성 및 데이터 입력
테이블 장애 발생 (DROP)
FLASHBACK 사용하여 복구
복구 확인 및 파라미터 비활성화

6.2. 결과

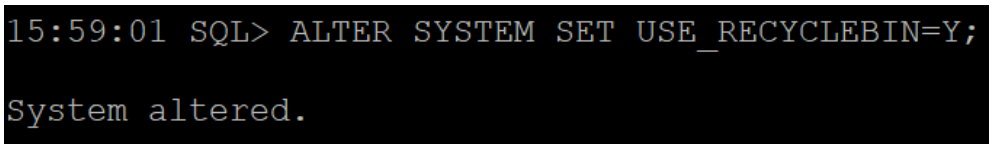
1. TEST 사용자 생성 및 권한 부여

```
CREATE USER TEST IDENTIFIED BY TEST;
GRANT CONNECT, RESOURCE TO TEST;
CONN TEST/TEST
```



2. 파라미터 활성화

```
CONN SYS/TIBERO
ALTER SYSTEM SET USE_RECYCLEBIN=Y;
```



3. 테이블 생성 후 데이터 입력

```
CONN TEST/TEST

CREATE TABLE FLASHBACK_TEST(A NUMBER, B NUMBER, C NUMBER);

DECLARE
BEGIN
FOR I IN 1..300 LOOP
INSERT INTO TEST.FLASHBACK_TEST VALUES(I,I,I);
END LOOP;
END;
/
```

```
COMMIT;
SELECT COUNT(*) FROM FLASHBACK_TEST;
```

```
16:00:34 SQL> SELECT COUNT(*) FROM FLASHBACK_TEST;

  COUNT(*)
----------
       300

1 row selected.
```

4. 테이블 장애 발생 (DROP)

```
DROP TABLE TEST.FLASHBACK_TEST;
SELECT COUNT(*) FROM FLASHBACK_TEST;
```

```
16:00:46 SQL> DROP TABLE TEST.FLASHBACK_TEST;

Table 'TEST.FLASHBACK_TEST' dropped.

16:00:53 SQL> SELECT COUNT(*) FROM FLASHBACK_TEST;
TBR-8033: Specified schema object was not found.
at line 1, column 23 of null:
```

5. FLASHBACK 사용하여 복구

```
FLASHBACK TABLE FLASHBACK_TEST TO BEFORE DROP;
```

```
16:01:57 SQL> FLASHBACK TABLE FLASHBACK_TEST TO BEFORE DROP;

Flashbacked.
```

6. 복구 확인 후 파라미터 비활성화

```
SELECT COUNT(*) FROM FLASHBACK_TEST;

CONN SYS/TIBERO
ALTER SYSTEM SET USE_RECYCLEBIN=N;
```

```
16:02:24 SQL> SELECT COUNT(*) FROM FLASHBACK_TEST;

  COUNT(*)
----------
       300

1 row selected.

16:02:34 SQL> CONN SYS/TIBERO
Connected to Tibero.

16:02:39 SQL> ALTER SYSTEM SET USE_RECYCLEBIN=N;

System altered.
```

7. TABLE LEVEL FLASHBACK (INSERT)

7.1. 수행

수행내역
TEST 사용자 연결
TEST 사용자의 테이블 생성
데이터 입력 후 가장 최근 SCN번호 출력
또 다른 데이터 입력후 COMMIT
SCN번호를 입력하여 첫번째 INSERT한 TABLE로 복구

7.2. 결과

1. USE_RECYCLEBIN 활성화 TEST 사용자 연결

```
CONN TEST/TEST
```

```
16:21:30 SQL> CONN TEST/TEST
Connected to Tibero.
```

2. TEST 사용자의 테이블 생성

```
CREATE TABLE T (A NUMBER, B NUMBER);
INSERT INTO T VALUES(1,1);
COMMIT;

SELECT * FROM T;
```

```
16:21:37 SQL> CREATE TABLE T (A NUMBER, B NUMBER);

Table 'T' created.

16:21:48 SQL> INSERT INTO T VALUES(1,1);

1 row inserted.

16:22:00 SQL> COMMIT;

Commit completed.

16:22:02 SQL> SELECT * FROM T;

         A          B
---------- ----------
         1          1
```

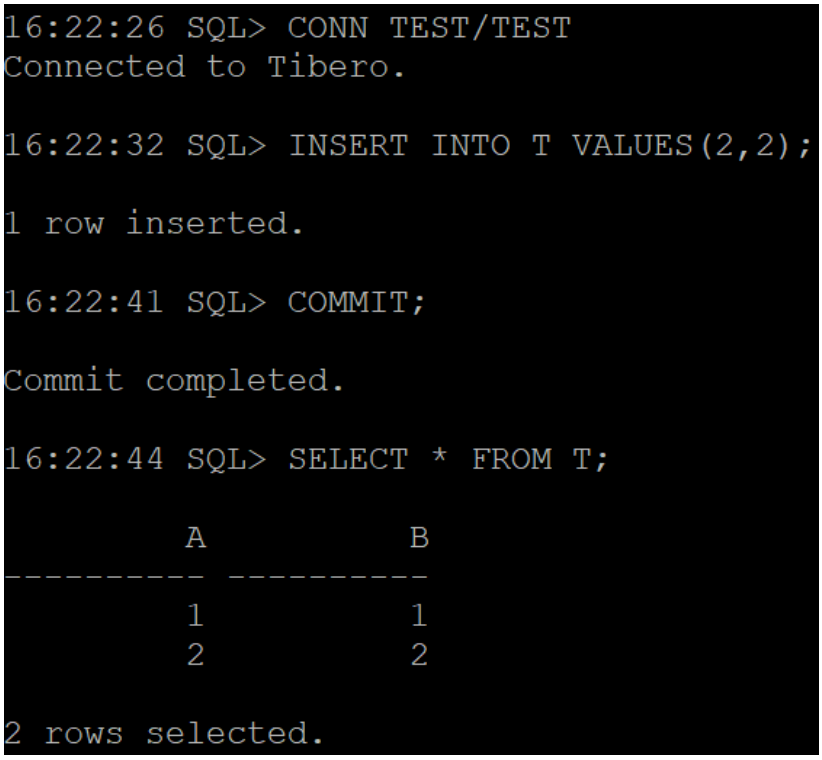
3. 데이터 입력 후 가장 최근 SCN번호 출력

```
CONN SYS/TIBERO
SELECT CURRENT_TSN FROM V$DATABASE;
```

```
16:22:07 SQL> CONN SYS/TIBERO
Connected to Tibero.

16:22:19 SQL> SELECT CURRENT_TSN FROM V$DATABASE;

CURRENT_TSN
-----------
     542907

1 row selected.
```

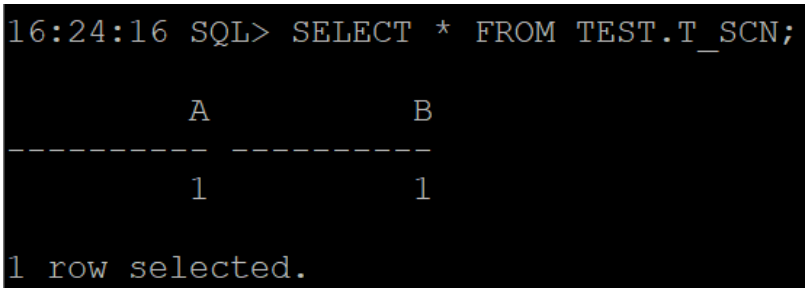
4. 또 다른 데이터 입력후 COMMIT

```
CONN TEST/TEST
INSERT INTO T VALUES(2,2);
COMMIT;

SELECT * FROM T;
```



5. SCN번호를 입력하여 첫번째 INSERT한 TABLE로 복구

```
FLASHBACK TABLE TEST.T TO SCN 542907 RENAME TO T_SCN;
SELECT * FROM TEST.T_SCN;
```



8. UNDO PARAMETER 변경

8.1. 수행

수행내역
undo parameter 확인
undo_retention 기본값 변경
undo 사용량 조회 (monitoring)
임시 undo tablespace 생성
기본 tablespace를 undotbs2로 변경가능

8.2. 결과

1. UNDO PARAMETER 확인

```
SHOW PARAM UNDO;
```

```
16:45:57 SQL> SHOW PARAM UNDO;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------
---
UNDO_RETENTION                       INT32       900
UNDO_TABLESPACE                      STRING      UNDO
```

2. UNDO_RETENTION 기본값 변경

```
ALTER SYSTEM SET UNDO_RETENTION=2400;
SHOW PARAM UNDO;
```

```
16:46:04 SQL> ALTER SYSTEM SET UNDO_RETENTION=2400;

System altered.

16:46:17 SQL> SHOW PARAM UNDO;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------
---
UNDO_RETENTION                       INT32       2400
UNDO_TABLESPACE                      STRING      UNDO
```

→ 15분 → 40분

3. undo 사용량 조회 (monitoring)

```
SELECT STATUS, SUM(BYTES)/1024/1024 MB FROM DBA_UNDO_EXTENTS GROUP BY STATUS;
```

```
SQL> SELECT STATUS, SUM(BYTES)/1024/1024 MB FROM DBA_UNDO_EXTENTS GROUP BY
STATUS;

STATUS                                 MB
------------------------------ ----------
Expired                        319.914063
Unexpired                       7.9921875

2 rows selected.
```

- EXPIRED : UNDO_RETENTION값이 지난 값으로 UNDO 부족시 REWRITE될 수 있는 영역

- UNEXPIRED : UNDO 작업 종료, UNDO_RETENTION값이 아직 지나지 않은 영역

4. 임시 undo tablespace 생성

```
CREATE UNDO TABLESPACE UNDOTBS2 DATAFILE '/tibero/s/undotbs02.dbf' SIZE 10G;
SELECT TABLESPACE_NAME FROM DBA_TABLESPACES;
```

```
SQL> CREATE UNDO TABLESPACE UNDOTBS2 DATAFILE '/tibero/s/undotbs02.dbf' SIZ
E 10G;

Tablespace 'UNDOTBS2' created.
```

```
SQL> SELECT TABLESPACE_NAME FROM DBA_TABLESPACES;

TABLESPACE_NAME
------------------------------------------------------------
-----
SYSTEM
UNDO
TEMP
USR
SYSSUB
UNDOTBS2

6 rows selected.
```

5. UNDO TABLESPACE 임시 TABLESPACE로 변경

```
ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2;
```

```
SQL> ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2;

System altered.
```

9. PIVOT/UNPIVOT-1

9.1. 수행

수행내역
부서별, 직군별로 직업의 합
PIVOT VIEW 생성
PIVOT VIEW 이용하여 UNPIVOT하기
모든 직업의 부서별 월급 합 조회

9.2. 결과

1. 부서별, 직군별로 직원의 합

```
SELECT *
FROM (SELECT DEPTNO, JOB, SALARY
FROM EMPLOYEE
)
PIVOT (SUM(SALARY) SALARY_SUM
FOR DEPTNO
IN  (10 DEPT10 ,20 DEPT20, 30 DEPT30)
);
```

#	JOB	DEPT10_SALARY_SUM	DEPT20_SALARY_SUM	DEPT30_SALARY_SUM
1	CLERK	<NULL>	1,900	950
2	ANALYST	<NULL>	3,000	2,500
3	ANALTST	<NULL>	3,000	<NULL>
4	PRESIDENT	5,000	<NULL>	<NULL>
5	SALESMAN	<NULL>	<NULL>	5,600
6	MANAGER	2,450	2,975	2,850

2. PIVOT VIEW 생성

```
CREATE VIEW PIVOTED_EMP
AS
SELECT *
FROM (SELECT DEPTNO, JOB, SALARY
FROM EMPLOYEE
)
PIVOT (SUM(SALARY) SALARY_SUM
FOR DEPTNO
IN  (10 DEPT10 ,20 DEPT20, 30 DEPT30)
);

SELECT * FROM PIVOTED_EMP;
```

[00:13:17.258]One DDL row has been executed.

#	JOB	DEPT10_...	DEPT20_...	DEPT30_...
1	CLERK	<NULL>	1,900	950
2	ANALYST	<NULL>	3,000	2,500
3	ANALTST	<NULL>	3,000	<NULL>
4	PRESIDENT	5,000	<NULL>	<NULL>
5	SALESMAN	<NULL>	<NULL>	5,600
6	MANAGER	2,450	2,975	2,850

3. PIVOT VIEW 이용하여 UNPIVOT하기

```
SELECT *
FROM PIVOTED_EMP
UNPIVOT (
SALARY_SUM
FOR DEPARTMENT
IN (DEPT10_SALARY_SUM as 10,
DEPT20_SALARY_SUM as 20,
DEPT30_SALARY_SUM as 30)
);
```

#	JOB	DEPARTMENT	SALARY_SUM
1	CLERK	20	1,900
2	CLERK	30	950
3	ANALYST	20	3,000
4	ANALYST	30	2,500
5	ANALTST	20	3,000
6	PRESIDENT	10	5,000
7	SALESMAN	30	5,600
8	MANAGER	10	2,450
9	MANAGER	20	2,975
10	MANAGER	30	2,850

4. 모든 직업의 부서별 월급 합 조회

```
SELECT *
FROM PIVOTED_EMP
UNPIVOT INCLUDE NULLS (
SALARY_SUM
FOR DEPARTMENT
IN (DEPT10_SALARY_SUM as 10,
DEPT20_SALARY_SUM as 20,
DEPT30_SALARY_SUM as 30)
);
```

#	JOB	DEPARTMENT	SALARY_SUM
1	CLERK	10	<NULL>
2	CLERK	20	1,900
3	CLERK	30	950
4	ANALYST	10	<NULL>
5	ANALYST	20	3,000
6	ANALYST	30	2,500
7	ANALTST	10	<NULL>
8	ANALTST	20	3,000
9	ANALTST	30	<NULL>
10	PRESIDENT	10	5,000
11	PRESIDENT	20	<NULL>
12	PRESIDENT	30	<NULL>
13	SALESMAN	10	<NULL>
14	SALESMAN	20	<NULL>
15	SALESMAN	30	5,600
16	MANAGER	10	2,450
17	MANAGER	20	2,975
18	MANAGER	30	2,850

10. PIVOT/UNPIVOT-2

[문제]

- 원본 리스트

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
, LEVEL v
FROM dual
CONNECT BY LEVEL <= 26
```

```
)
SELECT *
FROM t
;
```

→ CHR( ) : 문자 리턴 함수 (CHR(65):A)

- 원본 테이블

#	C	V
1	A	1
2	B	2
3	C	3
4	D	4
5	E	5
6	F	6
7	G	7
8	H	8
9	I	9
10	J	10
11	K	11
12	L	12
13	M	13
14	N	14
15	O	15
16	P	16
17	Q	17
18	R	18
19	S	19
20	T	20
21	U	21
22	V	22
23	W	23
24	X	24
25	Y	25
26	Z	26

- 결과 테이블

GB1	GB2	V1	V2	V3	V4	V5	V6	V7
1	1	A	B	C	D	E	F	G
1	2	1	2	3	4	5	6	7
2	1	H	I	J	K	L	M	N
2	2	8	9	10	11	12	13	14
3	1	O	P	Q	R	S	T	U
3	2	15	16	17	18	19	20	21
4	1	V	W	X	Y	Z		
4	2	22	23	24	25	26		

10.1. 수행

수행내역
GB1 (번호 1~7까지를 1, 8~14까지를 2,… 즉, 7개의 행을 하나의 그룹으로 구별하는 식별자)
GB3 (V1~V7까지 7개 항목으로 구별하는 식별자)
GB2 (동일한 GB1 그룹 내에서 코드행과 번호행을 구별하는 식별자) - DECODE (열을 행으로 변환)
GB2 - UNPIVOT (열을 행으로 변환)
행을 열로 변환 (GB3) - DECODE,MIN 사용
행을 열로 변환 (GB3) - PIVOT 사용

10.2. 결과

1. GB1

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
, LEVEL v
FROM dual
CONNECT BY LEVEL <= 26
)
SELECT c
, v
, CEIL(v / 7) gb1
FROM t
;
```

#	C	V	GB1
1	A	1	1
2	B	2	1
3	C	3	1
4	D	4	1
5	E	5	1
6	F	6	1
7	G	7	1
8	H	8	2
9	I	9	2
10	J	10	2
11	K	11	2
12	L	12	2
13	M	13	2
14	N	14	2
15	O	15	3
16	P	16	3
17	Q	17	3
18	R	18	3
19	S	19	3
20	T	20	3
21	U	21	3
22	V	22	4
23	W	23	4
24	X	24	4
25	Y	25	4
26	Z	26	4

→ CEIL 함수 : 소수점 무조건 올림

2. GB3

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
, LEVEL v
FROM dual
CONNECT BY LEVEL <= 26
)
SELECT c
, v
, CEIL(v / 7) gb1
, MOD(v - 1, 7) + 1 gb3
FROM t
;
```

→ MOD 함수 : 나눈 나머지 (v를 7로 나누면 0~6이므로 1을 빼고 나머지를 구한 다음 1을 더해줌)

3. GB2 - DECODE함수 사용

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
, LEVEL v
FROM dual
CONNECT BY LEVEL <= 26
)
SELECT DECODE(gb2, 1, c, v) v
, CEIL(v / 7) gb1
, gb2
, MOD(v - 1, 7) + 1 gb3
FROM t
, (SELECT LEVEL gb2 FROM dual CONNECT BY LEVEL <= 2)
ORDER BY c
;
```

#	V	GB1	GB2	GB3
1	A	1	1	1
2	1	1	2	1
3	B	1	1	2
4	2	1	2	2
5	C	1	1	3
6	3	1	2	3
7	D	1	1	4
8	4	1	2	4
9	E	1	1	5
10	5	1	2	5
11	F	1	1	6
12	6	1	2	6
13	7	1	2	7
14	G	1	1	7
15	H	2	1	1
16	8	2	2	1
17	I	2	1	2
18	9	2	2	2
19	J	2	1	3
20	10	2	2	3
21	K	2	1	4
22	11	2	2	4
23	L	2	1	5
24	12	2	2	5
25	13	2	2	6
26	M	2	1	6
27	N	2	1	7
28	14	2	2	7

→ DECODE 함수 : GB2 칼럼이 1이면 C 아니면 V → 칼럼 V

4. GB2-UNPIVOT 사용 (열→행)

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
, LEVEL v
FROM dual
CONNECT BY LEVEL <= 26
)
SELECT *
FROM (SELECT c
, TO_CHAR(v) v
, CEIL(v / 7) gb1
, MOD(v - 1, 7) + 1 gb3
FROM t
)
UNPIVOT ( v FOR gb2 IN (c AS 1, v AS 2) )
;
```

#	GB1	GB3	GB2	V
1	1	1	1	A
2	1	1	2	1
3	1	2	1	B
4	1	2	2	2
5	1	3	1	C
6	1	3	2	3
7	1	4	1	D
8	1	4	2	4
9	1	5	1	E
10	1	5	2	5
11	1	6	1	F
12	1	6	2	6
13	1	7	1	G
14	1	7	2	7
15	2	1	1	H
16	2	1	2	8
17	2	2	1	I
18	2	2	2	9
19	2	3	1	J
20	2	3	2	10
21	2	4	1	K
22	2	4	2	11
23	2	5	1	L
24	2	5	2	12
25	2	6	1	M
26	2	6	2	13
27	2	7	1	N
28	2	7	2	14

→ 앞에 (3)과 같은 결과

[UNPIVOT문]

```
SELECT *
FROM (SELECT c
, TO_CHAR(v) v
, CEIL(v / 7) gb1
, MOD(v - 1, 7) + 1 gb3
FROM t
)
UNPIVOT ( v FOR gb2 IN (c AS 1, v AS 2) )
;
```

→ c를 unpivot하여 v로 합침

→ gb2칼럼 지정 : c를 1, v를 2

5. 행을 열로 변환 (GB3) - DECODE,MIN 사용

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
     , LEVEL v
  FROM dual
CONNECT BY LEVEL <= 26
)
SELECT gb1, gb2
, MIN(DECODE(gb3, 1, v)) v1
, MIN(DECODE(gb3, 2, v)) v2
```

```
, MIN(DECODE(gb3, 3, v)) v3
, MIN(DECODE(gb3, 4, v)) v4
, MIN(DECODE(gb3, 5, v)) v5
, MIN(DECODE(gb3, 6, v)) v6
, MIN(DECODE(gb3, 7, v)) v7
FROM (SELECT DECODE(gb2, 1, c, v) v
, CEIL(v / 7) gb1
, gb2
, MOD(v - 1, 7) + 1 gb3
FROM t
, (SELECT LEVEL gb2 FROM dual CONNECT BY LEVEL <= 2)
)
GROUP BY gb1, gb2
ORDER BY gb1, gb2
;
```

#	GB1	GB2	V1	V2	V3	V4	V5	V6	V7
1	1	1	A	B	C	D	E	F	G
2	1	2	1	2	3	4	5	6	7
3	2	1	H	I	J	K	L	M	N
4	2	2	8	9	10	11	12	13	14
5	3	1	O	P	Q	R	S	T	U
6	3	2	15	16	17	18	19	20	21
7	4	1	V	W	X	Y	Z	<NULL>	<NULL>
8	4	2	22	23	24	25	26	<NULL>	<NULL>

→ MIN 함수 : 1~26에서 최솟값이 1인 수들을 V1, 2인 수들을 V2… → GROUP BY해줌

→ DECODE 함수 : GB2에서 1인 것을 C, 아닌건 V

6. 행을 열로 변환 (GB3) - PIVOT 사용

```
WITH t AS
(
SELECT CHR(LEVEL + 64) c
     , LEVEL v
  FROM dual
CONNECT BY LEVEL <= 26
)
SELECT *
FROM (SELECT *
FROM (SELECT c
, TO_CHAR(v) v
, CEIL(v / 7) gb1
, MOD(v - 1, 7) + 1 gb3
FROM t
)
UNPIVOT ( v FOR gb2 IN (c AS 1, v AS 2) )
)
PIVOT ( MIN(v) FOR gb3 IN (1 v1, 2 v2, 3 v3, 4 v4, 5 v5, 6 v6, 7 v7) )
ORDER BY gb1, gb2
;
```

#	GB1	GB2	V1	V2	V3	V4	V5	V6	V7
1	1	1	A	B	C	D	E	F	G
2	1	2	1	2	3	4	5	6	7
3	2	1	H	I	J	K	L	M	N
4	2	2	8	9	10	11	12	13	14
5	3	1	O	P	Q	R	S	T	U
6	3	2	15	16	17	18	19	20	21
7	4	1	V	W	X	Y	Z	<NULL>	<NULL>
8	4	2	22	23	24	25	26	<NULL>	<NULL>

→ PIVOT 함수 : 최솟값에 따라 행→ 열로 변환

- PIVOT은 행→열로 변환할 때 사용하는 방법으로 GROUP BY, MIN, DECODE 사용 대체
- UNPIVOT은 열→행으로 변환할 때 사용 CROSS JOIN과 DECODE 사용 대체

11. 동등 조인 (EQUI JOIN)

11.1. 수행

수행내역
동등 조인의 명시적 표현법
동등 조인의 묵시적 표현법
동등 조인 VS 자연 조인(NATURAL JOIN)

11.2. 결과

1. 동등 조인의 명시적 표현법

```
SELECT *
FROM EMPLOYEE
INNER JOIN DEPARTMENT ON EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPT
1	7,369	SMITH	CLERK	7,902	0080/1...	800	
2	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	
3	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	
4	7,566	JONES	MANAGER	7,839	0081/0...	2,975	
5	7,654	MARTIN	SALESMAN	7,698	0081/0...	1,250	
6	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	
7	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	
8	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	

2. 동등 조인의 묵시적 표현법

```
SELECT *
FROM EMPLOYEE EMP, DEPARTMENT DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPT
1	7,369	SMITH	CLERK	7,902	0080/1...	800	
2	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	
3	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	
4	7,566	JONES	MANAGER	7,839	0081/0...	2,975	
5	7,654	MARTIN	SALESMAN	7,698	0081/0...	1,250	
6	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	
7	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	
8	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	

3. 동등 조인 VS 자연 조인(NATURAL JOIN)

```
/*동등 조인*/
SELECT *
FROM EMPLOYEE
INNER JOIN DEPARTMENT ON EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
/*INNER JOIN*/
SELECT *
FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

3	MANAGERNO	STARTDATE	SALARY	DEPTNO	DEPTNO	DNAME	LOC
NAGER	7,839	0081/0...	2,450	40	40	OPERAT...	BOSTON
NAGER	7,839	0081/0...	2,450	40	40	ACCOUN...	AMERICA
NAGER	7,839	0081/0...	2,850	40	40	OPERAT...	BOSTON
NAGER	7,839	0081/0...	2,850	40	40	ACCOUN...	AMERICA
ERK	7,902	0080/1...	800	20	20	RESEARCH	DALLAS
LESMAN	7,839	0081/0...	1,600	30	30	SALES	CHICAGO
LESMAN	7,839	0081/0...	1,600	30	30	RESEARCH	SALLAS
LESMAN	7,698	0081/0...	1,250	30	30	SALES	CHICAGO

#	DEPTNO	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DI
1	40	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	OI
2	40	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	A(
3	40	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	OI
4	40	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	A(
5	20	7,369	SMITH	CLERK	7,902	0080/1...	800	RI
6	30	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	S/
7	30	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	RI
8	30	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	S/

→ EQUI JOIN은 DEPTNO칼럼 두번 나옴 ↔ INNER JOIN은 DEPTNO 한번 나옴

12. 교차 조인 (CROSS JOIN)

- 카티션 프로덕트와 같다.
- 발생하는 상황
  - JOIN 조건 잘못 기술했을 때
  - JOIN 조건 정의하지 않았을 때
  - 조인 조건이 조인 조건에 참여하는 테이블의 모든 행이 조인될 때

12.1. 수행

수행내역
교차 조인의 명시적 표현법
교차 조인의 묵시적 표현법

12.2. 결과

1. 교차 조인의 명시적 표현법

```
SELECT COUNT(*)
FROM EMPLOYEE
CROSS JOIN DEPARTMENT;
```

#	COUNT(*)
1	96

2. 교차 조인의 묵시적 표현법

```
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT;
```

#	COUNT(*)
1	96

13. 셀프 조인 (SELF JOIN)

13.1. 수행

수행내역
사원 번호와 매니저 번호를 같이 조회
셀프 조인의 명시적 표현법
셀프 조인의 묵시적 표현법

13.2. 결과

1. 셀프 조인의 명시적 표현법

```
SELECT E.EMPNO, E.ENAME, E.JOB, M.EMPNO,M.ENAME,M.JOB
FROM EMPLOYEE E
INNER JOIN EMPLOYEE M ON E.MANAGERNO=M.EMPNO;
```

#	EMPNO	ENAME	JOB	EMPNO	ENAME	JOB
1	7,082	CLARK	MANAGER	7,839	KING	PRESIDENT
2	7,098	BLAKE	MANAGER	7,839	KING	PRESIDENT
3	7,499	ALLEN	SALESMAN	7,839	KING	PRESIDENT
4	7,521	WARD	SALESMAN	7,698	BLAKE	MANAGER
5	7,566	JONES	MANAGER	7,839	KING	PRESIDENT
6	7,698	BLAKE	MANAGER	7,839	KING	PRESIDENT
7	7,782	CLARK	MANAGER	7,839	KING	PRESIDENT
8	7,788	SCOTT	ANALTST	7,566	JONES	MANAGER
9	7,844	TURNER	SALESMAN	7,698	BLAKE	MANAGER

2. 셀프 조인의 묵시적 표현법

```
SELECT E.EMPNO, E.ENAME, E.JOB, M.EMPNO,M.ENAME,M.JOB
FROM EMPLOYEE E, EMPLOYEE M
WHERE E.MANAGERNO=M.EMPNO;
```

#	EMPNO	ENAME	JOB	EMPNO	ENAME	JOB
1	7,082	CLARK	MANAGER	7,839	KING	PRESIDENT
2	7,098	BLAKE	MANAGER	7,839	KING	PRESIDENT
3	7,499	ALLEN	SALESMAN	7,839	KING	PRESIDENT
4	7,521	WARD	SALESMAN	7,698	BLAKE	MANAGER
5	7,566	JONES	MANAGER	7,839	KING	PRESIDENT
6	7,698	BLAKE	MANAGER	7,839	KING	PRESIDENT
7	7,782	CLARK	MANAGER	7,839	KING	PRESIDENT
8	7,788	SCOTT	ANALTST	7,566	JONES	MANAGER
9	7,844	TURNER	SALESMAN	7,698	BLAKE	MANAGER

14. 외부 조인 (OUTER JOIN)

14.1. 수행

수행내역
왼쪽 외부 조인 (JOIN문)
왼쪽 외부 조인 (+)
오른쪽 외부 조인 (JOIN문)
오른쪽 외부 조인 (+)
완전 외부 조인

14.2. 결과

1. 왼쪽 외부 조인 (JOIN문)

```
SELECT *
FROM EMPLOYEE
LEFT OUTER JOIN DEPARTMENT ON EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO	DEPTNO	DNAME	LOC
1	7,034	MILLER	CLERK	7,782	0082/0...	1,300	70	<NULL>	<NULL>	<NULL>
2	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	OPERAT...	BOSTON
3	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	ACCOUN...	AMERICA
4	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	OPERAT...	BOSTON
5	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	ACCOUN...	AMERICA
6	7,369	SMITH	CLERK	7,902	0080/1...	800	20	20	RESEARCH	DALLAS
7	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	30	30	SALES	CHICAGO
8	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	30	30	RESEARCH	SALLAS
9	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	30	30	SALES	CHICAGO
10	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	30	30	RESEARCH	SALLAS
11	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20	20	RESEARCH	DALLAS
12	7,654	MARTIN	SALESMAN	<NULL>	0081/0...	1,250	40	40	OPERAT...	BOSTON
13	7,654	MARTIN	SALESMAN	<NULL>	0081/0...	1,250	40	40	ACCOUN...	AMERICA
14	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	30	30	SALES	CHICAGO
15	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	30	30	RESEARCH	SALLAS
16	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	10	10	ACCOUN...	NEW YORK
17	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	20	20	RESEARCH	DALLAS
18	7,839	KING	PRESIDENT	<NULL>	0081/1...	5,000	10	10	ACCOUN...	NEW YORK
19	7,844	TURNER	SALESMAN	7,698	<NULL>	1,500	40	40	OPERAT...	BOSTON
20	7,844	TURNER	SALESMAN	7,698	<NULL>	1,500	40	40	ACCOUN...	AMERICA
21	7,876	ADAMS	<NULL>	7,788	0083/0...	1,100	20	20	RESEARCH	DALLAS
22	7,900	JAMES	CLERK	7,698	0081/1...	950	30	30	SALES	CHICAGO
23	7,900	JAMES	CLERK	7,698	0081/1...	950	30	30	RESEARCH	SALLAS
24	7,934	MILLER	CLERK	7,782	0082/0...	1,300	10	10	ACCOUN...	NEW YORK
25	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	SALES	CHICAGO
26	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	RESEARCH	SALLAS

2. 왼쪽 외부 조인 (+)

```
SELECT *
FROM EMPLOYEE,DEPARTMENT
WHERE EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO(+);
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO	DEPTNO	DNAME	LOC
1	7,034	MILLER	CLERK	7,782	0082/0...	1,300	70	<NULL>	<NULL>	<NULL>
2	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	OPERAT...	BOSTON
3	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	ACCOUN...	AMERICA
4	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	OPERAT...	BOSTON
5	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	ACCOUN...	AMERICA
6	7,369	SMITH	CLERK	7,902	0080/1...	800	20	20	RESEARCH	DALLAS

→ LEFT OUTER JOIN과 결과가 같음

3. 오른쪽 외부 조인 (JOIN문)

```
SELECT *
FROM EMPLOYEE
RIGHT OUTER JOIN DEPARTMENT ON EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO	DEPTNO	DNAME	LOC
1	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	OPERAT...	BOSTON
2	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	ACCOUN...	AMERICA
3	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	OPERAT...	BOSTON
4	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	ACCOUN...	AMERICA
5	7,369	SMITH	CLERK	7,902	0080/1...	800	20	20	RESEARCH	DALLAS
6	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	30	30	SALES	CHICAGO
7	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	30	30	RESEARCH	SALLAS
8	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	30	30	SALES	CHICAGO
9	7,521	WARD	SALESMAN	7,698	0081/0...	1,250	30	30	RESEARCH	SALLAS
10	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20	20	RESEARCH	DALLAS
11	7,654	MARTIN	SALESMAN	<NULL>	0081/0...	1,250	40	40	OPERAT...	BOSTON
12	7,654	MARTIN	SALESMAN	<NULL>	0081/0...	1,250	40	40	ACCOUN...	AMERICA
13	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	30	30	SALES	CHICAGO
14	7,698	BLAKE	MANAGER	7,839	0081/0...	2,850	30	30	RESEARCH	SALLAS
15	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	10	10	ACCOUN...	NEW YORK
16	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	20	20	RESEARCH	DALLAS
17	7,839	KING	PRESIDENT	<NULL>	0081/1...	5,000	10	10	ACCOUN...	NEW YORK
18	7,844	TURNER	SALESMAN	7,698	<NULL>	1,500	40	40	OPERAT...	BOSTON
19	7,844	TURNER	SALESMAN	7,698	<NULL>	1,500	40	40	ACCOUN...	AMERICA
20	7,876	ADAMS	<NULL>	7,788	0083/0...	1,100	20	20	RESEARCH	DALLAS
21	7,900	JAMES	CLERK	7,698	0081/1...	950	30	30	SALES	CHICAGO
22	7,900	JAMES	CLERK	7,698	0081/1...	950	30	30	RESEARCH	SALLAS
23	7,934	MILLER	CLERK	7,782	0082/0...	1,300	10	10	ACCOUN...	NEW YORK
24	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	SALES	CHICAGO
25	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	RESEARCH	SALLAS
26	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	50	SALES	AMERICA
27	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	60	ACCOUN...	SEOUL

4. 오른쪽 외부 조인 (+)

```
SELECT *
FROM EMPLOYEE,DEPARTMENT
WHERE EMPLOYEE.DEPTNO(+)=DEPARTMENT.DEPTNO;
```

23	7,934	MILLER	CLERK	7,782	0082/0...	1,300	10	10	ACCOUN...	NEW YORK
24	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	SALES	CHICAGO
25	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	RESEARCH	SALLAS
26	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	50	SALES	AMERICA
27	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	60	ACCOUN...	SEOUL

→ RIGHT OUTER JOIN과 결과 같음

5. 완전 외부 조인

```
SELECT *
FROM EMPLOYEE
FULL OUTER JOIN DEPARTMENT ON EMPLOYEE.DEPTNO=DEPARTMENT.DEPTNO;
```

21	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30	30	RESEARCH	SALLAS
22	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40	40	ACCOUN...	AMERICA
23	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	40	40	ACCOUN...	AMERICA
24	7,654	MARTIN	SALESMAN	<NULL>	0081/0...	1,250	40	40	ACCOUN...	AMERICA
25	7,844	TURNER	SALESMAN	7,698	<NULL>	1,500	40	40	ACCOUN...	AMERICA
26	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	50	SALES	AMERICA
27	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>	60	ACCOUN...	SEOUL
28	7,034	MILLER	CLERK	7,782	0082/0...	1,300	70	<NULL>	<NULL>	<NULL>

15. 안티 조인

- 안티 조인 : 서브 쿼리의 B 테이블에는 없는 메인 쿼리의 A 테이블의 데이터만 추출하는 조인 방법

15.1. 수행

수행내역
안티조인 (NOT IN)
안티조인 (NOT EXISTS)
NOT IN vs NOT EXISTS

15.2. 결과

1. 안티조인 (NOT IN)

```
SELECT A.EMPNO,A.ENAME,A.DEPTNO,B.DNAME
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
AND A.DEPTNO NOT IN (SELECT DEPTNO
FROM DEPARTMENT
WHERE MANAGERNO IS NULL);
```

#	EMPNO	ENAME	DEPTNO
1	7,082	CLARK	40
2	7,369	SMITH	20
3	7,499	ALLEN	30
4	7,521	WARD	30
5	7,566	JONES	20
6	7,782	CLARK	10
7	7,788	SCOTT	20
8	7,844	TURNER	40
9	7,876	ADAMS	20
10	7,900	JAMES	30
11	7,934	MILLER	10
12	7,990	NEW	30

2. 안티조인 (NOT EXISTS)

```
SELECT A.EMPNO,A.ENAME,A.DEPTNO
FROM EMPLOYEE A
WHERE NOT EXISTS ( SELECT 1
FROM DEPARTMENT C
WHERE A.DEPTNO=C.DEPTNO
AND MANAGERNO IS NULL) ;
```

#	EMPNO	ENAME	DEPTNO
1	7,082	CLARK	40
2	7,098	BLAKE	<NULL>
3	7,369	SMITH	20
4	7,499	ALLEN	30
5	7,521	WARD	30
6	7,566	JONES	20
7	7,782	CLARK	10
8	7,788	SCOTT	20
9	7,844	TURNER	40
10	7,876	ADAMS	20
11	7,900	JAMES	30
12	7,934	MILLER	10
13	7,990	NEW	30

3. NOT IN vs NOT EXISTS

```
/*NOT IN*/
SELECT COUNT(*)
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
AND A.DEPTNO NOT IN (SELECT DEPTNO
FROM DEPARTMENT
WHERE MANAGERNO IS NULL);

/*NOT EXISTS*/
SELECT COUNT(*)
FROM EMPLOYEE A
WHERE NOT EXISTS ( SELECT 1
FROM DEPARTMENT C
WHERE A.DEPTNO=C.DEPTNO
AND MANAGERNO IS NULL) ;
```

#	COUNT(*)
1	12

#	COUNT(*)
1	13

→ NOT IN : DEPTNO의 NULL인 값은 제외

→ NOT EXISTS : DEPTNO의 NULL인 값 포함

16. 세미 조인

- 세미 조인: 서브 쿼리의 B 테이블에 있는 메인 쿼리의 A 테이블의 데이터만 추출하는 조인 방법

16.1. 수행

수행내역
세미조인 (IN)
세미조인 (EXISTS)
IN vs EXISTS

16.2. 결과

1. 세미조인 (IN)

```
SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND D.DEPTNO IN (SELECT E.DEPTNO FROM EMPLOYEE E
WHERE E.SALARY <= 3000 AND E.DEPTNO=D.DEPTNO);
```

#	DEPTNO	DNAME	LOC
1	20	RESEARCH	DALLAS
2	10	ACCOUN...	NEW YORK

2. 세미조인 (EXISTS)

```
SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND EXISTS (SELECT 1 FROM EMPLOYEE E
WHERE E.SALARY <=3000 AND E.DEPTNO=D.DEPTNO);
```

#	DEPTNO	DNAME	LOC
1	20	RESEARCH	DALLAS
2	10	ACCOUN...	NEW YORK

3. IN vs EXISTS

```
SET TIMING ON
/*IN*/
SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND D.DEPTNO IN (SELECT E.DEPTNO FROM EMPLOYEE E
WHERE E.SALARY <= 3000 AND E.DEPTNO=D.DEPTNO);
/*EXISTS*/
SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND EXISTS (SELECT 1 FROM EMPLOYEE E
WHERE E.SALARY <=3000 AND E.DEPTNO=D.DEPTNO);
```

```
14:28:14 SQL> SET TIMING ON
14:28:16 SQL> SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND D.DEPTNO IN (SELECT E.DEPTNO FROM EMPLOYEE E
WHERE E.SALARY <= 3000 AND E.DEPTNO=D.DEPTNO);14:
21    4

0 row selected.

Total elapsed time 00:00:00.003636
```

```
14:28:22 SQL> SET TIMING ON
14:28:27 SQL> SELECT * FROM DEPARTMENT D
WHERE D.DEPTNO IN (10,20)
AND EXISTS (SELECT 1 FROM EMPLOYEE E
WHERE E.SALARY <=3000 AND E.DEPTNO=D.DEPTNO);
2    4

0 row selected.

Total elapsed time 00:00:00.003236
```

→ IN 이 EXISTS보다 더 오래걸린다.

→ 쿼리마다 걸리는 시간 달라지기 때문에 더 효율적인것 선택

17. 부질의 결과값 하나의 열

- 부질의 : SELECT, INSERT, UPDATE, DELETE문에 중첩된 내부 SELECT문을 포함할 수 있으며, 이렇게 포함된 SELECT 문장을 말한다.

17.1. 수행

수행내역
‘BLAKE’ 사원보다 많은 급여를 받는 사원정보를 검색
조건절의 부질의 vs SELF 조인
10번 부서 사원의 직무와 같은 직무를 가지고 있는 다른 부서 사원의 정보
10번 부서의 사원정보가 나오는 것을 빼고 같은 직무 가지고 있는 부서 사원 정보
ANY 사용하기 (하나 이상의 결과가 조건을 만족)
ALL 사용하기 (3개 값 모두 조건 만족해야함)

17.2. 결과

1. ‘BLAKE’ 사원보다 많은 급여를 받는 사원정보를 검색

```
SELECT SALARY FROM EMPLOYEE WHERE ENAME='BLAKE';

SELECT DEPTNO, SALARY
```

```
FROM EMPLOYEE
WHERE SALARY >
(SELECT SALARY
FROM EMPLOYEE
WHERE ENAME='BLAKE');
```

#	SALARY
1	2,850

#	DEPTNO	SALARY
1	20	2,975
2	20	3,000
3	10	5,000

2. 조건절의 부질의 vs SELF 조인

```
/*SELF 조인*/
SELECT A.EMPNO,A.ENAME,A.SALARY
FROM EMPLOYEE A, EMPLOYEE B
WHERE A.SALARY>B.SALARY AND B.ENAME='BLAKE';
```

#	EMPNO	ENAME	SALARY
1	7,566	JONES	2,975
2	7,788	SCOTT	3,000
3	7,839	KING	5,000

→ 부질의랑 같은 값을 가짐.

3. 10번 부서 사원의 직무와 같은 직무를 가지고 있는 다른 부서 사원의 정보

```
/*subquery*
SELECT JOB
FROM EMPLOYEE
WHERE DEPTNO=10;

SELECT *
FROM EMPLOYEE
WHERE JOB IN
(SELECT JOB
FROM EMPLOYEE
WHERE DEPTNO=10);
```

#	JOB
1	MANAGER
2	PRESIDENT
3	CLERK

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO
1	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40
2	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	<NULL>
3	7,369	SMITH	CLERK	7,902	0080/1...	800	20
4	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20
5	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	10
6	7,839	KING	PRESIDENT	<NULL>	0081/1...	5,000	10
7	7,900	JAMES	CLERK	7,698	0081/1...	950	30
8	7,934	MILLER	CLERK	7,782	0082/0...	1,300	10

4. 10번 부서의 사원정보가 나오는 것을 빼고 같은 직무 가지고 있는 부서 사원 정보

```
SELECT *
FROM EMPLOYEE
WHERE JOB IN
(SELECT JOB
FROM EMPLOYEE
WHERE DEPTNO=10)
AND DEPTNO!=10;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO
1	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40
2	7,369	SMITH	CLERK	7,902	0080/1...	800	20
3	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20
4	7,900	JAMES	CLERK	7,698	0081/1...	950	30

5. ANY 사용하기 (하나 이상의 결과가 조건을 만족)

```
/*subquery*/
SELECT DEPTNO,ROUND(AVG(SALARY)) AVG_SALARY
FROM EMPLOYEE
GROUP BY DEPTNO
ORDER BY DEPTNO;

SELECT *
FROM EMPLOYEE
WHERE SALARY > ANY
(SELECT AVG(SALARY)
FROM EMPLOYEE
GROUP BY DEPTNO)
ORDER BY DEPTNO;
```

#	DEPTNO	AVG_SALARY
1	10	2,917
2	20	1,969
3	30	1,575
4	40	1,733
5	<NULL>	2,850

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO
1	7,782	CLARK	MANAGER	7,839	0081/0...	2,450	10
2	7,839	KING	PRESIDENT	<NULL>	0081/1...	5,000	10
3	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20
4	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	20
5	7,499	ALLEN	SALESMAN	7,839	0081/0...	1,600	30
6	7,990	NEW	ANALYST	7,902	0081/0...	2,500	30
7	7,082	CLARK	MANAGER	7,839	0081/0...	2,450	40
8	7,098	BLAKE	MANAGER	7,839	0081/0...	2,850	<NULL>

6. ALL 사용하기 (3개 값 모두 조건 만족해야함)

```
SELECT *
FROM EMPLOYEE
WHERE SALARY > ALL
(SELECT AVG(SALARY)
FROM EMPLOYEE
GROUP BY DEPTNO)
ORDER BY DEPTNO;
```

#	EMPNO	ENAME	JOB	MANAGERNO	STARTDATE	SALARY	DEPTNO
1	7,566	JONES	MANAGER	7,839	0081/0...	2,975	20
2	7,788	SCOTT	ANALTST	7,566	0082/1...	3,000	20
3	7,839	KING	PRESIDENT	<NULL>	0081/1...	5,000	10

→ 최댓값 2917

18. 계층 질의 의사 칼럼

- 계층 질의 : 테이블에 포함된 로우 사이에 상하 계층 관계가 성립된 경우 그 상관 관계에 따라 로우를 출력하는 질의
- SELECT문장 내 START WITH…CONNECT BY절 이용
  - START WITH절 : 루트 로우 지정
  - CONNECT BY절 : 로우 간의 상하 관계 정의

18.1. 수행

수행내역
TREE 테이블 생성
계층 질의 결과 레벨 확인
계층 질의 결과 행이 리프노드인지 확인
TREE_CYCLE 테이블 생성
계층 질의 결과 행이 루프를 발생시키는 행인지 확인

18.2. 결과

1. TREE 테이블 생성

```
CREATE TABLE tree(ID INT, MgrID INT, Name VARCHAR(32), BirthYear INT);

INSERT INTO tree VALUES (1,NULL,'Kim', 1963);
INSERT INTO tree VALUES (2,NULL,'Moy', 1958);
INSERT INTO tree VALUES (3,1,'Jonas', 1976);
INSERT INTO tree VALUES (4,1,'Smith', 1974);
INSERT INTO tree VALUES (5,2,'Verma', 1973);
INSERT INTO tree VALUES (6,2,'Foster', 1972);
INSERT INTO tree VALUES (7,6,'Brown', 1981);

SELECT * FROM TREE;
```

#	ID	MGRID	NAME	BIRTHYEAR
1	1	<NULL>	Kim	1,963
2	2	<NULL>	Moy	1,958
3	3	1	Jonas	1,976
4	4	1	Smith	1,974
5	5	2	Verma	1,973
6	6	2	Foster	1,972
7	7	6	Brown	1,981

2. 계층 질의 결과 레벨 확인

```
SELECT id, mgrid, name, LEVEL
FROM tree
WHERE LEVEL=2
START WITH mgrid IS NULL
```

```
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	LEVEL
1	3	1	Jonas	2
2	4	1	Smith	2
3	5	2	Verma	2
4	6	2	Foster	2

→ CONNECT BY : 특정 (부모) 로우의 EMPNO 컬럼 값과 같은 MGRNO 컬럼 값을 갖는 모든 로우는, EMPNO 컬럼 값을 갖는 로우의 자식 로우가 된다.

3. 계층 질의 결과 행이 리프노드인지 확인

```
SELECT id, mgrid, name, CONNECT_BY_ISLEAF
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	CONNECT...
1	1	<NULL>	Kim	0
2	2	<NULL>	Moy	0
3	3	1	Jonas	1
4	4	1	Smith	1
5	5	2	Verma	1
6	6	2	Foster	0
7	7	6	Brown	1

→ LEAF 노드이면 1, 아니면 0

4. TREE_CYCLE 테이블 생성

```
CREATE TABLE tree_cycle(ID INT, MgrID INT, Name VARCHAR(32));

INSERT INTO tree_cycle VALUES (1,NULL,'Kim');
INSERT INTO tree_cycle VALUES (2,11,'Moy');
INSERT INTO tree_cycle VALUES (3,1,'Jonas');
INSERT INTO tree_cycle VALUES (4,1,'Smith');
INSERT INTO tree_cycle VALUES (5,3,'Verma');
INSERT INTO tree_cycle VALUES (6,3,'Foster');
INSERT INTO tree_cycle VALUES (7,4,'Brown');
INSERT INTO tree_cycle VALUES (8,4,'Lin');
INSERT INTO tree_cycle VALUES (9,2,'Edwin');
INSERT INTO tree_cycle VALUES (10,9,'Audrey');
INSERT INTO tree_cycle VALUES (11,10,'Stone');

SELECT * FROM TREE_CYCLE;
```

#	ID	MGRID	NAME
1	1	<NULL>	Kim
2	2	11	Moy
3	3	1	Jonas
4	3	1	Jonas
5	4	1	Smith
6	5	3	Verma
7	6	3	Foster
8	7	4	Brown
9	8	4	Lin
10	9	2	Edwin
11	10	9	Audrey
12	11	10	Stone

5. 계층 질의 결과 행이 루프를 발생시키는 행인지 확인

```
SELECT id, mgrid, name, CONNECT_BY_ISCYCLE
FROM tree_cycle
START WITH name in ('Kim', 'Moy')
CONNECT BY NOCYCLE PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	CONNECT...
1	1	<NULL>	Kim	0
2	2	11	Moy	0
3	3	1	Jonas	0
4	3	1	Jonas	0
5	4	1	Smith	0
6	5	3	Verma	0
7	5	3	Verma	0
8	6	3	Foster	0
9	6	3	Foster	0
10	7	4	Brown	0
11	8	4	Lin	0
12	9	2	Edwin	0
13	10	9	Audrey	0
14	11	10	Stone	1

→ 현재 행의 자식이 다른 노드의 조상도 된다면 1, 아니면 0

→ CONNECT_BY_ISCYCLE은 CONNECT BY절에 NOCYCLE 키워드가 명시되는 경우에만 사용

19. 계층 질의 연산자 & 함수

19.1. 수행

수행내역
칼럼값으로 루트 행의 값 반환
칼럼값으로 부모 행의 값 반환 (루트행은 NULL)
루트 행으로부터 해당 행까지의 상-하 관계의 PATH를 문자열로 반환

19.2. 결과

1. 칼럼값으로 루트 행의 값 반환 (CONNECT_BY_ROOT)

```
SELECT id, mgrid, name, CONNECT_BY_ROOT id
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	CONNECT...
1	1	<NULL>	Kim	1
2	2	<NULL>	Moy	2
3	3	1	Jonas	1
4	4	1	Smith	1
5	5	2	Verma	2
6	6	2	Foster	2
7	7	6	Brown	2

→ 각 ID의 루트 ID는 CONNECT_BY_ROOT에 조회

→ SELECT문 내의 WHERE절 및 ORDER BY절에서 사용

2. 칼럼값으로 부모 행의 값 반환 (루트행은 NULL) - (PRIOR)

```
SELECT id, mgrid, name, PRIOR id as "prior_id"
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	prior_id
1	1	<NULL>	Kim	<NULL>
2	2	<NULL>	Moy	<NULL>
3	3	1	Jonas	1
4	4	1	Smith	1
5	5	2	Verma	2
6	6	2	Foster	2
7	7	6	Brown	6

→ 각 ID의 매니저 ID는 PRIOR에 조회

→ SELECT문 내의 WHERE절, ORDER BY절 및 CONNECT BY절에서 사용

3. 루트 행으로부터 해당 행까지의 상-하 관계의 PATH를 문자열로 반환

```
SELECT id, mgrid, name, SYS_CONNECT_BY_PATH(name,'/') as hierarchy
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

#	ID	MGRID	NAME	HIERARCHY
1	1	<NULL>	Kim	/Kim
2	2	<NULL>	Moy	/Moy
3	3	1	Jonas	/Kim/Jonas
4	4	1	Smith	/Kim/Smith
5	5	2	Verma	/Moy/Verma
6	6	2	Foster	/Moy/Foster
7	7	6	Brown	/Moy/Foster/Brown

→ JONAS의 매니저는 KIM이다.

→ SMITH의 매니저도 KIM이다.

→ KIM은 ROOT이다.

20. 병렬 질의

- 병렬 질의 : 하나의 SQL문장을 여러 WORKING THREAD를 사용하여 처리하는 것
- 장점 : 빠르게 작업 실행 가능
- HINT PARALLEL을 사용한다.

20.1. 수행

수행내역
EMPLOYEE 테이블 사용
일반 SELECT문 TRACE 확인
병렬 질의문 TRACE 확인
TEST 테이블 생성
일반 vs 병렬 질의 속도 확인

20.2. 결과

1. 일반 SELECT문 TRACE 확인

```
SET AUTOT TRACE EXPLAIN
SELECT EMPNO, ENAME, SALARY FROM EMPLOYEE ORDER BY SALARY;
```

```
SQL> SET AUTOT TRACE EXPLAIN
SQL> SELECT EMPNO, ENAME, SALARY FROM EMPLOYEE ORDER BY SALARY;

SQL ID: 47jz0pusgjw4u
Child number: 1891
Plan hash value: 1224307385

Execution Plan
----------------------------------------------------------------------------------
   1  ORDER BY (SORT) (Cost:7, %%CPU:0, Rows:15)
   2    TABLE ACCESS (ROWID): EMPLOYEE (Cost:7, %%CPU:0, Rows:15)
   3      INDEX (FAST FULL SCAN): EMP_ID_PK (Cost:6, %%CPU:0, Rows:15)


Note
----------------------------------------------------------------------------------
   3 - dynamic sampling used for this table (13 blocks)
```

2. 병렬 질의문 TRACE 확인

```
SELECT /*+PARALLEL (4) */ EMPNO, ENAME, SALARY FROM EMPLOYEE ORDER BY SALARY;
```

```
SQL> SELECT /*+PARALLEL (4) */ EMPNO, ENAME, SALARY FROM EMPLOYEE ORDER BY SALARY;

SQL ID: gy6wymjp6v2v1
Child number: 1898
Plan hash value: 4033622264

Execution Plan
----------------------------------------------------------------------------------
   1  PE MANAGER (Cost:0, %%CPU:0, Rows:15)
   2    PE SEND QC (ORDER) (Cost:0, %%CPU:0, Rows:15)
   3      ORDER BY (SORT) (Cost:7, %%CPU:0, Rows:15)
   4        PE RECV (Cost:0, %%CPU:0, Rows:15)
   5          PE SEND (RANGE) (Cost:0, %%CPU:0, Rows:15)
   6            TABLE ACCESS (ROWID): EMPLOYEE (Cost:7, %%CPU:0, Rows:15)
   7              PE BLOCK ITERATOR (Cost:6, %%CPU:0, Rows:15)
   8                INDEX (FAST FULL SCAN): EMP_ID_PK (Cost:6, %%CPU:0, Rows:15)
```

→ PE, RECV, PE SEND 노드 추가됨

3. TEST 테이블 생성

```
CREATE TABLE TEST (A NUMBER, B NUMBER, C NUMBER);

DECLARE
BEGIN
FOR I IN 1..100000 LOOP
INSERT INTO TEST VALUES(I,I,I);
END LOOP;
END;
/

COMMIT;

SELECT COUNT(*) FROM TEST;
```

```
SQL> CREATE TABLE TEST (A NUMBER, B NUMBER, C NUMBER);

Table 'TEST' created.

SQL> DECLARE
   2 BEGIN
   3 FOR I IN 1..100000 LOOP
   4 INSERT INTO TEST VALUES(I,I,I);
   5 END LOOP;
   6 END;
   7 /

PSM completed.

SQL> COMMIT;

Commit completed.

SQL> SELECT COUNT(*) FROM TEST;

  COUNT(*)
----------
    100000

1 row selected.
```

4. 일반 vs 병렬 질의 속도 확인

```
SET TIMING ON
/*일반*/
SELECT A,B, FROM TEST;
/*병렬 질의*/
SELECT /*+PARALLEL*/ A,B FROM TEST;
```

```
100000 rows selected.

Total elapsed time 00:00:01.192809
```

```
100000 rows selected.

Total elapsed time 00:00:01.096449
```

→ 일반보다 병렬 질의가 속도가 더 빠르다. (속도 향상)

21. 듀얼(DUAL) 테이블

- 한 행으로 결과를 출력하기 위한 테이블

21.1. 수행

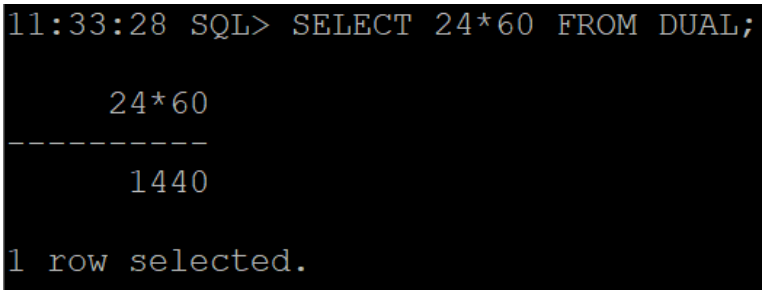
수행내역
산술 연산의 결과 한줄로 출력

DUAL 테이블의 구성
현재 날짜 출력
여러 칼럼의 데이터 만들기
기존 쿼리 결과에 데이터 붙이기 (집합 연산자)
여러행의 데이터 만들기 (CONNECT BY)
현재 날짜 기준 일주일치 날짜 불러오기
시작일자~종료일자 사이 모든 날짜 출력

21.2. 결과

1. 산술 연산의 결과 한줄로 출력

```
SELECT 24*60 FROM DUAL;
```



2. DUAL 테이블의 구성

```
DESC DUAL;
SELECT * FROM DUAL;
```

```
11:33:41 SQL> DESC DUAL;

COLUMN_NAME                              TYPE               CONSTRAINT
---------------------------------------- ------------------ --------------------
DUMMY                                    VARCHAR(1)

11:33:52 SQL> SELECT * FROM DUAL;

DUMMY
-----
X

1 row selected.
```

→ 칼럼 X는 아무 의미없는 값

3. 현재 날짜 출력

```
SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD')
FROM DUAL;
```

```
11:33:56 SQL> SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD')
11:34:26    2 FROM DUAL;

TO_CHAR(SYSDATE,'YYYY-MM-DD')
------------------------------
2022-11-25

1 row selected.
```

4. 여러 칼럼의 데이터 만들기

```
SELECT 7088 EMPNO
,'SCOTT' ENAME
,'ANALYST' JOB
FROM DUAL;
```

```
11:34:28 SQL> SELECT 7088 EMPNO
11:34:40    2 ,'SCOTT' ENAME
11:34:45    3 ,'ANALYST' JOB
11:34:52    4 FROM DUAL;

     EMPNO ENAME JOB
---------- ----- -------
      7088 SCOTT ANALYST

1 row selected.
```

5. 기존 쿼리 결과에 데이터 붙이기 (집합 연산자)

```
(SELECT 7088 EMPNO
,'AMY' ENAME
,'ANALYST' JOB
FROM DUAL)
UNION ALL
(SELECT EMPNO,ENAME,JOB FROM EMPLOYEE WHERE JOB IN ('MANAGER','CLERK'));
```

```
11:41:38 SQL> (SELECT 7088 EMPNO
11:41:44    2 ,'AMY' ENAME
11:41:49    3 ,'ANALYST' JOB
11:41:56    4 FROM DUAL)
11:41:58    5 UNION ALL
11:42:00    6 (SELECT EMPNO,ENAME,JOB FROM EMPLOYEE WHERE JOB IN ('MANAGER','CLERK'));

     EMPNO ENAME          JOB
---------- -------------- --------------
      7088 AMY            ANALYST
      7082 CLARK          MANAGER
      7098 BLAKE          MANAGER
      7369 SMITH          CLERK
      7566 JONES          MANAGER
      7782 CLARK          MANAGER
      7900 JAMES          CLERK
      7934 MILLER         CLERK

8 rows selected.
```

→ 칼럼명들이 같아야 합칠 수 있음

6. 여러행의 데이터 만들기 (CONNECT BY)

```
SELECT LEVEL FROM DUAL CONNECT BY LEVEL<=10;
```

```
11:44:08 SQL> SELECT LEVEL FROM DUAL CONNECT BY LEVEL<=10;

     LEVEL
----------
         1
         2
         3
         4
         5
         6
         7
         8
         9
        10

10 rows selected.
```

7. 현재 날짜 기준 일주일치 날짜 불러오기

```
SELECT TRUNC(SYSDATE) + LEVEL
FROM DUAL
CONNECT BY LEVEL<=7;
```

```
11:44:20 SQL> SELECT TRUNC(SYSDATE) + LEVEL
11:44:31    2 FROM DUAL
11:44:33    3 CONNECT BY LEVEL<=7;

TRUNC(SYSDATE)+LEVEL
--------------------
2022/11/26
2022/11/27
2022/11/28
2022/11/29
2022/11/30
2022/12/01
2022/12/02

7 rows selected.
```

→ TRUNC( ) : 시간 절사

8. 시작일자~종료일자 사이 모든 날짜 출력

```
SELECT TO_DATE('2022-11-28','YYYY-MM-DD')+(LEVEL-1) DTE
FROM DUAL
CONNECT BY LEVEL <= (TO_DATE('2022-12-02','YYYY-MM-DD')-TO_DATE('2022-11-28','YYYY-MM-DD')+1);
```

```
12:02:27 SQL> SELECT TO_DATE('2022-11-28','YYYY-MM-DD')+(LEVEL-1) DTE
FROM DUAL
CONNECT BY LEVEL <= (TO_DATE('2022-12-02','YYYY-MM-DD')-TO_DATE('2022-
DD')+1);12:02:28    2 12:02:28    3

DTE
--------------------
2022/11/28
2022/11/29
2022/11/30
2022/12/01
2022/12/02

5 rows selected.
```

22. 단순(SIMPLE) VIEW

22.1. 수행

수행내역
TEST 사용자 생성
뷰 생성&테이블 조회 권한 할당
단순 뷰 생성
단순 뷰 조회

22.2. 결과

1. TEST 사용자 생성

```
CREATE USER IDENTIFIED BY TEST;
GRANT CONNECT, RESOURCE TO TEST;
```

```
13:48:20 SQL> CREATE USER TEST IDENTIFIED BY TEST;

User 'TEST' created.
```

2. 뷰 생성&테이블 조회 권한 할당

```
GRANT CREATE VIEW TO TEST;
GRANT SELECT ON EMPLOYEE TO TEST;
GRANT SELECT ON DEPARTMENT TO TEST;

CONN TEST/TEST
```

```
13:49:52 SQL> CONN SYS/TIBERO
Connected to Tibero.

13:50:07 SQL> GRANT SELECT ON EMPLOYEE TO TEST;

Granted.

13:50:16 SQL> GRANT SELECT ON DEPARTMENT TO TEST;

Granted.
```

3. 단순 뷰 생성

```
CREATE OR REPLACE VIEW DEPT_SAL
AS
SELECT DEPTNO,SALARY
FROM SYS.EMPLOYEE;
```

```
13:53:18 SQL> CREATE OR REPLACE VIEW DEPT_SAL
13:54:17    2 AS
13:54:17    3 SELECT DEPTNO,SALARY
13:54:36    4 FROM SYS.EMPLOYEE
13:54:43    5 ;

View 'DEPT_SAL' created.
```

4. 단순 뷰 조회

```
SELECT * FROM DEPT_SAL;
```

```
13:54:50 SQL> SELECT * FROM DEPT_SAL;

    DEPTNO     SALARY
---------- ----------
        40       2450
                 2850
        20        800
        30       1600
        30       1250
        20       2975
        40       1250
        10       2450
        20       3000
        10       5000
        40       1500
        20       1100
        30        950
        10       1300
        30       2500

15 rows selected.
```

23. 복합(COMPLEX) VIEW

- 복합 뷰 : SUB QUERY 부분에 여러 개의 테이블 조인되어 생성되는 뷰

22.1. 수행

수행내역
TEST 사용자 생성
뷰 생성&테이블 조회 권한 할당
복합 뷰 생성
복합 뷰 조회

22.2. 결과

1. TEST 사용자 생성

```
CREATE USER IDENTIFIED BY TEST;
GRANT CONNECT, RESOURCE TO TEST;
```

```
13:48:20 SQL> CREATE USER TEST IDENTIFIED BY TEST;

User 'TEST' created.
```

2. 뷰 생성&테이블 조회 권한 할당

```
GRANT CREATE VIEW TO TEST;
GRANT SELECT ON EMPLOYEE TO TEST;
GRANT SELECT ON DEPARTMENT TO TEST;

CONN TEST/TEST
```

```
13:49:52 SQL> CONN SYS/TIBERO
Connected to Tibero.

13:50:07 SQL> GRANT SELECT ON EMPLOYEE TO TEST;

Granted.

13:50:16 SQL> GRANT SELECT ON DEPARTMENT TO TEST;

Granted.
```

3. 복합 뷰 생성 (부서번호 별 월급의 합)

```
CREATE OR REPLACE VIEW DEPT_SAL
AS
SELECT SUM(A.SALARY) SAL, A.DEPTNO DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
14:05:34 SQL> CREATE OR REPLACE VIEW DEPT_SAL
14:06:15    2 AS
14:06:16    3 SELECT SUM(A.SALARY) SAL, A.DEPTNO DEPTNO
14:06:40    4 FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
14:06:49    5 WHERE A.DEPTNO=B.DEPTNO
14:06:56    6 GROUP BY A.DEPTNO;

View 'DEPT_SAL' created.
```

→ OR REPLACE : 같은 이름의 VIEW가 존재할 경우 대체

4. 복합 뷰 조회 (부서번호 별 월급의 합)

```
SELECT * FROM DEPT_SAL;
```

```
14:07:04 SQL> SELECT * FROM DEPT_SAL;

       SAL     DEPTNO
---------- ----------
      8750         10
      5200         40
      7875         20
      6300         30

4 rows selected.
```

24. 인라인(INLINE) VIEW

- 인라인 뷰
  - 뷰는 필요할 때 생성한 후 여러 번 반복해서 재사용 가능
  - 1회만 필요한 뷰일 경우 FROM절에 VIEW의 서브쿼리 부분을 바로 적어주면된다.

24.1. 수행

수행내역
TEST 사용자 생성
뷰 생성&테이블 조회 권한 할당
인라인 뷰 생성
인라인 뷰 조회

24.2. 결과

1. TEST 사용자 생성

```
CREATE USER IDENTIFIED BY TEST;
GRANT CONNECT, RESOURCE TO TEST;
```

```
13:48:20 SQL> CREATE USER TEST IDENTIFIED BY TEST;

User 'TEST' created.
```

2. 뷰 생성&테이블 조회 권한 할당

```
GRANT CREATE VIEW TO TEST;
GRANT SELECT ON EMPLOYEE TO TEST;
GRANT SELECT ON DEPARTMENT TO TEST;

CONN TEST/TEST
```

```
13:49:52 SQL> CONN SYS/TIBERO
Connected to Tibero.

13:50:07 SQL> GRANT SELECT ON EMPLOYEE TO TEST;

Granted.

13:50:16 SQL> GRANT SELECT ON DEPARTMENT TO TEST;

Granted.
```

3. 인라인 뷰 생성 및 조회

```
SELECT D.DNAME DNAME
,E.MAX_SALARY MAX_SAL
,E.MIN_SALARY MIN_SAL
FROM (SELECT DEPTNO, MAX(SALARY) MAX_SALARY,MIN(SALARY) MIN_SALARY
FROM SYS.EMPLOYEE
GROUP BY DEPTNO) E, SYS.DEPARTMENT D
WHERE E.DEPTNO=D.DEPTNO;
```

```
14:17:40 SQL> ^C
14:17:47 SQL> SELECT D.DNAME DNAME
,E.MAX_SALARY MAX_SAL
,E.MIN_SALARY MIN_SAL
FROM (SELECT DEPTNO, MAX(SALARY) MAX_SALARY,MIN(SALARY) MIN_SALARY
FROM SYS.EMPLOYEE
GROUP BY DEPTNO) E, SYS.DEPARTMENT D
WHERE E.DEPTNO=D.DEPTNO;14:18:26    2 14:18:26    3 14:18:26    4 1
26    6 14:18:26    7

DNAME            MAX_SAL    MIN_SAL
-------------- ---------- ----------
ACCOUNTING          5000       1300
RESEARCH            3000        800
SALES               2500        950
ACCOUNTING          2450       1250

4 rows selected.
```

→ 부서이름별 최대월급과 최소월급 뷰 생성

25. 실체화 뷰

- 실체화 뷰 : 뷰는 쿼리만 저장하고 있을 뿐 데이터를 가지고 있지 않지만 실체호 뷰는 ‘MATERIALIZE’가 의미하는 것처럼 물리적으로 실제 데이터를 갖는다.
- 특징
  - REFRESH 옵션을 이용해 집계 테이블 자동 관리하도록 할 수 있다.
  - OPTIMIZER에의해 QUERY REWRITE 지원된다.
- [일반적인 제약 조건]

- SYSDATE, ROWNUM과 같이 반복할 수 없는 표현식 포함하면 안된다.
- LONG 또는 LONG RAW데이터 타입을 포함하면 안된다.
- SELECT 리스트에 부질의를 초함하면 안된다.
- SELECT 리스트에 분석 함수를 포함하면 안된다.
- HAVING, ANY, ALL, NOT EXISTS 포함하면 안된다.
- [START WITH] CONNECY BY절을 포함하면 안된다.
- UNION등 SET연산자 포함하면 안된다.

- [집합 함수 포함한 제약 조건]
  - SUM, COUNT, AVG, STDDEV, VARIANCE, MIN, MAX 함수 지원 가능
  - COUNT(*)는 항상 포함해야 한다.
  - 집합 함수가 항상 표현식 최상위에 있어야 한다.
  - SELECT 리스트에 모든 GROUP BY컬럼 있어야된다.

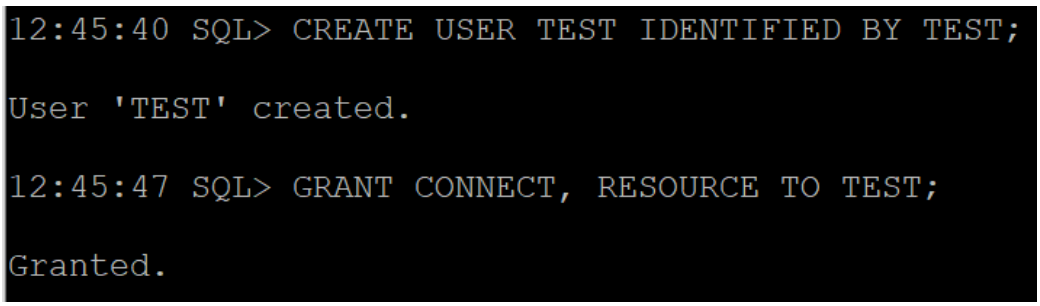
25.1. 수행

수행내역
TEST 사용자 생성
QUERY REWRITE 권한 부여 - ERROR
실제화 뷰 생성 권한 부여
TEST 사용자에 접속
실체화 뷰 생성
실체화 뷰 조회

25.2. 결과

1. TEST 사용자 생성

```
CREATE USER IDENTIFIED BY TEST;
GRANT CONNECT, RESOURCE TO TEST;
```



2. QUERY REWRITE 권한 부여 - ERROR

```
GRANT QUERY REWRITE TO TEST;
```

```
12:45:55 SQL> GRANT QUERY REWRITE TO TEST;
TBR-7001: General syntax error.
at line 1, column 13 of null:
GRANT QUERY REWRITE TO TEST
            ^^^^^^^
```

- Grant

다음은 Tibero의 Grant 미지원 사항에 대한 설명으로, Grant Count 계산할 때 제외시킨다.

항목	설명
COMMIT REFRESH	Mview를 생성할 때 생성 구문으로 지원하며 권한으로는 지원하지 않는다.
QUERY REWRITE	Mview를 생성할 때 생성 구문으로 지원하며 권한으로는 지원하지 않는다.
DEBUG	PL/SQL DEBUGGING 권한으로 현재 Tibero는 지원하지 않는다.

→ TIBERO는 현재 COMMIT REFRESH와 QUERY REWRITE GRANT 지원하지않는다.

3. 실제화 뷰 생성 권한 부여

```
GRANT CREATE MATERIALIZED VIEW TO TEST;
```

```
12:46:05 SQL> GRANT CREATE MATERIALIZED VIEW TO TEST;

Granted.
```

4. TEST 사용자에서 실체화 뷰 생성

```
CONN TEST/TEST

CREATE  MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
12:55:40 SQL> CREATE  MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;12:56:07    2 12:56:07    3 12:
6 12:56:07    7 12:56:07    8 12:56:07    9 12:56

Materialized View 'DEPT_SAL' created.
```

→ BUILD IMMEDIATE : MVIEW 생성과 동시에 데이터들도 생성

→ REFRESH : 데이터를 언제 어떻게 REFRESH

- ON DEMAND : DBMS_MVIEW 패키지 (REFRESH, REFRESH_ALL_MVIEWS, REFRESH_DEPENDENT) 를 실행 한 경우 Refresh 되는 경우

→ ENABLE QUERY REWRITE : MVIEW 생성시 이 옵션을 주어야 QUERY REWRITE고려 (지정안해도 ALTER MATERIALIZED VIEW 이용하여 수정 가능)

5. 실체화 뷰 조회

```
SELECT * FROM DEPT_SAL;
```

```
12:56:08 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         6300         10
         5200         40
        10875         20
         6300         30

4 rows selected.
```

→ 부서번호별 월급의 합 테이블

26. 실체화 뷰 관리하기

26.1. 수행

수행내역
실체화 뷰 생성 및 조회
INDEX 생성

원본 테이블에 데이터 추가하고 실체화 뷰와 비교
수동으로 원본 테이블과 실체화 뷰 데이터 동기화
TEST 사용자가 생성한 실체화 뷰 내용 조회

26.2. 결과

1. 실체화 뷰 생성 및 조회

```
CREATE  MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;

SELECT * FROM DEPT_SAL;
```

```
14:29:00 SQL> CREATE  MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;14:29:11    2 14:29:11    3 14:
6 14:29:11    7 14:29:11    8 14:29:11    9 14:29

Materialized View 'DEPT_SAL' created.

14:29:12 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30

4 rows selected.
```

2. INDEX 생성

```
CREATE INDEX IDX_DEPT ON DEPT_SAL(DEPTNO);
DESC DEPT_SAL;
```

```
14:29:19 SQL> CREATE INDEX IDX_DEPT ON DEPT_SAL(DEPTNO);

Index 'IDX_DEPT' created.

14:31:07 SQL> DESC DEPT_SAL;

COLUMN_NAME                               TYPE               CONSTRAINT
----------------------------------------- ------------------ --------------------
SUM(A.SALARY)                             NUMBER
DEPTNO                                    NUMBER(3)


INDEX_NAME                       TYPE                      COLUMN_NAME
-------------------------------- ------------------------- ----------------------
IDX_DEPT                         NORMAL                    DEPTNO
I_SNAP$_DEPT_SAL                 FUNCTION-BASED NORMAL     - EXPRESSION COLUMN -
```

3. 원본 테이블에 데이터 추가하고 실체화 뷰와 비교

```
CONN SYS/TIBERO
INSERT INTO EMPLOYEE VALUES(7000,'AMY','ANALYST',7902,'90-05-12',3000,50                                                );

CONN TEST/TEST
/*원본 테이블*/
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;

/*실체화 뷰*/
SELECT * FROM DEPT_SAL;
```

```
14:35:11 SQL> CONN TEST/TEST
Connected to Tibero.

14:35:18 SQL> SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;14:35:36    2 14:35:36    3

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30
         3000         50

5 rows selected.

14:35:37 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         7875         20
         6300         30
         5200         40

4 rows selected.
```

→ 뷰 아직 동기화되지 않아서 결과 다름.

4. 수동으로 원본 테이블과 실체화 뷰 데이터 동기화

```
BEGIN
DBMS_MVIEW.REFRESH('DEPT_SAL');
END;
/

SELECT * FROM DEPT_SAL;
```

```
14:35:51 SQL> BEGIN
14:35:59    2 DBMS_MVIEW.REFRESH('DEPT_SAL');
14:36:11    3 END;
14:36:13    4 /

PSM completed.

14:36:14 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         7875         20
         6300         30
         5200         40
         3000         50

5 rows selected.
```

5. TEST 사용자가 생성한 실체화 뷰 내용 조회 (USER_MVIEWS)

```
COL MVIEW_NAME FOR A20SELECT MVIEW_NAME, QUERY
FROM USER_MVIEWS
WHERE MVIEW_NAME='DEPT_SAL';
```

```
14:45:10 SQL> COL MVIEW_NAME FOR A20
14:45:19 SQL> SELECT MVIEW_NAME, QUERY
14:45:29    2 FROM USER_MVIEWS
14:45:33    3 WHERE MVIEW_NAME='DEPT_SAL';

MVIEW_NAME
--------------------
QUERY
--------------------------------------------------------------------------------
DEPT_SAL
SELECT SUM(A.SALARY), A.DEPTNO FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B WHERE A.DEP


1 row selected.
```

27. SQL 플랜

- SET AUTOT[RACE] {OFF|ON|TRACE[ONLY]}[옵션절];

27.1. 수행

수행내역
플랜 노드 정보 (EXECUTION PLAN)
수행 후 전체 통계 정보 (STATISTICS)
플랜 노드 별 수행 정보 (PLANSTAT)
PLUSTRACE ROLE TEST사용자에게 부여

27.2. 결과

1. 플랜 노드 정보 (EXECUTION PLAN)

```
CONN SYS/TIBERO

SET AUTOT ON EXPLAIN
SELECT SUM(SALARY), DEPTNO
FROM EMPLOYEE
GROUP BY DEPTNO;
```

```
SQL> SET AUTOT ON EXPLAIN
SQL> SELECT SUM(SALARY), DEPTNO
   2 FROM EMPLOYEE
   3 GROUP BY DEPTNO;

SUM(SALARY)     DEPTNO
----------- ----------
       3000         60
       8750         10
       5200         40
       7875         20
       6300         30
       2850
       3000         50

7 rows selected.

SQL ID: 454bkhmhypwzb
Child number: 126
Plan hash value: 4017526471

Execution Plan
--------------------------------------------------------------------------------
   1  GROUP BY (HASH) (Cost:7, %%CPU:0, Rows:5)
   2    TABLE ACCESS (ROWID): EMPLOYEE (Cost:7, %%CPU:0, Rows:15)
   3      INDEX (FAST FULL SCAN): EMP_ID_PK (Cost:6, %%CPU:0, Rows:15)


Note
--------------------------------------------------------------------------------
   3 - dynamic sampling used for this table (13 blocks)
```

2. 수행 후 전체 통계 정보 (STATISTICS)

```
CONN SYS/TIBERO

SET AUTOT ON EXPLAIN
SELECT SUM(SALARY), DEPTNO
FROM EMPLOYEE
GROUP BY DEPTNO;
```

```
SQL> SET AUTOT OFF
SQL> SET AUTOT ON STATISTICS
SQL> SELECT SUM(SALARY), DEPTNO
   2 FROM EMPLOYEE
   3 GROUP BY DEPTNO;

SUM(SALARY)     DEPTNO
----------- ----------
       3000         60
       8750         10
       5200         40
       7875         20
       6300         30
       2850
       3000         50

7 rows selected.


NAME                                VALUE
------------------------------ ----------
db block gets                           2
consistent gets                         5
physical reads                          0
redo size                               0
sorts (disk)                            0
sorts (memory)                          0
rows processed                          7
```

3. 플랜 노드 별 수행 정보 (PLANSTAT)

```
CONN SYS/TIBERO

SET AUTOT ON EXPLAIN
SELECT SUM(SALARY), DEPTNO
FROM EMPLOYEE
GROUP BY DEPTNO;
```

```
SQL> SET AUTOT OFF
SQL> SET AUTOT ON PLANSTAT
SQL> SELECT SUM(SALARY), DEPTNO
   2 FROM EMPLOYEE
   3 GROUP BY DEPTNO;

SUM(SALARY)     DEPTNO
----------- ----------
       3000         60
       8750         10
       5200         40
       7875         20
       6300         30
       2850
       3000         50

7 rows selected.

SQL ID: 454bkhmhypwzb
Child number: 126
Plan hash value: 4017526471

Execution Stat
--------------------------------------------------------------------------------
   1  GROUP BY (HASH) (Time:0. ms, Rows:0, Starts:0)
   2    TABLE ACCESS (ROWID): EMPLOYEE (Time:0. ms, Rows:0, Starts:0)
   3      INDEX (FAST FULL SCAN): EMP_ID_PK (Time:0. ms, Rows:0, Starts:0)
```

4. PLUSTRACE ROLE TEST사용자에게 부여

```
CONN SYS/TIBERO
CREATE ROLE PLUSTRACE'
GRANT SELECT ON VT_AUTOTRACESTAT TO PLUSTRACE;
GRANT SELECT ON V$SQL_PLAN TO PLUSTRACE;
GRANT SELECT ON V$SQL_PLAN_STATISTICS TO PLUSTRACE;
GRANT PLUSTRACE TO DBA WITH ADMIN OPTION;
GRANT PLUSTRACE TO TEST;

CONN TEST/TEST
```

```
13:06:55 SQL> CREATE ROLE PLUSTRACE;

Role 'PLUSTRACE' created.

13:06:59 SQL> GRANT SELECT ON VT_AUTOTRACESTAT TO PLUSTRACE;

Granted.

13:07:18 SQL> GRANT SELECT ON V$SQL_PLAN TO PLUSTRACE;

Granted.

13:07:31 SQL> GRANT SELECT ON V$SQL_PLAN_STATISTICS TO PLUSTRACE;

Granted.

13:07:45 SQL> GRANT PLUSTRACE TO DBA WITH ADMIN OPTION;

Granted.

13:07:53 SQL> GRANT PLUSTRACE TO TEST;

Granted.

13:07:59 SQL> CONN TEST/TEST
Connected to Tibero.
```

28. QUERY REWRITE-1

- QUERY REWRITE: A 사용자가 실체화 뷰를 생성해 놓았을 경우, 이 사실을 모르는 B가 A가 만든 똑같은 뷰와 같은 결과가 나오는 SQL문을 수행하였을 경우 → B는 SQL문장을 수행했지만, 같은 문장에 실체화 뷰가 존재한다면, 실체화 뷰를 조회하도록 쿼리가 다시 쓰여지는 것

28.1. 수행

수행내역
QUERY REWRITE 할 수 있도록 변환
TEST 사용자로 접속
실체화 뷰가 존재하는 SQL문장 실행
현재 OPRIMIZER_MODE 조회 및 변경
OPTIMIZER_MODE 변경 후 다시 SQL문장 실행

28.2. 결과

1. QUERY REWRITE 할 수 있도록 변환

```
/*SYS사용자에서 변경*/
CONN SYS/TIBERO
ALTER SYSTEM SET QUERY_REWRITE_ENABLED='TRUE';
```

```
13:02:14 SQL> CONN SYS/TIBERO
Connected to Tibero.

13:02:19 SQL> ALTER SYSTEM SET QUERY_REWRITE_ENABLED='TRUE';

System altered.
```

2. TEST 사용자로 접속

```
CONN TEST/TEST
```

```
13:02:28 SQL> CONN TEST/TEST
Connected to Tibero.
```

→ TEST사용자에게 TRACE하는 권한이 없음

3. 실체화 뷰가 존재하는 SQL문장 실행

```
SET AUTOTRACE ON
SELECT SUM(A.SALARY), A.DEPTNO
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
13:08:02 SQL> SET AUTOTRACE ON
13:08:13 SQL> SELECT SUM(A.SALARY), A.DEPTNO
13:08:21    2 FROM EMPLOYEE A, DEPARTMENT B
13:08:30    3 WHERE A.DEPTNO=B.DEPTNO
13:08:35    4 GROUP BY A.DEPTNO;

SUM(A.SALARY)     DEPTNO
------------- ----------
         6300         10
         5200         40
        10875         20
         6300         30

4 rows selected.
```

```
SQL ID: 702h2wbvwm9b1
Child number: 3266
Plan hash value: 3802988761

Execution Plan
----------------------------------------------------------------------------
   1  TABLE ACCESS (FULL) MV REWRITE: DEPT_SAL (Cost:23, %%CPU:0, Rows:4032)


NAME                                 VALUE
-------------------------------- ----------
db block gets                           298
consistent gets                          92
physical reads                            0
redo size                                60
sorts (disk)                              0
sorts (memory)                            3
rows processed                            4
```

4. 현재 OPRIMIZER_MODE 조회 및 변경

```
SHOW PARAM OPTIMIZER_MODE;
ALTER SESSION SET OPTIMIZER_MODE='FIRST_ROWS_1';
```

```
13:29:42 SQL> SHOW PARAM OPTIMIZER_MODE;

NAME                           TYPE     VALUE
------------------------------ -------- -----------
OPTIMIZER_MODE                 STRING   ALL_ROWS
```

```
13:09:07 SQL> ALTER SESSION SET OPTIMIZER_MODE='FIRST_ROWS_1';

Session altered.
```

5. OPTIMIZER_MODE 변경 후 다시 SQL문장 실행

```
SELECT SUM(A.SALARY), A.DEPTNO
FROM EMPLOYEE A, DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
SQL ID: 702h2wbvwm9b1
Child number: 3278
Plan hash value: 1895546394

Execution Plan
----------------------------------------------------------------------------
   1  TABLE ACCESS (ROWID): DEPT_SAL (Cost:410, %%CPU:0, Rows:4032)
   2    INDEX (FAST FULL SCAN) MV REWRITE: I_SNAP$_DEPT_SAL (Cost:6, %%CPU:0, Ro
ws:4032)
```

→ ALL_ROWS : 통계 정보의 유무와 상관없이 최대 처리량을 목표로 최적의 경로 찾음

→ FIRST_ROWS)N : 통계정보 유무와 관계없이 처음 N개 빠르게 추출할 수 있도록 최적의 경로 찾음

→ 실제 테이블을 사용하지 않고 실체회 뷰를 이용하여 QUERY REWRITE됨

29. QUERY REWRITE (HINT 사용)

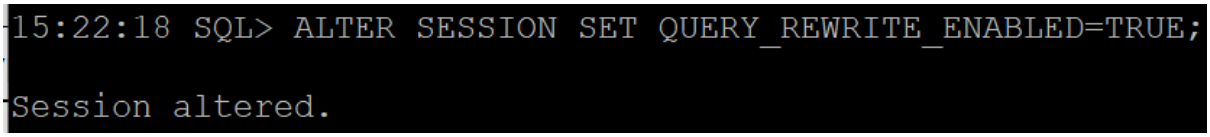
29.1. 수행

수행내역
QUERY REWRITE SESSION 변경
원본 테이블 조회
실체화 뷰 생성
실체화 뷰 사용하도록 힌트 구성

29.2. 결과

1. QUERY REWRITE SESSION 변경

```
ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE;
```



2. 원본 테이블 조회

```
SET AUTOTRACE ON
SET TIMING ON

SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
6 rows selected.

Total elapsed time 00:00:00.000982

SQL ID: 7at0um7n73n4q
Child number: 3544
Plan hash value: 1971386726

Execution Plan
----------------------------------------------------------------------------------
   1  GROUP BY (HASH) (Cost:30, %%CPU:0, Rows:5)
   2    HASH JOIN (Cost:30, %%CPU:0, Rows:15)
   3      TABLE ACCESS (FULL): DEPARTMENT (Cost:23, %%CPU:0, Rows:6)
   4      TABLE ACCESS (ROWID): EMPLOYEE (Cost:7, %%CPU:0, Rows:15)
   5        INDEX (FAST FULL SCAN): EMP_ID_PK (Cost:6, %%CPU:0, Rows:15)


Predicate Information
----------------------------------------------------------------------------------
   2 - access: ("B"."DEPTNO" = "A"."DEPTNO") (0.167)


Note
----------------------------------------------------------------------------------
   3 - dynamic sampling used for this table (13 blocks)
   5 - dynamic sampling used for this table (13 blocks)


NAME                                VALUE
------------------------------ ----------
db block gets                           3
consistent gets                        19
physical reads                          0
redo size                               0
sorts (disk)                            0
sorts (memory)                          0
rows processed                          6
```

3. 실체화 뷰 생성

```
CREATE MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
15:55:09 SQL> CREATE MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;15:55:18    2 15:55:18    3 15
6 15:55:18    7 15:55:18    8 15:55:18    9 15:5

Materialized View 'DEPT_SAL' created.
```

4. 실체화 뷰 사용하도록 힌트 구성

```
SELECT /*+REWRITE(DEPT_SAL)*/ SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
```

```
SUM(A.SALARY)     DEPTNO
------------- ----------
         3000         60
         8750         10
         5200         40
         7875         20
         6300         30
         3000         50

6 rows selected.

Total elapsed time 00:00:00.020252

SQL ID: fhv1jwt64nq2z
Child number: 3584
Plan hash value: 1872705364

Execution Plan
----------------------------------------------------------------------------------
   1  TABLE ACCESS (FULL) MV REWRITE: DEPT_SAL (Cost:23, %%CPU:0, Rows:4032)


NAME                                VALUE
------------------------------ ----------
db block gets                         662
consistent gets                       479
physical reads                          0
redo size                            1052
sorts (disk)                            0
sorts (memory)                         29
rows processed                          6
```

30. 실체화뷰 REFRESH하기

- DBMS_MVIEW PACKAGE 사용

30.1. 수행

수행내역
ON DEMAND 실체화 뷰 생성
ON COMMIT 실체화 뷰 생성
원본 테이블에 데이터 추가후 COMMIT
원본 테이블 vs 실체화 뷰
뷰 기준 갱신
테이블 기준 실체화 뷰 갱신
모든 실체화 뷰 모두 갱신

30.2. 결과

1. ON DEMAND 실체화 뷰 생성

```
CREATE MATERIALIZED VIEW DEPT_SAL
BUILD IMMEDIATE
REFRESH
COMPLETE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;

SELECT * FROM DEPT_SAL;
```

```
Materialized View 'DEPT_SAL' created.

14:53:56 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30

4 rows selected.
```

2. ON COMMIT 실체화 뷰 생성

```
CREATE MATERIALIZED VIEW DEPT_SAL_COMM
BUILD IMMEDIATE
REFRESH
COMPLETE
ON COMMIT
ENABLE QUERY REWRITE
```

```
AS
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;

SELECT * FROM DEPT_SAL_COMM;
```

```
Materialized View 'DEPT_SAL_COMM' created.

14:54:38 SQL> SELECT * FROM DEPT_SAL_COMM;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30

4 rows selected.
```

→ ON COMMIT조건은 원본 테이블에 변경이 일어난 후 COMMIT을 하면 실체화 뷰도 REFRESH됨

3. 원본 테이블에 데이터 추가후 COMMIT

```
CONN SYS/TIBERO
INSERT INTO EMPLOYEE VALUES(7000,'AMY','ANALYST',7902,'90-05-12',3000,50);
COMMIT;
```

```
14:51:31 SQL> INSERT INTO EMPLOYEE VALUES(7000,'AMY','ANALYST',7902,'90-05-12',3000,50
                                          );

1 row inserted.

14:51:33 SQL> COMMIT;

Commit completed.
```

4. 원본 테이블 vs 실체화 뷰

```
CONN TEST/TEST
/*원본 테이블*/
SELECT SUM(A.SALARY), A.DEPTNO
FROM SYS.EMPLOYEE A, SYS.DEPARTMENT B
WHERE A.DEPTNO=B.DEPTNO
GROUP BY A.DEPTNO;
/*ON DEMAND VIEW */
SELECT * FROM DEPT_SAL;
/*ON COMMIT VIEW*/
SELECT * FROM DEPT_SAL_COMMIT;
```

```
SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30
         3000         50

5 rows selected.
```

```
SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30

4 rows selected.

14:55:35 SQL> SELECT * FROM DEPT_SAL_COMM;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30
         3000         50

5 rows selected.
```

5. 뷰 기준 갱신

```
BEGIN
DBMS_MVIEW.REFRESH('DEPT_SAL');
END;
/

SELECT * FROM DEPT_SAL;
```

```
14:55:45 SQL> BEGIN
14:55:56    2 DBMS_MVIEW.REFRESH('DEPT_SAL');
14:56:06    3 END;
14:56:08    4 /

PSM completed.

14:56:08 SQL> SELECT * FROM DEPT_SAL;

SUM(A.SALARY)     DEPTNO
------------- ----------
         8750         10
         5200         40
         7875         20
         6300         30
         3000         50

5 rows selected.
```

6. 테이블 기준 실체화 뷰 갱신 & 모든 실체화 뷰 갱신 - ERROR

```
BEGIN
DBMS_MVIEW.REFRESH_DEPENDENT('EMPLOYEE');
END;
/

BEGIN
DBMS_MVIEW.REFRESH_ALL_VIEWS;
END;
/
```

```
14:56:12 SQL> BEGIN
14:56:17    2 DBMS_MVIEW.REFRESH_DEPENDENT('EMPLOYEE');
14:56:30    3 END;
14:56:33    4 /
TBR-15146: PSM compilation error.

TBR-15046: Identifier is out of scope.
at line 2, column 1 of null:
DBMS_MVIEW.REFRESH_DEPENDENT('EMPLOYEE');
^^^^^^^^^^
```

```
15:00:41 SQL> BEGIN
15:00:57    2 DBMS_MVIEW.REFRESH_ALL_VIEWS;
15:01:06    3 END;
15:01:08    4 /
TBR-15146: PSM compilation error.

TBR-15046: Identifier is out of scope.
at line 2, column 1 of null:
DBMS_MVIEW.REFRESH_ALL_VIEWS;
^^^^^^^^^^
```