티베로 실무SQL 튜닝가이드

TmaxTibero

Tibero Practical SQL Tuning Guide

저자 Park jong hyun

책 소개

티베로 SQL실무 가이드는 티베로7 DBMS를 사용하여 실무에서 SQL 튜닝 능력 향상을 목표로 한다. 이 책은 이론과 실기로 구성되어 있으며, 최소한 SQL(문법, 인덱스, 조인)에 대한 기본 지식이 있어야 한다.

이론 부분은 실기 부분을 하기 전에 반드시 알아야 할 튜닝 기술을 다룬다. 각 튜닝 기술에 대해 심도 있게 알아 볼 것이다.

- SQL 성능정보를 추출하는 방법
- 인덱스 사용과 Table Full Scan 비교
- NL 조인과 Hash 조인 비교
- 힌트 종류
- 파티션 인덱스의 종류와 차이

실기 부분은 실무에서 자주 접하는 튜닝 포인터를 8가지로 정리하였다. 아울러 티베로7에서 실기 과정을 따라 할 수 있는 스크립트(테이블과 데이터 생성)가 제공되어 튜닝 전/후를 체감할 수 있다.

- NL 조인과 Hash 조인을 언제 선택해야 하는가
- IN 서브쿼리를 상황에 맞게 처리하는 방법
- EXISTS 서브쿼리를 상황에 맞게 처리하는 방법
- 쿼리를 UNION으로 분리해야 하는 이유
- 동일 테이블의 반복 사용을 제거하는 방법
- 테이블을 한 번만 액세스하기 위해 분석 함수를 사용하는 방법
- SQL을 페이지 처리시 튜닝하는 방법
- 대용량 테이블 및 배치에서 파티션을 사용하는 방법

티베로 SQL실무 가이드의 실기 부분을 충분히 이해하면, 실무에서 많은 SQL 튜닝에 적용할 수 있으며, 특히 SQLP의 실기 시험에서 좋은 점수를 받을 것으로 생각한다.

SQLP 시험은 이론 70점과 실기 30점으로 구성되며 합격 기준은 75점이다. 많은 시험 응시자가 실기를 힘들어 한다. SQLP 실기는 2문항이며, 지문에 나온 실행계획을 보고 SQL을 작성(힌트 사용)하거나, 주어진 SQL을 힌트와 인덱스를 사용하여 튜닝하는 식이다. 티베로 SQL실무 가이드의 실기와 동일한 유형이다.

과목명	필기		실기		검정시험시간
	문항수	배점	문항수	배점	
데이터 모델링의 이해	10	10 (문항당 1점)			
SQL 기본 및 활용	20	20 (문항당 1점)	2	30 (문항당 15점)	180분(3시간)
SQL 고급활용 및 튜닝	40	40 (문항당 1점)			
계	70	70	2	30	

끝으로, 티베로 실무SQL 튜닝가이드 제작에 많은 도움을 주신 박명애 사장님, 상품화정책팀, 연구소, Expert팀 에게 감사를 드립니다.

스크립트

실기 과정을 따라 할 수 있도록 테이블과 테스트 데이터를 생성하는 스크립트가 제공된다. 스크립트를 실행하기 전에 다음 사항을 준비한다.

- 티베로7 설치 (Online Redo Log 그룹은 3개 이상, 멤버 크기 3G 이상, GATHER_SQL_PLAN_STAT=YES)
- 테스트 데이터 적재 용 테이블스페이스 생성 (약 60G)
- 테스트 DB 계정 생성(CONNECT/RESOURCE/SELECT ANY DICTIONARY 권한 필요)

스크립트 파일명은 생성되는 테이블 이름과 동일하다.
하나의 스크립트를 실행하면 해당 테이블에 대해 Drop Table → Create Table → PK 생성 → Data Load → 통계정보 수집 → 건수 확인으로 진행된다.

생성되는 테이블 이름은 T_ 또는 P_ 로 시작되는데, T_ 는 Non-Partitioned 테이블이며, P_ 는 Partitioned 테이블이다. 테이블 이름 뒤에 번호(1,2,...)가 붙는 경우 해당 장(Chapter)에서만 사용되는 테이블이며(예로 T_ORDER8 테이블은 8장 에서만 사용되는 테이블), 테이블 이름 뒤에 번호가 없는 경우(예로 T_CUST 테이블은 여러 장에서 공용으로 사용하는 테이블) 이다.

각 장에서 필요한 스크립트 목록은 다음과 같다. 스크립트 실행 순서는 없으나 T_ORDER_D3.sql 과 P_PAY8.sql 는 내부적으로 "Insert Into ... Select From ..." 을 사용하므로 먼저 실행되어야 할 스크립트가 있다.(아래 표 참조)

장(Chapter)	필요한 스크립트 파일
1.조인	T_ORDER_STA1.sql T_ORDER1.sql T_CUST.sql
2.IN 서브쿼리	T_COMP2.sql T_AGENCY2.sql T_CONT2.sql T_CONT_STA2.sql
3.EXISTS 서브쿼리	T_CUST.sql T_CUST_GRAD.sql T_GOODS.sql T_ORDER3.sql T_ORDER_D3.sql (T_ORDER3 테이블을 먼저 생성하고 실행할 것) T_GOODS_P3.sql T_PARTS3.sql
4.쿼리 분리	T_EMP.sql T_EMP_TX4.sql T_ORDER4.sql T_CUST4.sql
5.반복 제거	T_CUST.sql T_GOODS.sql T_SALE_LIST.sql T_SCORE_RESULT6.sql (원래 6장에서 사용할 시나리오가 5장으로 이동) T_HIGH_SCORE6.sql (원래 6장에서 사용할 시나리오가 5장으로 이동)
6.분석 함수	T_EMP.sql T_SALE_LIST.sql T_SCORE_RESULT6.sql T_SCORE_SM6.sql
7.페이지 처리	T_CUST_LOGIN7.sql T_CUST.sql T_SALE_LIST.sql T_GOODS.sql
8.파티션	P_ORDER8.sql T_CUST.sql T_CUST_GRAD.sql T_ORDER8.sql P_PAY8.sql (P_ORDER8 테이블을 먼저 생성하고 실행할 것) P_ORDER_D8.sql T_GOODS.sql

목차

이론

실기

이론

1. SQL 성능정보

1. SQL 성능정보

특정 SQL의 성능정보를 수집하는 방법에는 여러 가지가 있다. 그 중에서 SQL 튜닝 시 꼭 필요한 성능정보(실행계획, 소요시간, 읽은 블럭양, 처리 Row개수 등)를 쉽게 수집하는 DBMS_XPLAN.DISPLAY_CURSOR 를 사용하는 방법에 대해 설명한다.

1. GATHER_PLAN_STAT 초기화 파라미터 설정

GATHER_PLAN_STAT 초기화 파라미터는 SQL 수행 시 Plysical Plan Cache에 SQL 성능정보의 수집 여부를 결정하기 때문에 켜져 있어야 한다. 디폴트 값은 N 이다. SELECT_ANY_DICTIONARY 권한은 필요 없다.

1) GATHER_PLAN_STAT 초기화 파라미터 확인 방법

```
SQL> SHOW PARAMETER GATHER_SQL_PLAN_STAT

NAME                          TYPE    VALUE
----------------------------- ------- -------
GATHER_SQL_PLAN_STAT          Y_N     YES
```

2) GATHER_PLAN_STAT 초기화 파라미터 설정 방법

(방법1)
$TB_HOME/config/$TB_SID.tip 파일(티베로 초기화 파라미터 파일, TIP 파일)에 GATHER_PLAN_STAT 초기화 파라미터 값을 Y로 설정한다.

```
[tibero@tibero7-svr ~]$ cd $TB_HOME
[tibero@tibero7-svr tibero7]$ cd config
[tibero@tibero7-svr config]$ cat tibero.tip
# tip file generated from /home/tibero/tibero7/config/tip.template (Mon Feb 19 18:53:14
KST 2024)
#------------------------------------------------------------------------------
# RDBMS initialization parameter
#------------------------------------------------------------------------------

DB_NAME=tibero
LISTENER_PORT=8629
CONTROL_FILES="/home/tibero/tibero7/database/tibero/c1.ctl"
#CERTIFICATE_FILE="/home/tibero/tibero7/config/tb_wallet/tibero.crt"
#PRIVKEY_FILE="/home/tibero/tibero7/config/tb_wallet/tibero.key"
#WALLET_FILE="/home/tibero/tibero7/config/tb_wallet/WALLET"
#ILOG_MAP="/home/tibero/tibero7/config/ilog.map"
MAX_SESSION_COUNT=20
MEMORY_TARGET=3G
TOTAL_SHM_SIZE=2G

gather_sql_plan_stat=Y
```

(방법2)
GATHER_PLAN_STAT 초기화 파라미터는 세션 또는 시스템 레벨에서 동적으로 적용 가능하다. 아래는 세션 레벨에서 GATHER_PLAN_STAT 초기화 파라미터를 Y로 설정한다.

```
SQL> ALTER SESSION SET GATHER_SQL_PLAN_STAT=Y;

SESSION ALTERED.

SQL> SHOW PARAMETER GATHER_SQL_PLAN_STAT

NAME                          TYPE     VALUE
----------------------------- -------- ------
GATHER_SQL_PLAN_STAT          Y_N      YES
```

1. SQL 성능정보 → CARDS + COST + PART + ELAPTIME + LAST

2. DBMS_XPLAN.DISPLAY_CURSOR 함수

Physical Plan Cache에 등록되어 있는 여러 SQL 중 에서 SQL_ID 와 CHILD_NO로 특정 SQL을 지정하고, FORMAT 으로 성능정보를 출력할 항목을 선택할 수 있다.

1) 프로토타입

```
DBMS_XPLAN.DISPLAY_CURSOR
(
   IN_SQL_ID       NUMBER     DEFAULT NULL,
   IN_CHILD_NO     NUMBER     DEFAULT NULL,
   FORMAT          VARCHAR2   DEFAULT 'BASIC LAST SQL'
)
RETURN DBMS_XPLAN_TYPE_TABLE PIPELINED;
```

파라미터	설명
IN_SQL_ID	성능정보를 출력하려는 대상 SQL의 SQL ID 값 이다. 입력을 생략한 경우에는 해당 세션의 마지막 수행 된 SQL의 SQL ID 값을 사용한다.
IN_CHILD_NO	성능정보를 출력하려는 대상 SQL의 CHILD NUMBER 값 이다. 입력을 생략한 경우에는 해당 SQL ID를 가지는 모든 성능정보를 출력한다. CHILD NO 값은 SQL ID 값이 입력된 경우에만 지정할 수 있다.
FORMAT	출력하고자 하는 항목을 지정한다. 항목에는 개별 항목과 개별 항목 여러 개를 묶은 그룹 항목이 있다. 항목 이름 앞에 하이픈(-)를 붙이면 해당 항목을 제외할 수 있다.

2) FORMAT의 개별항목

개별 항목	설명
CARDS	OPTIMIZER가 예측한 Row 개수이다. 기본값이다.
COST	OPTIMIZER에서 예측한 COST이다. 기본값이다.
PARTITION	파티션 관련 정보이다. 기본값이다.
PARALLEL	PARALLEL EXECUTION 관련 정보이다.
PREDICATE	PREDICATE 정보이다.
REMOTE	데이터베이스 링크를 수행한 쿼리 내용이다.
ROWS	실제 처리된 Row 개수이다. 오라클의 ROWS는 예상 Row 개수이다.
ELAPTIME	실제 수행된 시간이다. 기본값이다.
USEDMEM	실제 사용된 메모리 양이다.
TEMPREAD	실제 수행된 TEMP READ 개수이다.
TEMPWRITE	실제 수행된 TEMP WRITE 개수이다.
BUFGETS	실제 액세스한 BUFFER GET 개수이다
STARTS	실제 수행한 횟수이다.
LAST	성능정보 값을 마지막에 수행한 값만 출력한다. 지정하지 않은 경우 성능정보 값은 모든 수행에 대한 누적 값을 보여준다. 기본값이다.
PRECISE	CARDS, ROWS에 대해 반올림 없이 실제 값을 보여준다
HEADER	플랜의 기본 정보(SQL ID, HASH VALUE, 총 수행 횟수, 총 패치 횟수, 플랜 수행 시간)를 보여준다. 기본값이다.
SQL	성능정보 출력에 사용된 SQL 문장을 보여준다. 기본값이다.
READS	실제 수행된 디스크 I/O 횟수개수이다.

1. SQL 성능정보

3) FORMAT의 그룹항목

그룹 항목	설명
IOSTATS	수행 정보 중 IO와 관련된 모든 항목을 보여준다. → TEMPREAD + TEMPWRITE + BUFGETS + READS
MEMSTATS	수행 정보 중 메모리와 관련된 모든 항목을 보여준다. → USEDMEM
ALLSTATS	모든 수행 정보를 보여 준다. → IOSTATS + MEMSTATS
BASIC	출력 기본 포맷으로 마지막 수행에 대한 optimizer에서 예측한 cardinality, cost와 노드별 수행 시간을 출력한다. → CARDS + COST + PART + ELAPTIME + LAST
TYPICAL	BASIC 포맷에 추가로 마지막 수행에 대한 노드별 처리 row 수와 predicate 정보, remote sql 정보를 출력한다. → BASIC + PE + ROWS + STARTS + PRED + REMOTE + PRECISE
ALL	TYPICAL 포맷에 ALLSTATS 항목을 보여준다. → TYPICAL + ALLSTATS

3. 성능정보 추출 방법

다음 예제는 DBMS_XPLAN.DISPLAY_CURSOR 함수의 기본값으로 성능정보를 출력한다.

```
SQL> select  /*+ leading(o c) use_nl(c)  */
       count(*),sum(o.ORDER_AMT), sum(o.ORDER_QY), max(o.ORDER_DY)
  from T_ORDER1 o,
       T_CUST   c
 where ORDER_DY between '20230901' and '20231030'
   and o.CUST_ID = c.CUST_ID
   and c.cust_cd = 'z0005';

COUNT(*) SUM(O.ORDER_AMT) SUM(O.ORDER_QY) MAX(O.ORDER_DY)
---------- --------------- --------------- ---------------
       700          140000            7000 20231015

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR());

SQL_ID          : 40yx6fvg9z3g9
CHILD NUMBER    : 161
HASH VALUE      : 3734998505
PLAN HASH VALUE: 1882707456
EXECUTIONS      : 7
FETCHES         : 7
LOADED AT       : 2024/06/01 20:34:34
TOT ELAPSED TIME: 00:00:03.8322
AVG ELAPSED TIME: 00:00:00.5475

SQL             : select  /*+ leading(o c) use_nl(c)  */
       count(*),sum(o.ORDER_AMT), sum(o.ORDER_QY), max(o.ORDER_DY)
  from T_ORDER1 o,
       T_CUST   c
 where ORDER_DY between '20230901' and '20231030'
   and o.CUST_ID = c.CUST_ID
   and c.cust_cd = 'z0005'

-------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name       | Cost (%CPU)    | Cards    | Elaps. Time   |
-------------------------------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION            |            | 12930    (.69)|        1 |00:00:00.0000 |
|   2 |  SORT AGGR                   |            | 12930    (.69)|        1 |00:00:00.0001 |
|   3 |   INDEX JOIN                 |            | 12930    (.69)|      968 |00:00:00.0018 |
|   4 |    TABLE ACCESS (FULL)       | T_ORDER1   |  4165    (.94)|     136K |00:00:00.1153 |
|   5 |    TABLE ACCESS (ROWID)      | T_CUST     |     3      (0)|        1 |00:00:00.0280 |
|   6 |     INDEX (UNIQUE SCAN)      | PK_T_CUST  |     2      (0)|        1 |00:00:00.1528 |
-------------------------------------------------------------------------------------------------------
```

기본값 (HEADER, SQL, COST, CARDS, PARTITION, ESTIMATE, LAST) 이 사용된다.

HEADER

SQL

COST + CARDS + PARTITION + ESTIMATE + LAST

1. SQL 성능정보

다음 예제는 DBMS_XPLAN.DISPLAY_CURSOR 함수를 사용할 때 실무에서 많이 사용하는 값으로 성능정보를 출력한다.

```
SQL> select  /*+ leading(o c) use_nl(c)  */
       count(*),sum(o.ORDER_AMT), sum(o.ORDER_QY), max(o.ORDER_DY)
  from T_ORDER1 o,
       T_CUST   c
 where ORDER_DY between '20230901' and '20231030'
   and o.CUST_ID = c.CUST_ID
   and c.cust_cd = 'z0005';

COUNT(*) SUM(O.ORDER_AMT) SUM(O.ORDER_QY) MAX(O.ORDER_DY)
---------- ---------------- --------------- ---------------
       700           140000            7000 20231015

SQL> SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(NULL,NULL, FORMAT=>'PARTITION ROWS
ELAPTIME BUFGETS STARTS LAST PREDICATE'));

-----------------------------------------------------------------------------------------------
|ID| Operation                | Name      |Rows     | Elaps. Time  | CR Gets | Starts  |
-----------------------------------------------------------------------------------------------
| 1| COLUMN PROJECTION        |           |      1  |00:00:00.0000 |       0 |       1 |
| 2|  SORT AGGR               |           |      1  |00:00:00.0001 |       0 |       1 |
| 3|   INDEX JOIN             |           |    700  |00:00:00.0018 |       0 |       1 |
| 4|    TABLE ACCESS (FULL)   | T_ORDER1  |    140K |00:00:00.0629 |    9381 |       1 |
| 5|    TABLE ACCESS (ROWID)  | T_CUST    |    700  |00:00:00.0228 |     140K|     140K|
| 6|     INDEX (UNIQUE SCAN)  | PK_T_CUST |    140K |00:00:00.1474 |     280K|     140K|
-----------------------------------------------------------------------------------------------

Pedicate Information
-----------------------------------------------------------------------------------------------
   4 - filter: ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY" <= '20231030') (0.090
* 1.000)
   5 - filter: ("C"."CUST_CD" = 'z0005') (0.006)
   6 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
```

1. SQL 성능정보

4. 성능정보 보는 방법

실행계획은 위에서 아래로, 안쪽에서 바깥쪽으로, 조인은 2개를 묶어서 한 쌍으로 해석한다.

항목	설명
Rows	각 단계에서 실제로 추출된(엑세스한) Row 개수
Elaps. Time	각 단계에서 실제로 소요된 시간
CR Gets	각 단계에서 실제 읽은 블록 개수 오라클의 경우 상위 단계는 하위 단계의 블록 개수를 포함한다
Starts	각 단계에서 실제 실행한 횟수

① 제일 안쪽에 있는 고객을 읽어 (이때 고객코드 컬럼의 인덱스 사용)
② 주문의 고객코드+주문일시 인덱스를 사용하여
③ Index 조인을 한다.
④ 주문상태표 테이블을 Full Scan으로 읽어
⑤ 고객과 주문을 조인한 중간집합(③)과 Hash 조인을 한다.
⑥ Hash 조인 결과를 Sort 방식을 사용하여 Order By 한다.

```
SELECT /*+  LEADING(C O S)
            USE_NL(O) USE_HASH(S)
            INDEX(C IX_T_CUST_CD) INDEX(O IX_T_ORDER1_ID_DT) FULL(S)
       */
       O.ORDER_DY, C.CUST_ID, C.CUST_NM,  O.ORDER_NO, O.ORDER_AMT, O.ORDER_QY
  FROM T_CUST         C,
       T_ORDER1       O,
       T_ORDER_STA1   S
 WHERE C.CUST_CD = 'Z0005'
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_STA_CD = S.ORDER_STA_CD
   AND O.ORDER_DY BETWEEN '20230901' AND '20231030'
   AND O.ORDER_QY = 1
   AND S.ORDER_STA = '구매확정'
   AND S.CM IS NOT NULL
 ORDER BY O.ORDER_DY, C.CUST_ID;
```

각 단계에서 추출된 Row 개수
각 단계에서 소요된 시간

```
-----------------------------------------------------------------------------------------------------------
|ID|   Operation                    | Name                |  Rows  | Elaps. Time  | CR Gets | Starts|
-----------------------------------------------------------------------------------------------------------
| 1|⑥ ORDER BY (SORT)               |                     |      1 | 00:00:00.0000|       0 |      1|
| 2|  ⑤ HASH JOIN                   |                     |      1 | 00:00:00.0000|       0 |      1|
| 3|    ③ INDEX JOIN                |                     |      1 | 00:00:00.0001|       0 |      1|
| 4|       ① TABLE ACCESS (ROWID)   | T_CUST              |    350 | 00:00:00.0022|     350 |      1|
| 5|           INDEX (RANGE SCAN)   | IX_T_CUST_CD        |    350 | 00:00:00.0001|       2 |      1|
| 6|       ② TABLE ACCESS (ROWID)   | T_ORDER1            |      1 | 00:00:00.0017|     700 |    350|
| 7|           INDEX (RANGE SCAN)   | IX_T_ORDER1_ID_DY   |    700 | 00:00:00.0019|     379 |    350|
| 8|    ④ TABLE ACCESS (FULL)       | T_ORDER_STA1        |      1 | 00:00:00.0000|      14 |      1|
-----------------------------------------------------------------------------------------------------------
```

각 단계에서 읽은 블록 개수
각 단계에서 수행한 횟수

각 단계에서 사용된 조건절

```
Predicate Information
-----------------------------------------------------------------------------------------------------------
   2 - access: ("O"."ORDER_STA_CD" = "S"."ORDER_STA_CD") (1.000)
   5 - access: ("C"."CUST_CD" = 'z0005') (0.006)
   6 - filter: ("O"."ORDER_QY" = 1) (0.000)
   7 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230901') AND
("O"."ORDER_DY" <= '20231030') (0.002 * 1.000 * 1.000)
   8 - filter: ("S"."ORDER_STA" = '구매확정') AND ("S"."CM" IS NOT NULL) (0.200 * 1.000)
```

이론

2. 인덱스

2. 인덱스

인덱스 처리 과정과 Table Full Scan 처리 과정을 확인하여 언제 인덱스를 사용하는 것이 성능에 좋은지 확인한다.
Buffer Pinning 과 Index Clustering Factor 를 이해하면 성능정보의 블록 개수의 차이를 알 수 있다.
여러 컬럼으로 구성된 결합 인덱스를 생성할 때, 컬럼 순서의 중요성을 확인한다.

1. Table Full Scan 과 Index Range Scan 비교

1) 테스트 환경 구성

```
SQL> CREATE TABLE TEST1 ( CUST_ID     VARCHAR2(10)    NOT NULL,  -- pk
                          CUST_NM     VARCHAR2(30)    NOT NULL,
                          CUST_TY     VARCHAR2(1)     NOT NULL,
                          TELNO       VARCHAR2(30),
                          ADDRESS     VARCHAR2(30),
                          BRTH_DY     VARCHAR2(8)          );
Table 'TEST1' created.

SQL> ALTER TABLE TEST1 ADD CONSTRAINT PK_TEST1 PRIMARY KEY (CUST_ID);
Table 'Test1' altered.

SQL> INSERT INTO TEST1
     SELECT CUST_ID, CUST_NM, CUST_TY, TELNO, ADDRESS, BIRTH_DY FROM T_CUST;
70000 rows inserted.

SQL> COMMIT;
Commit completed.

SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('TEST02','TEST1');
PSM completed.

SQL> SELECT TABLE_NAME, NUM_ROWS,  BLOCKS
       FROM USER_TABLES
      WHERE TABLE_NAME = 'TEST1';

TABLE_NAME                     NUM_ROWS  BLOCKS
------------------------------ --------- -------
TEST1                             70000     512

SQL> SELECT OWNER, SEGMENT_NAME, EXTENT_ID, BLOCKS
       FROM DBA_EXTENTS
      WHERE OWNER ='TEST02'
        AND SEGMENT_NAME ='TEST1';

OWNER      SEGMENT_NA EXTENT_ID   BLOCKS
---------- ---------- --------- --------
TEST02     TEST1               0       16
TEST02     TEST1               1       16
TEST02     TEST1               2       16
TEST02     TEST1               3       16
TEST02     TEST1               4       16
TEST02     TEST1               5       16
TEST02     TEST1               6       16
TEST02     TEST1               7       16
TEST02     TEST1               8      128
TEST02     TEST1               9      128
TEST02     TEST1              10      128

SQL> SHOW PARAMETER DB_FILE_MULTIBLOCK_READ_COUNT;

NAME                                 TYPE    VALUE
------------------------------ --------- --------
DB_FILE_MULTIBLOCK_READ_COUNT      INT32       32
```

2. 인덱스

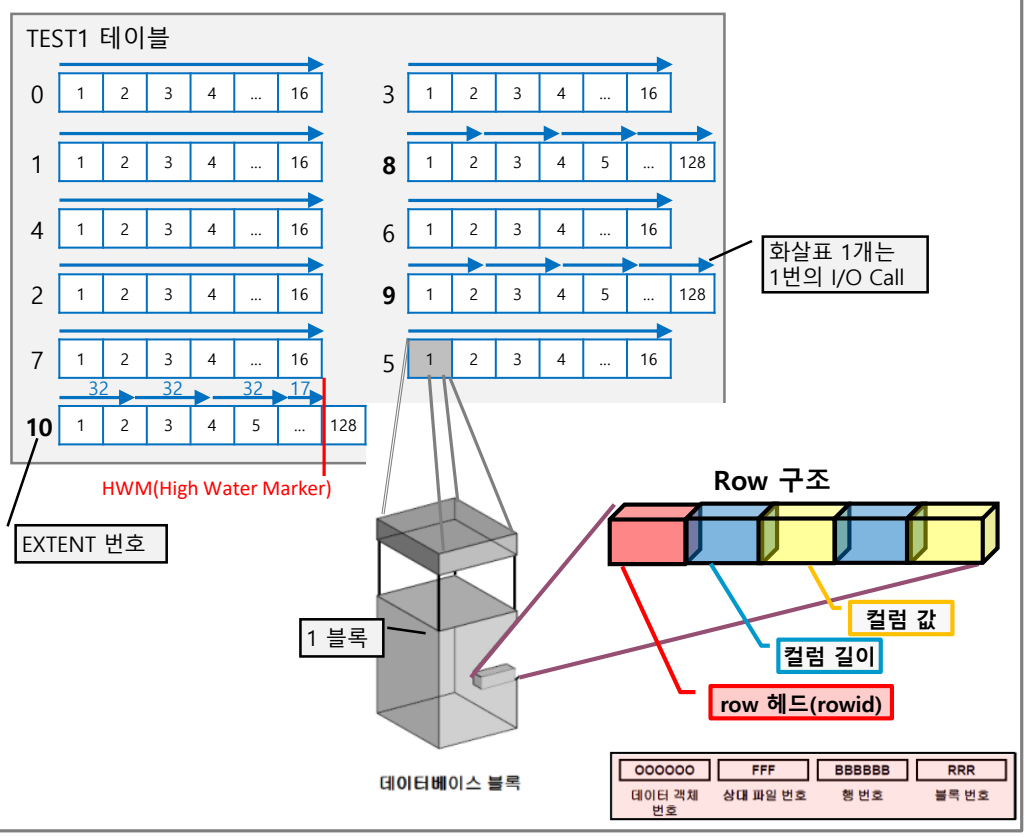
2) Table Full Scan 테스트

아래 SQL 실행 시 CUST_NM 컬럼에 인덱스가 없으면 어떻게 수행될까?

```
SQL> SELECT *
       FROM TEST1
      WHERE CUST_NM = 'KIM C RI';

CUST_ID     CUST_NM   CUST_TY  TELNO          ADDRESS
----------  --------  -------  -------------  -------
0000000367  KIM C RI  A        010-1234-0366  C City
0000006217  KIM C RI  A        010-1234-6216  C City
0000009701  KIM C RI  C        010-1234-9700  C City
0000067707  KIM C RI  A        010-1234-7706  C City

-------------------------------------------------------------------------------
| ID  | Operation          | Name  | Rows     | Elaps. Time | CR Gets | Starts |
-------------------------------------------------------------------------------
|   1 | TABLE ACCESS (FULL)| TEST1 |        4 |00:00:00.0043|     497 |      1 |
-------------------------------------------------------------------------------
   1 - filter: ("TEST1"."CUST_NM" = 'KIM C RI') (0.000)
```

실행계획을 보면 인덱스를 사용할 수 없기 때문에 당연히 TEST1 테이블을 Full Scan 해서 총 497 블록을 읽어 4건을 조회한다. 이때 0번 EXTENT → 1번 EXTENT … → 10번 EXTENT 순서로 읽는다. 1번의 I/O Call(파랑색 화살표)로 최대 연속된 DB_FILE_MULTIBLOCK_READ_COUNT 개수의 블록을 읽는다.

따라서 0번 ~ 7번 EXTENT 까지는 EXTENT 마다 I/O Call이 1번 씩 발생하고, 8번~ 10번 EXTENT는 I/O Call이 4번 씩 발생해서 총 20번 I/O Call이 발생한다. Table Full Scan 시 HWM 까지 읽기 때문에 TEST1 테이블 전체가 512 블록인데 497 블록만 읽는다.



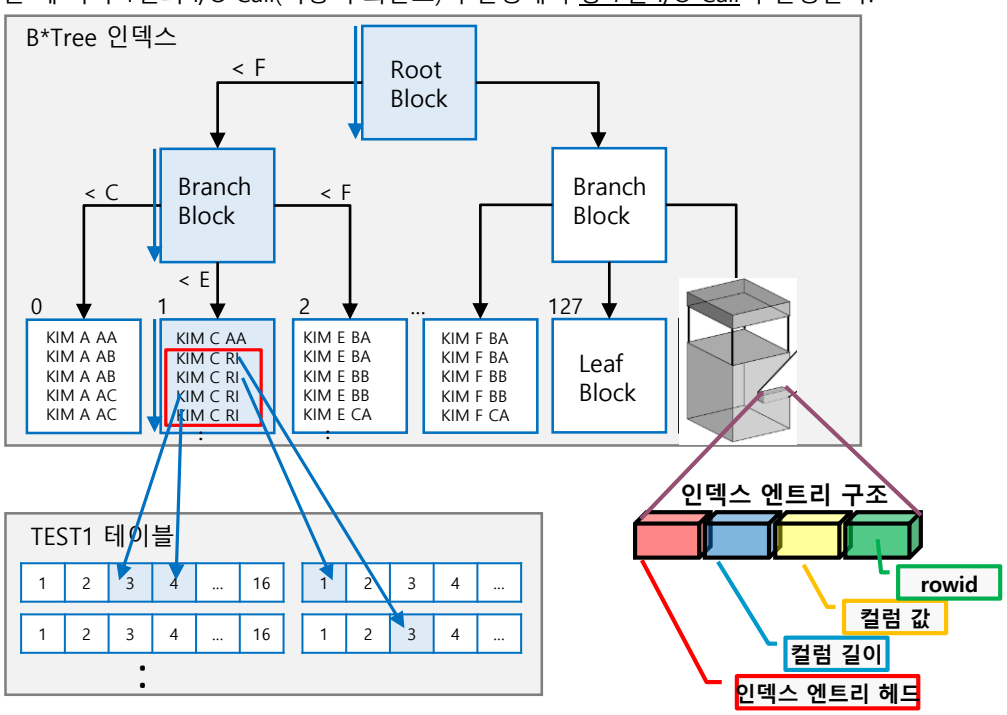
2. 인덱스

3) Index Range Scan 테스트

아래 SQL 실행 시 CUST_NM 컬럼에 인덱스가 있으면 어떻게 수행될까?

```
SQL> CREATE INDEX IX_TEST_CUSNM ON TEST1 ( CUST_NM );

Index 'IX_TEST_CUSTTY' created.

SQL> SELECT INDEX_NAME, BLEVEL, LEAF_BLOCKS
       FROM DBA_INDEXES
      WHERE OWNER ='TEST02'
        AND INDEX_NAME ='IX_TEST_CUSTNM';

     INDEX_NAME  BLEVEL LEAF_BLOCKS
---------------- ------- -----------
IX_TEST_CUSTNM        2         128

SQL> SELECT * FROM TEST1 WHERE CUST_NM = 'KIM C RI';

CUST_ID     CUST_NM   CUST_TY  TELNO          ADDRESS
----------  --------  -------  -------------  -------
0000000367  KIM C RI  A        010-1234-0366  C City
0000006217  KIM C RI  A        010-1234-6216  C City
0000009701  KIM C RI  C        010-1234-9700  C City
0000067707  KIM C RI  A        010-1234-7706  C City

-------------------------------------------------------------------------------------------
|ID |  Operation            | Name           | Rows  | Elaps. Time    | CR Gets | Starts |
-------------------------------------------------------------------------------------------
| 1 | TABLE ACCESS (ROWID)  | TEST1          |     4 |00:00:00.0000 |       4 |      1 |
| 2 |  INDEX (RANGE SCAN)   | IX_TEST_CUSTNM |     4 |00:00:00.0002 |       3 |      1 |
-------------------------------------------------------------------------------------------
   2 - access: ("T"."CUST_NM" = 'KIM C RI') (0.000)
```

실행계획을 보면 IX_TEST_CUSTNM 인덱스를 Index Range Scan 으로 3 블럭을 읽는다. 인덱스에서 찾은 4건의 Rowid로 테이블 블록을 4개 읽어 총 7 블록을 읽어 4건 조회한다.
이때 Root 블록 → Branch 블록 → Leaf 블록 → 테이블 블록 순서로 읽는다. 1번개의 블록을 읽을 때 마나 1번의 I/O Call(파랑색 화살표)이 발생해서 총 7번 I/O Call이 발생한다.



2. 인덱스

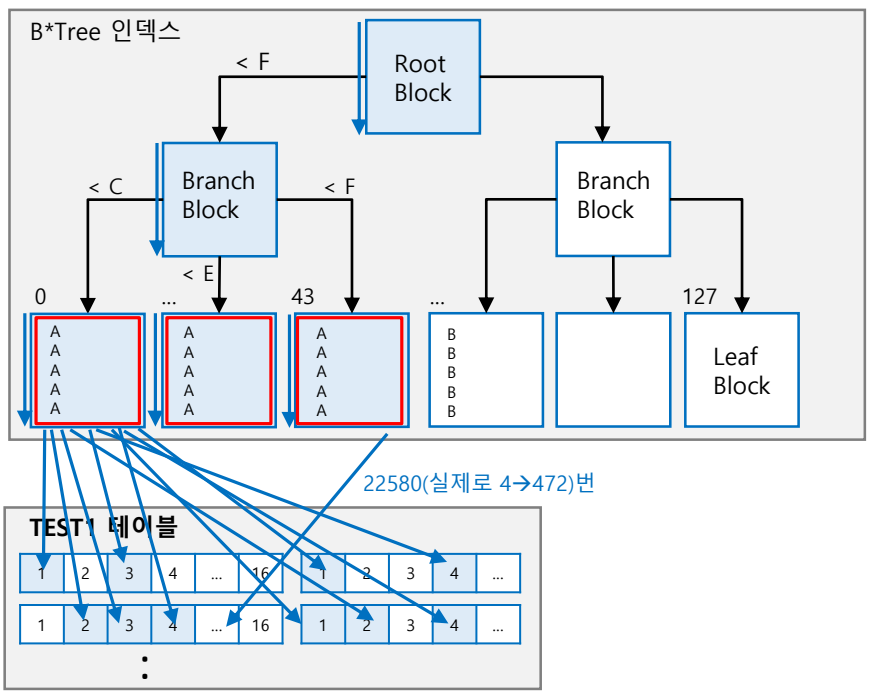
4) 그럼 언제 인덱스를 사용해야 하나?

테스트 SQL을 Table Full Scan 과 Index Range Scan을 읽은 블록 개수와 I/O Call 횟수 관점에서 비교하면 Index Range Scan의 성능이 더 좋다.

구분	읽은 블록 개수	I/O Call 횟수
Full Table Scan	497 블록	20 번
Index Range Scan	7 블록	7 번

그러나 아래 SQL의 경우 인덱스에서 CUST_TY가 A를 만족하는 건수가 많기 때문에 (225804건) 인덱스의 Leaf 블록을 읽는 양이 증가한다.(3블록→45블록).
더구나 인덱스 Leaf 블록에서 찾은 건수(22580건) 만큼 테이블 블록을 읽어야 하기 때문에 테이블 블록에 대해 I/O Call이 22580(실제로 4→472)번 발생하고, 테이블 블록을 22580 (실제로 4→472) 블록 읽는다. 즉 인덱스로 찾은 건수가 많으면 테이블 블록을 읽는 양과 I/O Call이 증가하여 Table Full Scan보다 성능이 나빠진다.

```
SQL> CREATE INDEX IX_TEST_CUSTTY ON TEST1 ( CUST_TY );
Index 'IX_TEST_CUSTTY' created.

SQL> SELECT * FROM TEST1 WHERE CUST_TY = 'A';

CUST_ID     CUST_NM   CUST_TY  TELNO          ADDRESS
----------  --------  -------  -------------  -------
 :
0000069998 KIM F IW          A  010-1234-9997   F City
0000070000 KIM H HS          A  010-1234-9999   H City

22580 rows selected.

-------------------------------------------------------------------------------------------
|ID |  Operation             | Name           | Rows  | Elaps. Time   | CR Gets | Starts |
-------------------------------------------------------------------------------------------
| 1 | TABLE ACCESS (ROWID)   | TEST1          | 22580 |00:00:00.0065 |     472 |      1 |
| 2 |  INDEX (RANGE SCAN)    | IX_TEST_CUSTTY | 22580 |00:00:00.0089 |      45 |      1 |
-------------------------------------------------------------------------------------------
   2 - access: ("T"."CUST_TY" = 'A') (0.324)
```



2. 인덱스

2. Buffer Pinning 과 Index Clustering Factor

말 그대로 메모리(Buffer Cache)에서 읽은 Buffer(블록)를 보관(Pinning)하므로써 다시 Buffer를 Buffer Cache에서 읽지 않는 것이다. Buffer Pinning에는 Index Buffer Pinning과 Table Buffer Pinning이 있다.

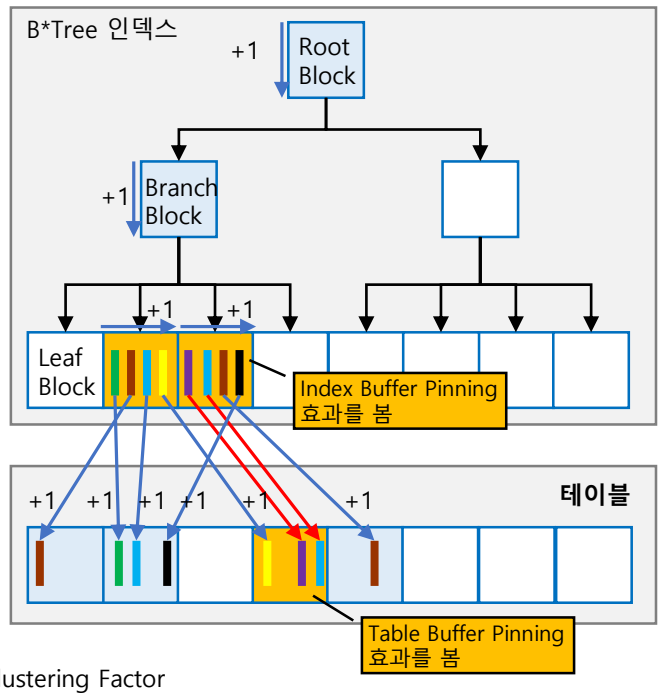
1) Index Buffer Pinning

인덱스를 Index Range Scan으로 읽을 때 최근에 읽은 인덱스 Leaf 블록 1개를 Pinning 한다. 그래서 인덱스에서 조건을 만족하는 첫 번째 Row의 Rowid로 테이블 블록을 읽고, 다시 인덱스 Leaf 블록에서 조건을 만족하는 두 번째 Row를 동일한 Leaf 블록에서 읽으면, 공유 메모리(Buffer Cache)에서 Leaf 블록을 읽지 않고 Pinning된 Leaf 블록을 사용하여 성능이 향상된다. 이 경우 Leaf 블록에 대한 CR Gets는 증가하지 않는다.

데이터베이스 종류와 버전에 따라 인덱스의 Root 블록과 Branch 블록에 대한 Buffer Pinning 이 있다.

2) Table Buffer Pinning

Index Range Scan으로 읽은 Row의 Rowid로 테이블 블록을 읽을 때 최근에 읽은 테이블 블록 1개를 Pinning 한다. 그래서 인덱스에서 조건을 만족하는 첫 번째 Row의 Rowid로 테이블 블록을 읽고(이때 테이블 블록이 Pinning 됨), 다시 인덱스에서 조건을 만족하는 두 번째 Row가 동일한 테이블 블록을 읽을 경우, 공유 메모리(Buffer Cache)에서 테이블 블록을 읽지 않고 Pinning된 테이블 블록을 사용하여 성능이 향상된다. 이 경우 테이블 블록에 대한 CR Gets는 증가하지 않는다.



3) Clustering Factor

인덱스 전체를 Index Range Scan으로 읽어서 Leaf 블록의 Rowid로 테이블 블록을 읽을 때, 최근에 읽은 테이블 블록과 다른 테이블 블록을 읽을 확률(0~1)을 나타낸다. 따라서 같은 테이블 블록을 읽을 경우 테이블 Buffer Pinning 기능을 사용하여 성능이 좋아진다. 즉 값이 1에 가까울 수록 다른 다른 테이블 블록을 읽어 성능이 저하되고 0에 가까울 수록 좋다. 오라클의 Clustering Factor는 테이블 블록을 총 읽은 개수이다.(티베로의 Clustering Factor * 전제 Row 개수)

```
SELECT INDEX_NAME,
       CLUSTERING_FACTOR,                              -- 티베로의 Clustering Factor
       ROUND(CLUSTERING_FACTOR * NUM_ROWS)  ORACLE_CF  -- 오라클의 Clustering Factor
  FROM USER_INDEXES
 WHERE TABLE_NAME ='T_EMP_TX4';
```

2. 인덱스

3. 결합 인덱스의 컬럼 구성

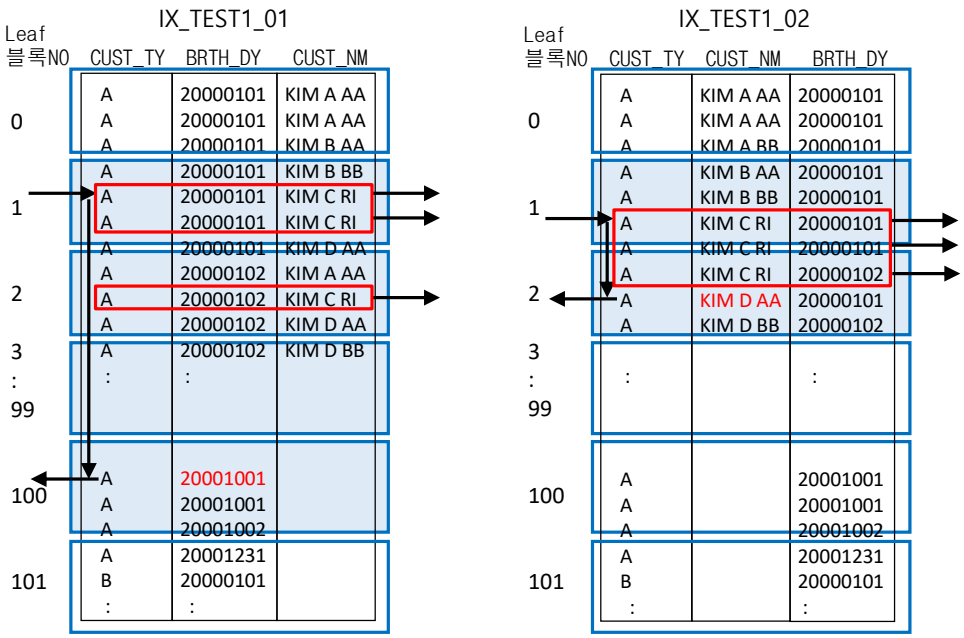
"범위 조건이 처음 나올 때 까지 사용된 모든 조건" 의 인덱스 Leaf 블록을 읽는다. 따라서 = 조건 컬럼을 인덱스 앞쪽에, 범위 조건 컬럼을 인덱스 뒤쪽에 위치하면 인덱스 Leaf 블록을 작게 읽는다.

예를 들어 01 인덱스가 CUST_TY + BRTH_DY+ CUST_NM 컬럼으로 구성되고, SQL이 CUST_TY = , BRTH_DY Between, CUST_NM = 조건을 사용하면, 범위(Between) 조건이 처음으로 나올 때 까지 사용된 CUST_TY = , BRTH_DY Between 조건을 모두 만족하는 인덱스 Leaf 블록을 읽는다.

이 경우 02 인덱스를 CUST_TY + CUST_NM + BRTH_DY 컬럼으로 구성하면, CUST_TY =, CUST_NM =, BRTH_DY Between 조건을 모두 만족하는 더 작은 인덱스 Leaf 블록을 읽는다.

아래 SQL에서 01 인덱스는 CUST_NM 조건이 틀려도 BIRTH_DY 값이 달라지면 다시 CUST_NM 조건이 맞을 수 있기 때문에 BRTH_DY 조건이 틀릴 때 까지 (20001001) 인덱스 Leaf 블록을 읽는다. CUST_NM 조건은 인덱스 Leaf 블록을 읽는 양에는 관여하지 못하고 오직 조건 확인하는 용도(필터)로만 사용된다. 그러나 02 인덱스는 CUST_TY, CUST_NM, BRTH_DY 조건을 모두 만족하는 인덱스 Leaf 블록을 읽는다. 하나라도 조건이 틀리면 더 이상 인덱스 Leaf 블록을 읽을 필요가 없다.

```
CREATE INDEX IX_TEST1_01  ON TEST1 ( CUST_TY, BRTH_DY, CUST_NM );
CREATE INDEX IX_TEST1_02  ON TEST1 ( CUST_TY, CUST_NM, BRTH_DY );

SELECT /*+ INDEX(T 인덱스명) */ *
  FROM TEST1   T
 WHERE CUST_TY ='A'
   AND BRTH_DY BETWEEN '20000101' AND '20100930'
   AND CUST_NM ='KIM C RI';

-------------------------------------------------------------------------------------------
| ID | Operation                   | Name         | Rows    | Elaps. Time     | CR Gets | Starts |
-------------------------------------------------------------------------------------------
|  1 | TABLE ACCESS (ROWID)        | TEST1        |       3 |00:00:00.0001 |       3 |      1 |
|  2 |  FILTER                     |              |       3 |00:00:00.0006 |       0 |      1 |
|  3 |   INDEX (RANGE SCAN)        | IX_TEST1_01  |   22580 |00:00:00.0005 |     102 |      1 |
-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------
| ID | Operation                   | Name         | Rows    | Elaps. Time     | CR Gets | Starts |
-------------------------------------------------------------------------------------------
|  1 | TABLE ACCESS (ROWID)        | TEST1        |       3 |00:00:00.0000 |       3 |      1 |
|  2 |  INDEX (RANGE SCAN)         | IX_TEST1_02  |       3 |00:00:00.0000 |       4 |      1 |
-------------------------------------------------------------------------------------------
```



이론

3.조인

3. 조인

일반적으로 많이 사용하는 Nested Loop 조인(NL 조인)과 Hash 조인에 대해 설명한다. 두 개의 조인은 서로의 단점을 보완하는 역할을 한다.

1. Nested Loop 조인

1) 처리 방법

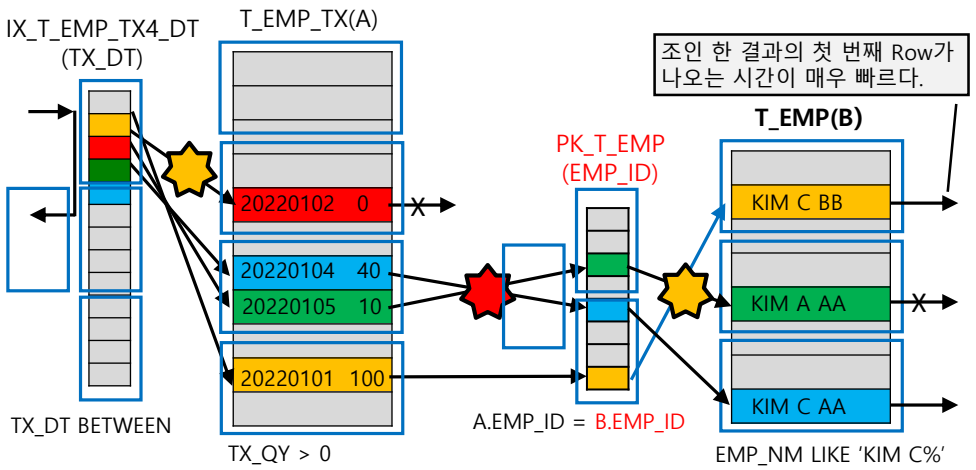
① 두 개의 테이블 중 하나를 먼저 액세스하여(선행 테이블, Driving Table) 해당 테이블 조건에 맞는 데이터를 찾아서,
② 다른 테이블(후행 테이블, Drived Table)의 조인 컬럼에 존재하는 인덱스를 이용하여 조인한다.

2) 특징

조인 결과의 첫 번째 Row가 나오는 시간이 매우 빠르다. OLTP 시스템에 최적이다.
순차적으로 처리하므로 부분범위 처리에 많이 사용된다.(조인하다가 멈출 수 있다)
후행 테이블의 조인 컬럼에 인덱스가 반드시 필요하다. 인덱스가 없으면 성능 저하가 발생한다.
선행 테이블은 조건을 만족하는 건수가 작은 테이블로 선택해야, 후행 테이블과 조인 횟수가 최소화 된다.

```
SELECT /*+ LEADING(A B) USE_NL(B) */
       A.TX_NO, A.TX_DT, A.TX_QY, B.EMP_NM, B.EMP_ID
  FROM T_EMP_TX4   A,
       T_EMP       B
 WHERE A.TX_DT BETWEEN TO_DATE('20220101','YYYYMMDD') AND TO_DATE('20220105','YYYYMMDD')
   AND A.TX_QY > 0
   AND A.EMP_ID = B.EMP_ID
   AND B.EMP_NM LIKE 'KIM C%';

-------------------------------------------------------------------------------------------------
|ID| Operation                    | Name            | Rows    | Elaps. Time   | CR Gets | Starts |
-------------------------------------------------------------------------------------------------
| 1| INDEX JOIN                   |                 |    1926 |00:00:00.0009 |       0  |      1 |
| 2|  TABLE ACCESS (ROWID)        | T_EMP_TX4       |   50000 |00:00:00.0130 |    1862  |      1 |
| 3|   INDEX (RANGE SCAN)         | IX_T_EMP_TX4_DT |   50000 |00:00:00.0004 |     146  |      1 |
| 4|  TABLE ACCESS (ROWID)        | T_EMP           |    1926 |00:00:00.0295 |   50000  |  50000 |
| 5|   INDEX (UNIQUE SCAN)        | PK_T_EMP        |   50000 |00:00:00.0548 |    100K  |  50000 |
-------------------------------------------------------------------------------------------------

   2 - filter: ("A"."TX_QY" > 0) (1.000)
   3 - access: ("A"."TX_DT" >= TO_DATE('2022-01-01 00:00:00','YYYY-MM-DD HH24:MI:SS'))
AND ("A"."TX_DT" <= TO_DATE('2022-01-05 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.014)
   4 - filter: ("B"."EMP_NM" LIKE 'KIM C%') (0.031)
   5 - access: ("B"."EMP_ID" = "A"."EMP_ID") (0.000)
```

NL 조인 시 **T_EMP(B)** 에서 읽은 블록 개수
= T_EMP 인덱스 블록 개수 + T_EMP 테이블 블록 개수
= (50000번 조인 * 2개 블록) + (5000번 액세스 * 1개 블록) = 100000 + 50000 = 150000 블록



3. 조인

2. Hash 조인

앞서 NL 조인 예제에서, 선행 테이블에서 많은 건(50000건)이 후행 테이블(T_EMP)로 조인하여, 후행 쪽에서 읽는 블록 양이 많아 성능 저하가 발생했다. 이러한 문제를 해결하는 것이 Hash 조인이다.

1) 처리 방법

① 두 개의 테이블 중 하나를 먼저 (인덱스 또는 Full Table Scan으로) 액세스하여(Build Input) 해당 테이블의 조건에 맞는 데이터를 찾아 Hash Table을 생성한다.
② 다른 테이블을 (인덱스 또는 Full Table Scan으로) 액세스하여 (Probe Input) 조건에 맞는 데이터를 찾아 한 건씩 생성 된 Hash Table을 탐침(Probe)하여 조인하므로 인덱스가 필요 없고 Hash 함수를 사용한 CPU 연산을 한다.

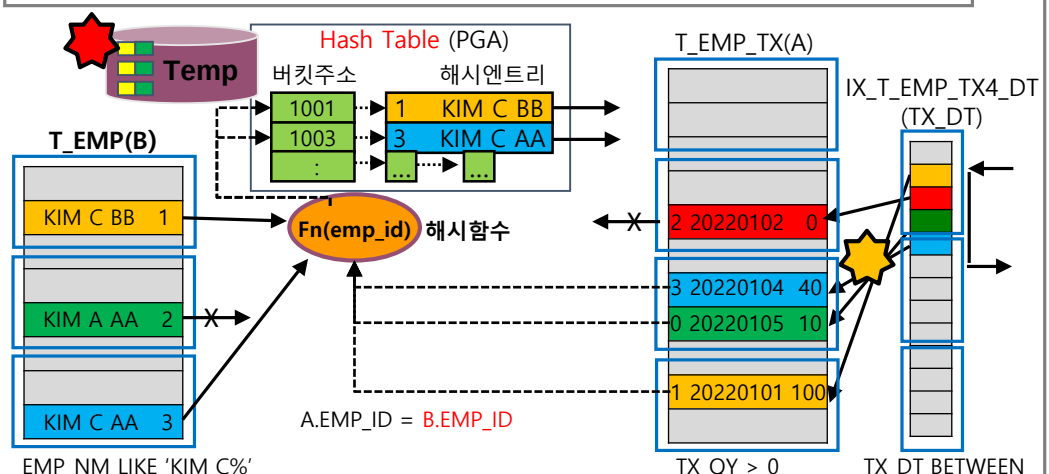
2) 특징

Hash Table은 한정된 메모리에 생성되므로 메모리가 부족하면 Temp를 사용하여 성능 저하가 발생한다. 따라서 조인이 아닌 조건을 만족하는 데이터가 작은 테이블로 Hash Table을 만들어야 한다. 먼저 액세스 되는 테이블이 Hash Table로 만들어지므로 힌트로 제어할 수 있다.
Hash 조인 결과의 첫 번째 Row가 나오는 시간이, NL 조인 보다 느리다.(Hash Table이 생성되고 나서 Hash 조인이 시작된다) = 조인 연산자 만 Hash 조인을 할 수 있다.

NL 조인에서 후행 테이블(T_EMP)의 전체 크기는 433 블록으로 작았지만, 선행 테이블(T_EMP_TX4)에서 많은 건(50000건)이 후행 테이블과 NL 조인하여, 1회 조인 시 3 블록을 읽어 50000번 조인 시 150000 블록을 읽어 성능 저하가 발생했다.
이 경우, 후행 테이블을 한 번만 읽어(Table Full Scan 또는 Index Scan) 조인이 아닌 조건으로 Hash Table을 만들어 Hash 조인을 하면 블록을 읽는 양을 크게 줄일 수 있다.

```
SELECT /*+ LEADING(B A) USE_HASH(A) */  A.TX_NO, A.TX_DT, A.TX_QY, B.EMP_NM, B.EMP_ID
  FROM T_EMP_TX4 A,   T_EMP B
 WHERE A.TX_DT BETWEEN TO_DATE('20220101','YYYYMMDD') AND TO_DATE('20220105','YYYYMMDD')
   AND A.TX_QY > 0
   AND A.EMP_ID = B.EMP_ID
   AND B.EMP_NM LIKE 'KIM C%' ;

-------------------------------------------------------------------------------------------
|ID| Operation            | Na                | Rows   | Elaps. Time  | CR Gets | Starts |
-------------------------------------------------------------------------------------------
| 1| HASH JOIN            |                   |   1926 |00:00:00.0042 |       0 |      1 |
| 2|  TABLE ACCESS (FULL) | T_EMP             |    781 |00:00:00.0028 |     433 |      1 |
| 3|  TABLE ACCESS (ROWID)| T_EMP_TX4         |  50000 |00:00:00.0137 |    1862 |      1 |
| 4|   INDEX (RANGE SCAN) | IX_T_EMP_TX4_DT   |  50000 |00:00:00.0006 |     146 |      1 |
-------------------------------------------------------------------------------------------
   1 - access: ("B"."EMP_ID" = "A"."EMP_ID") (0.000)
   2 - filter: ("B"."EMP_NM" LIKE 'KIM C%') (0.031)
   3 - filter: ("A"."TX_QY" > 0) (1.000)
   4 - access: ("A"."TX_DT" >= TO_DATE('2022-01-01 00:00:00','YYYY-MM-DD HH24:MI:SS'))
AND ("A"."TX_DT" <= TO_DATE('2022-01-05 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000*0.014)
```

Hash Table



이론

4. 힌트

4. 힌트

데이터베이스의 옵티마이저는 SQL이 사용하는 오브젝트(테이블, 인덱스 등)와 시스템의 통계정보를 이용하여 가장 적은 Cost를 가지는 실행계획을 생성하여 SQL을 수행한다. 따라서 Cost 계산에 이용되는 통계정보가 정확하지 않거나, Cost 산정 방식이 정확하지 않으면 계산된 Cost 값이 부정확하여 나쁜 실행계획을 생성하여 SQL을 수행한다.

이 경우, SQL을 튜닝하는 가장 일반적인 방법은 힌트를 사용하여 실행계획을 변경하는 것이다. 서로 다른 데이터베이스가 힌트를 동일 이름으로 제공한다고 기능도 같은 것이 아니다.

티베로에서 제공하는 힌트는 다음과 같다.

분류	힌트	설명
옵티마이저 모드	ALL_ROWS	전체 결과를 빨리 처리하도록 한다.
	FIRST_ROWS	결과의 첫 번째 Row를 빨리 처리하도록 한다.
테이블 액세스 순서	LEADING(t1 t2)	t1 → t2 순서로 액세스 한다.
	ORDERED	From 절에 표기된 테이블 순서대로 액세스 한다.
조인 방법	USE_NL(t2)	t2 와 Nested Loop 조인(이하 NL 조인)을 한다. 이때 인덱스를 사용하여 NL조인을 하면 실행계획에는 Index Join으로 표시되고, 그렇지 않은 경우 실행계획에는 Nested Loop Join으로 표시된다.
	NO_USE_NL(t2)	t2 와 NL 조인을 하지 않는다.
	USE_NL_WITH_INDEX(t2 idx2)	t2 와 NL 조인을 할 때 idx2 인덱스를 사용한다. 실행계획에는 Index Join 으로 표시된다.
	USE_HASH(t2)	t2 와 Hash 조인을 한다.
	NO_USE_HASH(t2)	t2 와 Hash 조인을 하지 않는다.
	SWAP_JOIN_INPUTS(t2)	Hash 조인을 할 때 t2로 Hash Table을 만든다.
	NO_SWAP_JOIN_INPUTS(t2)	Hash 조인을 할 때 t2로 Hash Table을 만들지 않는다.
	USE_MERGE(t2)	t2 와 Sort Merge 조인을 한다.
	NO_USE_MERGE(t2)	t2 와 Sort Merge 조인을 하지 않는다.
	NL_SJ	NL 세미조인을 한다.
	NL_AJ	NL 안티조인을 한다.
	HASH_SJ	Hash 세미조인을 한다.
	HASH_AJ	Hash 안티조인을 한다.
	MERGE_SJ	Sort Merge 세미조인을 한다.
	MERGE_AJ	Sort Merge 안티조인을 한다.
액세스 방법	FULL(t1)	t1 테이블을 Full Scan 한다.
	INDEX(t1 idx1)	t1 테이블의 idx1 인덱스를 읽는다.
	NO_INDEX(t1 idx1)	t1 테이블의 idx1 인덱스를 읽지 않는다.
	INDEX_ASC(t idx1)	t1 테이블의 idx1 인덱스를 오름차순으로 읽는다.
	INDEX_DESC(t1 idx1)	t1 테이블의 idx1 인덱스를 내림차순으로 읽는다.
	INDEX_FFS(t1 idx1)	t1 테이블의 idx1 인덱스를 Fast Full Scan 방식으로 읽는다.
	NO_INDEX_FFS(t1 idx1)	t1 테이블의 idx1 인덱스를 Fast Full Scan 방식으로 읽지 않는다.
	INDEX_RS(t1 idx1)	t1 테이블의 idx1 인덱스를 Index Range Scan 방식으로 읽는다.

4. 힌트

분류	힌트	설명
엑세스 방법	NO_INDEX_RS(t1 idx1)	t1 테이블의 idx1 인덱스를 Index Range Scan 방식으로 읽지 않는다.
	INDEX_SS(t1 idx1)	t1 테이블의 idx1 인덱스를 Index Skip Scan 방식으로 읽는다.
	NO_INDEX_SS(t1 idx1)	t1 테이블의 idx1 인덱스를 Index Skip Scan 방식으로 읽지 않는다.
	INDEX_JOIN(t1)	명시한 테이블의 인덱스 2개를 조인하여 액세스한다.
병렬 처리	PARALLEL(n) PARALLEL(t1 n)	n개의 쓰레드를 사용하여 병렬 처리한다.
	NO_PARALLEL	병렬 처리를 하지 않는다.
	PQ_DISTRIBUTE(inner테이블 outer테이블_row분배방식 Inner테이블_row분배방식)	조인 작업에서 Row 분배 방식을 지정한다. - none none : 힌트가 없을 때와 동일. - broadcast none : outer테이블 전체를 inner쪽 병렬 프로세스 모두에게 복제 - none broadcast : inner테이블 전체를 outer쪽 병렬 프로세스 모두에게 복제 - hash hash : outer테이블과 inner테이블 모두 해시 함수로 분배
쿼리 변환	NO_MERGE	뷰 머지를 하지 않는다.
	UNNEST	서브쿼리를 Unnest 하여 서브쿼리 블록을 메인 쿼리 블록과 합쳐 하나의 쿼리 블록으로 만든다.
	NO_UNNEST	서브쿼리를 Unnest 하지 않도록 해서, 서브쿼리 블록과 메인 쿼리 블록을 하나로 합치지 않는다.
	REWRITE	Materialized View를 사용하도록 SQL을 수정한다.
	NO_REWRITE	SQL을 수정하지 않아 Materialized View를 사용하지 않는다.
	MATERIALIZE	WITH 문에 사용하여 WITH 문을 별도의 쿼리 블록으로 만든다.
	INLINE	WITH 문에 사용하여 WITH 문을 별도의 쿼리 블록으로 만들지 않고 메인 쿼리 블록에 합친다.
	USE_CONCAT	OR 를 서로 다른 쿼리 블록으로 만든다.
	NO_EXPAND	OR 를 서로 다른 쿼리 블록으로 만들지 않는다.
기타	APPEND	데이터 입력 시 메모리가 아닌 데이터 파일에 직접 쓰는 방식(Direct Path Write)을 사용한다. Insert /*+ APPEND */ into t1 select ...
	NO_APPEND	Direct Path Write 방식을 사용하지 않는다.
	APPEND_VALUES	Values 절을 사용하여 데이터 입력 시 데이터 파일에 직접 쓰는 방식을 사용한다. Insert /*+ APPEND_VALUES */ into t1 values(...)
	IGNORE_ROW_ON_DUP_KEY_INDEX	Unique 제약조건을 위배하는 데이터 입력 시 에러를 발생시키지 않는다.
	MONITOR	해당 SQL 수행 시 성능 정보를 수집해서 데이터 딕셔너리에 저장한다.
	NO_MONITOR	해당 SQL 수행 시 성능 정보를 수집하지 않는다.
	RESULT_CACHE	SQL 수행 결과를 Result Cache에 캐싱하여 동일 SQL 수행 시 Result Cache 값을 사용한다.
	CARD(t1 n)	t1 테이블의 예상 Row 개수(Cardinality)를 n개로 설정한다.
	NO_SUBQUERY_CACHE	SQL을 수행할 때 특정 서브쿼리의 수행 결과를 캐싱하지 않는다.

이론

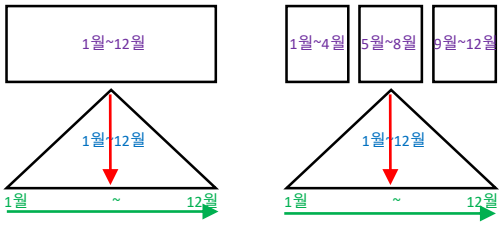
5.파티션 인덱스

5. 파티션 인덱스(Partitioned Index)

테이블 파티션
인덱스 파티션
인덱스 컬럼

테이블의 파티션 유무와 상관 없이 인덱스를 파티션 할 수 있다. 인덱스를 파티션 하는 이유는 관리 및 성능 목적이며, 파티션 기준으로 다음과 같이 분류할 수 있다.

1. Non-Partitioned Index



대용량 테이블 인 경우, Non-Partitioned Index는 인덱스의 Leaf 블록이 많기 때문에 인덱스 Depth가 깊어져서 인덱스 액세스(수직 탐색) 시 성능 저하가 발생한다.

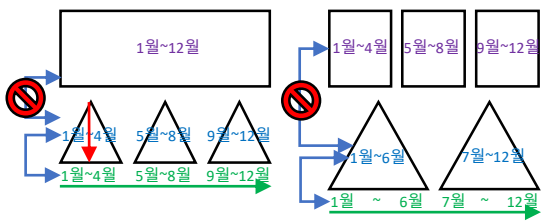
2. Partitioned Index

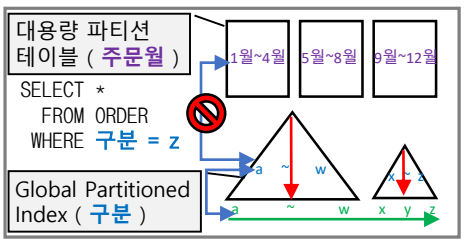
인덱스를 파티션 함으로써 인덱스 Depth가 얕아지고, 대용량 테이블 인 경우 개별 인덱스 파티션의 Depth가 깊어져도 인덱스 파티션 별로 Index Rebuild 같은 인덱스 관리(Global은 제한적)를 독립적으로 할 수 있다. 모든 파티션 인덱스는 인덱스 파티션 키 컬럼이 검색 조건으로 사용되어야 특정 인덱스 파티션 만 액세스할 수 있다. 그러치 않으면 모든 인덱스 파티션을 액세스해야 하므로 (여러 번의 수직 탐색이 발생하여) 성능 저하가 발생한다.
Global Index 와 Local Index의 가장 큰 차이점은 인덱스의 Leaf 블럭에서의 정렬이다. Global Index는 테이블 전체에 대해 인덱스 키 컬럼의 정렬을 보장하고, Local Index는 테이블 전체에 대해 인덱스 키 컬럼의 정렬은 보장하지 않고 각 각의 인덱스 파티션 안에서만 인덱스 키 컬럼의 정렬을 보장한다.

1) Global Partitioned Index

테이블과 인덱스의 파티션(키 컬럼/범위)이 다르고, 인덱스 파티션 키 컬럼이 인덱스 컬럼의 Prefix에 위치하는 Prefixed Index 만 가능하다. 따라서 Global Partitioned Index는 인덱스 키 컬럼 값이 전체 테이블에 대해 정렬되어 있다. 인덱스 키 컬럼이 인덱스 파티션 키 컬럼과 같기 때문에 검색 조건에 인덱스 파티션 키(인덱스 선두 컬럼) 컬럼이 사용되지 않으면 인덱스를 사용할 수 없다. 사용시 특정 인덱스 파티션 만 액세스 한다.

아래 SQL처럼 대용량 파티션 테이블에서 파티션 키 컬럼(주문월)이 검색 조건으로 사용되지 않고, 구분 컬럼을 검색 조건으로 빠른 응답이 필요한 경우 Global Partitioned Index를 고려해야 한다. 특히 Critical 한 상수 Range(x ~ z)에 대해 작은 인덱스 파티션을 생성하여 인덱스 크기를 줄여 인덱스 액세스 시간을 줄일 수 있다.



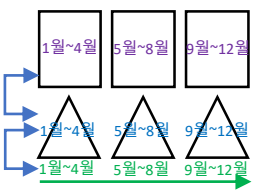


2) Local Partitioned Index

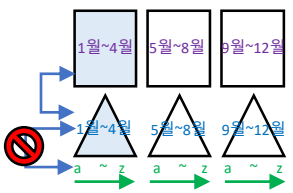
Local Index는 인덱스 파티션 키 컬럼이 인덱스 키 컬럼의 Prefix에 위치하지 않아도 된다.
즉 Prefixed Index와 Non-Prefixed Index 모두 존재한다. 따라서 Prefixed Index 만 인덱스 키 컬럼 값이 전체 테이블에 대해 정렬되어 있다.
검색 조건에 인덱스 파티션 키 컬럼이 사용되지 않으면 모든 인덱스 파티션을 액세스 한다.
사용시 특정 인덱스 파티션 만 액세스 한다.

아래 SQL에서 ORDER 테이블(주문월 컬럼 RANGE 파티션), IX_ORDER_MON (구분) Local Partitioned Index인 경우, 파티션 키 컬럼(주문월)이 검색 조건에
사용되었기 때문에 특정 인덱스 파티션(1~4월 인덱스 파티션)을 액세스하여 구분 a를 찾는다.

Local Partitioned Prefixed Index



Local Partitioned Non-Prefixed Index




```
SELECT *
FROM ORDER
WHERE 주문월 = 3월
  AND 구분 = a;
```

실기

1.조인

1. 조인

문제1	테이블 조인 시, 한쪽 테이블의 조건이 아주 좋은 경우
분석	주문→고객으로 NL 조인 시, 고객에 액세스하는 횟수가 많아 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	먼저 주문을 CUST_ID로 Group By 해서 건수를 줄인 후, 주문→고객으로 NL 조인한다. (참고) Sort Group By 와 Hash Group By
튜닝 2단계	변별력이 좋은 조건을 가지는 고객을 먼저 액세스해서, 고객→주문 순서로 NL 조인한다.
문제2	테이블 조인 시, 양쪽 모두 넓은 범위 조건인 경우
분석	고객→주문으로 NL 조인 시, 주문에 액세스하는 횟수가 많아 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	NL 조인을 Hash 조인으로 변경한다.
튜닝 2단계	Hash 조인 시, LEADING 힌트로 Hash Table을 지정한다.
튜닝 3단계	고객→주문 순서로 Hash 조인 시, 주문을 Group By 해서 건수를 줄여 Hash Table을 탐색하는 횟수를 줄인다.
문제3	세 번째 액세스하는 테이블과 Hash 조인할 경우
분석	주문상태표 테이블은 작지만, NL 조인 시, 주문상태표에 액세스하는 횟수가 많아 주문상태표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많다
튜닝 1단계	NL 조인을 Hash 조인으로 변경한다.
튜닝 2단계	세 번째 액세스되는 테이블로 Hash Table을 만들어 Hash 조인을 할 경우, SWAP_JOIN_INPUTS 힌트를 사용한다.

실기

1.조인 - 문제1

테이블 조인 시, 한쪽 테이블의 조건이 아주 좋은 경우

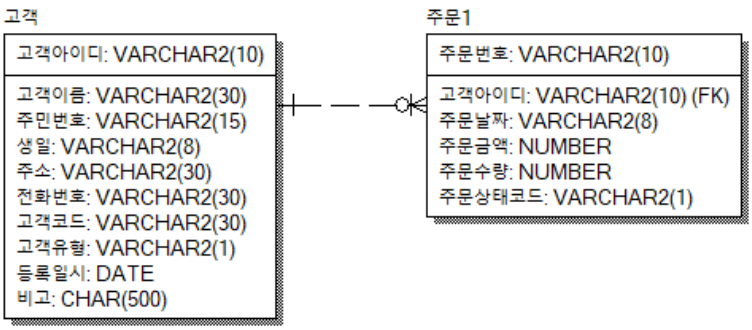
1. 조인

문제1	테이블 조인 시, 한쪽 테이블의 조건이 아주 좋은 경우

문제설명

- A 온라인 쇼핑몰의 관리 담당자는 z0005 고객코드를 가지는 고객이 2023/9/1 ~ 2023/10/30 동안 주문한 주문 정보를 고객 별로 총주문횟수, 총주문금액, 총주문수량, 최종주문날짜 를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CUST (CUST_TY);
- 주문 테이블은 월 7만 건, 22개월(2022/1 ~ 2023/10) 동안 총 150만 건 주문정보를 저장한다. 주문날짜는 문자(8자리) Type이다. 주문은 모두 주문상태코드(1~5)를 가지며, 대부분 5 값을 가진다.

SQL

```
SELECT  /*+ LEADING(O C) USE_NL(C)  */
        O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
        COUNT(*)              "총주문횟수",
        SUM(O.ORDER_AMT)      "총주문금액",
        SUM(O.ORDER_QY)       "총주문량",
        MAX(O.ORDER_DY)       "최종주문일"
  FROM T_ORDER1  O,
        T_CUST    C
 WHERE ORDER_DY BETWEEN '20230901' AND '20231030'   -- 2개월, 14만 건
   AND O.CUST_ID = C.CUST_ID
   AND C.CUST_CD = 'z0005'          -- 0.05%, 350건
 Group By O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO;
```

1. 조인

문제1	테이블 조인 시, 한쪽 테이블의 조건이 아주 좋은 경우

TRACE

NL조인 시 후행 테이블의 인덱스가 사용되면, 실행계획에 Index Join 으로 표기 됨.

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name       | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | Group By (SORT)              |            |      350 |00:00:00.0005 |        0 |        1 |
|   2 |  INDEX JOIN                  |            |      700 |00:00:00.0021 |        0 |        1 |
|   3 |   TABLE ACCESS (FULL)        | T_ORDER1   |     140K |00:00:00.0679 |     9381 |        1 |
|   4 |  TABLE ACCESS (ROWID)        | T_CUST     |      700 |00:00:00.0393 |     140K |     140K | ③
|   5 |   INDEX (UNIQUE SCAN)        | PK_T_CUST  |     140K |00:00:00.1505 |     280K |     140K | ②
-------------------------------------------------------------------------------------------------
   3 - filter: ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY" <= '20231030') (0.094 * 1.000)
   4 - filter: ("C"."CUST_CD" = 'z0005') (0.006)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
```

분석

① 주문 테이블을 Full Scan 해서 2개월에 해당하는 14만 건을 찾아, 주문→고객으로 NL 조인 (Nested Loop Join)한다.

② 고객의 PK인덱스를 14만 번 액세스 한 후, 14만 건을 추출한다. 이때 인덱스 블록을 랜덤 액세스 하는 양(28만 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 14만 건이 테이블 블록을 액세스해서 z0005 고객코드를 만족하는 700건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(14만 블록)이 많아 성능 저하가 발생하다.

④ 테이블에서 추출한 14만 건을 Group By 해서 350건을 추출한다.

1. 조인

문제1 – 튜닝 1단계	먼저 주문을 Group By 해서 건수 줄인 후, 주문→고객으로 NL 조인한다.

튜닝내역

1. 주문→고객으로 NL 조인 횟수가 많아, 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 따라서 고객과 NL 조인하기 전에 주문을 CUST_ID로 Group By 해서 주문을 고객 수준으로 건수를 줄인 후, 주문→고객으로 NL 조인한다.

   ① O2 인라인뷰를 사용하여 주문을 CUST_ID로 Group By 한다. 이때 NO_MERGE 힌트를 사용하여, O2 인라인뷰가 메인과 머지되는 것을 방지하여 O2 인라인뷰에서 Group By가 먼저 되도록 한다.

   ② LEADING 힌트와 USE_NL 힌트로 O2 인라인뷰→고객으로 NL 조인(Nested Loop 조인)을 한다.

SQL

```
SELECT /*+ LEADING(O2 C)  USE_NL(C)  */
       O2.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       CNT              "총주문횟수",
       SUM_ORDER_AMT    "총주문금액",
       SUM_ORDER_QY     "총주문수량",
       MAX_ORDER_DY     "최종주문날짜"
  FROM (
        SELECT /*+ NO_MERGE */
               O.CUST_ID,
               COUNT(*)                CNT,
               SUM(O.ORDER_AMT)        SUM_ORDER_AMT,
               SUM(O.ORDER_QY)         SUM_ORDER_QY,
               MAX(O.ORDER_DY)         MAX_ORDER_DY
          FROM T_ORDER1 O
         WHERE O.ORDER_DY BETWEEN '20230901' AND '20231030'   -- 2개월, 14만 건
         Group By O.CUST_ID
       ) O2, -- 7만 건
       T_CUST C
 WHERE O2.CUST_ID = C.CUST_ID
   AND C.CUST_CD = 'z0005'; -- 0.05%, 350 건
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                | Name      | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | INDEX JOIN               |           |      350 |00:00:00.0017 |        0 |        1 |
|   2 |  Group By (SORT)         |           |    70000 |00:00:00.1026 |        0 |        1 |
|   3 |   TABLE ACCESS (FULL)    | T_ORDER1  |     140K |00:00:00.0678 |     9381 |        1 |
|   4 | TABLE ACCESS (ROWID)     | T_CUST    |      350 |00:00:00.0452 |    70000 |    70000 |
|   5 |  INDEX (UNIQUE SCAN)     | PK_T_CUST |    70000 |00:00:00.0790 |     140K |    70000 |
-------------------------------------------------------------------------------------------------
   3 - filter: ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY" <= '20231030') (0.094 * 1.000)
   4 - filter: ("C"."CUST_CD" = 'z0005') (0.006)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
```

1. 조인

문제1 – 튜닝 1단계	먼저 주문을 Group By 해서 건수 줄인 후, 주문→고객으로 NL 조인한다.

참고

- 만약, O2 인라인뷰 안에서 주문을 CUST_ID로 Group By 할 때, 옵티마이저가 Sort가 아닌 Hash로 Group By 하면 Group By 된 중간 집합이 CUSR_ID로 정렬되지 않아, 주문→고객으로 NL 조인 시, PK_T_CUST 인덱스 블록에 대한 Buffer Pinning 효과가 떨어져 인덱스 블록을 많이 읽게 된다. 이 때는 NO_USE_HASH_GROUPBY 힌트를 사용해서 Sort로 Group By 하면 CUST_ID로 정렬된다.
  OPT_PARAM(_ENABLE_SORT_GROUPBY, Y/N), OPT_PARAM(_ENABLE_HASH_GROUPBY, Y/N)

- 오라클은 NO_USE_HASH_GROUPBY 대신 NO_USE_HASH_AGGREGATION 힌트를 사용한다.

```
SELECT /*+ LEADING(O2 C)  USE_NL(C)  */
       O2.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       CNT             "총주문횟수",
       SUM_ORDER_AMT   "총주문금액",
       SUM_ORDER_QY    "총주문수량",
       MAX_ORDER_DY    "최종주문날짜"
  FROM ( SELECT /*+ NO_MERGE  NO_USE_HASH_AGGREGATION */
                CUST_ID,
                COUNT(*)              CNT,
               SUM(O.ORDER_AMT)      SUM_ORDER_AMT,
               SUM(O.ORDER_QY)       SUM_ORDER_QY,
               MAX(O.ORDER_DY)       MAX_ORDER_DY
          FROM T_ORDER1 O
         WHERE O.ORDER_DY BETWEEN '20230901' AND '20231030'   -- 2개월, 14만 건
         Group By CUST_ID
       ) O2, -- 7만 건
       T_CUST C
 WHERE O2.CUST_ID = C.CUST_ID
   AND C.CUST_CD =  'Z0005'; -- 0.05%, 350건
```

- 오라클 옵티마이저가 Hash Group By 한 경우

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |    350 |00:00:00.24 |     109K|
|   1 |  NESTED LOOPS                |           |      1 |    350 |00:00:00.24 |     109K|
|   2 |   NESTED LOOPS               |           |      1 |  70000 |00:00:00.18 |   39795 |
|   3 |    VIEW                      |           |      1 |  70000 |00:00:00.12 |    9365 |
|   4 |     HASH Group By            |           |      1 |  70000 |00:00:00.11 |    9365 |
|*  5 |      TABLE ACCESS FULL       | T_ORDER1  |      1 |   140K|00:00:00.05 |    9365 |
|*  6 |    INDEX UNIQUE SCAN         | PK_T_CUST |  70000 |  70000 |00:00:00.04 |   30430 |
|*  7 |   TABLE ACCESS BY INDEX ROWID| T_CUST    |  70000 |    350 |00:00:00.04 |   70000 |
-----------------------------------------------------------------------------------------------------
```

- 오라클에서 NO_USE_HASH_AGGREGATION 힌트를 사용하여 Sort Group By 한 경우

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |    350 |00:00:00.22 |   80668 |
|   1 |  NESTED LOOPS                |           |      1 |    350 |00:00:00.22 |   80668 |
|   2 |   NESTED LOOPS               |           |      1 |  70000 |00:00:00.21 |   10668 |
|   3 |    VIEW                      |           |      1 |  70000 |00:00:00.16 |    9365 |
|   4 |     SORT Group By            |           |      1 |  70000 |00:00:00.15 |    9365 |
|*  5 |      TABLE ACCESS FULL       | T_ORDER1  |      1 |   140K|00:00:00.06 |    9365 |
|*  6 |    INDEX UNIQUE SCAN         | PK_T_CUST |  70000 |  70000 |00:00:00.03 |    1303 |
|*  7 |   TABLE ACCESS BY INDEX ROWID| T_CUST    |  70000 |    350 |00:00:00.03 |   70000 |
-----------------------------------------------------------------------------------------------------
```

1. 조인

문제1 – 튜닝 2단계	변별력이 좋은 조건을 가지는 고객을 먼저 액세스해서, 고객→주문으로 NL 조인한다.

튜닝내역

1. 주문을 CUST_ID로 Group By 해서 주문을 고객 수준으로 건수를 줄인 후 주문→고객으로 NL 조인을 하여도, 고객에 액세스하는 횟수가 7만 건으로 다소 많아 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 따라서 변별력이 좋은 조건(CUST_CD = z0005)을 가지는 고객을 먼저 액세스해서, 고객→주문으로 NL 조인한다.

   ① 고객에서 z0005 고객코드를 찾기 위해 고객코드 컬럼에 신규 인덱스를 생성한다.

   ② 고객→주문으로 NL 조인 시, 사용할 신규 인덱스를 생성한다.

   ③ INDEX, LEADING 과 USE_NL 힌트로 특정 인덱스를 사용하여 주문→고객으로 NL 조인한다.

SQL

```
CREATE INDEX IX_T_CUST_CD ON T_CUST ( CUST_CD );                    -- ① 액세스
CREATE INDEX IX_T_ORDER1_ID_DY  ON T_ORDER1 ( CUST_ID, ORDER_DY );  -- ② 조인


SELECT /*+ LEADING(C O)  USE_NL(O)  INDEX(C IX_T_CUST_CD)  INDEX(O IX_T_ORDER1_ID_DY) */
       O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       COUNT(*)              "총주문횟수",
       SUM(O.ORDER_AMT)      "총주문금액",
       SUM(O.ORDER_QY)       "총주문량",
       MAX(O.ORDER_DY)       "최종주문일"
  FROM T_ORDER1  O,
       T_CUST    C
 WHERE O.ORDER_DY BETWEEN '20230901' AND '20231030'   -- 2개월, 14만 건
   AND O.CUST_ID = C.CUST_ID
   AND C.CUST_CD =  'z0005'          -- 0.05%, 350 건
 Group By O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO;


DROP INDEX IX_T_CUST_CD;
DROP INDEX IX_T_ORDER1_ID_DY;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name              | Rows     | Elaps. Time  | CR Gets  | Starts    |
-------------------------------------------------------------------------------------------------------------
|   1 | Group By (SORT)              |                   |      350 |00:00:00.0006 |        0 |         1 |
|   2 |  INDEX JOIN                  |                   |      700 |00:00:00.0005 |        0 |         1 |
|   3 |   TABLE ACCESS (ROWID)       | T_CUST            |      350 |00:00:00.0006 |      350 |         1 |
|   4 |    INDEX (RANGE SCAN)        | IX_T_CUST_CD      |      350 |00:00:00.0000 |        2 |         1 |
|   5 |   TABLE ACCESS (ROWID)       | T_ORDER1          |      700 |00:00:00.0031 |      700 |       350 |
|   6 |    INDEX (RANGE SCAN)        | IX_T_ORDER1_ID_DY |      700 |00:00:00.0868 |      379 |       350 |
-------------------------------------------------------------------------------------------------------------
   4 - access: ("C"."CUST_CD" = 'z0005') (0.004)
   6 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY"
<= '20231030') (0.000 * 0.090 * 1.000)
```

실기

1.조인 – 문제2

테이블 조인 시, 양쪽 모두 넓은 범위 조건인 경우

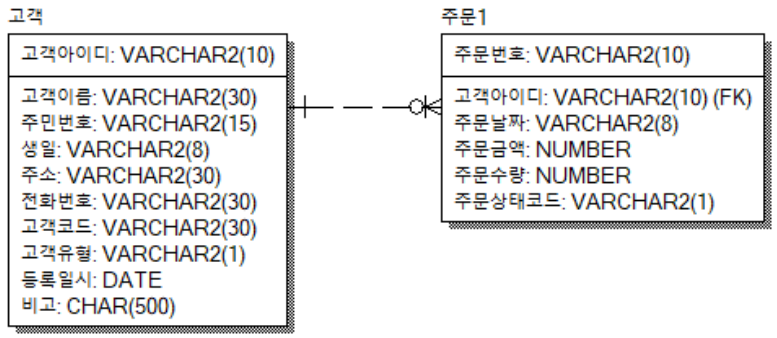
1. 조인

문제2	테이블 조인 시, 양쪽 모두 넓은 범위 조건인 경우

문제설명

- A 온라인 쇼핑몰의 관리 담당자는 z0006 고객코드를 가지는 고객이 2023/1/1 ~ 2023/12/31 동안 주문한 주문 정보를 고객 별로 총주문횟수, 총주문금액, 총주문수량, 최종주문날짜 를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호의 뒷자리는 암호화되어 있고, 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을 , 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CUST (CUST_TY);
- 주문 테이블은 월 7만 건, 22개월(2022/1 ~ 2023/10) 동안 총 150만 건 주문정보를 저장한다. 주문은 모두 주문상태코드(1 ~ 5)를 가지며, 대부분 5 값을 가진다.

1. 조인

문제2	테이블 조인 시, 양쪽 모두 넓은 범위 조건인 경우

SQL

```
CREATE INDEX IX_T_CUST_CD ON T_CUST ( CUST_CD );
CREATE INDEX IX_T_ORDER1_ID_DY  ON T_ORDER1 ( CUST_ID, ORDER_DY );


SELECT /*+ LEADING(C O)  USE_NL(O)  INDEX(C IX_T_CUST_CD)  INDEX(O IX_T_ORDER1_ID_DY) */
       O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       COUNT(*)              "총주문횟수",
       SUM(O.ORDER_AMT)      "총주문금액",
       SUM(O.ORDER_QY)       "총주문량",
       MAX(O.ORDER_DY)       "최종주문일"
  FROM T_ORDER1  O,
       T_CUST    C
 WHERE ORDER_DY BETWEEN '20230101' AND '20231231'  -- 12개월, 7만*12개월=84만 건
   AND O.CUST_ID = C.CUST_ID
   AND C.CUST_CD = 'z0006'        -- 99.5%, 69650 건
 Group By O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO;


DROP INDEX IX_T_CUST_CD;
DROP INDEX IX_T_ORDER1_ID_DY;
```

TRACE

```
-----------------------------------------------------------------------------------------------------
| ID | Operation                | Name             | Rows    | Elaps. Time  | CR Gets  | Starts   |
-----------------------------------------------------------------------------------------------------
|  1 | Group By (HASH)          |                  |  69650  |00:00:00.1600 |       0  |       1  |
|  2 |  INDEX JOIN              |                  |    696K |00:00:00.0964 |       0  |       1  |
|  3 |   TABLE ACCESS (ROWID)   | T_CUST           |  69650  |00:00:00.0206 |    5841  |       1  |
|  4 |    INDEX (RANGE SCAN)    | IX_T_CUST_CD     |  69650  |00:00:00.0302 |     171  |       1  |
|  5 |   TABLE ACCESS (ROWID)   | T_ORDER1         |    696K |00:00:01.6424 |    696K  |   69650  | ③
|  6 |    INDEX (RANGE SCAN)    | IX_T_ORDER1_ID_DY|    696K |00:00:01.4041 |   75450  |   69650  | ②
-----------------------------------------------------------------------------------------------------
   4 - access: ("C"."CUST_CD" = 'z0006') (0.996)
   6 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230101') AND ("O"."ORDER_DY"
<= '20231231') (0.000 * 0.451 * 1.000)
```

분석

① 고객의 인덱스로 CUST_CD=z0006 에 해당하는 69650 건을 찾아, 고객→주문으로 NL 조인한다.

② 주문의 인덱스(CUST_ID + ORDER_DY)를 69650번 액세스 한 후, 69만 건을 추출한다. 이때 인덱스 블록을 랜덤 액세스하는 양(7만 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 69만 건이 테이블 블록을 랜덤 액세스해서 69만 건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(69만 블록)이 많아 성능 저하가 발생한다.

④ 테이블에서 추출한 69만 건을 Group By 해서 69650건을 추출한다.

1. 조인

문제2, 튜닝 1단계	NL 조인을 Hash 조인으로 변경한다.

튜닝내역

1. 고객→주문으로 NL 조인 시, 주문으로 액세스 횟수가 많아, 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. z0006 고객코드가 전체 고객의 99.5%에 해당하고, 주문날짜가 전체 주문의 55%에 해당하여 어느 쪽을 먼저 액세스해서도 NL 조인 시, 블록을 랜덤 액세스 양이 많다. 따라서 NL 조인을 Hash 조인으로 변경한다.

   ① FULL, LEADING 과 USE_HASH 힌트로 주문 테이블을 Full Scan 해서 z0006 고객코드를 만족하는 70만 건을 가지고 Hash Table을 만든다.(Build 과정, 먼저 액세스 된 테이블을 가지고 hash Table을 만든다)

   ② Z0006 고객코드 범위가 넓으므로 FULL 힌트로 고객 테이블을 Full Scan 해서 고객코드를 만족하는 69650건을 가지고, 69650번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe 과정)

   ③ Hash 조인 후, Group By 한다.

SQL

```
SELECT /*+ LEADING(O C) USE_HASH(C)  FULL(O)  FULL(C) */
       O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       COUNT(*)              "총주문횟수",
       SUM(O.ORDER_AMT)      "총주문금액",
       SUM(O.ORDER_QY)       "총주문량",
       MAX(O.ORDER_DY)       "최종주문일"
  FROM T_ORDER1  O,
       T_CUST    C
 WHERE ORDER_DY BETWEEN '20230101' AND '20231231'    -- 12개월, 7만*12개월=84만 건
   AND O.CUST_ID = C.CUST_ID
   AND C.CUST_CD =  'z0006'          -- 99.5%, 69650건
 Group By O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO;
```

TRACE

```
-------------------------------------------------------------------------------------------
| ID  | Operation              | Name     | Rows     | Elaps. Time  | CR Gets  | Starts  |
-------------------------------------------------------------------------------------------
|   1 | Group By (HASH)        |          |   69650  |00:00:00.0000 |       0  |      1  |
|   2 |  HASH JOIN             |          |     696K |00:00:00.0000 |       0  |      1  |
|   3 |   TABLE ACCESS (FULL)  | T_ORDER1 |     700K |00:00:00.0426 |    9381  |      1  |①Hash Table
|   4 |   TABLE ACCESS (FULL)  | T_CUST   |   69650  |00:00:00.0141 |    5858  |      1  |②
-------------------------------------------------------------------------------------------
   2 - access: ("O"."CUST_ID" = "C"."CUST_ID") (0.000)
   3 - filter: ("O"."ORDER_DY" >= '202300101') AND ("O"."ORDER_DY" <= '20231231') (0.451 * 1.000)
   4 - filter: ("C"."CUST_CD" = 'Z0005') (0.000)
```

1. 조인

문제2, 튜닝 2단계	Hash 조인 시, LEADING 힌트로 Hash Table을 지정한다.

튜닝내역

1. Hash Table은 메모리에 생성되며, 메모리가 부족할 경우 디스크(Temporary Tablespace)를 같이 사용하므로 성능 저하가 발생한다. 따라서 작은 중간 집합(테이블에서 검색 조건을 만족하는 데이터)을 가지고 Hash Table을 만드는 것이 좋다. 주문→고객 순서로 액세스하면, 주문의 주문날짜를 만족하는 70만 건을 가지고 Hash Table을 만드나, 고객→주문 순서로 액세스하면 고객에서 고객코드를 만족하는 69650건을 가지고 Hash Table을 만든다.

   ① LEADING 과 USE_HASH 힌트로 고객 테이블을 Full Scan(z0006 고객코드 범위가 넓으므로) 해서 z0006 고객코드를 만족하는 69650건을 가지고 Hash Table을 만든다.(Build 과정)

   ② FULL 힌트로 주문 테이블을 Full Scan(주문날짜 범위가 넓으므로) 해서 주문날짜를 만족하는 70만 건을 가지고, 70만 번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe 과정)

   ③ Hash 조인 후, Group By 한다.

SQL

```
SELECT /*+ LEADING(C O)  USE_HASH(O)  FULL(O)  FULL(C) */
       O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       COUNT(*)              "총주문횟수",
       SUM(O.ORDER_AMT)      "총주문금액",
       SUM(O.ORDER_QY)       "총주문량",
       MAX(O.ORDER_DY)       "최종주문일"
  FROM T_ORDER1  O,
       T_CUST    C
 WHERE ORDER_DY BETWEEN '20230101' AND '20231231'    -- 12개월, 7만*12개월=84만 건
   AND O.CUST_ID = C.CUST_ID
   AND C.CUST_CD =  'z0006'          -- 99.5%, 69650 건
 Group By O.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO;
```

TRACE

```
-------------------------------------------------------------------------------------------
| ID | Operation              | Name     | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------
|  1 | Group By (HASH)        |          |   69650  |00:00:00.2154 |       0  |       1  |
|  2 |  HASH JOIN             |          |     696K |00:00:00.2617 |       0  |       1  |
|  3 |   TABLE ACCESS (FULL)  | T_CUST   |   69650  |00:00:00.0285 |    5858  |       1  |①Hash Table
|  4 |   TABLE ACCESS (FULL)  | T_ORDER1 |     700K |00:00:00.1503 |    9381  |       1  |②
-------------------------------------------------------------------------------------------
Predicate Information
-------------------------------------------------------------------------------------------------------
   2 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   3 - filter: ("C"."CUST_CD" = 'z0006') (0.996)
   4 - filter: ("O"."ORDER_DY" >= '20230101') AND ("O"."ORDER_DY" <= '20231231') (0.451 * 1.000)
```

1. 조인

문제2, 튜닝 3단계	고객→주문 순서로 Hash 조인 시, 주문을 Group By 해서 건수를 줄여 Hash Table을 탐색하는 횟수를 줄인다.

튜닝내역

1. 고객→주문 순서로 액세스하면, 먼저 액세스 된 고객으로 Hash Table을 만들고(Build 과정), 주문을 읽어 Hash Table에서 조인 조건을 만족하는 것을 찾는다(Probe 과정). 만약, 주문을 고객으로 Group By 해서 주문을 고객 수준으로 건수를 줄이면, Hash Table을 탐색하는 횟수가 줄어들어(70만 번→7만 번) 성능 향상을 기대할 수 있다. 이 경우, 블록 액세스 양은 변하지 않고, CPU 사용양이 줄어든다.

   ① LEADING 과 USE_HASH 힌트로 고객→주문 순서로 Hash 조인한다. 고객 테이블을 Full Scan(z0006 고객코드 범위가 넓으므로) 해서 z0006 고객코드를 만족하는 69650건을 가지고 Hash Table을 만든다.(Build 과정)

   ② FULL 힌트로 주문 테이블을 Full Scan(주문날짜 범위가 넓으므로) 해서 주문날짜를 만족하는 70만 건을 CUST_ID로 Group By 해서 만든 7만 건을 가지고, 7만 번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe 과정)

SQL

```
SELECT /*+ LEADING(C O2)  USE_HASH(O2)  FULL(C)  */
       O2.CUST_ID, C.CUST_NM, C.ADDRESS, C.TELNO,
       CNT              "총주문횟수",
       SUM_ORDER_AMT    "총주문금액",
       SUM_ORDER_QY     "총주문수량",
       MAX_ORDER_DY     "최종주문날짜"
  FROM (
       SELECT /*+ NO_MERGE  FULL(O) */
              CUST_ID,
              COUNT(*)              CNT,
                 SUM(O.ORDER_AMT)      SUM_ORDER_AMT,
                 SUM(O.ORDER_QY)       SUM_ORDER_QY,
                 MAX(O.ORDER_DY)       MAX_ORDER_DY
         FROM T_ORDER1 O
        WHERE ORDER_DY BETWEEN '20230101' AND '20231231'    -- 12개월, 7만*12개월=84만 건
        Group By CUST_ID
       ) O2, -- 7만 건
       T_CUST C
 WHERE O2.CUST_ID = C.CUST_ID
   AND C.CUST_CD =  'z0006'; -- 99.5%, 69650 건
```

TRACE

```
-------------------------------------------------------------------------------------
|ID | Operation              | Name     | Rows    | Elaps. Time  | CR Gets  |Starts  |
-------------------------------------------------------------------------------------
| 1 | HASH JOIN              |          |  69650  |00:00:00.0509 |       0  |     1  |
| 2 |  TABLE ACCESS (FULL)   | T_CUST   |  69650  |00:00:00.0167 |    5858  |     1  |①Hash Table
| 3 |  Group By (HASH)       |          |  70000  |00:00:00.1671 |       0  |     1  |②
| 4 |   TABLE ACCESS (FULL)  | T_ORDER1 |    700K |00:00:00.1118 |    9381  |     1  |
-------------------------------------------------------------------------------------

   1 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   2 - filter: ("C"."CUST_CD" = 'z0006') (0.996)
   4 - filter: ("O"."ORDER_DY" >= '20230101') AND ("O"."ORDER_DY" <= '20231231') (0.451 * 1.000)
```

실기

1.조인 – 문제3

세 번째 액세스하는 테이블과 Hash 조인 할 경우

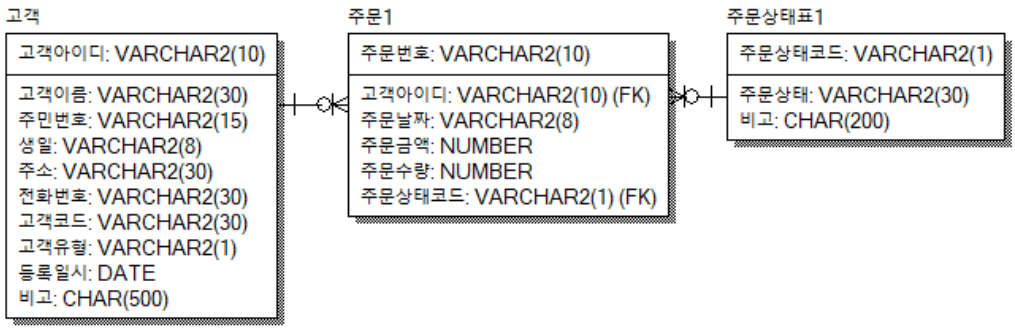
1. 조인

문제3	세 번째 액세스하는 테이블과 Hash 조인 할 경우

문제설명

- A 온라인 쇼핑몰의 관리 담당자는 z0005 고객코드를 가지는 고객이 2023/9/1 ~ 2023/10/30 동안 주문하고 구매확정 한 주문 정보를 고객 별로 총주문횟수, 총주문금액, 총주문수량, 최종주문날짜를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호의 뒷자리는 암호화되어 있고, 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을 , 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CUST (CUST_TY);
- 주문 테이블은 월 7만 건, 22개월(2022/1 ~ 2023/10) 동안 총 150만 건 주문정보를 저장한다. 주문은 모두 주문상태코드(1 ~ 5)를 가지며, 대부분 5 값을 가진다.
- 주문상태표 테이블은 총 5건의 주문상태 정보를 저장한다. 주문상태는 다음과 같다.
  1: 결재완료
  2: 상품준비중
  3: 배송중
  4: 배송완료
  5: 구매확정

1. 조인

문제3	세 번째 액세스하는 테이블과 Hash 조인 할 경우

SQL

```
CREATE INDEX IX_T_CUST_CD       ON T_CUST ( CUST_CD );
CREATE INDEX IX_T_ORDER1_ID_DY  ON T_ORDER1 ( CUST_ID, ORDER_DY );

SELECT  /*+ LEADING(C O S) USE_NL(O) USE_NL(S)
            INDEX(C IX_T_CUST_CD) INDEX(O IX_T_ORDER1_ID_DY) INDEX(S PK_T_ORDER_STA1) */
       O.ORDER_DY, C.CUST_ID, C.CUST_NM,  O.ORDER_AMT, O.ORDER_QY
  FROM T_CUST        C,
       T_ORDER1      O,
       T_ORDER_STA1  S
 WHERE C.CUST_CD = 'z0005'    -- 0.05%, 350건
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DY BETWEEN '20230901' AND '20231030' -- 2개월, 14만 건
   AND O.ORDER_STA_CD = S.ORDER_STA_CD
   AND S.ORDER_STA = '구매확정'      -- 대부분 구매확정
   AND S.CM IS NOT NULL
 ORDER BY O.ORDER_DY, C.CUST_ID ;

DROP INDEX IX_T_CUST_CD;
DROP INDEX IX_T_ORDER1_ID_DY;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name                | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------
| 1 | ORDER BY (SORT)              |                     |      700 |00:00:00.0003 |        0 |        1 |
| 2 |  INDEX JOIN                  |                     |      700 |00:00:00.0000 |        0 |        1 |
| 3 |   INDEX JOIN                 |                     |      700 |00:00:00.0017 |        0 |        1
| 4 |    TABLE ACCESS (ROWID)      | T_CUST              |      350 |00:00:00.0006 |      350 |      350
| 5 |     INDEX (RANGE SCAN)       | IX_T_CUST_CD        |      350 |00:00:00.0000 |        2 |        1
| 6 |    TABLE ACCESS (ROWID)      | T_ORDER1            |      700 |00:00:00.0043 |      700 |      350
| 7 |     INDEX (RANGE SCAN)       | IX_T_ORDER1_ID_DY   |      700 |00:00:00.0650 |      379 |      350
| 8 |   TABLE ACCESS (ROWID)       | T_ORDER_STA1        |      700 |00:00:00.0002 |      700 |      700   ④
| 9 |    INDEX (UNIQUE SCAN)       | PK_T_ORDER_STA1     |      700 |00:00:00.0003 |      700 |      700   ③
-------------------------------------------------------------------------------------------------------
```

중간집합

```
   5 - access: ("C"."CUST_CD" = 'z0005') (0.004)
   7 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY"
<= '20231030') (0.000 * 0.090 * 1.000)
   8 - filter: ("S"."ORDER_STA" = '구매확정') AND ("S"."CM" IS NOT NULL) (0.200 * 1.000)
   9 - access: ("S"."ORDER_STA_CD" = "O"."ORDER_STA_CD") (1.000)
```

분석

① 고객의 인덱스로 CUST_ID=z0005 에 해당하는 350건을 찾아, 고객→주문으로 NL 조인한다.

② 주문의 인덱스(CUST_ID + ORDER_DY)를 350번 액세스 한 후, 테이블에서 700건을 추출한다.

③ ②에서 조인 한 중간 집합(700건)이 주문상태표로 NL조인한다. 주문상태표의 인덱스 (ORDER_STAT_CD)를 700번 액세스 한 후, 700건을 추출한다.

④ 인덱스로 추출한 700건이 테이블 블록을 랜덤 액세스해서 700건을 추출한다. 이때 ③+④ 에서 블록을 랜덤 액세스하는 양이 주문상태표 테이블의 총 블록 개수 보다 많다.

1. 조인

문제3 – 튜닝 1단계	NL 조인을 Hash 조인으로 변경한다.

튜닝내역

1. 고객→주문으로 NL 조인 한 중간 집합(700건)이, 주문상태표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이, 주문상태표 테이블 크기 보다 많아 성능 저하가 발생한다. 따라서 주문상태표 테이블을 Full Scan 해서 Hash 조인한다.

   ① INDEX, LEADING 과 USE_NL 힌트로 지정한 인덱스를 사용해서 고객→주문으로 NL 조인된 중간 집합(700건)으로 Hash Table을 만든다.(Build 과정)

   ② FULL, LEADING 과 USE_HASH 힌트로 주문상태표 테이블을 Full Scan 해서 구매확정과 CM Is Not NULL을 만족하는 1건을 가지고, 1번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe과정)

SQL

```
CREATE INDEX IX_T_CUST_CD       ON T_CUST ( CUST_CD );
CREATE INDEX IX_T_ORDER1_ID_DY  ON T_ORDER1 ( CUST_ID, ORDER_DY );

SELECT  /*+  LEADING(C O S) USE_NL(O) USE_HASH(S)
             INDEX(C IX_T_CUST_CD) INDEX(O IX_T_ORDER1_ID_DY)  FULL(S)  */
       O.ORDER_DY, C.CUST_ID, C.CUST_NM,  O.ORDER_AMT, O.ORDER_QY
  FROM T_CUST        C,
       T_ORDER1      O,
       T_ORDER_STA1  S
 WHERE C.CUST_CD = 'z0005'    -- 0.05%, 350건
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DY BETWEEN '20230901' AND '20231030' -- 2개월, 14만 건
   AND O.ORDER_STA_CD = S.ORDER_STA_CD
   AND S.ORDER_STA = '구매확정'      -- 대부분 구매확정
   AND S.CM IS NOT NULL
 ORDER BY O.ORDER_DY, C.CUST_ID ;

DROP INDEX IX_T_CUST_CD;
DROP INDEX IX_T_ORDER1_ID_DY;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID  | Operation                | Name             | Rows    | Elaps. Time  | CR Gets | Starts   |
-------------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)          |                  |     700 |00:00:00.0003 |       0 |        1 |
|  2 |  HASH JOIN               |                  |     700 |00:00:00.0002 |       0 |        1 |
|  3 |   INDEX JOIN             |                  |     700 |00:00:00.0007 |       0 |        1 |  ①
|  4 |    TABLE ACCESS (ROWID)  | T_CUST           |     350 |00:00:00.0014 |     350 |        1 |
|  5 |     INDEX (RANGE SCAN)   | IX_T_CUST_CD     |     350 |00:00:00.0000 |       2 |        1 |
|  6 |    TABLE ACCESS (ROWID)  | T_ORDER1         |     700 |00:00:00.0038 |     700 |      350 |
|  7 |     INDEX (RANGE SCAN)   | IX_T_ORDER1_ID_DY|     700 |00:00:00.0039 |     379 |      350 |
|  8 |   TABLE ACCESS (FULL)    | T_ORDER_STA1     |       1 |00:00:00.0085 |      14 |        1 |  ②
-------------------------------------------------------------------------------------------------------
   2 - access: ("O"."ORDER_STA_CD" = "S"."ORDER_STA_CD") (1.000)
   5 - access: ("C"."CUST_CD" = 'z0005') (0.004)
   7 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY"
<= '20231030') (0.000 * 0.090 * 1.000)
   8 - filter: ("S"."ORDER_STA" = '구매확정') AND ("S"."CM" IS NOT NULL) (0.200 * 1.000)
```

Hash Table

1. 조인

문제3 – 튜닝 2단계	세 번째 액세스하는 테이블로 Hash Table을 만들어서 Hash 조인을 할 경우, SWAP_JOIN_INPUTS 힌트를 추가로 사용한다.

튜닝내역

1. LEADING 힌트로 고객→주문→주문상태표 순서로 액세스하기 때문에, 고객→주문으로 NL 조인 된 중간 집합(700건)이 주문상태표와 Hash 조인한다. 구매확정은 1건으로, 기존 중간 집합(700건)보다 작기 때문에, SWAP_JOIN_INPUTS 힌트를 사용하여 주문상태표를 가지고 Hash Table을 만든다.(사실 700건 이나 1건 모두 메모리에 Hash Table을 만들기 때문에 성능 차이가 없다).

   ① INDEX, LEADING 과 USE_NL 힌트로 지정한 인덱스를 사용하여 고객→주문으로 NL 조인 된 중간 집합(700건)으로 Hash Table을 만드는 것이 원칙이지만,

   ② SWAP_JOIN_INPUTS 힌트로 주문상태표(1건)로 Hash Table을 만들기 때문에(Build 과정), 700번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe 과정)

SQL

```
CREATE INDEX IX_T_CUST_CD       ON T_CUST ( CUST_CD );
CREATE INDEX IX_T_ORDER1_ID_DY  ON T_ORDER1 ( CUST_ID, ORDER_DY );

SELECT  /*+ LEADING(C O S) USE_NL(O) USE_HASH(S)
            INDEX(C IX_T_CUST_CD) INDEX(O IX_T_ORDER1_ID_DY) FULL(S) SWAP_JOIN_INPUTS(S)  */
       O.ORDER_DY, C.CUST_ID, C.CUST_NM,  O.ORDER_AMT, O.ORDER_QY
  FROM T_CUST        C,
       T_ORDER1      O,
       T_ORDER_STA1  S
 WHERE C.CUST_CD = 'z0005'   -- 0.05%, 350건
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DY BETWEEN '20230901' AND '20231030' -- 2개월, 14만 건
   AND O.ORDER_STA_CD = S.ORDER_STA_CD
   AND S.ORDER_STA = '구매확정'      -- 대부분 구매확정
   AND S.CM IS NOT NULL
 ORDER BY O.ORDER_DY, C.CUST_ID ;

DROP INDEX IX_T_CUST_CD;
DROP INDEX IX_T_ORDER1_ID_DY;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID  | Operation                | Name             | Rows    | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)          |                  |     700 |00:00:00.0008 |        0 |        1 |
|  2 |  HASH JOIN               | Hash Table       |     700 |00:00:00.0002 |        0 |        1 |
|  3 |   TABLE ACCESS (FULL)    | T_ORDER_STA1     |       1 |00:00:00.0001 |       14 |        1 |  ②
|  4 |   INDEX JOIN             |                  |     700 |00:00:00.0010 |        0 |        1 |  ①
|  5 |    TABLE ACCESS (ROWID)  | T_CUST           |     350 |00:00:00.0024 |      350 |        1 |
|  6 |     INDEX (RANGE SCAN)   | IX_T_CUST_CD     |     350 |00:00:00.0000 |        2 |        1 |
|  7 |    TABLE ACCESS (ROWID)  | T_ORDER1         |     700 |00:00:00.0045 |      700 |      350 |
|  8 |     INDEX (RANGE SCAN)   | IX_T_ORDER1_ID_DY|     700 |00:00:00.0045 |      379 |      350 |
-------------------------------------------------------------------------------------------------------

   2 - access("S"."ORDER_STA_CD" = "O"."ORDER_STA_CD") (1.000)
   3 - filter("S"."ORDER_STA" = '구매확정') AND ("S"."CM" IS NOT NULL) (0.200 * 1.000)
   6 - access("C"."CUST_CD" = 'z0005') (0.004)
   8 - access("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DY" >= '20230901') AND ("O"."ORDER_DY"
<= '20231030') (0.000 * 0.090 * 1.000)
```

1. 조인

정리

1. 테이블 조인 시, 한쪽 테이블의 조건이 아주 좋은 경우, 조건이 아주 좋은 테이블을 먼저 액세스 해서 다른 테이블과 NL 조인한다. 이때 먼저 액세스하는 좋은 조건을 가진 컬럼에 인덱스를 생성하고, 다른 쪽 테이블의 조인되는 컬럼에도 인덱스를 생성해야 한다.
2. 테이블 조인 시, 양쪽 테이블 모두 넓은 범위 조건인 경우, NL 조인 시 어느 쪽을 먼저 읽어도, 조인되는 다른 테이블에 액세스하는 횟수가 많아 인덱스와 테이블 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생하기 때문에 Hash 조인을 사용한다. Hash 조인을 할 때, 양쪽 테이블은 Full Scan 또는 Index Scan 모두 허용된다.
3. Hash 조인 시, 먼저 액세스 되는 테이블(또는 중간 집합)로 Hash Table을 만든다. Hash Table은 메모리에 생성되므로 상대적으로 작은 테이블(또는 중간 집합)로 Hash Table을 만드는 것이 좋다.
4. /* LEADING(A B C) USE_NL(B) USE_HASH(C) */ 힌트를 사용하여, A→B→C 순서로 액세스해서, A→B로 NL 조인한 중간 집합과 C가 Hash 조인을 할 때, C로 Hash Table을 만들어서 Hash 조인을 할 경우, SWAP_JOIN_INPUTS(C) 힌트를 추가로 사용한다.
   /* LEADING(A B C) USE_NL(B) USE_HASH(C) SWAP_JOIN_INPUTS(C) */
5. Prefetch 기능은 디스크에 대한 물리적인 I/O 발생 횟수를 줄여 성능을 개선한다. Prefetch 종류에는 인덱스 Prefetch, 테이블 Prefetch 그리고 Batch I/O가 있다.
6. Buffer Pinning 기능은 Buffer Cache(메모리)에 대한 논리적인 I/O 발생 횟수를 줄여 성능을 개선한다. Buffer Pinning 종류에는 인덱스 Buffer Pinning 과 테이블 Buffer Pinning이 있다.

실기

2.IN 서브쿼리

2. IN 서브쿼리

문제1	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장될 때
분석	계약 테이블을 Full Scan 한다. 계약→대리점→계약상태표로 NL 조인 시, 대리점과 계약상태표를 액세스하는 횟수가 많아, 대리점과 계약상태표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 PK가 있어, 메인과 서브쿼리가 조인(여기서는 NL 조인)으로 풀린다.
튜닝 1단계	계약일시 컬럼의 가공을 제거하고, 신규 인덱스를 생성한다.
튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다.
튜닝 3단계	변별력이 좋은 조건을 가지는 대리점을 먼저 액세스하여, 대리점→계약으로 NL 조인한다.
튜닝 4단계	NL 조인을 Hash 조인으로 변경하고, SWAP_JOIN_INPUTS 힌트를 사용한다.
문제2	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때
분석	계약 테이블을 Full Scan 한다. 계약→대리점→계약상태표로 NL 조인 시, 대리점과 계약상태표를 액세스하는 횟수가 많아, 대리점과 계약상태표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 Unique를 보장하는 것(PK 등)이 없어, 메인과 서브쿼리가 세미조인(여기서는 NL 세미조인)으로 풀린다.
튜닝 1단계	계약일시 컬럼의 가공을 제거하고, 신규 인덱스를 생성한다. NL 조인이 NL 세미조인으로 변경된다.
튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다. NL 조인이 NL 세미조인으로 변경된다
튜닝 3단계	변별력이 좋은 조건을 가지는 대리점을 먼저 액세스하여, 대리점→계약으로 NL 조인한다. NL 조인이 NL 세미조인으로 변경된다.
튜닝 4단계	NL 조인을 Hash 조인으로 변경하고, SWAP_JOIN_INPUTS 힌트를 사용한다. Hash 조인이 선 Unique, 후 Hash 조인으로 변경된다.

실기

2.IN 서브쿼리 – 문제1

IN 서브쿼리 안의 조인 컬럼에 Unique가 보장될 때

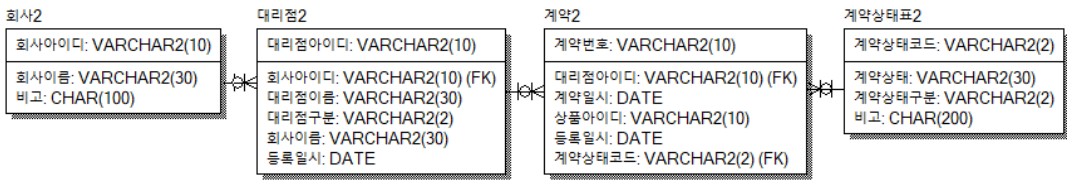
2. IN 서브쿼리

문제1	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장될 때

문제설명

- A 회사의 계약 관리 담당자는 0000000300 회사 아이디를 가지는 대리점이 2023/6/1 ~ 2023/7/31 동안 계약한 계약 정보(회사 아이디, 회사 이름, 대리점 아이디, 상품 아이디, 계약건수, 최근 계약일시, 최근 등록일시)를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라. 해당 쿼리는 온라인에서 자주 사용하기 때문에 최적의 성능을 보장해야 한다.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 회사 테이블은 600개 회사에 대한 정보를 저장한다.
- 대리점 테이블은 회사에 소속된 3600개 대리점에 대한 정보를 저장한다. 회사마다 6개 대리점이 있다. 회사 이름을 반정규화하여 저장한다.
- 계약 테이블은 월 7만 건, 24개월(2022/1 ~ 2023/12) 동안 총 170만 건 대리점의 계약 정보를 저장한다. 대리점 마다 월 20건 계약하며, 계약일시는 Date Type이다. 모든 상품을 골고루 계약하며, 계약상태코드(01 ~ 10) 값도 골고루 분포된다.
  대리점아이디 + 계약일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_CONT2_AGEN_DT ON T2_CONT (AGENY_ID, CONT_DT);
- 계약상태표 테이블은 총 10건의 계약상태 정보를 저장한다. 계약상태구분은 현재 3개(AA, BB, SS)가 있으며, 각 계약상태구분 별로 평균 3건의 계약상태코드를 가진다.

SQL

```
SELECT /*+  LEADING(C A S)  USE_NL(A)  INDEX(A PK_T_AGENCY2)  */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시,
       MAX(C.RGS_DT)    최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND TO_CHAR(C.CONT_DT,'YYYYMM') BETWEEN '202306' AND '202307'--2/24개월,14만건,컬럼가공
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S PK_T_CONT_STA2)  */
                                 S.CONT_STA_CD  -- PK
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS'  -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

2. IN 서브쿼리

문제1	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장될 때

TRACE

```
-------------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name           | Rows     | Elaps. Time  | CR Gets  | Starts    |
-------------------------------------------------------------------------------------------------------------
|   1 | GROUP BY (SORT)              |                |       53 |00:00:00.0002 |        0 |         1 |
|   2 |  INDEX JOIN                  |                |      120 |00:00:00.0000 |        0 |         1 |
|   3 |   INDEX JOIN                 |                |      240 |00:00:00.0015 |        0 |         1 |
|   4 |    TABLE ACCESS (FULL)       | T_CONT2        |     144K |00:00:00.7062 |    12185 |         1 | ①
|   5 |    TABLE ACCESS (ROWID)      | T_AGENCY2      |      240 |00:00:00.0087 |     144K |      144K | ③
|   6 |     INDEX (UNIQUE SCAN)      | PK_T_AGENCY2   |     144K |00:00:00.1433 |     288K |      144K | ②
|   7 |   TABLE ACCESS (ROWID)       | T_CONT_STA2    |      120 |00:00:00.0000 |      240 |       240 | ⑤
|   8 |    INDEX (UNIQUE SCAN)       | PK_T_CONT_STA2 |      240 |00:00:00.0000 |      240 |       240 | ④
-------------------------------------------------------------------------------------------------------------
   4 - filter: (TO_CHAR("C"."CONT_DT",'YYYYMM') >= '202306') AND (TO_CHAR("C"."CONT_DT",'YYYYMM') <=
'202307') (0.100 * 1.000)
   5 - filter: ("A"."COMP_ID" = '0000000300') (0.002)
   6 - access: ("C"."AGENCY_ID" = "A"."AGENCY_ID") (0.000)
   7 - filter: ("S"."CONT_STA_FLAG" = 'SS') (0.500)
   8 - access: ("S"."CONT_STA_CD" = "C"."CONT_STA_CD") (0.100)
```

분석

① 계약→대리점→계약상태표 순서로 액세스한다. 계약 테이블을 Full Scan 해서 2개월에 해당하는 14만 건을 찾아, 계약→대리점으로 NL 조인한다.

② 대리점의 PK인덱스를 14만 번 액세스 한 후, 14만 건을 추출한다. 이때 인덱스 블록을 랜덤 액세스 양(28만 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 14만 건이 테이블의 블록을 랜덤 액세스하여 0000000300 회사 아이디를 확인해서 240건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(14만 블록)이 많아 성능 저하가 발생한다.

④ IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 PK가 있으면, CONT_STA_CD 컬럼과 조인으로 풀려도 건수가 증가되지 않는 것이 보장되기 때문에 메인과 서브쿼리가 조인으로 풀린다. 계약→대리점으로 NL 조인 한 중간 집합(240건)이 계약상태표와 NL 조인을 하면서 인덱스(PK_T_CONT_STAT2)를 240번 액세스해서 240건을 추출한다.

⑤ 인덱스로 추출한 240건이 테이블 블록을 랜덤 액세스해서 120건을 추출한다. ④+⑤ 에서 인덱스와 테이블의 블록을 랜덤 액세스하는 양(480 블록)이 계약상태표 테이블 크기(14 블록) 보다 많다.

⑥ 중간 집합→서브쿼리로 NL 조인해서 추출한 120건을 Group By 해서 53건을 추출한다.

2. IN 서브쿼리

문제1 – 튜닝 1단계	계약일시 컬럼의 가공을 제거하고, 신규 인덱스를 생성한다.

튜닝내역

1. 계약일시 컬럼이 가공되어 계약 테이블을 Full Scan 한다. 좋은 조건(계약일시의 전체 24개월에서 2개월 조회)이 인덱스를 사용하도록 컬럼 가공을 제거하고, 신규 인덱스를 생성한다

   ① 계약일시 컬럼의 가공을 제거한다.

   ② 계약에서 계약일시 컬럼에 신규 인덱스를 생성한다.

SQL

```
CREATE INDEX IX_T_CONT2_DT ON T_CONT2 ( CONT_DT );

SELECT /*+  LEADING(C A S)  USE_NL(A)  INDEX(C IX_T_CONT2_DT)  INDEX(A PK_T_AGENCY2)  */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, A.COMP_NM, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시,
       MAX(C.RGS_DT)    최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM') AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S PK_T_CONT_STA2)  */
                                 S.CONT_STA_CD  -- PK
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS'  -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;

DROP INDEX IX_T_CONT2_DT;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name           | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------
|   1 | GROUP BY (SORT)              |                |       53 |00:00:00.0001 |        0 |        1 |
|   2 |  INDEX JOIN                  |                |      120 |00:00:00.0000 |        0 |        1 |
|   3 |   INDEX JOIN                 |                |      240 |00:00:00.0024 |        0 |        1 |
|   4 |    TABLE ACCESS (ROWID)      | T_CONT2        |     144K |00:00:00.0261 |     1028 |        1 |
|   5 |     INDEX (RANGE SCAN)       | IX_T_CONT2_DT  |     144K |00:00:00.0797 |      412 |        1 |  ②
|   6 |    TABLE ACCESS (ROWID)      | T_AGENCY2      |      240 |00:00:00.0166 |     144K |     144K |
|   7 |     INDEX (UNIQUE SCAN)      | PK_T_AGENCY2   |     144K |00:00:00.1257 |     288K |     144K |
|   8 |   TABLE ACCESS (ROWID)       | T_CONT_STA2    |      120 |00:00:00.0000 |      240 |      240 |
|   9 |    INDEX (UNIQUE SCAN)       | PK_T_CONT_STA2 |      240 |00:00:00.0000 |      240 |      240 |
-------------------------------------------------------------------------------------------------------

   5 - access: ("C"."CONT_DT" >= TO_DATE('2023-06-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("C"."CONT_DT" <= TO_DATE('2023-07-31 23:59:59','YYYY-MM-DD HH24:MI:SS')) (0.296 * 1.000)
   6 - filter: ("A"."COMP_ID" = '0000000300') (0.002)
   7 - access: ("A"."AGENCY_ID" = "C"."AGENCY_ID") (0.000)
   8 - filter: ("S"."CONT_STA_FLAG" = 'SS') (0.500)
   9 - access: ("S"."CONT_STA_CD" = "C"."CONT_STA_CD") (0.100)
```

2. IN 서브쿼리

문제1 – 튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다.

튜닝내역

1. 계약→대리점으로 NL 조인 시, 대리점에 액세스하는 횟수가 많아 대리점의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 인덱스로 추출한 14만 건이 테이블 블록을 액세스하여 0000000300 회사 아이디를 확인해서 240건으로 대폭 줄어든다. 따라서 변별력이 좋은 조건(COMP_ID=0000000300)을 대리점 테이블이 아닌 조인하는 인덱스에서 먼저 확인하면, 테이블 블록을 랜덤 액세스하는 양이 줄어든다(14만 블록→240 블록).

   ① 계약→대리점으로 NL 조인 시, 사용할 신규 인덱스를 AGENCY_ID+COMP_ID 로 생성한다.

   ② INDEX, LEADING 과 USE_NL 힌트로 지정한 인덱스를 사용하여 계약→대리점→계약상태표로 NL 조인한다. UNNEST 힌트는 서브쿼리의 괄호를 제거해서 서브쿼리의 테이블을 메인과 조인이 되도록 한다.

SQL

```
CREATE INDEX IX_T_CONT2_DT ON T_CONT2 ( CONT_DT );
CREATE INDEX IX_T_AGENCY2_AGEN_COMPID ON T_AGENCY2 ( AGENCY_ID, COMP_ID );   -- ①


SELECT /*+  LEADING(C A S)  USE_NL(A) INDEX(C IX_T_CONT2_DT) INDEX(A IX_T_AGENCY2_AGEN_COMPID) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시,
       MAX(C.RGS_DT)    최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM' )
                     AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S PK_T_CONT_STA2)  */
                                 S.CONT_STA_CD  -- PK
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS'  -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;


DROP INDEX IX_T_CONT2_DT;
DROP INDEX IX_T_AGENCY2_AGEN_COMPID;
```

2. IN 서브쿼리

문제1 – 튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다

TRACAE

```
-------------------------------------------------------------------------------------------------------------
|ID| Operation                | Name                  | Rows    | Elaps. Time  | CR Gets  |Starts    |
-------------------------------------------------------------------------------------------------------------
| 1| GROUP BY (SORT)          |                       |      53 |00:00:00.0002 |        0 |       1  |
| 2|  INDEX JOIN              |                       |     120 |00:00:00.0000 |        0 |       1  |
| 3|   INDEX JOIN             |                       |     240 |00:00:00.0022 |        0 |       1  |
| 4|    TABLE ACCESS (ROWID)  | T_CONT2               |    144K |00:00:00.0295 |     1028 |       1  |
| 5|     INDEX (RANGE SCAN)   | IX_T_CONT2_DT         |    144K |00:00:00.0014 |      412 |       1  |
| 6|    TABLE ACCESS (ROWID)  | T_AGENCY2             |     240 |00:00:00.0012 |      240 |     240  |
| 7|     INDEX (RANGE SCAN)   | IX_T_AGENCY2_AGEN_COMPID |   240 |00:00:00.1649 |     144K |    144K  |
| 8|   TABLE ACCESS (ROWID)   | T_CONT_STA2           |     120 |00:00:00.0000 |      240 |     240  |
| 9|    INDEX (UNIQUE SCAN)   | PK_T_CONT_STA2        |     240 |00:00:00.0000 |      240 |     240  |
-------------------------------------------------------------------------------------------------------------
   5 - access: ("C"."CONT_DT" >= TO_DATE('2023-06-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("C"."CONT_DT" <= TO_DATE('2023-07-31 23:59:59','YYYY-MM-DD HH24:MI:SS')) (0.296 * 1.000)
   7 - access: ("A"."AGENCY_ID" = "C"."AGENCY_ID") AND ("A"."COMP_ID" = '0000000300') (0.000 * 0.002)
   8 - filter: ("S"."CONT_STA_FLAG" = 'SS') (0.500)
   9 - access: ("S"."CONT_STA_CD" = "C"."CONT_STA_CD") (0.100)
```

2. IN 서브쿼리

문제1 – 튜닝 3단계	변별력이 좋은 조건을 가지는 대리점을 먼저 액세스하여, 대리점→계약으로 NL 조인한다.

튜닝내역

1. 변별력이 좋은 조건(COMP_ID=0000000300)을 조인하는 인덱스에서 먼저 확인하여 대리점 테이블 블록을 랜덤 액세스하는 양은 줄어드나(14만 블록→240 블록), 인덱스 블록을 랜덤 액세스 하는 양은 여전히 많다. 따라서 변별력이 좋은 조건(COMP_ID=0000000300)을 가지는 대리점을 먼저 액세스하여 대리점→계약으로 NL 조인한다.

   ① 대리점 테이블은 작기 때문에, 대리점에서 0000000300 조건을 찾는 인덱스가 필요 없다.

   ② INDEX, LEADING 과 USE_NL 힌트로 지정한 인덱스를 사용하여 대리점→계약→계약상태표로 NL 조인한다.

SQL

```
SELECT /*+  LEADING(A C S)  USE_NL(C) INDEX(C IX_T_CONT2_AGEN_DT) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID =  '0000000300'     -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE( '202306' ,' YYYYMM' )
                     AND LAST_DAY(TO_DATE( '202307' ,' YYYYMM' )) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S PK_T_CONT_STA2)  */
                                 S.CONT_STA_CD  -- PK
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG =  'SS'    -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------------
|ID  | Operation                    | Name                | Rows     | Elaps. Time  | CR Gets  | Starts    |
-------------------------------------------------------------------------------------------------------------------
|  1 | GROUP BY (SORT)              |                     |       53 |00:00:00.0003 |        0 |         1 |
|  2 |  INDEX JOIN                  |                     |      120 |00:00:00.0000 |        0 |         1 |
|  3 |   INDEX JOIN                 |                     |      240 |00:00:00.0000 |        0 |         1 |
|  4 |    TABLE ACCESS (FULL)       | T_AGENCY2           |        6 |00:00:00.0003 |       44 |         1 |
|  5 |    TABLE ACCESS (ROWID)      | T_CONT2             |      240 |00:00:00.0001 |       13 |         6 |
|  6 |     INDEX (RANGE SCAN)       | IX_T_CONT2_AGEN_DT  |      240 |00:00:00.0001 |       12 |         6 |
|  7 |   TABLE ACCESS (ROWID)       | T_CONT_STA2         |      120 |00:00:00.0003 |      240 |       240 |
|  8 |    INDEX (UNIQUE SCAN)       | PK_T_CONT_STA2      |      240 |00:00:00.0003 |      240 |       240 |
-------------------------------------------------------------------------------------------------------------------
   4 - filter: ("A"."COMP_ID" = '0000000300') (0.002)
   6 - access: ("C"."AGENCY_ID" = "A"."AGENCY_ID") AND ("C"."CONT_DT" >= TO_DATE('2023-06-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("C"."CONT_DT" <= TO_DATE('2023-07-31 23:59:59','YYYY-MM-DD
HH24:MI:SS')) (0.000 * 0.296 * 1.000)
   7 - filter: ("S"."CONT_STA_FLAG" = 'SS') (0.500)
   8 - access: ("S"."CONT_STA_CD" = "C"."CONT_STA_CD") (0.100)
```

2. IN 서브쿼리

문제1 – 튜닝 4단계	NL 조인을 Hash 조인으로 변경하고, SWAP_JOIN_INPUTS 힌트를 사용한다.

튜닝내역

1. 대리점→계약으로 NL 조인 한 중간 집합(240건)이 계약상태표와 NL 조인 시, 계약상태표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양(480 블록)이 계약상태표 테이블의 크기(14 블록) 보다 많아 성능 저하가 발생한다. 따라서 계약상태표 테이블과 NL 조인을 Hash 조인으로 변경한다.
   'CONT_STA_FLAG = SS' 을 만족하는 것은 평균 3건으로 중간 집합(240건) 보다 작기 때문에, SWAP_JOIN_INPUTS 힌트를 사용하여 계약상태표로 Hash Table을 만드는 것이 더 좋다(물론 240건, 5건 모두 작아서 메모리에 Hash Table이 생성되어 성능 차이는 없다).

   ① 대리점→계약으로 NL 조인을 한 중간 집합(240건)을, USE_HASH 와 FULL 힌트로 계약상태표 테이블을 Full Scan 해서 Hash 조인한다.

   ② 원칙적으로 먼저 액세스 한 중간 집합으로 Hash Table이 생성되므로, SWAP_JOIN_INPUTS 힌트를 추가로 사용하여 계약상태표에서 'CONT_STA_FLAG=SS' 를 만족하는 것으로 Hash Table을 만들고, 중간 집합을 읽어 Hash Table에서 조인 조건을 만족하는 것을찾는다.

SQL

```
SELECT /*+  LEADING(A C S)  USE_NL(C) INDEX(C IX_T_CONT2_AGEN_DT) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM') AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_HASH(S)  FULL(S)  SWAP_JOIN_INPUTS(S)  */
                                 S.CONT_STA_CD  -- PK
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS'  -- (2~5)/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name                | Rows     | Elaps. Time  | CR Gets  | Starts    |
-------------------------------------------------------------------------------------------------------------------
|   1 | GROUP BY (SORT)              |                     |       53 |00:00:00.0001 |        0 |         1 |
|   2 |  HASH JOIN                   |                     |      120 |00:00:00.0001 |        0 |         1 |
|   3 |   TABLE ACCESS (FULL)        | T_CONT_STA2         |        5 |00:00:00.0000 |       14 |         1 |
|   4 |   INDEX JOIN                 |                     |      240 |00:00:00.0000 |        0 |         1 |
|   5 |    TABLE ACCESS (FULL)       | T_AGENCY2           |        6 |00:00:00.0002 |       44 |         1 |
|   6 |    TABLE ACCESS (ROWID)      | T_CONT2             |      240 |00:00:00.0001 |       13 |         6 |
|   7 |     INDEX (RANGE SCAN)       | IX_T_CONT2_AGEN_DT  |      240 |00:00:00.0000 |       12 |         6 |
-------------------------------------------------------------------------------------------------------------------

   2 - access("S"."CONT_STA_CD" = "C"."CONT_STA_CD") (0.100)
   3 - filter("S"."CONT_STA_FLAG" = 'SS') (0.500)
   5 - filter("A"."COMP_ID" = '0000000300') (0.002)
   7 - access("C"."AGENCY_ID" = "A"."AGENCY_ID") AND ("C"."CONT_DT" >= TO_DATE('2023-06-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("C"."CONT_DT" <= TO_DATE('2023-07-31 23:59:59','YYYY-MM-DD
HH24:MI:SS')) (0.000 * 0.296 * 1.000)
```

실기

2.IN 서브쿼리 – 문제2

IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때

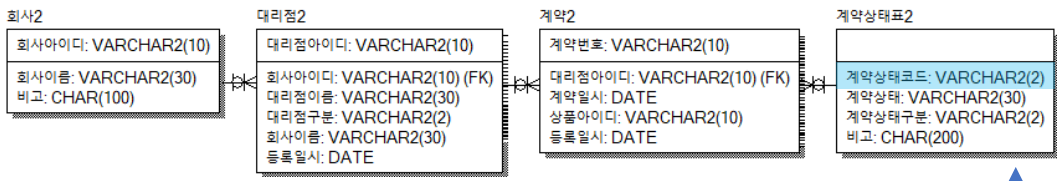
2. IN 서브쿼리

문제2	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때

문제설명

- A 회사의 계약 관리 담당자는 0000000300 회사 아이디를 가지는 대리점이 2023/6/1 ~ 2023/7/31 동안 계약한 계약 정보를 회사 아이디, 회사 이름, 대리점 아이디, 상품 아이디, 계약건수, 최근 계약일시, 최근 등록일시로 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라. 해당 쿼리는 온라인에서 자주 사용하기 때문에 최적의 성능을 보장해야 한다.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 회사 테이블은 600개 회사에 대한 정보를 저장한다.
- 대리점 테이블은 회사에 소속된 3600개 대리점에 대한 정보를 저장한다. 회사마다 6개 대리점이 있다. 회사이름을 반정규화하여 저장한다.
- 계약 테이블은 월 7만 건, 24개월(2022/1 ~ 2023/10) 동안 총 170만 건 대리점의 계약 정보를 저장한다. 대리점 마다 월 20건 계약하며, 계약일시는 DATE TYPE이다. 모든 상품을 골고루 계약하며, 계약상태코드(01 ~ 10) 값도 골고루 분포된다.
  대리점아이디 + 계약일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_CONT2_AGEN_DT ON T2_CONT (AGENT_ID, CONT_DT);
- 계약상태표 테이블은 총 10건의 계약상태 정보를 저장한다. 계약상태구분은 현재 3개(AA, BB, SS)가 있으며, 각 계약상태구분 별로 평균 3건의 계약상태코드를 가진다.

계약상태표 테이블의 계약상태코드는 PK가 아니며, NON-UNIQUE 인덱스가 존재한다.

```
ALTER TABLE T_CONT_STA2 DROP CONSTRAINT PK_T_CONT_STA2;
DROP INDEX PK_T_CONT_STA2;
CREATE INDEX IX_T_CONT_STA2_CD ON T_CONT_STA2 ( CONT_STA_CD );
```

2. IN 서브쿼리

문제2	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때,

SQL

```
SELECT /*+  LEADING(C A S)  USE_NL(A)   INDEX(A PK_T_AGENCY2)  */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)        계약건수,
       MAX(C.CONT_DT)  최근계약일시,
       MAX(C.RGS_DT)   최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND TO_CHAR(C.CONT_DT,'YYYYMM') BETWEEN '202306' AND '202307' --2/24개월, 14만건, 컬럼 가공
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)   INDEX(S IX_T_CONT_STA2_CD)  */
                                S.CONT_STA_CD  -- non-unique 인덱스
                           FROM T_CONT_STA2  S
                          WHERE S.CONT_STA_FLAG = 'SS'  -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

TRACE

CONT_STA2_CD에 PK가 있는 경우(NL 조인)

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation                   | Name             |
-------------------------------------------------------------------------------------------------------
|   1 | GROUP BY (SORT)             |                  |    53  |00:00:00.0002 |      0  |       1  |
|   2 |  INDEX JOIN                 |                  |   120  |00:00:00.0000 |      0  |       1  |
|   3 |  INDEX JOIN                 |                  |   240  |00:00:00.0015 |      0  |       1  |
|   4 |   TABLE ACCESS (FULL)       | T_CONT2          |   144K |00:00:00.7062 |  12185  |       1  |
|   5 |   TABLE ACCESS (ROWID)      | T_AGENCY2        |   240  |00:00:00.0087 |   144K  |    144K  |
|   6 |    INDEX (UNIQUE SCAN)      | PK_T_AGENCY2     |   144K |00:00:00.1433 |   288K  |    144K  |
|   7 |  TABLE ACCESS (ROWID)       | T_CONT_STA2      |   120  |00:00:00.0000 |    240  |     240  |
|   8 |   INDEX (UNIQUE SCAN)       | PK_T_CONT_STA2   |   240  |00:00:00.0000 |    240  |     240  |
-------------------------------------------------------------------------------------------------------
```

CONT_STA2_CD에 PK가 없는 경우(NL 세미조인)

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation                   | Name               |
-------------------------------------------------------------------------------------------------------
|   1 | GROUP BY (SORT)             |                    |    53  |00:00:00.0002 |      0  |       1  |
|   2 |  INDEX JOIN (SEMI)          |                    |   120  |00:00:00.0000 |      0  |       1  |
|   3 |  INDEX JOIN                 |                    |   240  |00:00:00.0034 |      0  |       1  |
|   4 |   TABLE ACCESS (FULL)       | T_CONT2            |   144K |00:00:00.6126 |  12185  |       1  |
|   5 |   TABLE ACCESS (ROWID)      | T_AGENCY2          |   240  |00:00:00.0108 |   144K  |    144K  |
|   6 |    INDEX (UNIQUE SCAN)      | PK_T_AGENCY2       |   144K |00:00:00.0245 |   288K  |    144K  |
|   7 |  TABLE ACCESS (ROWID)       | T_CONT_STA2        |   120  |00:00:00.0003 |    240  |     240  |
|   8 |   INDEX (RANGE SCAN)        | IX_T_CONT_STA2_CD  |   240  |00:00:00.0001 |    240  |     240  |
-------------------------------------------------------------------------------------------------------
   4 - filter: (TO_CHAR("C"."CONT_DT",'YYYYMM') >= '202306') AND (TO_CHAR("C"."CONT_DT",'YYYYMM') <=
'202307') (0.100 * 1.000)
   5 - filter: ("A"."COMP_ID" = '0000000300') (0.002)
   6 - access: ("C"."AGENCY_ID" = "A"."AGENCY_ID") (0.000)
   7 - filter: ("S"."CONT_STA_FLAG" = 'SS') (0.500)
   8 - access: ("C"."CONT_STA_CD" = "S"."CONT_STA_CD") (0.500)
```

2. IN 서브쿼리

문제2	IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때,

분석

① 계약→대리점→계약상태표 순서로 액세스한다. 계약 테이블을 Full Scan 해서 2개월에 해당하는 14만 건을 찾아, 계약→대리점으로 NL 조인한다.

② 대리점의 PK인덱스를 14만 번 액세스 한 후, 14만 건을 추출한다. 이때 인덱스 블록을 랜덤 액세스하는 양(28만 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 14만 건이 테이블 블록을 랜덤 액세스하여 0000000300 회사 아이디를 확인해서 240건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(14만 블록)이 많아 성능 저하가 발생한다.

④ IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 Unique를 보장하는 것(PK 등)이 없어, 메인과 서브쿼리가 세미조인(여기서는 NL 세미조인)으로 풀렸다. 세미조인 외에도 선 Unique/후 조인, 선 조인/후 Unique 또는 FILTER 로 풀릴 수 있다. 세미조인은 서브쿼리 안에서 조인이 성공하는 순간 서브쿼리를 멈추기 때문에 메인 건수가 증가하지 않는다. 계약→대리점으로 NL 조인 한 중간 집합(240건)이 계약상태표의 인덱스(IX_T_CONT_STAT2)를 240번 액세스해서 240건을 추출한다.

⑤ 인덱스로 추출한 240건이 테이블 블록을 랜덤 액세스해서 120건을 추출한다. ④+⑤ 에서 인덱스와 테이블의 블록을 랜덤 액세스하는 양(480 블록)이 계약상태표 테이블 크기(14 블록) 보다 많다.

⑥ 중간 집합→서브쿼리로 NL 조인해서 추출한 120건을 Group By 해서 53건을 추출한다.

2. IN 서브쿼리

문제2 – 튜닝 1단계	계약일시 컬럼의 가공을 제거하고, 신규 인덱스를 생성한다.

튜닝내역

1. 좋은 조건이 인덱스를 사용하도록 컬럼 가공을 제거하고, 신규 인덱스를 생성한다. IN 서브쿼리 안의 CONT_STA_CD 컬럼에 Unique가 보장되지 않아 NL 세미조인으로 풀린다.

SQL

```
CREATE INDEX IX_T_CONT2_DT ON T_CONT2 ( CONT_DT );

SELECT /*+  LEADING(C A S)  USE_NL(A)  INDEX(C IX_T_CONT2_DT)  INDEX(A PK_T_AGENCY2)  */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, A.COMP_NM, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시,
       MAX(C.RGS_DT)    최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM') AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S IX_T_CONT_STA2_CD)  */
                                 S.CONT_STA_CD  -- Non-unique 인덱스
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS' ) -- 3/10
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID;

DROP INDEX IX_T_CONT2_DT;
```

TRACE

CONT_STA2_CD에 PK가 있는 경우(NL 조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0001	0	1
2	INDEX JOIN		120	00:00:00.0000	0	1
3	INDEX JOIN		240	00:00:00.0024	0	1
4	TABLE ACCESS (ROWID)	T_CONT2	144K	00:00:00.0261	1028	144K
5	INDEX (RANGE SCAN)	IX_T_CONT2_DT	144K	00:00:00.0797	412	1
6	TABLE ACCESS (ROWID)	T_AGENCY2	240	00:00:00.0166	144K	144K
7	INDEX (UNIQUE SCAN)	PK_T_AGENCY2	144K	00:00:00.1257	288K	144K
8	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0000	240	240
9	INDEX (UNIQUE SCAN)	PK_T_CONT_STA2	240	00:00:00.0000	240	240

NL 세미조인으로 풀렸다.

CONT_STA2_CD에 PK가 없는 경우(NL 세미조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0002	0	1
2	INDEX JOIN (SEMI)		120	00:00:00.0000	0	1
3	INDEX JOIN		240	00:00:00.0027	0	1
4	TABLE ACCESS (ROWID)	T_CONT2	144K	00:00:00.0232	1028	1
5	INDEX (RANGE SCAN)	IX_T_CONT2_DT	144K	00:00:00.0973	412	1
6	TABLE ACCESS (ROWID)	T_AGENCY2	240	00:00:00.0090	144K	144K
7	INDEX (UNIQUE SCAN)	PK_T_AGENCY2	144K	00:00:00.0249	288K	144K
8	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0003	240	240
9	INDEX (RANGE SCAN)	IX_T_CONT_STA2_CD	240	00:00:00.0001	240	240

2. IN 서브쿼리

문제2 – 튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다.

튜닝내역

1. 변별력이 좋은 조건(COMP_ID=0000000300)을 대리점 테이블이 아닌 조인하는 인덱스에서 먼저 확인하면, 테이블 블록을 랜덤 액세스하는 양이 줄어든다(14만 불럭→240만 블록). IN 서브쿼리 안의 CONT_STA_CD 컬럼에 Unique가 보장되지 않아 NL 세미조인으로 풀리다.

SQL

```
CREATE INDEX IX_T_CONT2_DT ON T_CONT2 ( CONT_DT );
CREATE INDEX IX_T_AGENCY2_AGEN_COMPID ON T_AGENCY2 ( AGENCY_ID, COMP_ID ); -- 조인 시 사용


SELECT /*+  LEADING(C A S)  USE_NL(A) INDEX(C IX_T_CONT2_DT) INDEX(A IX_T_AGENCY2_AGEN_COMPID) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)        계약건수,
       MAX(C.CONT_DT)  최근계약일시,
       MAX(C.RGS_DT)   최근등록일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM' )
                     AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S IX_T_CONT_STA2_CD)  */
                                 S.CONT_STA_CD  -- Non-unique 인덱스
                            FROM T_CONT_STA2  S
                             WHERE S.CONT_STA_FLAG = 'SS'  -- 3/10
                         )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;


DROP INDEX IX_T_CONT2_DT;
DROP INDEX IX_T_AGENCY2_AGEN_COMPID;
```

2. IN 서브쿼리

문제2 – 튜닝 2단계	계약→대리점으로 NL 조인 시, 변별력이 좋은 조건을 대리점 테이블이 아닌 조인 인덱스에서 먼저 확인한다.

TRACE

CONT_STA2_CD에 PK가 있는 경우(NL 조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0002	0	1
2	INDEX JOIN		120	00:00:00.0000	0	1
3	INDEX JOIN		240	00:00:00.0022	0	1
4	TABLE ACCESS (ROWID)	T_CONT2	144K	00:00:00.0295	1028	1
5	INDEX (RANGE SCAN)	IX_T_CONT2_DT	144K	00:00:00.0014	412	1
6	TABLE ACCESS (ROWID)	T_AGENCY2	240	00:00:00.0012	240	240
7	INDEX (RANGE SCAN)	IX_T_AGENCY2_AGEN_COMPID	240	00:00:00.1649	144K	144K
8	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0000	240	240
9	INDEX (UNIQUE SCAN)	PK_T_CONT_STA2	240	00:00:00.0000	240	240

NL 세미조인으로 풀렸다.

CONT_STA2_CD에 PK가 없는 경우(NL 세미조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0001	0	1
2	INDEX JOIN (SEMI)		120	00:00:00.0000	0	1
3	INDEX JOIN		240	00:00:00.0124	0	1
4	TABLE ACCESS (ROWID)	T_CONT2	144K	00:00:00.0292	1028	1
5	INDEX (RANGE SCAN)	IX_T_CONT2_DT	144K	00:00:00.0013	412	1
6	INDEX (RANGE SCAN)	IX_T_AGENCY2_AGEN_COMPID	240	00:00:00.1768	144K	144K
7	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0003	240	240
8	INDEX (RANGE SCAN)	IX_T_CONT_STA2_CD	240	00:00:00.0003	240	240

2. IN 서브쿼리

문제2 – 튜닝 3단계	변별력이 좋은 조건을 가지는 대리점을 먼저 액세스하여, 대리점→계약으로 NL 조인한다.

튜닝내역

1. 변별력이 좋은 조건(COMP_ID='0000000300')을 가지는 대리점을 먼저 액세스하여 대리점→계약으로 NL 조인한다. IN 서브쿼리 안의 CONT_STA_CD 컬럼에 Unique가 보장되지 않아 NL 세미조인으로 풀린다.

SQL

```
SELECT /*+  LEADING(A C S)  USE_NL(C) INDEX(C IX_T_CONT2_AGEN_DT) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID =  '0000000300'    -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM' )
                     AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_NL(S)  INDEX(S IX_T_CONT_STA2_CD)  */
                                 S.CONT_STA_CD  -- Non-unique 인덱스
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG =  'SS'    -- 3/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

TRACE

CONT_STA2_CD에 PK가 있는 경우(NL 조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0003	0	1
2	INDEX JOIN		120	00:00:00.0000	0	1
3	INDEX JOIN		240	00:00:00.0000	0	1
4	TABLE ACCESS (FULL)	T_AGENCY2	6	00:00:00.0003	44	1
5	TABLE ACCESS (ROWID)	T_CONT2	240	00:00:00.0001	13	240
6	INDEX (RANGE SCAN)	IX_T_CONT2_AGEN_DT	240	00:00:00.0001	12	6
7	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0003	240	240
8	INDEX (UNIQUE SCAN)	PK_T_CONT_STA2	240	00:00:00.0003	240	240

NL 세미조인으로 풀렸다.

CONT_STA2_CD에 PK가 없는 경우(NL 세미조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0002	0	1
2	INDEX JOIN (SEMI)		120	00:00:00.0063	0	1
3	INDEX JOIN		240	00:00:00.0001	0	1
4	TABLE ACCESS (FULL)	T_AGENCY2	6	00:00:00.0007	44	1
5	TABLE ACCESS (ROWID)	T_CONT2	240	00:00:00.0003	13	6
6	INDEX (RANGE SCAN)	IX_T_CONT2_AGEN_DT	240	00:00:00.0002	12	6
7	TABLE ACCESS (ROWID)	T_CONT_STA2	120	00:00:00.0013	240	240
8	INDEX (RANGE SCAN)	IX_T_CONT_STA2_CD	240	00:00:00.0038	240	240

2. IN 서브쿼리

문제2 – 튜닝 4단계	NL 조인을 Hash 조인으로 변경하고, SWAP_JOIN_INPUTS 힌트를 사용한다.

튜닝내역

1. 대리점→계약으로 NL 조인 한 중간 집합(240건)과 계약상태표를 Hash 조인한다. 이때 SWAP_JOIN_INPIUTS 힌트를 사용하여 계약상태표로 Hash Table을 만든다. 그리고 IN 서브쿼리 안의 CONT_STA_CD 컬럼에 Unique가 보장되지 않아, 서브쿼리 안의 CONT_STA_CD 컬럼을 Distinct로 Unique하게 만들고, Hash 조인으로 풀린다.

SQL

```
SELECT /*+  LEADING(A C S)  USE_NL(C) INDEX(C IX_T_CONT2_AGEN_DT) */
       A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID,
       COUNT(*)         계약건수,
       MAX(C.CONT_DT)   최근계약일시
  FROM T_AGENCY2  A,
       T_CONT2    C
 WHERE A.COMP_ID = '0000000300'  -- 1/600
   AND A.AGENCY_ID = C.AGENCY_ID
   AND C.CONT_DT BETWEEN TO_DATE('202306','YYYYMM' )
                     AND LAST_DAY(TO_DATE('202307','YYYYMM')) + 0.99999
   AND C.CONT_STA_CD IN ( SELECT /*+  UNNEST  USE_HASH(S)  FULL(S)  SWAP_JOIN_INPUTS(S)  */
                                 S.CONT_STA_CD  -- Non-unique 인덱스
                            FROM T_CONT_STA2  S
                           WHERE S.CONT_STA_FLAG = 'SS'  -- (2~5)/10
                        )
 GROUP BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID
 ORDER BY A.COMP_ID, A.COMP_NM, A.AGENCY_ID, C.GOODS_ID ;
```

TRACE

CONT_STA2_CD에 PK가 있는 경우(Hash 조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0001	0	1
2	HASH JOIN		120	00:00:00.0001	0	1
3	TABLE ACCESS (FULL)	T_CONT_STA2	5	00:00:00.0000	14	1
4	INDEX JOIN		240	00:00:00.0000	0	1
5	TABLE ACCESS (FULL)	T_AGENCY2	6	00:00:00.0002	44	1
6	TABLE ACCESS (ROWID)	T_CONT2	240	00:00:00.0001	13	6
7	INDEX (RANGE SCAN)	IX_T_CONT2_AGEN_DT	240	00:00:00.0000	12	6

Unique하게 만들고, Hash 조인

CONT_STA2_CD에 PK가 없는 경우(Unique후 Hash 조인)

ID	Operation	Name				
1	GROUP BY (SORT)		53	00:00:00.0002	0	1
2	HASH JOIN		120	00:00:00.0000	0	1
3	DISTINCT (HASH)		5	00:00:00.0001	0	1
4	TABLE ACCESS (FULL)	T_CONT_STA2	5	00:00:00.0000	14	1
5	INDEX JOIN		240	00:00:00.0000	0	1
6	TABLE ACCESS (FULL)	T_AGENCY2	6	00:00:00.0002	44	1
7	TABLE ACCESS (ROWID)	T_CONT2	240	00:00:00.0001	13	6
8	INDEX (RANGE SCAN)	IX_T_CONT2_AGEN_DT	240	00:00:00.0000	12	6

2. IN 서브쿼리

정리

1. IN 서브쿼리를 처리하는 방법

IN 서브쿼리 안의 조인 컬럼에 Unique가 보장될 때,	일반 조인(NL 조인, Hash 조인)으로 풀린다.
IN 서브쿼리 안의 조인 컬럼에 Unique가 보장되지 않을 때,	세미조인(NL 세미조인, Hash 세미조인)으로 풀린다.
	서브쿼리 안의 조인 컬럼을 Unique하게 만들고, 일반 조인으로 풀린다.
	일반 조인 후, 조인 결과를 Unique하게 만든다.
	서브쿼리가 FILTER로 풀린다.
기타 경우,	서브쿼리가 메인쿼리와 종속관계에 있지 않고, 서브쿼리의 결과가 1건으로 보장되면, Execute 시점에 서브쿼리를 먼저 수행해서 1건인 서브쿼리 값을 메인쿼리에 상수로 제공한다.
	서브쿼리에 집계함수가 있으면 집계함수를 제거하고, 메인쿼리에 분석함수를 추가한다.

2. IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 Unique를 보장하는 것(PK 등)이 있으면, 일반 조인으로 풀려도 건수가 증가되지 않기 때문에, 메인과 서브쿼리가 일반 조인으로 풀린다.
3. IN 서브쿼리 안에서 조인되는 CONT_STA_CD 컬럼에 Unique를 보장하는 것(PK 등)이 없으면, 세미조인, 선 Unique/후 조인, 선 조인/후 Unique 또는 FILTER 로 풀린다. 세미조인은 서브쿼리 안에서 조인이 성공하면 조인을 멈추어 메인 건수가 증가되지 않는 것을 보장한다.

실기

3. EXISTS 서브쿼리

3. EXISTS 서브쿼리

문제1	2023/3/1 이후 주문 된 상품에 사용된 부품 정보를 조회한다.
분석	주문상세에서 2023/3/1 이후 주문된 모든 상품을 상품, 상품부품, 부품과 조인해서 150만 건을 추출한 후, 중복 제거를 해서 500건 만 조회한다. 주문상세가 상품, 상품부품, 부품과 NL 조인 시, 테이블과 인덱스의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	부품의 사용 여부를 확인하기 위해 EXISTS 서브쿼리를 FILTER로 처리한다.
문제2	구매확정을 한 번도 하지 않고 고객유형이 A, B, C 가 아닌 고객의 정보를 조회한다.
분석	D 고객유형(741건)을 찾기 위해 부정형(NOT IN) 조건 사용으로 고객 테이블을 Full Scan 한다. 서브쿼리에서 NL 조인해서 구매확정(5)을 한 주문 건수를 찾는다. 이때 주문의 테이블과 인덱스의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 FILTER로 처리한다.
튜닝 2단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 NL (세미)안티조인로 처리한다.
튜닝 3단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 Hash (세미)안티조인로 처리한다.
문제3	최근 7일 이내 주문 중, 20살 고객의 고객정보와 고객등급을 조회한다.
분석	20살 고객을 찾기 위해 고객 테이블을 Full Scan 한다. 20살 고객을 고객등급 및 주문과 조인해서 최근 7일 이내 주문한 19248건을 추출한 후, 중복 제거를 해서 6659건을 조회한다. 20살 고객을 고객등급표 및 주문과 NL 조인 시, 테이블과 인덱스의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 FILTER로 처리한다.
튜닝 2단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 NL 세미조인으로 처리한다.
튜닝 3단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 Hash 세미조인으로 처리한다.

실기

3. EXISTS 서브쿼리 - 문제1

2023/3/1 이후 주문 된 상품에 사용된 부품 정보를 조회한다.

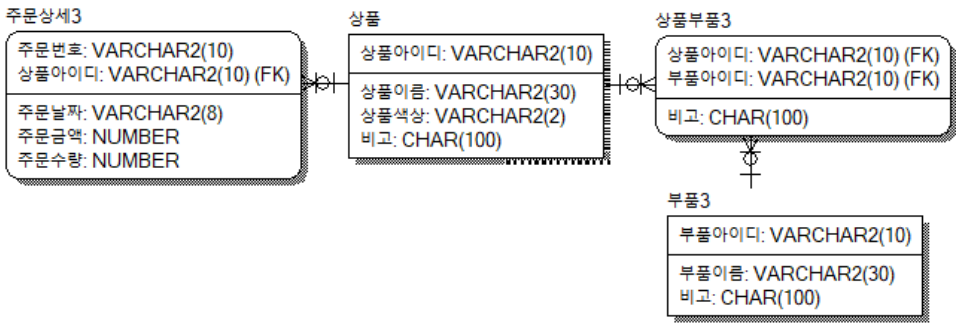
3. EXISTS 서브쿼리

문제1	2023/3/1 이후 주문 된 상품에 사용된 부품 정보를 조회한다.

문제설명

- A 회사의 자재 담당자는 2023/3/1 이후에 주문된 상품에 사용된 부품 정보(부품아이디, 부품이름)를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문상세 테이블은 월 70만 건, 15개월(2022/1 ~ 2023/3) 동안 총 1천만 건 주문상세(주문한 상품) 정보를 저장한다. 하나의 주문은 여러 상품을 주문하는데, 1개 상품이 1개 주문상세에 해당한다. 주문날짜는 문자 8자리이다.
- 상품 테이블은 총 1만개 상품에 대한 정보를 저장한다.
- 상품부품은 상품을 구성하는 부품에 대한 정보를 저장한다. 1개 상품은 평균 2개의 부품으로 만들기 때문에 총 2만 건을 저장한다.
- 부품 테이블은 총 500개 부품에 대한 정보를 저장한다. 부품은 상품을 만드는 데 사용된다.

SQL

```
SELECT /*+ LEADING(OD G GP P) USE_NL(G) USE_NL(GP) USE_NL(P) */
       DISTINCT P.PARTS_ID, P.PARTS_NM
  FROM T_ORDER_D3    OD,  -- 월 70만 건, 15개월
       T_GOODS       G,
       T_GOODS_P3    GP,
       T_PARTS3      P
 WHERE OD.ORDER_DY >= '20230301'
   AND OD.GOODS_ID  = G.GOODS_ID
   AND G.GOODS_ID   = GP.GOODS_ID
   AND GP.PARTS_ID  = P.PARTS_ID;
```

3. EXISTS 서브쿼리

문제1	2023/3/1 이후 주문 된 상품에 사용된 부품 정보를 조회한다.

TRACE

```
-------------------------------------------------------------------------------------------------
|ID| Operation                    | Name           | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|1 | DISTINCT (HASH)              |                |      500 |00:00:00.1953 |        0 |        1 |
|2 |  INDEX JOIN                  |                |    1525K |00:00:00.1231 |        0 |        1 | 부품
|3 |   INDEX JOIN                 |                |    1525K |00:00:00.2739 |        0 |        1 | 상품부품
|4 |    INDEX JOIN                |                |     762K |00:00:00.0344 |        0 |        1 | 상품
|5 |     TABLE ACCESS (FULL)      | T_ORDER_D3     |     762K |00:00:02.2371 |    62353 |        1 | 주문상세
|6 |     INDEX (UNIQUE SCAN)      | PK_T_GOODS     |     762K |00:00:01.1398 |     1525K|     762K |
|7 |    INDEX (RANGE SCAN)        | PK_T_GOODS_P3  |    1525K |00:00:02.1143 |      779K|     762K |
|8 |   TABLE ACCESS (ROWID)       | T_PARTS3       |    1525K |00:00:00.6720 |     1525K|    1525K |
|9 |    INDEX (UNIQUE SCAN)       | PK_T_PARTS3    |    1525K |00:00:02.1310 |     3050K|    1525K |
-------------------------------------------------------------------------------------------------

   5 - filter: ("O"."ORDER_DY" >= '20230301') (0.063)
   6 - access: ("G"."GOODS_ID" = "O"."GOODS_ID") (0.000)
   7 - access: ("GP"."GOODS_ID" = "G"."GOODS_ID") (0.000)
   9 - access: ("P"."PARTS_ID" = "GP"."PARTS_ID") (0.002)
```

분석

① 주문상세→상품→상품부품→부품 순서로 액세스 한다. 주문상세 테이블을 Full Scan 해서 1개월에 해당하는 76만 건을 찾아, 주문상세→상품으로 NL 조인해서 76만 건을 추출한다. 이때 주문상세 테이블을 Full Scan(62353 블록)하고, 상품과 NL 조인 시 상품 PK인덱스 블록을 랜덤 액세스 양(1525K 블록)이 많아 성능 저하가 발생한다.

② 주문상세와 상품이 조인 된 중간 집합(76만 건)→상품부품으로 NL 조인해서 150만 건을 추출한다. 이때 상품부품과 NL 조인 시 상품부품 PK인덱스 블록을 랜덤 액세스 양(779K 블록)이 많아 성능 저하가 발생한다.

③ 주문상세, 상품 그리고 상품부품이 조인 된 중간 집합(150만 건)→부품으로 NL 조인해서 150만 건을 추출한다. 이때 부품과 NL 조인 시 부품의 PK인덱스와 테이블의 블록을 랜덤 액세스 양(3050K + 1525K 블록)이 많아 성능 저하가 발생한다.

④ 주문상세에서 2023/3/1 이후 주문된 모든 상품을 상품, 상품부품, 부품과 조인해서 150만 건을 추출한 후, 중복 제거를 해서 500건 만 조회한다.

3. EXISTS 서브쿼리

문제1 – 튜닝 1단계	부품의 사용 여부를 확인하기 위해 EXISTS 서브쿼리를 FILTER로 처리한다.

튜닝내역

1. 주문상세에서 2023/3/1 이후 주문된 모든 상품을 상품, 상품부품, 부품과 조인해서 150만 건을 추출한 후, 중복 제거를 해서 500건 만 조회한다. 필요한 것은 부품이 사용된 상품의 주문 내역이 아닌 부품의 사용 여부이다. 따라서 부품은 건수가 작기 때문에 부품을 기준으로 EXISTS 서브쿼리를 사용해서 2023/3/1 이후 주문된 상품에서 해당 부품의 사용이 확인되는 순간 부품과 조인을 멈추도록 하고, NO_UNNEST 힌트 등으로 EXISTS 서브쿼리를 FILTER로 처리한다.

SQL

```
CREATE UNIQUE INDEX IX_T_GOODS_P3_PART_GOOD ON T_GOODS_P3 ( PARTS_ID, GOODS_ID );
CREATE INDEX IX_T_ORDER_D3_GOOD_DY ON T_ORDER_D3 ( GOODS_ID, ORDER_DY );

SELECT /*+  LEADING(P)  */  P.PARTS_ID, P.PARTS_NM
  FROM T_PARTS3  P
 WHERE EXISTS (  SELECT /*+  NO_UNNEST  LEADING(GP G OD)  INDEX(GP IX_T_GOODS_P3_PART_GOOD)
                             USE_NL(G) USE_NL(OD) INDEX(GP IX_T_ORDER_D3_GOOD_DY) */  'X'
                   FROM T_ORDER_D3   OD,
                        T_GOODS      G,
                        T_GOODS_P3   GP
                  WHERE OD.ORDER_DY >= '20230301'
                    AND OD.GOODS_ID  = G.GOODS_ID  --3 주문상세와 조인
                    AND G.GOODS_ID   = GP.GOODS_ID --2 상품과 조인
                    AND GP.PARTS_ID  = P.PARTS_ID  --1 상품부품과 조인
);

DROP INDEX IX_T_GOODS_P3_PART_GOOD;
DROP INDEX IX_T_ORDER_D3_GOOD_DY;
```

(주의) 티베로는 각 단계별로 Rows와 CR Gets를 표시하나, FILTER 처리 시 해당 부분(ID 3 ~ 9)에 대한 CR Gets 합을 표시한다.

TRACE

EXISTS를 FILTER로 처리

```
-----------------------------------------------------------------------------------------------------------
|ID| Operation                      |Name                  |Rows | Elaps. Time |CR Gets|Starts|
-----------------------------------------------------------------------------------------------------------
|1|FILTER                          |                      |  500|00:00:00.0259|   2871|     1 |
|2| TABLE ACCESS (FULL)            |T_PARTS3              |  500|00:00:00.0002|     14|     1 |부품
|3| CACHE                          |                      |  500|00:00:00.0006|      0|   500 |
|4|  COUNT (STOP NODE) (STOP LIMIT 2)|                    |  500|00:00:00.0007|      0|   500 |
|5|   INDEX JOIN                   |                      |  500|00:00:00.0004|      0|   500 |주문상세
|6|    INDEX JOIN                  |                      |  500|00:00:00.0005|      0|   500 |상품
|7|     INDEX (RANGE SCAN) DESCENDING|IX_T_GOODS_P3_PART_GOOD|18309|00:00:00.0050|   1184|   500 |상품부품
|8|     INDEX (UNIQUE SCAN)        |PK_T_GOODS            |  500|00:00:00.0024|   1000|   500 |
|9|    INDEX (RANGE SCAN)          |IX_T_ORDER_D3_GOOD_DY |31570|00:00:00.0085|    687|   500 |
-----------------------------------------------------------------------------------------------------------

   1 - filter: EXISTS (SELECT /*+ NO_UNNEST LEADING(GP G OD) INDEX(GP IX_T_GOODS_P3_PART_GOOD) INDEX(OD IX_T_ORDER_D3_GOOD_DY) USE_NL(G) USE_NL(OD)
*/ 'X'                    FROM T_ORDER_D3 OD,    T_GOODS    G,    T_GOODS_P3 GP
                          WHERE OD.ORDER_DY>='20230301'
                            AND OD.GOODS_ID=G.GOODS_ID AND G.GOODS_ID=GP.GOODS_I
                         ) (1.000)
   4 - filter: (ROWNUM = 1) (0.010)
   7 - access: ("GP"."PARTS_ID" = :0) (0.002)
   8 - access: ("GP"."GOODS_ID" = "G"."GOODS_ID") (0.000)
   9 - access: ("G"."GOODS_ID" = "OD"."GOODS_ID") AND ("OD"."
```

(주의) 티베로는 (NOT) EXISTS/IN 서브쿼리 안에서 Index Range Scan 시, 해당 Leaf 블록 1개에 속한 Rows를 가져온다. 이는 1개의 Row 만 따로 추출하여 별도 메모리에서 관리하는 부하를 없애기 위해서다. 테이블 블록 액세스 시 Rownum < 1 을 적용하므로 최종적으로 테이블에서 추출되는 Rows 수는 정확하다.

실기

3. EXISTS 서브쿼리 – 문제2

구매확정을 한 번도 하지 않고, 고객유형이 A, B, C 가 아닌 고객 정보를 조회한다.

3. EXISTS 서브쿼리

문제2	구매확정을 한 번도 하지 않고, 고객유형이 A, B, C 가 아닌 고객의 정보를 조회한다.

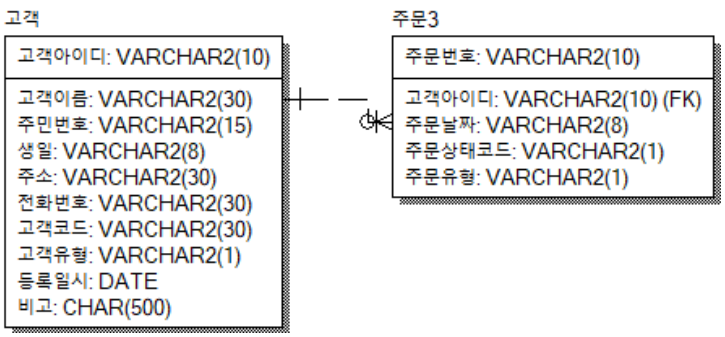
문제설명

- A 회사의 주문 담당자는 고객유형이 A, B, C 가 아닌 고객 중에서 구매확정을 한 번도 하지 않은 고객의 정보를 조회하는 쿼리를 사용하고 있다.

```
CUST_ID    CUST_NM   JUMINNO         BRTH_DY   ADDRESS   TELNO          CUST_TY
---------- --------  --------------- --------  --------- -------------- -------
0000069308 KIM R DD  000104*******    20000104 R City    010-1234-9307  D
0000069346 KIM D PH  070525*******    20070525 D City    010-1234-9345  D
0000069467 KIM U CJ  090922*******    20090922 U City    010-1234-9466  D
0000069506 KIM H TL  030808*******    20030808 H City    010-1234-9505  D
0000069516 KIM H AA  030809*******    20031109 X City    010-1234-9522  D
 :
10 rows selected.
```

- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CUST (CUST_TY);
- 주문 테이블은 월 70만 건, 15개월(2022/1 ~ 2023/3) 동안 총 1천만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장되며, 한번도 주문하지 않은 고객은 전체 7만 고객의 1%(700명)를 차지한다. 주문날짜는 문자 8자리이다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
  고객아이디 컬럼에 인덱스가 있다. CREATE INDEX IX_T_ORDER3_CUSTID ON T_ORDER3 (CUST_ID);

3. EXISTS 서브쿼리

문제2	구매확정을 한 번도 하지 않고, 고객유형이 A, B, C 가 아닌 고객의 정보를 조회한다.

SQL

```
SELECT CUST_ID, CUST_NM, JUMINNO, BRTH_DY, ADDRESS, TELNO, CUST_TY
  FROM T_CUST A
WHERE A.CUST_TY NOT IN ('A','B','C')          -- 1% 를 부정형으로 찾는다
  AND ( SELECT /*+ NO_UNNEST */  COUNT(*)
          FROM T_ORDER3  B
         WHERE B.CUST_ID = A.CUST_ID
           AND B.ORDER_STA_CD = '5'           -- 대부분 5(구매확정)
      ) < 1 ;
```

TRACE

(주의) 티베로는 각 단계별로 Rows와 CR Gets를 표시하나, FILTER 처리 시 해당 부분(ID 3 ~ 6)에 대한 CR Gets 합을 표시한다.

서브쿼리 < 1 을 FILTER로 처리

```
-------------------------------------------------------------------------------------------
|ID | Operati                          | Rows  | Elaps. Time   | CR Gets  | Starts |
-------------------------------------------------------------------------------------------
| 1 | FILTER                           |   10  |00:00:06.5005 |    113K  |      1 | ③
| 2 |  TABLE ACCESS (FULL)    | T_CUST |  741  |00:00:00.0811 |    5858  |      1 | ①
| 3 |  CACHE                           |  741  |00:00:00.0006 |       0  |    741 | ②
| 4 |   SORT AGGR                      |  741  |00:00:00.0039 |       0  |    741 |
| 5 |    TABLE ACCESS (ROWID)| T_ORDER3  |  109K |00:00:06.3263 |    109K  |    741 |
| 6 |     INDEX (RANGE SCAN) | IX_T_ORDER3_CUSTID | 109K |00:00:00.1604 |  3644  |    741 |
-------------------------------------------------------------------------------------------

  1 - filter: (1 > ( SELECT /*+ NO_UNNEST */ COUNT(*)
                       FROM T_ORDER3  B
                      WHERE B.CUST_ID = A.CUST_ID
                        AND B.ORDER_STA_CD = '5'
                   )) (0.100)

  2 - filter: (("A"."CUST_TY") NOT IN (('A'),('B'),('C'))) (0.300)
  5 - filter: ("B"."ORDER_STA_CD" = '5') (1.000)
  6 - access: ("B"."CUST_ID" = :0) (0.000)
```

분석

① 고객에서 1%를 차지하는 D 고객유형(741건)을 찾기 위해 부정형(NOT IN) 조건 사용으로 인덱스를 사용하지 못하고 고객 테이블을 Full Scan 한다.

② 메인 쿼리의 D 고객유형(741건)으로 서브쿼리에서 IX_T_ORDER3_CUSTID 인덱스로 주문과 NL 조인해서 해당 고객의 구매확정(5) 주문 건수를 구한다. 이때 주문에 액세스하는 횟수가 많아 주문의 테이블과 인덱스의 블록을 랜덤 액세스하는 양(3644 + 109K 블록)이 많아 성능 저하가 발생한다.

③ 구매확정(5)을 한 건수가 1보다 작은 지 확인한다.

3. EXISTS 서브쿼리

문제2 – 튜닝 1단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 FILTER로 처리한다.

튜닝내역

1. 고객에서 1%를 차지하는 D 고객유형(741건)을 찾기 위해 부정형(NOT IN) 조건 사용으로 인덱스를 사용하지 못한다. 따라서 부정형(NOT IN) 조건을 긍정형(IN) 조건으로 변경한다.
2. 메인 쿼리의 D 고객유형을 가진 고객을 서브쿼리에서 구매확정(5) 한 주문 건수를 구해서 1보다 작은지 확인한다. 필요한 것은 구매 확정(5) 주문 건수가 아닌 구매확정(5) 주문의 존재 여부이다. 따라서 D 고객유형(741건)은 건수가 작기 때문에 고객을 기준으로 NOT EXISTS 서브쿼리를 사용해서 주문에서 해당 고객의 구매확정(5) 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 하고, NO_UNNEST 힌트로 NOT EXISTS 서브쿼리를 FILTER로 처리한다.
   - ① 고객에서 인덱스로 D 고객유형을 741건 찾아서, 테이블에서 690 블록(총 4722블록 – (주문 인덱스 3301블록 + 주문 테이블 731블록)을 액세스 해서 741건 추출한다.
   - ② D 고객유형 741건에 대해 NOT EXISTS 서브쿼리 안에서 IX_T_ORDER3_CUSTID 인덱스를 사용하여 구매확정(5) 주문을 가진 고객 731건을 확인한다.
   - ③ 구매확정(5) 주문을 가진 D 고객유형 731건을 제외한 10건의 고객 정보를 추출한다.

SQL

```
SELECT /*+ INDEX(A IX_T_CUST_TY) */
       CUST_ID, CUST_NM, JUMINNO, BRTH_DY, ADDRESS, TELNO, CUST_TY
  FROM T_CUST A
 WHERE A.CUST_TY IN ('D')          -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND NOT EXISTS ( SELECT /*+ NO_UNNEST */  1
                      FROM T_ORDER3  B
                     WHERE B.CUST_ID = A.CUST_ID
                       AND B.ORDER_STA_CD = '5'     -- 대부분 5(구매확정)
                  );
```

TRACE

(주의) 티베로는 각 단계별로 Rows와 CR Gets를 표시하나, FILTER 처리 시 해당 부분(ID 4 ~ 6)에 대한 CR Gets 합을 표시한다. 여기서는 ID 1 T_CUST 테이블 블록(690)도 추가한다.

Operator에 FILTER 표시가 없으나 NOT EXISTS를 FILTER로 처리

```
-----------------------------------------------------------------------------------------------------------
|ID|Operation                          | Name              | Rows  | Elaps. Time | CR Gets | Starts |
-----------------------------------------------------------------------------------------------------------
| 1|TABLE ACCESS (ROWID)               | T_CUST            |    10 |00:00:00.0195|    4722 |      1 |③
| 2| INDEX (RANGE SCAN)                | IX_T_CUST_TY      |   741 |00:00:00.0001|       6 |      1 |①
| 3| CACHE                             |                   |   741 |00:00:00.0006|       0 |    741 |
| 4| COUNT (STOP NODE) (STOP LIMIT 2)  |                   |   731 |00:00:00.0003|       0 |    741 |②
| 5|  TABLE ACCESS (ROWID)             | T_ORDER3          |   731 |00:00:00.0023|     731 |    741 |
| 6|   INDEX (RANGE SCAN)              | IX_T_ORDER3_CUSTID| 83594 |00:00:00.0092|    3301 |    741 |
-----------------------------------------------------------------------------------------------------------

  1 - filter: NOT EXISTS ( SELECT /*+ NO_UNNEST */  1
                             FROM T_ORDER3 B
                            WHERE B.CUST_ID = A.CUST_ID
                              AND B.ORDER_STA_CD = '5'
                         ) (1.000)

  2 - access: ("A"."CUST_TY" = 'D') (0.009)
  4 - filter: (ROWNUM = 1) (0.010)
  5 - filter: ("B"."ORDER_STA_CD" = '5') (1.000)
  6 - access: ("B"."CUST_ID" = :0) (0.000)
```

(주의) 티베로는 (NOT) EXISTS/IN 서브쿼리 안에서 Index Range Scan 시, Leaf 블록 1개에 속한 Rows를 가져온다. 이는 1개의 Row 만 따로 추출하여 별도 메모리에서 관리하는 부하를 없애기 위함이다. 테이블 블록 액세스 시 Rownum < 1 을 적용하므로 최종적으로 테이블에서 추출되는 Rows 수는 정확하다.

3. EXISTS 서브쿼리

문제2 – 튜닝 2단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 NL (세미)안티조인으로 처리한다.

튜닝내역

1. 고객에서 1%를 차지하는 D 고객유형(741건)을 찾기 위해 부정형(NOT IN) 조건 사용으로 인덱스를 사용하지 못한다. 따라서 부정형(NOT IN) 조건을 긍정형(IN) 조건으로 변경한다.
2. 메인 쿼리의 D 고객유형을 가진 고객을 서브쿼리에서 구매확정(5) 주문 건수를 구해서 1보다 작은지 확인한다. 필요한 것은 구매 확정(5) 주문 건수가 아닌 구매확정(5) 주문의 존재 여부이다. 따라서 D 고객유형(741건)은 건수가 작기 때문에 고객을 기준으로 NOT EXISTS 서브쿼리를 사용해서 주문에서 해당 고객의 구매확정(5) 주문 존재가 확인되는 순간 주문과 조인을 멈추도록 하고, UNNEST 와 NL_AJ 힌트를 사용하여 NOT EXISTS 서브쿼리를 NL (세미)안티조인으로 처리한다.

SQL

```
SELECT /*+ INDEX(A IX_T_CUST_TY) */
       CUST_ID, CUST_NM, JUMINNO, BRTH_DY, ADDRESS, TELNO, CUST_TY
  FROM T_CUST A
 WHERE A.CUST_TY IN ('D')                    -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND NOT EXISTS ( SELECT /*+ UNNEST  NL_AJ */  1
                      FROM T_ORDER3  B
                     WHERE B.CUST_ID = A.CUST_ID
                       AND B.ORDER_STA_CD = '5'    -- 대부분 5(구매확정)
                  );
```

TRACE

NOT EXISTS를 안티 조인으로 처리

```
-----------------------------------------------------------------------------------------------------
|ID| Operation                    | Name                | Rows  | Elaps. Time  | CR Gets|Starts  |
-----------------------------------------------------------------------------------------------------
|1 | INDEX JOIN (ANTI)            |                     |    10 |00:00:00.0003 |      0 |     1  |
|2 |  ORDER BY (SORT)             |                     |   741 |00:00:00.0008 |      0 |     1  |
|3 |   TABLE ACCESS (ROWID)       | T_CUST              |   741 |00:00:00.0058 |    690 |     1  |
|4 |    INDEX (RANGE SCAN)        | IX_T_CUST_TY        |   741 |00:00:00.0001 |      6 |     1  |
|5 |  TABLE ACCESS (ROWID)        | T_ORDER3            |   641 |00:00:00.1083 |   109K |   741  |
|6 |   INDEX (RANGE SCAN)         | IX_T_ORDER3_CUSTID  | 68700 |00:00:00.0050 |   1535 |   741  |
-----------------------------------------------------------------------------------------------------

   4 - access: ("A"."CUST_TY" = 'D') (0.009)
   5 - filter: ("B"."ORDER_STA_CD" = '5') (1.000)
   6 - access: ("A"."CUST_ID" = "B"."CUST_ID") (1.000)
```

(주의) 티베로는 (NOT) EXISTS/IN 서브쿼리 안에서 Index Range Scan 시, Leaf 블록 1개에 속한 Rows를 가져온다. 이는 1개의 Row 만 따로 추출하여 별도 메모리에서 관리하는 부하를 없애기 위함이다. 테이블 블록 액세스 시 Rownum < 1 을 적용하므로 최종적으로 테이블에서 추출되는 Rows 수는 정확하다.

3. EXISTS 서브쿼리

문제2 – 튜닝 3단계	D 고객유형(741건)의 구매확정(5) 존재 여부를 확인하기 위해 NOT EXISTS 서브쿼리를 Hash (세미)안티조인로 처리한다.

튜닝내역

1. 고객에서 1%를 차지하는 D 고객유형을 찾기 위해 부정형(NOT IN) 조건을 사용으로 인덱스를 사용하지 못한다. 따라서 부정형(NOT IN)을 긍정형(IN)으로 변경한다.
2. 메인 쿼리애서 D 고객유형을 가진 고객 아이디로 서브쿼리에서 구매확정(5)을 한 주문 건수를 구해서, 1보다 작은지 확인한다. 구매확정(5) 한 주문 건수를 구했지만 실제로는 구매확정(5) 한 주문 존재만 확인한다. 따라서 NOT EXISTS를 사용하면 메인 쿼리에서 D 고객유형을 가진 고객 아이디로 NOT EXISTS 서브쿼리에서 구매확정(5)을 한 주문의 존재를 확인하는 순간 False가 되어 서브쿼리 내 처리를 멈추고. 메인쿼리의 다음 D 고객유형을 가진 고객을 확인하도록 사용하고, UNNEST 와 HASH_AJ 힌트로 Hash (세미)안티조인으로 처리한다.

SQL

```
SELECT /*+ INDEX(A IX_T_CUST_TY) */
       CUST_ID, CUST_NM, JUMINNO, BRTH_DY, ADDRESS, TELNO, CUST_TY
  FROM T_CUST A
 WHERE A.CUST_TY IN ('D' )          -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND NOT EXISTS ( SELECT /*+ UNNEST  HASH_AJ  */  1
                      FROM T_ORDER3  B
                     WHERE B.CUST_ID = A.CUST_ID
                       AND B.ORDER_STA_CD = '5'     -- 대부분 5(구매확정)
                  );
```

TRACE

NOT EXISTS를 안티 조인으로 처리

```
-------------------------------------------------------------------------------------------------
| ID  | Operati...                  | Name          | Rows     | Elaps. Time  | CR Gets  | Starts  |
-------------------------------------------------------------------------------------------------
|   1 | HASH JOIN (ANTI)            |               |       10 |00:00:00.6749 |        0 |       1 |
|   2 |  TABLE ACCESS (ROWID)       | T_CUST        |      741 |00:00:00.0021 |      690 |       1 |
|   3 |   INDEX (RANGE SCAN)        | IX_T_CUST_TY  |      741 |00:00:00.0108 |        6 |       1 |
|   4 |  TABLE ACCESS (FULL)        | T_ORDER3      |      10M |00:00:02.4096 |    57561 |       1 |
-------------------------------------------------------------------------------------------------

   1 - access: ("A"."CUST_ID" = "B"."CUST_ID") (1.000)
   3 - access: ("A"."CUST_TY" = 'D') (0.009)
   4 - filter: ("B"."ORDER_STA_CD" = '5') (1.000)
```

실기

3. EXISTS 서브쿼리 – 문제3

최근 7일 이내 주문 중, 20살 고객의 고객정보와 고객등급을 조회한다.

3. EXISTS 서브쿼리

문제3	최근 7일 이내 주문 한 20살 고객의 고객정보와 고객등급을 조회한다.

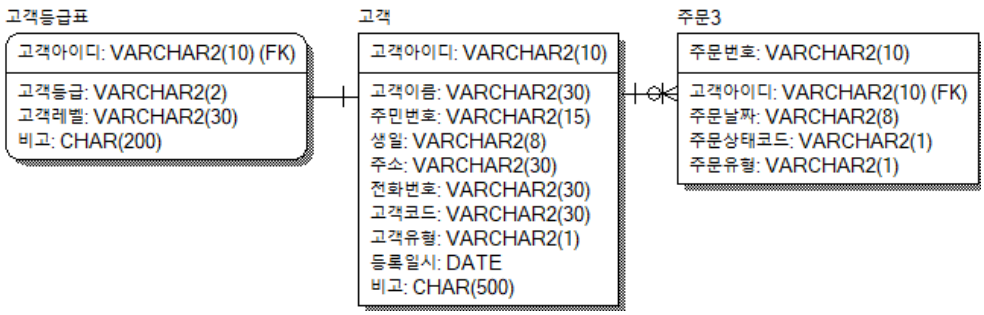
문제설명

- A 회사의 주문 담당자는 최근 7일 이내 주문한 나이가 20살 고객의 고객정보(고객이름, 고객 아이디, 생일, 나이)와 고객등급을 고객이름으로 정렬하여 조회하는 쿼리를 사용하고 있다. 나이는 Trunc((현재의 년월일 – 생일의 년월일)/365) 이다.

```
CCUST_NM    CUST_ID    CUST_GRAD  BRTH_DY     AGE
----------- ---------- ---------- ---------- ---
KIM A AB    0000043317 08         2002/07/19  20
KIM A AB    0000043318 08         2002/07/19  20
KIM A AB    0000043319 08         2002/07/19  20
KIM A AB    0000043320 08         2002/07/19  20
KIM A AD    0000052880 05         2003/01/15  20
KIM A AD    0000052882 05         2003/01/15  20
KIM A AD    0000052890 05         2003/01/15  20
KIM A AD    0000052899 05         2003/01/15  20
KIM A AH    0000016173 05         2002/07/23  20
  :
6515 rows selected.
```

- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객등급표 테이블은 7만명 고객에 대한 고객등급과 고객레벨을 저장한다. 고객등급은 01 ~ 11 값을 가지며, VIP 고객은 고객레벨에 VIP 값을 가지며, 전체 고객의 1%를 차지한다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CUST (CUST_TY);
- 주문 테이블은 월 70만 건, 15개월(2022/1 ~ 2023/3) 동안 총 1천만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장되며, 한번도 주문하지 않은 고객은 전체 7만 고객의 1%(700명)를 차지한다. 주문날짜는 문자 8자리이다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
  고객아이디 컬럼에 인덱스가 있다. CREATE INDEX IX_T_ORDER3_CUSTID ON T_ORDER3 (CUST_ID);

3. EXISTS 서브쿼리

문제3	최근 7일 이내 주문 한 20살 고객의 고객정보와 고객등급을 조회한다.

SQL

```
SELECT  /*+ LEADING(C G O) USE_NL(G) USE_NL(O) */
       DISTINCT C.CUST_NM, C.CUST_ID, G.CUST_GRAD, C.BRTH_DY,
       TRUNC((TO_DATE('20230301','YYYYMMDD')-TO_DATE(C.BRTH_DY,'YYYYMMDD')) / 365) AS AGE
  FROM T_CUST       C,
       T_CUST_GRAD  G,
       T_ORDER3     O
WHERE C.BRTH_DY >  TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 21 * 365 ) ,'YYYYMMDD')
  AND C.BRTH_DY <= TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 20 * 365 ) ,'YYYYMMDD')
  AND C.CUST_ID = G.CUST_ID(+)
  AND C.CUST_ID = O.CUST_ID
  AND O.ORDER_DY >= TO_CHAR((TO_DATE('20230301','YYYYMMDD') - 7),'YYYYMMDD')
  AND O.ORDER_DY <= TO_CHAR( TO_DATE('20230301','YYYYMMDD'),'YYYYMMDD')
  AND O.ORDER_TY IS NOT NULL
ORDER BY C.CUST_NM;
```

테스트를 위해 오늘을 2023/3/1 로 가정한다.

TRACE

```
-------------------------------------------------------------------------------------------------
| ID| Operation                    | Name                | Rows    | Elaps. Time  | CR Gets | Starts |
-------------------------------------------------------------------------------------------------
| 1 | DISTINCT (SORT)              |                     |  6659   |00:00:00.0499 |      0  |      1 |③
| 2 |  INDEX JOIN                  |                     | 19248   |00:00:00.0041 |      0  |      1 |
| 3 |   INDEX JOIN (LEFT OUTER)    |                     |  7016   |00:00:00.0001 |      0  |      1
| 4 |    ORDER BY (SORT)           |                     |  7016   |00:00:00.0024 |      0  |      1
| 5 |     TABLE ACCESS (FULL)      | T_CUST              |  7016   |00:00:00.0115 |   5858  |      1
| 6 |    TABLE ACCESS (ROWID)      | T_CUST_GRAD         |  7016   |00:00:00.0014 |   7016  |   7016
| 7 |     INDEX (UNIQUE SCAN)      | PK_T_CUST_GRAD      |  7016   |00:00:00.0055 |  14032  |   7016 |①
| 8 |   TABLE ACCESS (ROWID)       | T_ORDER3            | 19248   |00:00:00.7969 |  1042K  |   7016
| 9 |    INDEX (RANGE SCAN)        | IX_T_ORDER3_CUSTID  |  1042K  |00:00:00.0392 |  13680  |   7016 |②
-------------------------------------------------------------------------------------------------
   5 - filter: ("C"."BRTH_DY" <= '20030306') AND ("C"."BRTH_DY" > '20020306') (0.321 * 1.000)
   7 - access: ("G"."CUST_ID" = "C"."CUST_ID") (0.000)
   8 - filter: ("O"."ORDER_DY" >= '20230222') AND ("O"."ORDER_DY" <= '20230301') AND ("O"."ORDER_TY"
IS NOT NULL) (0.094 * 1.000 * 1.000)
   9 - access: ("O"."CUST_ID" = "C"."CUST_ID") (0.000)
```

분석

① 고객→고객등급표→주문 순서로 액세스 한다. 고객 테이블을 Full Scan 해서 20살에 해당하는 7016건을 찾아, 고객→고객등급표로 NL 조인해서 7016건을 추출한다. 이때 고객 테이블을 Full Scan(5858 블록) 하고, 고객등급표와 NL 조인 시 고객등급표의 PK인덱스와 테이블의 블록을 랜덤 액세스하는 양(14032 + 7016 블록)이 많아 성능 저하가 발생한다.

② 고객과 고객등급표가 조인 된 중간집합(7016건)→주문으로 NL 조인해서 19248건을 추출한다. 이때 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양(13680 + 1042K 블록)이 많아 성능 저하가 발생한다.

③ 20살 고객이 최근 7일 이내 주문한 19248건을 추출한 후, 중복 제거를 해서 6659건만 조회한다.

3. EXISTS 서브쿼리

문제3 – 튜닝 1단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 FILTER로 처리한다.

튜닝내역

1. 고객 테이블의 생일 컬럼에 인덱스를 생성하고 INDEX 힌트를 사용한다.
2. 고객등급표와 NL 조인 시, 고객등급표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 고객등급표 테이블의 크기가 작기 때문에 FULL 과 USE_HASH 힌트로 NL 조인을 Hash 조인으로 변경한다.
3. 최근 7일 이내 모든 주문을 찾기 위해 주문과 NL 조인해서 19248건을 추출한 후, 중복 제거를 해서 6659건을 조회한다. 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 필요한 것은 20살 고객의 최근 7일 이내의 모든 주문이 아닌 주문의 존재 여부이다. 따라서 20살 고객과 고객등급표의 조인 결과는 건수가 작기 때문에 이것을 기준으로 EXISTS 서브쿼리를 사용해서 최근 7일 이내 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 하고, NO_UNNEST 힌트 등으로 EXISTS 서브쿼리를 FILTER로 처리한다.

TRACE

```
CREATE INDEX IX_T_CUST_BRTHDY ON T_CUST ( BRTH_DY );                    --고객
CREATE INDEX IX_T3_ORDER_CUSTID_DY ON T_ORDER3 ( CUST_ID, ORDER_DY );  --주문


SELECT /*+ INDEX(C IX_T_CUST_BRTHDY) FULL(G) USE_HASH(G) */
       C.CUST_NM, C.CUST_ID, G.CUST_GRAD, C.BRTH_DY,
       TRUNC((TO_DATE('20230301','YYYYMMDD') - TO_DATE(C.BRTH_DY,'YYYYMMDD') ) /365) AS AGE
  FROM T_CUST       C,
       T_CUST_GRAD  G
 WHERE C.BRTH_DY >  TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 21 * 365 ) ,'YYYYMMDD')
   AND C.BRTH_DY <= TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 20 * 365 ) ,'YYYYMMDD')
   AND C.CUST_ID = G.CUST_ID(+)
   AND EXISTS ( SELECT /*+ NO_UNNEST  INDEX(O IX_T3_ORDER_CUSTID_DY) */ 1
                  FROM T_ORDER3  O
                 WHERE O.CUST_ID = C.CUST_ID
                   AND O.ORDER_DY >= TO_CHAR((TO_DATE('20230301','YYYYMMDD')-7),'YYYYMMDD')
                   AND O.ORDER_DY <= TO_CHAR( TO_DATE('20230301','YYYYMMDD'),'YYYYMMDD')
                   AND O.ORDER_TY IS NOT NULL
              )
  ORDER BY C.CUST_NM;


DROP INDEX IX_T_CUST_BRTHDY;
DROP INDEX IX_T3_ORDER_CUSTID_DY;
```

3. EXISTS 서브쿼리

문제3 – 튜닝 1단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 FILTER로 처리한다.

튜닝내역

(주의) 티베로는 각 단계별로 Rows와 CR Gets를 표시하나, FILTER 처리 시 해당 부분(ID 5 ~ 8)에 대한 CR Gets 합을 표시한다. 여기서는 ID 3 T_CUST 테이블 블록(4223)도 추가한다.

Operator에 FILTER 표시가 없으나 NOT EXISTS를 FILTER로 처리

중간집합(Hash Table)

```
-------------------------------------------------------------------------------------------------------
|ID| Operation                      | Name                | Rows  | Elaps. Time |CR Gets|Starts|
-------------------------------------------------------------------------------------------------------
|1|ORDER BY                         |                     |  6659 |00:00:00.0190|     0 |    1 |
|2| HASH JOIN (LEFT OUTER)          |                     |  6659 |00:00:00.0071|     0 |    1 |
|3|  TABLE ACCESS (ROWID)           |T_CUST               |  6659 |00:00:01.3127|  32201|    1 |
|4|   INDEX (RANGE SCAN)            |IX_T_CUST_BRTHDY     |  7016 |00:00:00.0001|     24|    1 |
|5|   CACHE                         |                     |  7016 |00:00:00.0067|     0 | 7016 |
|6|    COUNT (STOP NODE) (STOP LIMIT2)|                   |  6659 |00:00:00.0008|     0 | 7016 |
|7|     TABLE ACCESS (ROWID)        |T_ORDER3             |  6659 |00:00:00.0245|   6659| 7016 |
|8|      INDEX (RANGE SCAN)         |IX_T3_ORDER_CUSTID_DY| 19166 |00:00:01.2254|  21319| 7016 |
|9|  TABLE ACCESS (FULL)            |T_CUST_GRAD          | 70000 |00:00:00.0068|   2199|    1 |
-------------------------------------------------------------------------------------------------------
```

```
   2 - access: ("C"."CUST_ID" = "G"."CUST_ID") (0.000)
   3 - filter: EXISTS (SELECT /*+ NO_UNNEST */ 1
                       FROM T_ORDER3 O
                       WHERE O.CUST_ID = C.CUST_ID
                         AND O.ORDER_DY >= TO_CHAR((TO_DATE('20230301','YYYYMMDD
                         AND O.ORDER_DY <= TO_CHAR(TO_DATE('20230301','YYYYMMDD
                         AND O.ORDER_TY IS NOT NULL
                       ) (1.000)
   4 - access: ("C"."BRTH_DY" > '20020306') AND ("C"."BRTH_DY" <= '20030306') (1.000 * 0.321)
   6 - filter: (ROWNUM = 1) (0.010)
   7 - filter: ("O"."ORDER_TY" IS NOT NULL) (1.000)
   8 - access: ("O"."CUST_ID" = :0) AND ("O"."ORDER_DY" >= '20230222') AND ("O"."ORDER_DY" <=
'20230301') (0.000 * 1.000 * 1.000)
```

(주의) 티베로는 (NOT) EXISTS/IN 서브쿼리 안에서 Index Range Scan 시, Leaf 블록 1개에 속한 Rows를 가져온다. 이는 1개의 Row 만 따로 추출하여 별도 메모리에서 관리하는 부하를 없애기 위해서다. 테이블 블록 액세스 시 Rownum < 1 을 적용하므로 최종적으로 테이블에서 추출되는 Rows 수는 정확하다.

3. EXISTS 서브쿼리

문제3 – 튜닝 2단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 NL 세미조인으로 처리한다.

튜닝내역

1. 고객 테이블의 생일 컬럼에 인덱스를 생성하고 INDEX 힌트를 사용한다.
2. 고객등급표와 NL 조인 시, 고객등급표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 고객등급표 테이블의 크기가 작기 때문에 FULL 과 USE_HASH 힌트로 NL 조인을 Hash 조인으로 변경한다.
3. 최근 7일 이내 모든 주문을 찾기 위해 주문과 NL 조인해서 19248건을 추출한 후, 중복 제거를 해서 6659건을 조회한다. 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 필요한 것은 20살 고객의 최근 7일 이내의 모든 주문이 아닌 주문의 존재 여부이다. 따라서 20살 고객과 고객등급표의 조인 결과는 건수가 작기 때문에 이것을 기준으로 EXISTS 서브쿼리를 사용해서 최근 7일 이내 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 하고, UNNEST 와 NL_SJ 힌트로 EXISTS 서브쿼리를 NL 세미조인으로 처리한다.

SQL

```
CREATE INDEX IX_T_CUST_BRTHDY ON T_CUST ( BRTH_DY );                    --고객
CREATE INDEX IX_T3_ORDER_CUSTID_DY ON T_ORDER3 ( CUST_ID, ORDER_DY );  --주문

SELECT /*+ INDEX(C IX_T_CUST_BRTHDY) FULL(G) USE_HASH(G) */
       C.CUST_NM, C.CUST_ID, G.CUST_GRAD, C.BRTH_DY,
       TRUNC((TO_DATE('20230301','YYYYMMDD') - TO_DATE(C.BRTH_DY,'YYYYMMDD') ) /365) AS AGE
  FROM T_CUST       C,
       T_CUST_GRAD  G
 WHERE C.BRTH_DY >  TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 21 * 365 ) ,'YYYYMMDD')
   AND C.BRTH_DY <= TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 20 * 365 ) ,'YYYYMMDD')
   AND C.CUST_ID = G.CUST_ID(+)
   AND EXISTS ( SELECT /*+ UNNEST  NL_SJ */ 1
                  FROM T_ORDER3  O
                 WHERE O.CUST_ID = C.CUST_ID
                   AND O.ORDER_DY >= TO_CHAR((TO_DATE('20230301','YYYYMMDD') -7),'YYYYMMDD')
                   AND O.ORDER_DY <= TO_CHAR( TO_DATE('20230301','YYYYMMDD'),'YYYYMMDD')
                   AND O.ORDER_TY IS NOT NULL
              )
  ORDER BY C.CUST_NM;

DROP INDEX IX_T_CUST_BRTHDY;
DROP INDEX IX_T3_ORDER_CUSTID_DY;
```

(주의) 티베로는 (NOT) EXISTS/IN 서브쿼리 안에서 Index Range Scan 시, Leaf 블록 1개에 속한 Rows를 가져온다. 이는 1개의 Row 만 따로 추출하여 별도 메모리에서 관리하는 부하를 없애기 위해서다. 테이블 블록 액세스 시 Rownum < 1 을 적용하므로 최종적으로 테이블에서 추출되는 Rows 수는 정확하다.

TRACE

EXISTS를 세미 조인으로 처리

```
| ID| Operation                    | Name                  | Rows    | Elaps. Time  | CR Gets | Starts  |
| 1 |HASH JOIN (LEFT OUTER)        |                       |    6659 |00:00:00.0254 |       0 |       1 |
| 2 | INDEX JOIN (SEMI)            |                       |    6659 |00:00:00.0003 |       0 |       1 |
| 3 |  TABLE ACCESS (ROWID)        | T_CUST                |    7016 |00:00:00.0094 |    4223 |       1 |
| 4 |   INDEX (RANGE SCAN)         | IX_T_CUST_BRTHDY      |    7016 |00:00:00.0001 |      24 |       1 |
| 5 |  TABLE ACCESS (ROWID)        | T_ORDER3              |   19248 |00:00:00.0289 |   19248 |    7016 |
| 6 |   INDEX (RANGE SCAN)         | IX_T3_ORDER_CUSTID_DY |   19248 |00:00:00.0305 |   11637 |    7016 |
| 7 | TABLE ACCESS (FULL)          | T_CUST_GRAD           |   70000 |00:00:00.0071 |    2199 |       1 |

   1 - access: ("C"."CUST_ID" = "G"."CUST_ID") (0.000)
   4 - access: ("C"."BRTH_DY" > '20020306') AND ("C"."BRTH_DY" <= '20030306') (1.000 * 0.321)
   5 - filter: ("O"."ORDER_TY" IS NOT NULL) (1.000)
   6 - access: ("C"."CUST_ID" = "O"."CUST_ID") AND ("O"."ORDER_DY" >= '20230222') AND ("O"."ORDER_DY"
<= '20230301') (1.000 * 0.094 * 1.000)
```

3. EXISTS 서브쿼리

문제3 – 튜닝 3단계	20살 고객의 최근 7일 이내 주문 여부를 확인하기 위해 EXISTS 서브쿼리를 Hash 세미조인으로 처리한다.

튜닝내역

1. 고객 테이블의 생일 컬럼에 인덱스를 생성하고 INDEX 힌트를 사용한다.
2. 고객등급표와 NL 조인 시, 고객등급표의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 고객등급표 테이블의 크기가 작기 때문에 FULL 과 USE_HASH 힌트로 NL 조인을 Hash 조인으로 변경한다.
3. 최근 7일 이내 모든 주문을 찾기 위해 주문과 NL 조인해서 19248건을 추출한 후, 중복 제거를 해서 6659건을 조회한다. 주문의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 필요한 것은 20살 고객의 최근 7일 이내의 모든 주문이 아닌 주문의 존재 여부이다. 따라서 20살 고객과 고객등급표의 조인 결과는 건수가 작기 때문에 이것을 기준으로 EXISTS 서브쿼리를 사용해서 최근 7일 이내 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 하고, UNNEST 와 HASH SJ 힌트로 EXISTS 서브쿼리를 Hash 세미조인으로 처리한다.

SQL

```
CREATE INDEX IX_T_CUST_BRTHDY ON T_CUST ( BRTH_DY );                    --고객
CREATE INDEX IX_T3_ORDER_DY ON T_ORDER3 ( ORDER_DY );                   --주문

SELECT /*+ INDEX(C IX_T_CUST_BRTHDY) FULL(G) USE_HASH(G) */
       C.CUST_NM, C.CUST_ID, G.CUST_GRAD, C.BRTH_DY,
       TRUNC((TO_DATE('20230301','YYYYMMDD') - TO_DATE(C.BRTH_DY,'YYYYMMDD') ) /365) AS AGE
  FROM T_CUST       C,
       T_CUST_GRAD  G
 WHERE C.BRTH_DY >  TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 21 * 365 ) ,'YYYYMMDD')
   AND C.BRTH_DY <= TO_CHAR(TO_DATE('20230301','YYYYMMDD') - ( 20 * 365 ) ,'YYYYMMDD')
   AND C.CUST_ID = G.CUST_ID(+)
   AND EXISTS ( SELECT /*+ UNNEST  HASH_SJ  INDEX(O IX_T3_ORDER_DY) */ 1
                  FROM T_ORDER3  O
                 WHERE O.CUST_ID = C.CUST_ID
                   AND O.ORDER_DY >= TO_CHAR((TO_DATE('20230301','YYYYMMDD') -7),'YYYYMMDD')
                 AND O.ORDER_DY <= TO_CHAR( TO_DATE('20230301','YYYYMMDD'),'YYYYMMDD')
                 AND O.ORDER_TY IS NOT NULL
              )
  ORDER BY C.CUST_NM;

DROP INDEX IX_T_CUST_BRTHDY;
DROP INDEX IX_T3_ORDER_DY;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID| Operation                | Name              | Rows  | Elaps. Time  | CR Gets  | Starts |
-------------------------------------------------------------------------------------------------
| 1 | ORDER BY (SORT)          |                   | 6659  |00:00:00.0160 |       0  |     1  |
| 2 |  HASH JOIN (LEFT OUTER)  |                   | 6659  |00:00:00.0064 |       0  |     1  |
| 3 |   HASH JOIN (SEMI)       |                   | 6659  |00:00:00.0184 |       0  |     1  |
| 4 |    TABLE ACCESS (ROWID)  | T_CUST            | 7016  |00:00:00.0085 |    4223  |     1  |
| 5 |     INDEX (RANGE SCAN)   | IX_T_CUST_BRTHDY  | 7016  |00:00:00.0001 |      24  |     1  |
| 6 |    TABLE ACCESS (ROWID)  | T_ORDER3          |  193K |00:00:00.0629 |   33835  |     1  |
| 7 |     INDEX (RANGE SCAN)   | IX_T3_ORDER_DY    |  193K |00:00:00.0775 |     552  |     1  |
| 8 |   TABLE ACCESS (FULL)    | T_CUST_GRAD       |70000  |00:00:00.0062 |    2199  |     1  |
-------------------------------------------------------------------------------------------------

   2 - access: ("C"."CUST_ID" = "G"."CUST_ID") (0.000)
   3 - access: ("C"."CUST_ID" = "O"."CUST_ID") (1.000)
   5 - access: ("C"."BRTH_DY" > '20020306') AND ("C"."BRTH_DY" <= '20030306') (1.000 * 0.321)
   6 - filter: ("O"."ORDER_TY" IS NOT NULL) (1.000)
   7 - access: ("O"."ORDER_DY" >= '20230222' AND "O"."ORDER_DY" <= '20230301') (0.094 * 1.000)
```

3. EXISTS 서브쿼리

정리

1. EXISTS 서브쿼리를 처리하는 방법

FILTER	메인 쿼리→EXISTS 서브쿼리 순서으로 조인한다. 이때 메인 쿼리의 1건이 EXISTS 서브쿼리 안에서 1건 이라도 조인에 성공하는 순간 서브쿼리 안에서 조인이 멈추고 메인 쿼리의 다음 건을 가지고 서브쿼리 안에서 조인을 한다.
NL 세미조인	EXISTS 서브쿼리의 괄호가 없어지고 메인 쿼리 테이블과 EXISTS 서브쿼리 테이블이 NL 조인으로 처리된다. 조인이므로 메인 쿼리 테이블과 EXISTS 서브쿼리 테이블 중 어떤 것이라도 먼저 액세스 될 수 있다. 메인 쿼리 테이블에서 1건이 EXISTS 서브쿼리 테이블과 1건 이라도 조인에 성공하는 순간 EXISTS 서브쿼리 테이블과의 조인을 멈추고, 메인 쿼리 테이블의 다음 건을 가지고 EXISTS 서브쿼리 테이블과 조인한다
Hash 세미조인	EXISTS 서브쿼리의 괄호가 없어지고 메인 쿼리 테이블과 EXISTS 서브쿼리 테이블이 Hash 조인으로 처리된다. 조인이므로 메인 쿼리 테이블과 EXISTS 서브쿼리 테이블 중 어떤 것이라도 먼저 액세스 될 수 있다. 메인 쿼리 테이블에서 1건이 EXISTS 서브쿼리 테이블과 1건 이라도 조인에 성공하는 순간 EXISTS 서브쿼리 테이블과의 조인을 멈추고, 메인 쿼리 테이블의 다음 건을 가지고 EXISTS 서브쿼리 테이블과 조인한다

2. 2023/3/1 이후 주문 된 상품에 사용된 부품 정보를 조회하기 위해서, 주문상세에서 2023/3/1 이후 주문된 모든 상품을 상품, 상품부품, 부품과 조인해서 150만 건을 추출한 후 중복 제거를 해서 500건 만 조회한다.
   ➔
   필요한 것은 부품이 사용된 상품의 주문 내역이 아닌 부품의 사용 여부이다.
   따라서 부품은 건수가 작기 때문에 부품을 기준으로 EXISTS 서브쿼리를 사용해서 2023/3/1 이후 주문된 상품에서 해당 부품의 사용이 확인되는 순간 부품과 조인을 멈추도록 한다.
   이때 EXISTS 서브쿼리를 NO_UNNEST 힌트로 FILTER로 처리, UNNEST 와 NL_SJ 힌트로 NL 세미조인으로 처리, UNNEST 와 HASH_SJ 힌트로 Hash 세미조인으로 처리한다.
3. 구매확정을 한 번도 하지 않고, 고객유형이 A, B, C 가 아닌 고객의 정보를 조회하기 위해서, 메인 쿼리의 D 고객유형을 가진 고객을 서브쿼리에서 구매확정(5) 한 주문 건수를 구해서 1보다 작은지 확인한다.
   ➔
   필요한 것은 구매 확정(5) 주문 건수가 아닌 구매확정(5) 주문의 존재 여부이고 고객 정보다.
   따라서 D 고객유형(741건)은 건수가 작기 때문에 고객을 기준으로 NOT EXISTS 서브쿼리를 사용해서 주문에서 해당 고객의 구매확정(5) 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 한다.
   이때 EXISTS 서브쿼리를 NO_UNNEST 힌트로 FILTER로 처리, UNNEST 와 NL_AJ 힌트로 NL 세미조인으로 처리, UNNEST 와 HASH_AJ 힌트로 Hash 세미조인으로 처리한다.
4. 최근 7일 이내 주문 한 20살 고객의 고객 정보와 고객 등급을 조회하기 위해서, 20살 고객과 최근 7일 주문을 모두 조인해서 19248건을 추출한 후, 중복 제거를 해서 6659건을 조회한다.
   ➔
   필요한 것은 20살 고객의 최근 7일 이내의 모든 주문이 아닌 주문의 존재 여부와 고객 정보다.
   따라서 20살 고객과 고객등급표의 조인 결과는 건수가 작기 때문에 이것을 기준으로 EXISTS 서브쿼리를 사용해서 최근 7일 이내 주문의 존재가 확인되는 순간 주문과 조인을 멈추도록 한다.
   이때 EXISTS 서브쿼리를 NO_UNNEST 힌트로 FILTER로 처리, UNNEST 와 NL_SJ 힌트로 NL 세미조인으로 처리, UNNEST 와 HASH_SJ 힌트로 Hash 세미조인으로 처리한다.

실기

4.쿼리 분리

4. 쿼리 분리

문제1	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 사원아이디와 거래일시(내림차순)로 조회한다.
분석	동적 쿼리가 아니므로 사원아이디(선택 조건)에 LIKE %를 사용했다. 선택 조건의 값 입력 여부와 상관 없이 하나의 고정된 실행계획이 생성된다.
튜닝 1단계	선택 조건의 값 입력 여부에 따라 인덱스를 3개 생성해도, 1개 인덱스 만 사용된다.
튜닝 2단계	Union All + Order By로 변경한다.
문제2	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 거래일시(내림차순)으로 조회한다.
분석	Union All 의 각 쿼리 블록 결과를 합친 후 Order By 컬럼으로 정렬한다.
튜닝 1단계	기존 Order By를 제거하고, Union All 의 각 쿼리 블록에 Order By를 추가하면 인덱스를 역순으로 읽으면서 검색 조건과 Order By를 처리한다.
문제3	서로 다른 테이블에 선택/필수 조건이 있는 Static 쿼리를 조회한다.
분석	NL 조인으로 풀릴 경우, 필수 조건(주문일시)이 있는 주문을 먼저 액세스하여 주문→고객으로 NL 조인 하는데, 이때 고객으로 액세스하는 횟수가 많아지면 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 량이 많아 성능 저하가 발생한다.
튜닝 1단계	주문(필수)→고객(선택)으로 NL 조인 시, 변별력이 좋은 선택 조건의 값을 테이블이 아닌 조인 인덱스(필요시 Covered 인덱스)에서 확인한다.
튜닝 2단계	Union All 로 변경한다.

실기

4.쿼리 분리 – 문제1

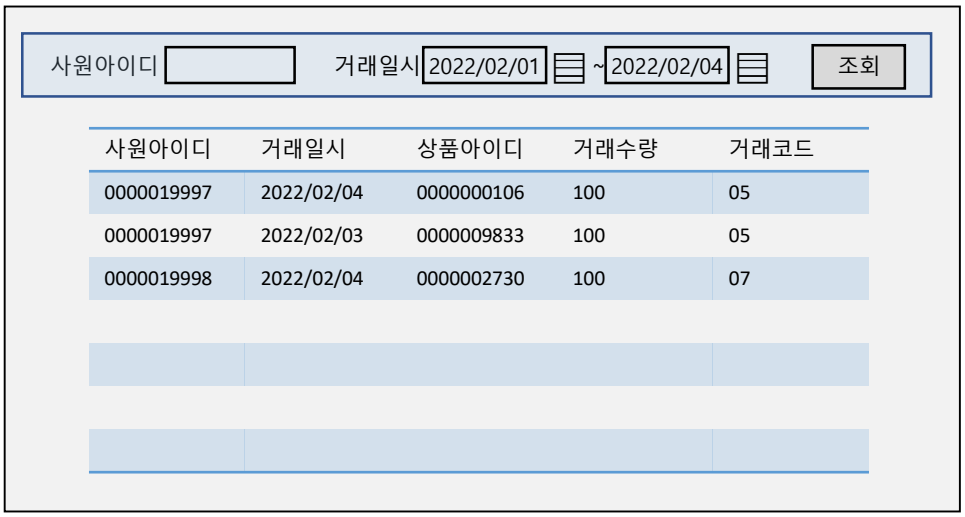
동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 사원아이디와 거래일시(내림차순)로 조회한다.

4. 쿼리 분리

문제1	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 사원아이디와 거래일시(내림차순)로 조회한다.

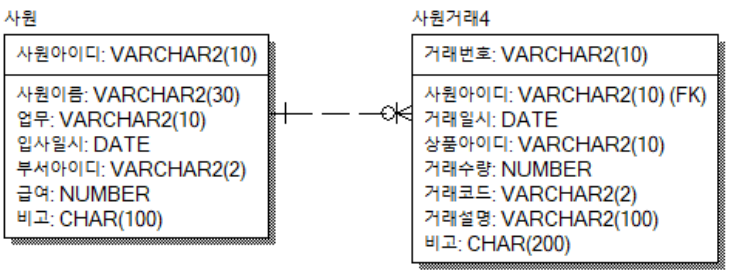
문제설명

- A씨는 사내 복지몰의 사원거래를 담당하고 있다. A씨는 검색 조건으로 사원아이디와 거래일시를, 정렬 조건으로 사원아이디와 거래일시(내림차순)로 조회하는 화면을 사용한다.
- 화면에서 사원아이디는 선택 조건이고, 거래일시는 필수 조건으로 최대 7일까지 입력할 수 있다.
- 동적 쿼리를 지원하지 않으며, 페이지 처리 없이 전체를 조회한다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.



사원아이디	거래일시	상품아이디	거래수량	거래코드
0000019997	2022/02/04	0000000106	100	05
0000019997	2022/02/03	0000009833	100	05
0000019998	2022/02/04	0000002730	100	07

ERD



- 사원 테이블은 2만 명 사원에 대한 정보를 저장한다. 사원은 업무(MANAGER, CLERK, SALESMAN, DEVELOPER, DESIGNER, PROGRAMMER, DBA)와 부서아이디(01 ~ 07)를 골고루 가진다. 입사일시은 Date Type으로 2022/1 ~ 2023/12 사이 값을 가진다.
- 사원거래 테이블은 월 30만 건, 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 사원거래 정보를 저장한다. 거래일시는 Date Type이다. 테이블은 사원아이디(2만 명), 상품아이디(1만 개), 거래코드(01 ~ 10)를 골고루 가진다.

4. 쿼리 분리

문제1	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 사원아이디와 거래일시(내림차순)로 조회한다.

SQL

```
VARIABLE EMP_ID VARCHAR2(10);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);


SELECT /*+  FULL(E) */
       EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4   E                -- 월 30만 건
 WHERE EMP_ID LIKE :EMP_ID || '%'  -- 선택 조건의 값이 입력되지 않은 경우를 위해 LIKE % 사용
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')
                 AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
 ORDER BY EMP_ID, TX_DT DESC;
```

TRACE

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

```
-------------------------------------------------------------------------------------------------
| ID  | Operation             | Name      | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)       |           |   40000  |00:00:00.0356 |       0  |       1  |
|   2 |  TABLE ACCESS (FULL)  | T_EMP_TX4 |   40000  |00:00:03.5632 |    135K  |       1  |
-------------------------------------------------------------------------------------------------
Predicate Information
-------------------------------------------------------------------------------------------------
   2 - filter: ("E"."TX_DT" <= TO_DATE(CONCAT(:2,'235959'),'YYYYMMDDHH24MISS')) AND
("E"."EMP_ID" LIKE CONCAT(:0,'%')) AND ("E"."TX_DT" >= TO_DATE(:1,'YYYYMMDD')) (0.102 * 1.000 *
1.000)


exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

```
-------------------------------------------------------------------------------------------------
| ID  | Operation             | Name      | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)       |           |       4  |00:00:00.0000 |       0  |       1  |
|   2 |  TABLE ACCESS (FULL)  | T_EMP_TX4 |       4  |00:00:00.5188 |    135K  |       1  |
-------------------------------------------------------------------------------------------------
```

분석

① 동적 쿼리가 아니므로 선택 조건인 사원아이디에 LIKE %를 사용했다. 선택 조건의 값 입력 여부와 상관 없이 고정된 하나의 실행계획이 생성된다. 고정된 하나의 실행계획은 선택 조건의 값이 입력되지 않은 경우와 선택 조건의 값이 입력된 경우 모두를 최적화할 수 없다.

② 사원거래 테이블을 Full Scan 해서, 선택 조건의 값이 입력되지 않은 경우 4일에 해당하는 4만 건을 찾고, 선택 조건의 값이 입력되면 특정 사원아이디와 4일에 해당하는 4건을 찾는다.

③ 찾은 4만 건 또는 4건을 사원아이디와 거래일시(내림차순)로 정렬해서 추출한다.

4. 쿼리 분리

문제1 – 튜닝 1단계	선택 조건의 값 입력 여부에 따라 인덱스를 3개 생성해도, 1개 인덱스 만 사용된다.

튜닝내역

1. 선택 조건의 값 입력 여부에 따라 적절한 인덱스가 사용되도록 여러 인덱스를 생성한다.

   ① 사원거래에 선택 조건(사원아이디) + 필수 조건(거래일시)으로 신규 인덱스를 생성한다.

   ② 사원거래에 필수 조건(거래일시)으로 신규 인덱스를 생성한다.

   ③ 사원거래에 필수 조건(거래일시) + 선택 조건(사원아이디)으로 신규 인덱스를 생성한다.

그러나 선택 조건의 값 입력 여부와 상관 없이 고정된 하나의 실행계획 만 생성된다. 고정된 하나의 실행계획은 ①, ②, ③번 인덱스 중 하나의 인덱스 만 사용하므로 다른 인덱스는 필요 없다.

➔ ①번 인덱스를 사용하는 실행계획이 생성되면, 선택 조건(사원아이디)의 값이 입력되지 않으면 인덱스 선두 컬럼이 조건에 사용되지 않아 ①번 인덱스를 사용하지 못하거나 인덱스 블록을 랜덤 엑세스하는 량이 많아진다. 따라서 ①번 인덱스를 생성하면 안된다.

➔ ②번 인덱스를 사용하는 실행계획이 생성되면, 선택 조건(사원아이디)의 값 입력 여부와 상관없이 필수 조건(거래일시)만으로 인덱스를 액세스하기 때문에, 만약 변별력이 좋은 선택 조건(사원아이디)이 입력 되어도 필수(거래일시) 조건 만 인덱스를 사용한다. 즉 사원아이디는 테이블에서 확인하기 때문에 테이블 블록을 랜덤 액세스하는 양이 많아진다.

➔ ③번 인덱스를 사용하는 실행계획이 생성되면, 선택 조건(사원아이디)이 입력되지 않아도 필수 조건(거래일시)이 인덱스 선두 컬럼이므로 인덱스 사용에 문제가 없다. 그리고 변별력이 좋은 선택 조건(사원아이디)까지 입력되면 필수 조건(거래일시)+선택 조건(사원아이디)으로 인덱스를 액세스하여 인덱스에서 찾은 건수가 작기 때문에 테이블 블록을 랜덤 액세스하는 양이 작다.

② 또는 ③번 인덱스를 사용하는 실행계획에서 필수 조건(거래일시) 만 입력되었을 때, 테이블 블록을 랜덤 액세스하는 양이 ②번 인덱스(1488 블록) 보다 ③번 인덱스(2964 블록)가 더 많다. 그 이유는 ②번 인덱스의 Clustering_Factor(0.0399) 보다 ③번 인덱스의 Clustering_Factor(0.9971)가 나쁘기 때문이다.

```
INDEX_NAME            CLUSTERING_FACTOR  BLEVEL ORACLE_CF
--------------------- ----------------- ------- ---------
IX_T_EMP_TX4_EMPID_DT         .999287051       2   3613053  -- ①
IX_T_EMP_TX4_DT               .039946345       2    155035  -- ② CF가 좋다
IX_T_EMP_TX4_DT_EMPID         .997192764       2   3606823  -- ③ CF가 나쁘다
```

SQL

```
CREATE INDEX IX_T_EMP_TX4_EMPID_DT ON T_EMP_TX4 ( EMP_ID, TX_DT ); -- ①
CREATE INDEX IX_T_EMP_TX4_DT       ON T_EMP_TX4 ( TX_DT );         -- ② CF가 좋다.
CREATE INDEX IX_T_EMP_TX4_DT_EMPID ON T_EMP_TX4 ( TX_DT, EMP_ID ); -- ③ CF가 나쁘다.

VARIABLE EMP_ID VARCHAR2(10);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);

SELECT /*+   인덱스 힌트   */
       EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4   E                -- 월 30만 건
 WHERE EMP_ID LIKE :EMP_ID || '%'   -- 선택 조건 값이 입력되지 않은 경우를 위해 LIKE % 사용
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD') AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
 ORDER BY EMP_ID, TX_DT DESC;

DROP INDEX IX_T_EMP_TX4_EMPID_DT;
DROP INDEX IX_T_EMP_TX4_DT;
DROP INDEX IX_T_EMP_TX4_DT_EMPID;
```

4. 쿼리 분리

문제1 – 튜닝 1단계	선택 조건의 값 입력 여부에 따라 인덱스를 3개 생성해도, 1개 인덱스 만 사용된다.

TRACE

②번 인덱스 사용

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

```
-----------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name           | Rows     | Elaps. Time  | CR Gets  | Starts    |
-----------------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)              |                |   40000  |00:00:00.0278 |        0 |         1 |
|   2 |  TABLE ACCESS (ROWID)        | T_EMP_TX4      |   40000  |00:00:00.0103 |     1488 |         1 |
|   3 |   INDEX (RANGE SCAN)         | IX_T_EMP_TX4_DT |  40000  |00:00:00.0287 |      118 |         1 |
-----------------------------------------------------------------------------------------------------------

3 - access: ("E"."TX_DT" >= TO_DATE(:0,'YYYYMMDD')) AND ("E"."TX_DT" <=
TO_DATE(CONCAT(:1,'235959'),'YYYYMMDDHH24MISS')) (1.000 * 0.102)
```

```
exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

```
-----------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name           | Rows     | Elaps. Time  | CR Gets  | Starts    |
-----------------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)              |                |       4  |00:00:00.0000 |        0 |         1 |
|   2 |  TABLE ACCESS (ROWID)        | T_EMP_TX4      |       4  |00:00:00.0184 |     1488 |         1 |
|   3 |   INDEX (RANGE SCAN)         | IX_T_EMP_TX4_DT |  40000  |00:00:00.0007 |      118 |         1 |
-----------------------------------------------------------------------------------------------------------
```

필수 조건(거래일시) 만으로 인덱스에서 추출한 4만건이 테이블 블록을 랜덤 액세스하는 량이 ②번 인덱스보다 ③번 인덱스가 더 많다. 그 이유는 ②번 인덱스의 Clustering Factor(0.0399) 보다 ③번 인덱스의 CF(0.9971)가 나쁘기 때문이다.

③번 인덱스 사용

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

```
-----------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name                 | Rows     | Elaps. Time  | CR Gets  | Starts    |
-----------------------------------------------------------------------------------------------------------
| 1 | ORDER BY (SORT)              |                      |  40000   |00:00:00.0220 |        0 |         1 |
| 2 |  TABLE ACCESS (ROWID)        | T_EMP_TX4            |  40000   |00:00:00.0228 |     2964 |         1 |
| 3 |   FILTER                     |                      |  40000   |00:00:00.0032 |        0 |         1 |
| 4 |    INDEX (RANGE SCAN)        | IX_T_EMP_TX4_DT_EMPID |  40000  |00:00:00.0007 |      181 |         1 |
-----------------------------------------------------------------------------------------------------------

3 - filter: ("E"."EMP_ID" LIKE CONCAT(:0,'%')) (0.031)
   4 - access: ("E"."TX_DT" >= TO_DATE(:1,'YYYYMMDD')) AND ("E"."EMP_ID" LIKE CONCAT(:0,'%')) AND
("E"."TX_DT" <= TO_DATE(CONCAT(:2,'235959'),'YYYYMMDDHH24MISS')) (1.000 * 0.031 * 1.000)
```

```
exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

```
-----------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name                 | Rows     | Elaps. Time  | CR Gets  | Starts    |
-----------------------------------------------------------------------------------------------------------
| 1 | ORDER BY (SORT)              |                      |       4  |00:00:00.0000 |        0 |         1 |
| 2 |  TABLE ACCESS (ROWID)        | T_EMP_TX4            |       4  |00:00:00.0000 |        4 |         1 |
| 3 |   FILTER                     |                      |       4  |00:00:00.0028 |        0 |         1 |
| 4 |    INDEX (RANGE SCAN)        | IX_T_EMP_TX4_DT_EMPID |  40000  |00:00:00.0442 |      181 |         1 |
-----------------------------------------------------------------------------------------------------------
```

4. 쿼리 분리

문제1 – 튜닝 1단계	선택 조건의 값 입력 여부에 따라 인덱스를 3개 생성해도, 1개 인덱스 만 사용된다.

참고

- 티베로의 Clustering Factor는 인덱스 전체를 순서대로 읽은 후 ROWID를 가지고 다시 테이블 블록을 읽을 때, 다른 테이블 블록을 읽을 확률(0 ~ 1)이다. 즉 값이 클수록 다른 테이블 블록을 읽게 된다. "티베로의 Clustering Factor * 전제 Row개수" 는 오라클의 Cluster Factor와 같다.

```
SELECT INDEX_NAME,
       CLUSTERING_FACTOR,                          -- 티베로의 Clustering Factor
       BLEVEL,
       ROUND(CLUSTERING_FACTOR * NUM_ROWS)  ORACLE_CF  -- 오라클의 Clustering Factor
  FROM USER_INDEXES
 WHERE TABLE_NAME ='T_EMP_TX4';
```

INDEX_NAME	CLUSTERING_FACTOR	BLEVEL	ORACLE_CF
IX_T_EMP_TX4_DT	.039946345	2	155035
IX_T_EMP_TX4_DT_EMPID	.997192764	2	3606823
IX_T_EMP_TX4_EMPID_DT	.999287051	2	3613053
PK_T_EMP_TX4	.040006002	2	150149

4. 쿼리 분리

문제1 – 튜닝 2단계	Union All + Order By로 변경한다

튜닝내역

1. 선택 조건의 값 입력 여부와 상관없이 고정된 하나의 실행계획이 생성되어, 고정된 하나의 실행계획은 선택 조건의 값이 입력되지 않은 경우와 선택 조건의 값이 입력된 경우 모두를 최적화할 수 없다. 따라서 Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 선택 조건의 값 입력 여부에 따른 최적화한 실행계획을 생성한다.

   ① 사원거래에 선택 조건(사원아이디)의 값이 입력되지 않은 경우 사용할 인덱스를 생성한다.

   ② 사원거래에 선택 조건(사원아이디)의 값이 입력된 경우 사용할 인덱스를 생성한다.

   ③ Union All 을 사용하여 선택 조건(사원아이디)의 값이 입력되지 않은 쿼리 블록과 입력된 쿼리 블럭으로 분리한다.

   ④ Union All 의 각 쿼리 블록 결과를 합친 후 Order By 컬럼으로 정렬한다.

SQL

```
CREATE INDEX IX_T_EMP_TX4_DT       ON T_EMP_TX4 ( TX_DT );          -- ①
CREATE INDEX IX_T_EMP_TX4_EMPID_DT ON T_EMP_TX4 ( EMP_ID, TX_DT );  -- ②

VARIABLE EMP_ID VARCHAR2(10);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);


-- 선택 조건의 값이 입력되지 않은 경우,
SELECT /*+ INDEX(A IX_T_EMP_TX4_DT) */
       EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4  A   -- 월 30만 건
 WHERE :EMP_ID IS NULL
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD' )
                 AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')

-- 선택 조건의 값이 입력된 경우,
UNION ALL
SELECT /*+ INDEX(B IX_T_EMP_TX4_EMPID_DT) */
       EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4  B   -- 월 30만 건
 WHERE :EMP_ID IS NOT NULL
   AND EMP_ID = :EMP_ID
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')
                 AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
 ORDER BY EMP_ID, TX_DT DESC;


DROP INDEX IX_T_EMP_TX4_DT;
DROP INDEX IX_T_EMP_TX4_EMPID_DT;
```

4. 쿼리 분리

문제1 – 튜닝 2단계	Union All + Order By로 변경한다.

TRACE

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	ORDER BY (SORT)		40000	00:00:00.0266	0	1
2	Union All		40000	00:00:00.0000	0	1
3	FILTER		40000	00:00:00.0000	0	1
4	TABLE ACCESS (ROWID)	T_EMP_TX4	40000	00:00:00.0102	1488	1
5	INDEX (RANGE SCAN)	IX_T_EMP_TX4_DT	40000	00:00:00.0166	118	1
6	FILTER		0	00:00:00.0000	0	1
7	TABLE ACCESS (ROWID)	T_EMP_TX4	0	00:00:00.0000	0	0
8	INDEX (RANGE SCAN)	IX_T_EMP_TX4_EMPID_DT	0	00:00:00.0000	0	0

선택 조건의 값이 입력된 경우와 입력되지 않은 2개의 쿼리 블록으로 실행계획이 만들어진다.

```
   3 - filter: (:0 IS NULL) (1.000)
   5 - access: ("A"."TX_DT" >= TO_DATE(:1,'YYYYMMDD')) AND ("A"."TX_DT" <=
TO_DATE(CONCAT(:2,'235959'),'YYYYMMDDHH24MISS')) (1.000 * 0.102)
   6 - filter: (:3 IS NOT NULL) (1.000)
   8 - access: ("B"."EMP_ID" = :4) AND ("B"."TX_DT" >= TO_DATE(:5,'YYYYMMDD')) AND ("B"."TX_DT" <=
TO_DATE(CONCAT(:6,'235959'),'YYYYMMDDHH24MISS')) (0.001 * 1.000 * 1.000)
```

```
exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	ORDER BY (SORT)		4	00:00:00.0001	0	1
2	Union All		4	00:00:00.0000	0	1
3	FILTER		0	00:00:00.0000	0	1
4	TABLE ACCESS (ROWID)	T_EMP_TX4	0	00:00:00.0000	0	0
5	INDEX (RANGE SCAN)	IX_T_EMP_TX4_DT	0	00:00:00.0000	0	0
6	FILTER		4	00:00:00.0000	0	1
7	TABLE ACCESS (ROWID)	T_EMP_TX4	4	00:00:00.0001	4	1
8	INDEX (RANGE SCAN)	IX_T_EMP_TX4_EMPID_DT	4	00:00:00.0000	3	1

실기

4.쿼리 분리 – 문제2

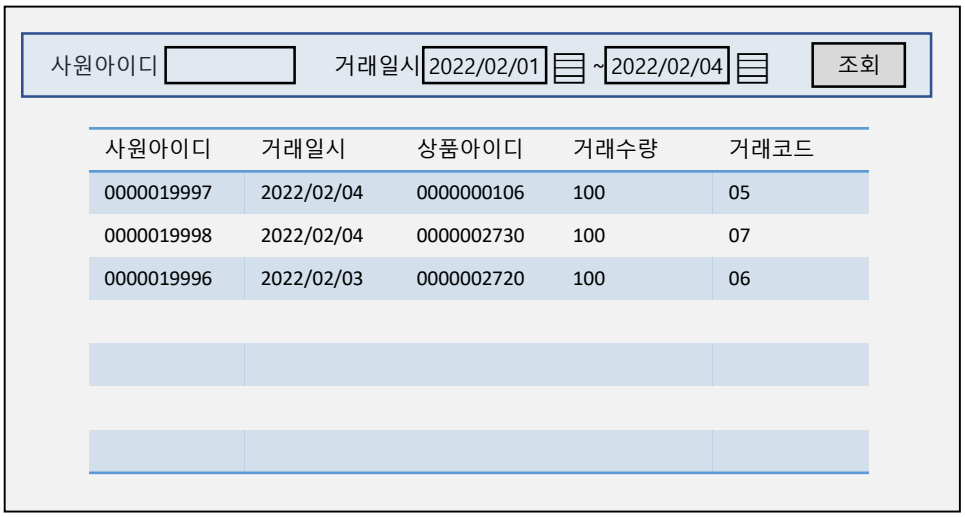
동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 거래일시(내림차순)으로 조회한다.

4. 쿼리 분리

문제2	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 거래일시(내림차순)으로 조회한다.

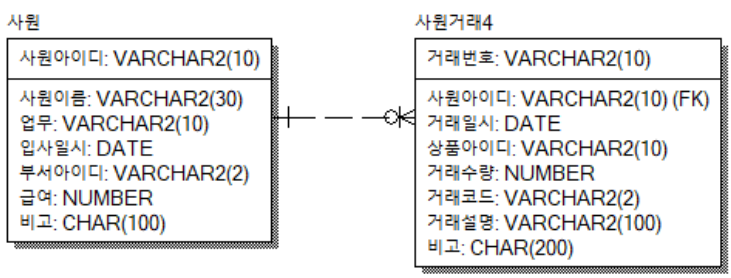
문제설명

- A씨는 사내 복지몰의 사원거래를 담당하고 있다. A씨는 검색 조건으로 사원아이디와 거래일시를, 정렬 조건으로 거래일시 내림차순으로 조회하는 화면을 사용한다.
- 화면에서 사원아이디는 선택 조건이고, 거래일시는 필수 조건으로 최대 7일까지 입력할 수 있다.
- 동적 쿼리를 지원하지 않으며, 페이지 처리 없이 전체를 조회한다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.



사원아이디	거래일시	상품아이디	거래수량	거래코드
0000019997	2022/02/04	0000000106	100	05
0000019998	2022/02/04	0000002730	100	07
0000019996	2022/02/03	0000002720	100	06

ERD



- 사원 테이블은 2만 명 사원에 대한 정보를 저장한다. 사원은 업무(MANAGER, CLERK, SALESMAN, DEVELOPER, DESIGNER, PROGRAMMER, DBA)와 부서아이디(01 ~ 07)를 골고루 가진다. 입사일시은 Date Type으로 2022/1 ~ 2023/12 사이 값을 가진다.
- 사원거래 테이블은 월 30만 건, 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 사원거래 정보를 저장한다. 거래일시는 Date Type이다. 테이블은 사원아이디(2만 명), 상품아이디(1만 개), 거래코드(01 ~ 10)를 골고루 가진다.

4. 쿼리 분리

문제2	동일 테이블에 선택/필수 조건이 있는 Static 쿼리를 거래일시(내림차순)으로 조회한다.

SQL

```
CREATE INDEX IX_T_EMP_TX4_DT       ON T_EMP_TX4 ( TX_DT );
CREATE INDEX IX_T_EMP_TX4_EMPID_DT ON T_EMP_TX4 ( EMP_ID, TX_DT );

VARIABLE EMP_ID VARCHAR2(10);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);

-- 선택 조건의 값이 입력되지 않은 경우,
SELECT EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4  A    -- 월 30만 건
 WHERE :EMP_ID IS NULL
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD') AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
-- 선택 조건의 값이 입력된 경우,
UNION ALL
SELECT EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
  FROM T_EMP_TX4  B    -- 월 30만 건
 WHERE :EMP_ID IS NOT NULL
   AND EMP_ID = :EMP_ID
   AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD') AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
ORDER BY TX_DT DESC;

DROP INDEX IX_T_EMP_TX4_DT;
DROP INDEX IX_T_EMP_TX4_EMPID_DT;
```

TRACE

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

정렬 작업이 꼭 필요한가?

```
| ID | Operation                    | Name                | Rows | Elaps. Time | CR Gets | Starts |
|  1 | ORDER BY (SORT)              |                     |40000 |00:00:00.0266|       0 |      1 |
|  2 |  Union All                   |                     |40000 |00:00:00.0000|       0 |      1 |
|  3 |   FILTER                     |                     |40000 |00:00:00.0000|       0 |      1 |
|  4 |    TABLE ACCESS (ROWID)      | T_EMP_TX4           |40000 |00:00:00.0102|    1488 |      1 |
|  5 |     INDEX (RANGE SCAN)       | IX_T_EMP_TX4_DT     |40000 |00:00:00.0166|     118 |      1 |
|  6 |   FILTER                     |                     |    0 |00:00:00.0000|       0 |      1 |
|  7 |    TABLE ACCESS (ROWID)      | T_EMP_TX4           |    0 |00:00:00.0000|       0 |      0 |
|  8 |     INDEX (RANGE SCAN)       | IX_T_EMP_TX4_EMPID_DT|    0 |00:00:00.0000|       0 |      0 |
```

```
exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

```
| ID | Operation                    | Name                | Rows | Elaps. Time | CR Gets | Starts |
|  1 | ORDER BY (SORT)              |                     |    4 |00:00:00.0001|       0 |      1 |
|  2 |  Union All                   |                     |    4 |00:00:00.0000|       0 |      1 |
|  3 |   FILTER                     |                     |    0 |00:00:00.0000|       0 |      1 |
|  4 |    TABLE ACCESS (ROWID)      | T_EMP_TX4           |    0 |00:00:00.0000|       0 |      0 |
|  5 |     INDEX (RANGE SCAN)       | IX_T_EMP_TX4_DT     |    0 |00:00:00.0000|       0 |      0 |
|  6 |   FILTER                     |                     |    4 |00:00:00.0000|       0 |      1 |
|  7 |    TABLE ACCESS (ROWID)      | T_EMP_TX4           |    4 |00:00:00.0001|       4 |      1 |
|  8 |     INDEX (RANGE SCAN)       | IX_T_EMP_TX4_EMPID_DT|    4 |00:00:00.0000|       3 |      1 |
```

4. 쿼리 분리

문제2	동적 쿼리 및 페이지 처리가 아니고, 동일 테이블에 선택 조건과 필수 조건이 있는 쿼리를 거래일시 내림차순으로 조회한다.

분석

① Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 선택 조건의 값 입력 여부에 따른 실행계획을 생성한다.

② Union All 의 각 쿼리 블록 결과를 합친 후 Order By 컬럼으로 정렬한다.

4. 쿼리 분리

문제2 – 튜닝 1단계	기존 Order By를 제거하고, Union All 의 각 쿼리 블록에 Order By를 추가하면 인덱스를 역순으로 읽으면서 검색 조건과 Order By를 처리한다.

튜닝내역

1. Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 실행한다. 그리고 ①, ②번 인덱스를 역순으로 읽으면 검색 조건과 Order By까지 처리할 수 있어 정렬 작업이 필요 없다.
   - ① 사원거래에 선택 조건(사원아이디)의 값이 입력되지 않은 경우 사용할 신규 인덱스를 생성한다.
   - ② 사원거래에 선택 조건(사원아이디)의 값이 입력된 경우 사용할 신규 인덱스를 생성한다.
   - ③ Union All 을 사용하여 선택 조건(사원아이디)의 값이 입력되지 않은 쿼리 블록과 입력된 쿼리 블록으로 분리한다.
   - ④ 제일 밖에 있는 기존 Order By를 제거하고, Union All 의 각 쿼리 블록에 Order By를 추가하면 인덱스를 역순으로 읽으면서 검색 조건과 Order By을 처리한다. 기존 Order By를 제거하지 않으면 정렬이 추가로 발생한다.

SQL

```
CREATE INDEX IX_T_EMP_TX4_DT        ON T_EMP_TX4 ( TX_DT );          -- ①
CREATE INDEX IX_T_EMP_TX4_EMPID_DT  ON T_EMP_TX4 ( EMP_ID, TX_DT );  -- ②

VARIABLE EMP_ID VARCHAR2(10);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);

-- 선택 조건의 값이 입력되지 않은 경우,
SELECT   *
 FROM (
        SELECT EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
          FROM T_EMP_TX4  A   -- 월 30만 건
         WHERE :EMP_ID IS NULL
           AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD') AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
        ORDER BY TX_DT DESC    -- 추가, 이것 때문에 IX_T_EMP_TX4_DT 를 역순으로 액세스
      )

-- 선택 조건의 값이 입력된 경우,
UNION ALL
SELECT *
 FROM (
        SELECT EMP_ID, TX_DT, GOODS_ID, TX_QY, TX_CD
          FROM T_EMP_TX4  B   -- 월 30만 건
         WHERE :EMP_ID IS NOT NULL
           AND EMP_ID = :EMP_ID
           AND TX_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD') AND TO_DATE(:END_DT || '235959','YYYYMMDDHH24MISS')
        ORDER BY TX_DT DESC    -- 추가, 이것 때문에 IX_T_EMP_TX4_EMPID_DT를 역순으로 액세스
      )
~~ORDER BY TX_DT DESC~~   -- 기존 Order By 제거
;

DROP INDEX IX_T_EMP_TX4_DT;
DROP INDEX IX_T_EMP_TX4_EMPID_DT;
```

4. 쿼리 분리

문제2 – 튜닝 1단계	Union All의 각 쿼리 블록에 Order By 를 추가해서 인덱스로 검색 조건과 정렬을 같이 처리하고, 기존 Order By를 제거한다.

TRACE

```
exec :EMP_ID := NULL;
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력되지 않은 경우,

```
-------------------------------------------------------------------------------------------------------------------
| ID| Operation                      | Name                | Rows  | Elaps. Time |CR Gets | Starts  |
-------------------------------------------------------------------------------------------------------------------
| 1 | Union All                      |                     | 40000 |00:00:00.0035|      0 |     1   |
| 2 |  FILTER                        |                     | 40000 |00:00:00.0000|      0 |     1   |
| 3 |   TABLE ACCESS (ROWID)         | T_EMP_TX4           | 40000 |00:00:00.0160|   1488 |     1   |
| 4 |    INDEX (RANGE SCAN) DESCENDING| IX_T_EMP_TX4_DT     | 40000 |00:00:00.0006|    118 |     1   |
| 5 |  FILTER                        |                     |     0 |00:00:00.0000|      0 |     1   |
| 6 |   TABLE ACCESS (ROWID)         | T_EMP_TX4           |     0 |00:00:00.0000|      0 |     0   |
| 7 |    INDEX (RANGE SCAN) DESCENDING| IX_T_EMP_TX4_EMPID_DT|     0 |00:00:00.0000|      0 |     0   |
-------------------------------------------------------------------------------------------------------------------

   2 - filter: (:0 IS NULL) (1.000)
   4 - access: ("A"."TX_DT" >= TO_DATE(:1,'YYYYMMDD')) AND ("A"."TX_DT" <=
TO_DATE(CONCAT(:2,'235959'),'YYYYMMDDHH24MISS')) (1.000 * 0.102)
   5 - filter: (:3 IS NOT NULL) (1.000)
   7 - access: ("B"."EMP_ID" = :4) AND ("B"."TX_DT" >= TO_DATE(:5,'YYYYMMDD')) AND ("B"."TX_DT" <=
TO_DATE(CONCAT(:6,'235959'),'YYYYMMDDHH24MISS')) (0.001 * 1.000 * 1.000)
```

인덱스를 역순으로 읽으면서
검색 조건과 정렬을 처리한다.

```
exec :EMP_ID := '0000000001';
exec :START_DT := '20220201';
exec :END_DT := '20220204';
```

선택 조건의 값이 입력된 경우,

```
-------------------------------------------------------------------------------------------------------------------
| ID| Operation                      | Name                | Rows  | Elaps. Time |CR Gets | Starts  |
-------------------------------------------------------------------------------------------------------------------
| 1 | Union All                      |                     |     4 |00:00:00.0000|      0 |     1   |
| 2 |  FILTER                        |                     |     0 |00:00:00.0000|      0 |     1   |
| 3 |   TABLE ACCESS (ROWID)         | T_EMP_TX4           |     0 |00:00:00.0000|      0 |     0   |
| 4 |    INDEX (RANGE SCAN) DESCENDING| IX_T_EMP_TX4_DT     |     0 |00:00:00.0000|      0 |     0   |
| 5 |  FILTER                        |                     |     4 |00:00:00.0000|      0 |     1   |
| 6 |   TABLE ACCESS (ROWID)         | T_EMP_TX4           |     4 |00:00:00.0001|      4 |     1   |
| 7 |    INDEX (RANGE SCAN) DESCENDING| IX_T_EMP_TX4_EMPID_DT|     4 |00:00:00.0000|      3 |     1   |
-------------------------------------------------------------------------------------------------------------------
```

실기

4.쿼리 분리 – 문제3

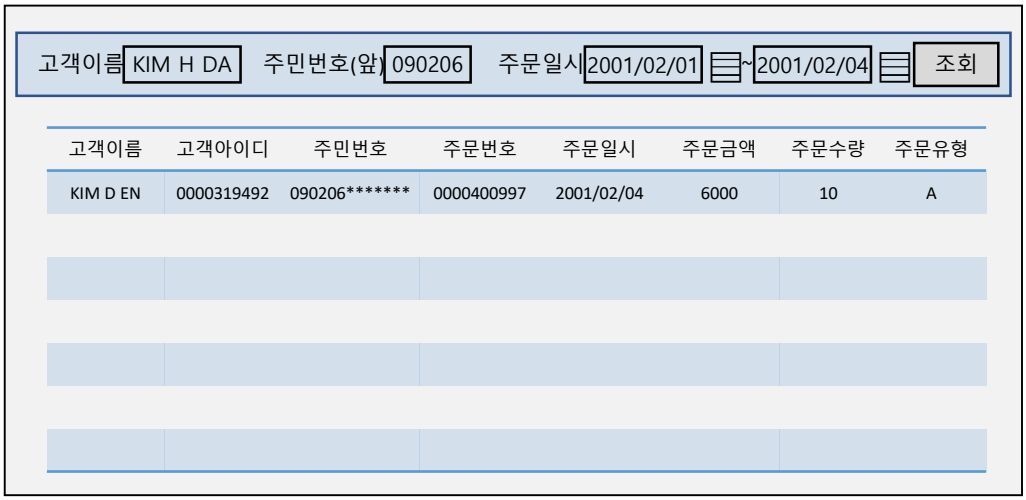
서로 다른 테이블에 선택/필수 조건이 있는 Static 쿼리를 조회한다.

4. 쿼리 분리

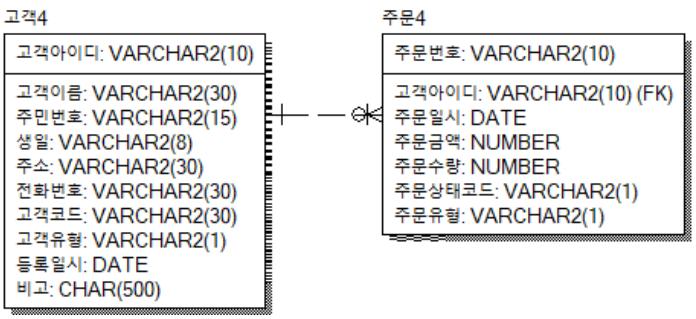
문제3	서로 다른 테이블에 선택/필수 조건이 있는 Static 쿼리를 조회한다.

문제설명

- A 온라인 쇼핑몰의 관리 담당자는 검색 조건으로 고객이름, 주민번호, 주문일시를 입력하여 조회하는 화면을 사용한다.
- 화면에서 고객이름과 주민번호는 선택 조건이고(동시에 입력하거나 입력하지 않는다), 주문일시는 필수 조건으로 최대 7일까지 입력할 수 있다.
- 동적 쿼리를 지원하지 않으며, 페이지 처리 없이 전체를 조회한다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.



ERD



- 고객 테이블은 100만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다.
- 주문 테이블은 월 3만 건, 99개월(2000/1 ~ 2008/3) 동안 총 300만 건 주문정보를 저장한다. 주문자인 고객아이디는 고객 테이블(100만 건)에 저장되어 있으며, 모든 고객이 골고루 주문한다. 주문일시는 Date Type이다. 주문은 모두 주문상태코드(1 ~ 5)를 가지며, 대부분 5 값을 가진다. 주문은 모두 주문유형(A, B, C, D)을 가지며, A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.

4. 쿼리 분리

문제3	서로 다른 테이블에 선택/필수 조건이 있는 Static 쿼리를 조회한다.

SQL

```
CREATE INDEX IX_T_ORDER4_DT       ON T_ORDER4 ( ORDER_DT );
CREATE INDEX IX_T_T_CUST4_JUMINNO ON T_CUST4 ( JUMINNO );

VARIABLE CUST_NM VARCHAR2(30);
VARIABLE JUMINNO VARCHAR2(13);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);


SELECT /*+ LEADING(O C) USE_NL(C) INDEX(O IX_T_ORDER4_DT) */
       C.CUST_NM,
       C.CUST_ID,
       C.JUMINNO,
       O.ORDER_NO,
       O.ORDER_DT,
       O.ORDER_AMT,
       O.ORDER_QY,
       O.ORDER_TY
  FROM T_CUST4     C,
       T_ORDER4    O    -- 월 3만 건
 WHERE C.CUST_NM  LIKE :CUST_NM || '%' -- 선택 조건 값이 입력되지 않은 경우를 위해 LIKE % 사용
   AND C.JUMINNO  LIKE :JUMINNO || '%' -- 인덱스 생성
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')   -- 인덱스 생성
                      AND TO_DATE(:END_DT || ' 235959','YYYYMMDD HH24MISS') ;


DROP INDEX IX_T_ORDER4_DT;
DROP INDEX IX_T_T_CUST4_JUMINNO;
```

TRACE

```
exec :CUST_NM := NULL;
exec :JUMINNO := NULL;
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력되지 않은 경우,

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation                    | Name          | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------
|   1 | INDEX JOIN                   |               |     4000 |00:00:00.0049 |        0 |        1 |
|   2 |  TABLE ACCESS (ROWID)        | T_ORDER4      |     4000 |00:00:00.0010 |       26 |        1 |
|   3 |   INDEX (RANGE SCAN)         | IX_T_ORDER4_DT|     4000 |00:00:00.0027 |       17 |        1 |
|   4 |  TABLE ACCESS (ROWID)        | T_CUST4       |     4000 |00:00:00.7202 |     4000 |     4000 |
|   5 |   INDEX (UNIQUE SCAN)        | PK_T_CUST4    |     4000 |00:00:00.4166 |    12000 |     4000 |
-------------------------------------------------------------------------------------------------------
   3 - access: ("O"."ORDER_DT" >= TO_DATE(:2,'YYYYMMDD')) AND ("O"."ORDER_DT" <= TO_DATE(CONCAT(:3,'
235959'),'YYYYMMDD HH24MISS')) (1.000 * 0.131)
   4 - filter: ("C"."CUST_NM" LIKE CONCAT(:0,'%')) AND ("C"."JUMINNO" LIKE CONCAT(:1,'%')) (1.000 *
1.000)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
```

4. 쿼리 분리

문제3	서로 다른 테이블에 선택/필수 조건이 있는 Static 쿼리를 조회한다.

TRACE

```
exec :CUST_NM := 'KIM H DA'
exec :JUMINNO := '090206';
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력된 경우,

```
-------------------------------------------------------------------------------------------------------------
| ID  | Operation                | Name            | Rows     | Elaps. Time  | CR Gets  | Starts    |
-------------------------------------------------------------------------------------------------------------
|   1 | INDEX JOIN               |                 |        1 |00:00:00.0000 |        0 |         1 |①
|   2 |  TABLE ACCESS (ROWID)    | T_ORDER4        |     4000 |00:00:00.0011 |       26 |         1 |
|   3 |   INDEX (RANGE SCAN)     | IX_T_ORDER4_DT  |     4000 |00:00:00.0001 |       17 |         1 |
|   4 | TABLE ACCESS (ROWID)     | T_CUST4         |        1 |00:00:00.0054 |     4000 |      4000 |③
|   5 |  INDEX (UNIQUE SCAN)     | PK_T_CUST4      |     4000 |00:00:00.0084 |    12000 |      4000 |②
-------------------------------------------------------------------------------------------------------------
   3 - access: ("O"."ORDER_DT" >= TO_DATE(:2,'YYYYMMDD')) AND ("O"."ORDER_DT" <= TO_DATE(CONCAT(:3,'
235959'),'YYYYMMDD HH24MISS')) (1.000 * 0.131)
   4 - filter: ("C"."CUST_NM" LIKE CONCAT(:0,'%')) AND ("C"."JUMINNO" LIKE CONCAT(:1,'%')) (1.000 *
1.000)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
```

변별력이 좋은 선택 조건의 값이 입력되었으나, 테이블 블록을 랜덤 액세스하는 양(4000 블록)은 동일하다. 고정된 실행계획에서는 변별력이 좋은 선택 조건을 사용하는 실행계획이 나오지 않는다.

분석

① 동적 쿼리가 아니므로 선택 조건인 고객이름과 주민번호에 LIKE %를 사용했다. NL 조인으로 풀릴 경우, 필수 조건이 있는 주문을 먼저 액세스해야 하므로 주문의 인덱스로 4일 치에 해당 하는 4000건을 찾아, 주문→고객으로 NL 조인 한다.

② 고객과 NL 조인 시, 고객의 PK 인덱스를 4000번 액세스해서 4000건을 추출한다. 이때 인덱스 블록을 랜덤 액세스하는 양(12000 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 4000건이 테이블 블록을 랜덤 액세스하여, 선택 조건(고객이름, 주민번호)의 값이 입력되지 않은 경우 4000건을 추출하고, 값이 입력된 경우 1건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(4000 블록)이 많아 성능 저하가 발생한다.

4. 쿼리 분리

문제3 – 튜닝 1단계	주문(필수)→고객(선택)으로 NL 조인시, 변별력이 좋은 선택 조건의 값을 테이블이 아닌 조인 인덱스(필요시 Covered 인덱스)에서 확인한다.

튜닝내역

1. NL 조인으로 풀릴 경우, 필수 조건(주문일시)이 있는 주문을 먼저 액세스하여 주문→고객으로 NL 조인하는데, 이때 고객으로 조인 시도 횟수가 많아지면 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 변별력이 좋은 주민번호는 선택 조건이기 때문에 고객을 먼저 액세스하여 고객→주문으로 NL 조인할 수 없다. 따라서 쿼리를 분리하지 않는다면, 주문→고객으로 NL 조인 시, 변별력이 좋은 선택 조건의 값을 고객 테이블이 아닌 조인 인덱스에서 확인하도록 조인 인덱스를 구성한다. 즉 조인 인덱스에서 추출되는 건을 줄이면 고객 테이블 블록을 랜덤 액세스하는 양이 작아진다.
   추가적인 성능 향상을 원한다면 조인 인덱스를 Covered 인덱스로 만들어 고객 테이블 블록을 액세스하지 않도록 한다.

   ① 주문의 필수 조건(주민번호)으로 신규 인덱스를 생성한다.

   ② 고객에 주문→고객으로 NL 조인 시 선택 조건의 값을 같이 확인하는 인덱스를 생성한다.

   ③ 고객에 주문→고객으로 NL 조인 시, 사용할 인덱스를 Covered 인덱스로 생성한다.

➔ ②번 인덱스를 사용하는 실행계획이 생성되면, 선택 조건(주민번호)의 값이 입력되면 조인 인덱스에서 추출되는 건수를 줄여 고객 테이블 블록을 랜덤 액세스하는 양이 작아진다. 그러나 선택 조건(주민번호)의 값이 입력되지 않으면 인덱스에서 추출되는 건수가 많아 고객 테이블 블록을 랜덤 액세스하는 양이 많아진다.

➔ ③번 Covered 인덱스를 사용하는 실행계획이 생성되면, 선택 조건의 값 입력 여부와 상관없이 고객 테이블 블록을 랜덤 액세스하지 않는다.

SQL

```
CREATE INDEX IX_T_ORDER4_DT         ON T_ORDER4 ( ORDER_DT );                  -- ① 필수 조건
CREATE INDEX IX_T_CUST4_ID_JUMIN    ON T_CUST4 ( CUST_ID, JUMINNO );           -- ② 조인 인덱스
CREATE INDEX IX_T_CUST4_ID_JUMIN_NM ON T_CUST4 ( CUST_ID, JUMINNO, CUST_NM );  -- ③ 조인 Covered 인덱스

VARIABLE CUST_NM VARCHAR2(30);
VARIABLE JUMINNO VARCHAR2(13);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);


SELECT /*+ LEADING(O C)  USE_NL(C)  INDEX(O IX_T_ORDER4_DT)   ② 또는 ③인덱스_힌트 */
       C.CUST_NM, C.CUST_ID, C.JUMINNO,
       O.ORDER_NO, O.ORDER_DT, O.ORDER_AMT, O.ORDER_QY, O.ORDER_TY
  FROM T_CUST4     C,
       T_ORDER4    O    -- 일1000 건
 WHERE C.CUST_NM  LIKE :CUST_NM || '%' -- 선택 조건 값이 입력되지 않은 경우를 위해 LIKE % 사용
   AND C.JUMINNO  LIKE :JUMINNO || '%'
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')
                      AND TO_DATE(:END_DT || ' 235959','YYYYMMDD HH24MISS') ;


DROP INDEX IX_T_ORDER4_DT;
DROP INDEX IX_T_CUST4_ID_JUMIN;
DROP INDEX IX_T_CUST4_ID_JUMIN_NM;
```

4. 쿼리 분리

문제3 – 튜닝 1단계	주문→고객으로 NL 조인 시, 변별력이 좋은 선택 조건 값을 테이블이 아닌 조인 인덱스(Covered 인덱스가 제일 좋음)에서 먼저 확인한다.

TRACE

②번 인덱스 사용

```
exec :CUST_NM := NULL;
exec :JUMINNO := NULL;
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력되지 않은 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	INDEX JOIN		4000	00:00:00.0034	0	1
2	TABLE ACCESS (ROWID)	T_ORDER4	4000	00:00:00.0010	26	1
3	INDEX (RANGE SCAN)	IX_T_ORDER4_DT	4000	00:00:00.0001	17	1
4	TABLE ACCESS (ROWID)	T_CUST4	4000	00:00:00.0145	4000	4000
5	INDEX (RANGE SCAN)	IX_T_CUST4_ID_JUMIN	4000	00:00:00.5758	4063	4000

```
   4 - filter: ("C"."CUST_NM" LIKE CONCAT(:0,'%')) (0.000)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") AND ("C"."JUMINNO" LIKE CONCAT(:1,'%')) (0.000 * 1.000)
```

```
exec :CUST_NM := 'KIM H DA'
exec :JUMINNO := '090206';
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력되면 테이블 블록을 랜덤 액세스하는 양이 작다.

선택 조건의 값이 입력된 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	INDEX JOIN		1	00:00:00.0005	0	1
2	TABLE ACCESS (ROWID)	T_ORDER4	4000	00:00:00.0052	26	1
3	INDEX (RANGE SCAN)	IX_T_ORDER4_DT	4000	00:00:00.0003	17	1
4	TABLE ACCESS (ROWID)	T_CUST4	1	00:00:00.0002	2	4000
5	INDEX (RANGE SCAN)	IX_T_CUST4_ID_JUMIN	2	00:00:00.0232	4030	4000

③번 Covered 인덱스 사용

```
exec :CUST_NM := NULL;
exec :JUMINNO := NULL;
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력되지 않은 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	INDEX JOIN		4000	00:00:00.0009	0	1
2	TABLE ACCESS (ROWID)	T_ORDER4	4000	00:00:00.0010	26	1
3	INDEX (RANGE SCAN)	IX_T_ORDER4_DT	4000	00:00:00.0001	17	1
4	FILTER		4000	00:00:00.0002	0	4000
5	INDEX (RANGE SCAN)	IX_T_CUST4_ID_JUMIN_NM	4000	00:00:00.0162	4067	4000

```
   4 - filter: ("C"."CUST_NM" LIKE CONCAT(:0,'%')) (1.000)
   5 - access: ("C"."CUST_ID" = "O"."CUST_ID") AND ("C"."JUMINNO" LIKE CONCAT(:1,'%')) (0.000 * 1.000)
```

```
exec :CUST_NM := 'KIM H DA'
exec :JUMINNO := '090206';
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

Covered Index 는 선택 조건의 값 입력 여부와 상관 없이 테이블 블록을 액세스하지 않는다.

선택 조건의 값이 입력된 경우,

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	INDEX JOIN		1	00:00:00.0031	0	1
2	TABLE ACCESS (ROWID)	T_ORDER4	4000	00:00:00.0009	26	1
3	INDEX (RANGE SCAN)	IX_T_ORDER4_DT	4000	00:00:00.0001	17	1
4	FILTER		1	00:00:00.0001	0	4000
5	INDEX (RANGE SCAN)	IX_T_CUST4_ID_JUMIN_NM	2	00:00:00.5564	4033	4000

4. 쿼리 분리

문제3 – 튜닝 2단계	Union All 로 변경한다.

튜닝내역

1. 선택 조건의 값 입력 여부와 상관없이 고정된 하나의 실행계획이 생성되어, 고정된 하나의 실행계획은 선택 조건의 값이 입력되지 않은 경우와 선택 조건의 값이 입력된 경우 모두를 최적화할 수 없다. 따라서 Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 선택 조건의 값 입력 여부에 따른 최적화한 실행계획을 생성한다.
   - 선택 조건의 값이 입력되지 않으면, 주문→고객으로 NL 조인하는 것이 좋고,
   - 선택 조건의 값이 입력되면, 변별력이 좋은 선택 조건을 가지는 고객을 먼저 액세스하여, 고객→주문으로 NL 조인을 하는 것이 좋다.

   ① 선택 조건의 값이 입력되지 않은 경우 사용할 인덱스를 생성한다.
   고객과 조인 시 사용되는 인덱스는 선택 조건의 값이 없기 때문에 테이블 블록을 랜덤 액세스하는 량을 없애기 위해 Covered 인덱스로 생성한다.

   ② 선택 조건의 값이 입력된 경우, 사용할 인덱스를 생성한다.
   고객에서 변별력이 좋은 선택 조건(주민번호)의 값으로 작은 건수를 추출하고 고객→주문으로 NL 조인을 한다.

   ③ Union All 을 사용하여 선택 조건(주민번호)의 값이 입력되지 않은 쿼리 블록과 입력된 쿼리 블록으로 분리한다.

SQL

```
-- 선택 조건의 값이 입력되지 않은 경우 사용할 인덱스
CREATE INDEX IX_T_ORDER4_DT         ON T_ORDER4 ( ORDER_DT );                  -- ① 테이블 액세스
CREATE INDEX IX_T_CUST4_ID_JUMIN_NM ON T_CUST4 ( CUST_ID, JUMINNO, CUST_NM );  -- ① 조인 인덱스

-- 선택 조건의 값이 입력된 경우 사용할 인덱스
CREATE INDEX IX_T_T_CUST4_JUMINNO   ON T_CUST4 ( JUMINNO );                    -- ② 테이블 액세스
CREATE INDEX IX_T_ORDER4_CUSTID     ON T_ORDER4 ( CUST_ID );                   -- ② 조인 인덱스

VARIABLE CUST_NM VARCHAR2(30);
VARIABLE JUMINNO VARCHAR2(13);
VARIABLE START_DT VARCHAR2(8);
VARIABLE END_DT VARCHAR2(8);

-- 선택 조건의 값이 입력되지 않은 경우
SELECT /*+  LEADING(O C)  USE_NL(C)
            INDEX(O IX_T_ORDER4_DT)  INDEX(C IX_T_CUST4_ID_JUMIN_NM) */
       C.CUST_NM,
       C.CUST_ID,
       C.JUMINNO,
       O.ORDER_NO,
       O.ORDER_DT,
       O.ORDER_AMT,
       O.ORDER_QY,
       O.ORDER_TY
  FROM T_CUST4     C,
       T_ORDER4    O
 WHERE :CUST_NM IS NULL  -- 선택 조건의 값 입력 여부 확인
   AND :JUMINNO IS NULL  -- 선택 조건의 값 입력 여부 확인
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')
                      AND TO_DATE(:END_DT || ' 235959','YYYYMMDD HH24MISS')
```

4. 쿼리 분리

문제3 – 튜닝 2단계	Union All 로 변경한다.

SQL

```
-- 선택 조건의 값이 입력된 경우
Union All
SELECT /*+ LEADING(C2 O2)  USE_NL(O2)
           INDEX(C2 IX_T_T_CUST4_JUMINNO)  INDEX(O2 IX_T_ORDER4_CUSTID) */
       C2.CUST_NM,
       C2.CUST_ID,
       C2.JUMINNO,
       O2.ORDER_NO,
       O2.ORDER_DT,
       O2.ORDER_AMT,
       O2.ORDER_QY,
       O2.ORDER_TY
  FROM T_CUST4      C2,
       T_ORDER4     O2
 WHERE :CUST_NM IS NOT NULL    -- 선택 조건의 값 입력 여부 확인
   AND :JUMINNO IS NOT NULL    -- 선택 조건의 값 입력 여부 확인
   AND C2.CUST_NM  = :CUST_NM
   AND C2.JUMINNO LIKE :JUMINNO ||'%'   -- 주민번호 뒷자리는 암호화
   AND C2.CUST_ID = O2.CUST_ID
 AND O2.ORDER_DT BETWEEN TO_DATE(:START_DT,'YYYYMMDD')
                     AND TO_DATE(:END_DT || ' 235959','YYYYMMDD HH24MISS') ;


DROP INDEX IX_T_ORDER4_DT;
DROP INDEX IX_T_CUST4_ID_JUMIN_NM;
DROP INDEX IX_T_T_CUST4_JUMINNO;
DROP INDEX IX_T_ORDER4_CUSTID;
```

4. 쿼리 분리

문제3 – 튜닝 2단계	Union All 로 변경한다.

```
exec :CUST_NM := NULL;
exec :JUMINNO := NULL;
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력되지 않은 경우, 고객 테이블의 블록을 랜덤 액세스하는 양이 많으므로, 테이블 블록을 랜덤 액세스 하지 않기 위해 Covered Index를 사용한다.

선택 조건의 값이 입력되지 않은 경우,

```
-------------------------------------------------------------------------------------------------------------------
| ID | Operation                | Name                  | Rows    | Elaps. Time  |CR Gets | Starts |
-------------------------------------------------------------------------------------------------------------------
|  1 | Union All                |                       |    4000 |00:00:00.0003 |      0 |      1 |
|  2 |  FILTER                  |                       |    4000 |00:00:00.0004 |      0 |      1 |
|  3 |   INDEX JOIN             |                       |    4000 |00:00:00.0012 |      0 |      1 |
|  4 |    TABLE ACCESS (ROWID)  | T_ORDER4              |    4000 |00:00:00.0013 |     26 |      1 |
|  5 |     INDEX (RANGE SCAN)   | IX_T_ORDER4_DT        |    4000 |00:00:00.0001 |     17 |      1 |
|  6 |    INDEX (RANGE SCAN)    | IX_T_CUST4_ID_JUMIN_NM|    4000 |00:00:00.0117 |   4067 |   4000 |
|  7 |  FILTER                  |                       |       0 |00:00:00.0000 |      0 |      1 |
|  8 |   INDEX JOIN             |                       |       0 |00:00:00.0000 |      0 |      0 |
|  9 |    ORDER BY (SORT)       |                       |       0 |00:00:00.0000 |      0 |      0 |
| 10 |     TABLE ACCESS (ROWID) | T_CUST4               |       0 |00:00:00.0000 |      0 |      0 |
| 11 |      INDEX (RANGE SCAN)  | IX_T_T_CUST4_JUMINNO  |       0 |00:00:00.0000 |      0 |      0 |
| 12 |    TABLE ACCESS (ROWID)  | T_ORDER4              |       0 |00:00:00.0000 |      0 |      0 |
| 13 |     INDEX (RANGE SCAN)   | IX_T_ORDER4_CUSTID    |       0 |00:00:00.0000 |      0 |      0 |
-------------------------------------------------------------------------------------------------------------------
```

```
exec :CUST_NM := 'KIM D EN'
exec :JUMINNO := '090206';
exec :START_DT := '20010201';
exec :END_DT := '20010204';
```

선택 조건의 값이 입력된 경우,

```
-------------------------------------------------------------------------------------------------------------------
| ID | Operation                | Name                  | Rows    | Elaps. Time  |CR Gets | Starts |
-------------------------------------------------------------------------------------------------------------------
|  1 | Union All                |                       |       1 |00:00:00.0000 |      0 |      1 |
|  2 |  FILTER                  |                       |       0 |00:00:00.0000 |      0 |      1 |
|  3 |   INDEX JOIN             |                       |       0 |00:00:00.0000 |      0 |      0 |
|  4 |    TABLE ACCESS (ROWID)  | T_ORDER4              |       0 |00:00:00.0000 |      0 |      0 |
|  5 |     INDEX (RANGE SCAN)   | IX_T_ORDER4_DT        |       0 |00:00:00.0000 |      0 |      0 |
|  6 |    INDEX (RANGE SCAN)    | IX_T_CUST4_ID_JUMIN_NM|       0 |00:00:00.0000 |      0 |      0 |
|  7 |  FILTER                  |                       |       1 |00:00:00.0000 |      0 |      1 |
|  8 |   INDEX JOIN             |                       |       1 |00:00:00.0000 |      0 |      1 |
|  9 |    ORDER BY (SORT)       |                       |       1 |00:00:00.0000 |      0 |      1 |
| 10 |     TABLE ACCESS (ROWID) | T_CUST4               |       1 |00:00:00.0616 |    313 |      1 |
| 11 |      INDEX (RANGE SCAN)  | IX_T_T_CUST4_JUMINNO  |     316 |00:00:00.0085 |      7 |      1 |
| 12 |    TABLE ACCESS (ROWID)  | T_ORDER4              |       1 |00:00:00.0000 |      4 |      1 |
| 13 |     INDEX (RANGE SCAN)   | IX_T_ORDER4_CUSTID    |       4 |00:00:00.0000 |      3 |      1 |
-------------------------------------------------------------------------------------------------------------------

   2 - filter: (:0 IS NULL) AND (:1 IS NULL) (1.000 * 1.000)
   5 - access: ("O"."ORDER_DT" >= TO_DATE(:2,'YYYYMMDD')) AND ("O"."ORDER_DT" <= TO_DATE(CONCAT(:3,'
235959'),'YYYYMMDD HH24MISS')) (1.000 * 0.131)
   6 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.00
   7 - filter: (:4 IS NOT NULL) AND (:5 IS NOT NULL)
  10 - filter: ("C2"."CUST_NM" = :6) (0.001)
  11 - access: ("C2"."JUMINNO" LIKE CONCAT(:7,'%')) (1.000)
  12 - filter: ("O2"."ORDER_DT" <= TO_DATE(CONCAT(:9,' 235959'),'YYYYMMDD HH24MISS')) AND
("O2"."ORDER_DT" >= TO_DATE(:8,'YYYYMMDD')) (0.131 * 1.000)
  13 - access: ("O2"."CUST_ID" = "C2"."CUST_ID") (0.000)
```

선택 조건의 값이 입력되면 변별력 좋은 주민번호를 가진 고객을 먼저 액세스해서 고객→주민으로 NL 조인한다.

4. 쿼리 분리

정리

1. 동일 테이블에 선택 조건과 필수 조건이 있는 쿼리를 사원아이디와 거래일시 내림차순으로 조회 시, 동적 쿼리가 아니므로 선택 조건인 사원아이디에 LIKE %를 사용했다. 그리고 선택 조건의 값 입력 여부에 따라 적절한 인덱스가 사용되도록 여러 인덱스를 생성하였으나, 선택 조건의 값 입력 여부와 상관 없이 하나의 고정된 실행계획이 생성된다. 고정된 하나의 실행계획은 여러 인덱스 중 오직 하나의 인덱스 만 사용한다. 고정된 하나의 실행계획은 선택 조건의 값이 입력되지 않은 경우와 입력된 모든 경우를 최적화 시킬 수 없다.
2. 사원거래 테이블은 거래일시 순서으로 2만 명 사원의 거래 정보가 12개월 동안 저장되고, ②번 인덱스는 거래일시 컬럼으로 구성되고, ③번 인덱스는 거래일시 + 사원번호 컬럼으로 구성된다. 이때 거래일시 조건으로 인덱스를 액세스하여 추출된 건이, 테이블 블록을 랜덤 액세스하는 양은 ②번 인덱스(1488 블록) 보다 ③번 인덱스(2964 블록)가 더 많다. 그 이유는 ②번 인덱스의 Clustering Factor(0.0399) 보다 ③번 인덱스의 Clustering Factor(0.9971)가 나쁘기 때문이다.
3. Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 선택 조건의 값 입력 여부에 따른 최적화한 실행계획을 생성한다.
4. 동일 테이블에 선택 조건과 필수 조건이 있는 쿼리를 거래일시 내림차순으로 조회 시, Union All 을 사용하여 선택 조건의 값 입력 여부에 따라 쿼리 블록을 분리하여 실행한다. 그리고 **①, ②** 인덱스를 역순으로 읽으면 검색 조건과 Order By까지 처리할 수 있어 Order By가 필요 없다.
5. 서로 다른 테이블에 선택 조건(주민번호)과 필수 조건(거래일시)이 있는 쿼리를 조회 시, NL 조인으로 풀릴 경우 필수 조건이 있는 주문을 먼저 액세스해야 하므로 주문(필수 조건)→고객(선택 조건)으로 NL 조인한다. 이때 고객으로 액세스하는 횟수가 많아지면 고객의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.

   1) 주문→고객으로 NL 조인 시, 변별력이 좋은 선택 조건의 값을 고객 테이블이 아닌 조인 인덱스에서 확인하도록 조인 인덱스를 구성한다. 즉 조인 인덱스에서 추출되는 건을 줄이면 고객 테이블 블록을 랜덤 액세스하는 양이 작아진다.
   추가 적인 성능 향상을 원한다면 조인 인덱스를 Covered 인덱스로 만들어 고객 테이블 블록을 액세스하지 않도록 한다.

   2) Union All 을 사용하여 선택 조건(주민번호)의 값 입력 여부에 따라 쿼리 블록을 분리한다.
   - 선택 조건의 값이 입력되지 않으면, 주문→고객으로 NL 조인하는 것이 좋고,
   - 선택 조건의 값이 입력되면, 변별력이 좋은 선택 조건을 가지는 고객을 먼저 액세스하여, 고객→주문으로 NL 조인을 하는 것이 좋다.

실기

5.반복 제거

5. 반복 제거

문제1	가전, 가전외, 냉장고 및 전체의 판매수량 합계를 조회한다.
분석	Union All 을 사용하여 판매내역 테이블을 4번 Full Scan 한다.
튜닝 1단계	판매일시 컬럼 가공을 제거하고, 판매내역을 1번 만 읽고 중간 집합(15만 건)을 만들어 4번 복제한다.
튜닝 2단계	더 작은 중간 집합(3건)을 만들어 3번 복제한다.
문제2	Group By 월,고객유형(함수) 과 Group By 일,고객유형(함수)로 판매수량 합계를 조회한다.
분석	Union All 을 사용하여 판매내역 테이블을 2번 읽으며, 함수 호출(360만 번)이 많아 성능 저하(블록, CPU)가 발생한다.
튜닝 1단계	F_CUST_TYPE 함수 호출을 고객과 조인하는 것으로 변경한다.
튜닝 2단계	Grouping Sets을 사용하여 판매내역을 1번 만 읽는다.
튜닝 3단계	Grouping Sets을 WITH 문으로 변경한다.
문제3	과목 성적이 90점 이상이면, 과목 성적을 고득점 테이블에 입력한다.
튜닝 1단계	행을 열로 바꾸기 위해 중간집합(7451건)을 만들어 3번 복제한다.
튜닝 2단계	행을 열로 바꾸기 위해 중간집합(7451건)을 만들어 UNPIVOT 기능을 사용한다.

실기

5. 반복 제거 – 문제1

가전, 가전외, 냉장고 및 전체의 판매수량 합계를 조회한다.

5. 반복 제거

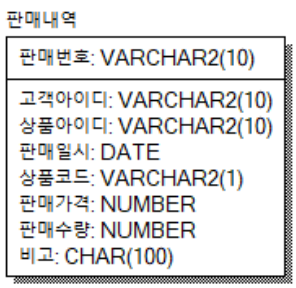
문제1	가전, 가전외, 냉장고 및 전체의 판매수량 합계를 조회한다.

문제설명

- A 온라인 쇼핑몰의 관리 담당자는 2022/7/1 ~ 2022/7/15 동안 판매한 판매 정보에 대해 가전, 가전외, 냉장고, 전체 별로 판매수량 합계를 조회하는 쿼리를 사용하고 있다.
- 판매일시 조회 기간은 최대 15일 이다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

NO	GOODS_TYPE	SALE_QY_SUM
1	가전	5358160
2	가전외	2887670
3	냉장고	2476920
4	전체	8245830

ERD



- 판매내역 테이블은 월30만 건(일1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A~X, 총 25개)를 가지며, A~I는 가전으로 전체 판매의 70%를 차지하고, J~X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);

SQL

```
SELECT 1                        AS NO,
       '가전'                    AS GOODS_TYPE,
       NVL(SUM(SALE_QY),0)    AS SALE_QY_SUM
  FROM T_SALE_LIST  S    -- 일 1만 건
 WHERE TO_CHAR(SALE_DT,'YYYY/MM/DD') BETWEEN '2022/07/01' AND '2022/07/15' --15일/12개월,컬럼가공
   AND GOODS_CD  IN ('A','B','C','D','E','F','H','I')
UNION ALL
SELECT 2,
       '가전외'                  AS GOODS_TYPE,
       NVL(SUM(SALE_QY),0)     AS SALE_QY_SUM
  FROM T_SALE_LIST  S    -- 일 1만 건
 WHERE TO_CHAR(SALE_DT,'YYYY/MM/DD') BETWEEN '2022/07/01' AND '2022/07/15' --15일/12개월,컬럼가공
   AND GOODS_CD  NOT IN ('A','B','C','D','E','F','H','I')
UNION ALL
SELECT 3,
       '냉장고'                    AS GOODS_TYPE,
       NVL(SUM(SALE_QY),0)     AS SALE_QY_SUM
  FROM T_SALE_LIST  S    -- 일 1만 건
 WHERE TO_CHAR(SALE_DT,'YYYY/MM/DD') BETWEEN '2022/07/01' AND '2022/07/15' --15일/12개월,컬럼가공
   AND GOODS_CD  = 'A'
```

5. 반복 제거

문제1	가전, 가전외, 냉장고 및 전체의 판매수량 합계를 조회한다.

SQL

```
UNION ALL
SELECT 4                      AS NO,
       '전체'                 AS GOODS_TYPE,
       NVL(SUM(SALE_QY),0)    AS SALE_QY_SUM
  FROM T_SALE_LIST  S    -- 일 1만 건
 WHERE TO_CHAR(SALE_DT,'YYYY/MM/DD') BETWEEN '2022/07/01' AND '2022/07/15' --15일/12개월,컬럼가공
```

ERD

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                | Name         | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | UNION ALL                |              |        4 |00:00:00.0000 |        0 |        1 |
|   2 |  UNION ALL               |              |        2 |00:00:00.0000 |        0 |        1 |
|   3 |   COLUMN PROJECTION      |              |        1 |00:00:00.0000 |        0 |        1 |
|   4 |    SORT AGGR             |              |        1 |00:00:00.0049 |        0 |        1 |
|   5 |     TABLE ACCESS (FULL)  | T_SALE_LIST  |    97500 |00:00:01.4066 |    79473 |        1 |
|   6 |   COLUMN PROJECTION      |              |        1 |00:00:00.0000 |        0 |        1 |
|   7 |    SORT AGGR             |              |        1 |00:00:00.0026 |        0 |        1 |
|   8 |     TABLE ACCESS (FULL)  | T_SALE_LIST  |    52500 |00:00:00.9421 |    79473 |        1 |
|   9 |  UNION ALL               |              |        2 |00:00:00.0000 |        0 |        1 |
|  10 |   COLUMN PROJECTION      |              |        1 |00:00:00.0000 |        0 |        1 |
|  11 |    SORT AGGR             |              |        1 |00:00:00.0024 |        0 |        1 |
|  12 |     TABLE ACCESS (FULL)  | T_SALE_LIST  |    45000 |00:00:00.6914 |    79473 |        1 |
|  13 |   COLUMN PROJECTION      |              |        1 |00:00:00.0000 |        0 |        1 |
|  14 |    SORT AGGR             |              |        1 |00:00:00.0072 |        0 |        1 |
|  15 |     TABLE ACCESS (FULL)  | T_SALE_LIST  |     150K |00:00:01.6450 |    79473 |        1 |
-------------------------------------------------------------------------------------------------
   5 - filter: (("S"."GOODS_CD") IN (('A'),('F'),('I'),('E'),('B'),('D'),('C'),('H'))) AND
(TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') >= '2022/07/01') AND (TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') <=
'2022/07/15') (1.000 * 0.100 * 1.000)
   8 - filter: (("S"."GOODS_CD") NOT IN (('A'),('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) AND
(TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') >= '2022/07/01') AND (TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') <= '
2022/07/15') (1.000 * 0.100 * 1.000)
  12 - filter: ("S"."GOODS_CD" = 'A') AND (TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') >= '2022/07/01') AND
(TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') <= '2022/07/15') (1.000 * 0.100 * 1.000)
  15 - filter: (TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') >= '2022/07/01') AND
(TO_CHAR("S"."SALE_DT",'YYYY/MM/DD') <= '2022/07/15') (0.100 * 1.000)
```

SQL

① Union All의 첫 번째 쿼리 블록에서 판매일시 컬럼 가공으로 판매내역 테이블을 Full Scan 해서, 판매일시와 가전 상품코드 조건에 맞는 97500건을 찾아 가전 판매수량 합계를 구한다.

② Union All의 두 번째 쿼리 블록에서 판매일시 컬럼 가공으로 판매내역 테이블을 Full Scan 해서, 판매일시와 가전외 상품코드 조건에 맞는 52500건을 찾아 가전외 판매수량 합계를 구한다.

③ Union All의 세 번째 쿼리 블록에서 판매일시 컬럼 가공으로 판매내역 테이블을 Full Scan 해서, 판매일시와 냉장고 상품코드 조건에 맞는 45000건을 찾아 냉장고 판매수량 합계를 구한다.

④ Union All의 네 번째 쿼리 블록에서 판매일시 컬럼 가공으로 판매내역 테이블을 Full Scan 해서, 판매일시 조건에 맞는 150만 건을 찾아 전체 판매수량 합계를 구한다.

⑤ Union All 의 각 쿼리 블록 결과를 합친다.

5. 반복 제거

문제1 – 튜닝 1단계	판매일시 컬럼 가공을 제거 하고, 판매내역을 1번 만 읽고 중간집합(15만 건)을 만들어 4번 복제한다.

튜닝내역

1. 판매일시 컬럼 가공으로 판매내역 테이블을 Full Scan 한다. 좋은 조건(판매일시 전체 12개월에서 15일 조회)에 인덱스를 사용하도록 컬럼 가공을 제거한다.
2. 가전, 가전 외, 냉장고, 전체 별로 판매수량 합계를 구하기 위해 판매내역을 4번 읽는 것을 1번 만 읽도록 한다. 판매일시 조건으로 1번 만 읽고 중간 집합을 만들어 4번 복제한 후, Group By를 사용하여 가전, 가전 외, 냉장고, 전체 별로 판매수량 합계를 계산한다.

   ① 판매내역을 판매일시 조건으로 1번 만 읽고 중간집합(15만 건)을 만든다.

가전(냉장고)	40
가전	20
가전	50
가전외	20
가전외	10

   ② 중간집합(15만 건)을 4번 복제하고, GOODS_TYPE(가전, 가전 외, 냉장고, 전체, 불필요)을 구해서, 그 중 '불필요'를 제외하고 NO, GOODS_TYPE로 Group By 해서 판매수량 합계를 계산한다.

SQL

```
SELECT NO, GOODS_TYPE, NVL(SUM(SALE_QY),0) AS SALE_QY_SUM
  FROM ( SELECT /*+ INDEX(S IX_T_SALE_LIST_DT) */
                NO,
                CASE WHEN NO = 1 AND GOODS_CD  IN    ('A','B','C','D','E','F','F','H','I') THEN '가전'
                     WHEN NO = 2 AND GOODS_CD  NOT IN ('A','B','C','D','E','F','F','H','I') THEN '가전외'
                     WHEN NO = 3 AND GOODS_CD  = 'A'                                        THEN '냉장고'
                     WHEN NO = 4                                                            THEN '전체'
                     ELSE                                                                        '불필요'
                END AS GOODS_TYPE,
                SALE_QY
           FROM T_SALE_LIST  S,    -- 일 1만 건, 전체 12개월
                (SELECT LEVEL NO
                   FROM DUAL
                CONNECT BY LEVEL <= 4
                )  -- 4번 복제
          WHERE SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD')
                            AND TO_DATE('2022/07/15','YYYY/MM/DD')  -- 컬럼 가공 제거
        ) S1
 WHERE GOODS_TYPE <> '불필요'
 GROUP BY NO, GOODS_TYPE
 ORDER BY NO, GOODS_TYPE ;
```

5. 반복 제거

문제1 – 튜닝 1단계	판매일시 컬럼 가공을 제거 하고, 인덱스로 판매내역을 1번 만 읽고 중간 집합(15만 건)을 만들어 4번 복제한다.

TRACE

```
-------------------------------------------------------------------------------------------------
|ID  | Operation                | Name              | Rows    | Elaps. Time  | CR Gets | Starts  |
-------------------------------------------------------------------------------------------------
|  1 | GROUP BY (SORT)          |                   |       4 |00:00:00.0967 |       0 |       1 |
|  2 |  NESTED LOOPS            |                   |    345K |00:00:00.1231 |       0 |       1 |
|  3 |   CONNECT BY (STACK)     |                   |       4 |00:00:00.0000 |       0 |       1 |  복제
|  4 |    DPV                   | _VT_DUAL          |       1 |00:00:00.0000 |       0 |       1 |
|  5 |    BUFF                  |                   |       0 |00:00:00.0000 |       0 |       0 |
|  6 |     DPV                  | _VT_DUAL          |       0 |00:00:00.0000 |       0 |       0 |
|  7 |   TABLE ACCESS (ROWID)   | T_SALE_LIST       |    150K |00:00:00.0298 |    3291 |       1 |  1번
|  8 |    INDEX (RANGE SCAN)    | IX_T_SALE_LIST_DT |    150K |00:00:00.0014 |     429 |       1 |
-------------------------------------------------------------------------------------------------

   2 - access: (CASE WHEN ((LEVEL = 1) AND (("S"."GOODS_CD") IN
(('A'),('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I')))) TH
(("S"."GOODS_CD") NOT IN (('A'),('B'),('C'),
('D'),('E'),('F'),('F'),('H'),('I')))) THEN '가전외' WHEN ((LEVEL = 3) AND ("S"."GOODS_CD" = 'A'))
THEN '냉장고' WHEN (LEVEL = 4) THEN '전체' ELSE '불필요' END <> '불필요') (0.990)
   8 - access: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-07-15 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
```

중간 집합의 크기가 15만 건이다.

참고

- 오라클의 경우, 4번의 복제가 Cartesian Product로 되지 않고, 판매내역을 실제로 4번 읽는 경우가 있다. 이때는 ROWNUM 을 사용하면 Cartesian Product가 된다.

```
SELECT NO, GOODS_TYPE, NVL(SUM(SALE_QY),0) AS SALE_QY_SUM
  FROM (
        SELECT /*+ INDEX(S IX_T_SALE_LIST_DT) */
               NO,
               ROWNUM AS IMSI,
               CASE WHEN NO = 1 AND GOODS_CD  IN    ('A','B','C','D','E','F','F','H','I') THEN '가전'
                    WHEN NO = 2 AND GOODS_CD  NOT IN ('A','B','C','D','E','F','F','H','I') THEN '가전외'
                    WHEN NO = 3 AND GOODS_CD  = 'A'                                        THEN '냉장고'
                    WHEN NO = 4                                                            THEN '전체'
                    ELSE                                                                       '불필요'
                END AS GOODS_TYPE,
                SALE_QY
          FROM T_SALE_LIST  S,
               (SELECT LEVEL NO
                  FROM DUAL
               CONNECT BY LEVEL <= 4
               )  -- 4번 복제
         WHERE SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD')
                           AND TO_DATE('2022/07/15','YYYY/MM/DD')  -- 컬럼 가공 제거
       ) S1
 WHERE GOODS_TYPE <> '불필요'
 GROUP BY NO, GOODS_TYPE
 ORDER BY NO, GOODS_TYPE ;
```

5. 반복 제거

문제1 – 튜닝 2단계	더 작은 중간집합(3건)을 만들어 3번 복제한다.

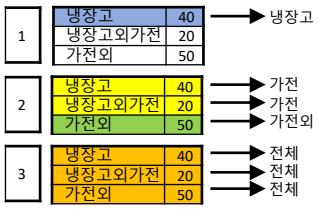
튜닝내역

1. 판매내역을 1번 만 읽고 중간 집합(15만 건)을 만들어 4번 복제한 후, Group By 한다. 이때 중간 집합(15만 건)이 커서 4번 복제하는 데 성능 저하가 발생한다. 따라서 판매내역을 1번 만 읽고 냉장고, 냉장고외가전, 가전외 별로 집계해서 중간 집합을 3건으로 줄인다.

   ① 판매내역을 판매일시 조건으로 1번 읽고 냉장고, 냉장고외가전, 가전외 별로 Group By 해서 중간 집합(3건)을 만든다.

냉장고	40
냉장고외가전	20
가전외	50

   ② 중간 집합(3건)을 3번 복제하고, GOODS_TYPE(가전, 가전외, 냉장고, 전체, 불필요)을 구해서, 그 중 '불필요'를 제외하고 NO, GOODS_TYPE로 Group By 해서 판매수량 합계를 계산한다.



SQL

```
SELECT NO,           GOOD_TYPE,          SUM(SALE_QY_SUM)  AS SALE_QY_SUM
  FROM (SELECT CASE WHEN NO=1 AND GOOD_TYPE = '냉장고'                    THEN 3
                    WHEN NO=2 AND GOOD_TYPE = '가전외'                    THEN 2
                    WHEN NO=2 AND GOOD_TYPE IN ('냉장고','냉장고외가전')  THEN 1
                    WHEN NO=3                                            THEN 4
                    ELSE                                                      5
               END  AS NO,
               CASE WHEN NO=1 AND GOOD_TYPE = '냉장고'                    THEN '냉장고'
                    WHEN NO=2 AND GOOD_TYPE = '가전외'                    THEN '가전외'
                    WHEN NO=2 AND GOOD_TYPE IN ('냉장고','냉장고외가전')  THEN '가전'
                    WHEN NO=3                                            THEN '전체'
                    ELSE                                                      '불필요'
               END  AS GOOD_TYPE,
               SALE_QY_SUM
          FROM (SELECT /*+ INDEX(S IX_T_SALE_LIST_DT) NO_MERGE */
                       CASE WHEN GOODS_CD = 'A'                                THEN '냉장고'
                            WHEN GOODS_CD IN    ('B','C','D','E','F','F','H','I')    THEN '냉장고외가전'
                            WHEN GOODS_CD NOT IN ('A','B','C','D','E','F','F','H','I') THEN '가전외'
                       END AS GOOD_TYPE,
                       NVL(SUM(SALE_QY),0)  AS SALE_QY_SUM
                  FROM T_SALE_LIST  S     -- 일1만 건, 전체 12개월
                 WHERE SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD')
                                   AND TO_DATE('2022/07/15','YYYY/MM/DD')
                 GROUP BY CASE WHEN GOODS_CD = 'A'                                THEN '냉장고'
                               WHEN GOODS_CD IN    ('B','C','D','E','F','F','H','I')    THEN '냉장고외가전'
                               WHEN GOODS_CD NOT IN ('A','B','C','D','E','F','F','H','I') THEN '가전외'
                          END
               ) A,
               ( SELECT LEVEL NO FROM DUAL CONNECT BY LEVEL <= 3 )
       ) A
 WHERE GOOD_TYPE <> '불필요'
 GROUP BY NO, GOOD_TYPE
 ORDER BY NO ;
```

이 부분은 없어도 로직에 문제가 없으나, 명확한 조건을 위해 추가 함

5. 반복 제거

문제1 – 튜닝 2단계	더 작은 중간 집합(3건)을 만들어 3번 복제한다.

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID | Operation                    | Name              | Rows  | Elaps. Time  | CR Gets | Starts  |
-------------------------------------------------------------------------------------------------------
|  1 | GROUP BY (SORT)              |                   |     4 |00:00:00.0001 |       0 |       1 |
|  2 |  NESTED LOOPS                |                   |     7 |00:00:00.0000 |       0 |       1 |
|  3 |   CONNECT BY (STACK)         |                   |     3 |00:00:00.0000 |       0 |       1 |   복제
|  4 |    DPV                       | _VT_DUAL          |     1 |00:00:00.0000 |       0 |       1 |
|  5 |    BUFF                      |                   |     0 |00:00:00.0000 |       0 |       0 |
|  6 |     DPV                      | _VT_DUAL          |     0 |00:00:00.0000 |       0 |       0 |
|  7 |   GROUP BY (HASH)            |                   |     3 |00:00:00.0345 |       0 |       1 |   1번
|  8 |    TABLE ACCESS (ROWID)      | T_SALE_LIST       |  150K |00:00:00.0305 |    3291 |       1 |
|  9 |     INDEX (RANGE SCAN)       | IX_T_SALE_LIST_DT |  150K |00:00:00.0017 |     429 |       1 |
-------------------------------------------------------------------------------------------------------
```

증간 집합의 크기가 3 건이다.

```
   2 - access: (CASE WHEN ((LEVEL = 1) AND (CASE WHEN ("S"."GOO                    HEN
(("S"."GOODS_CD") IN (('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '냉장고외가전' WHEN
(("S"."GOODS_CD") NOT IN (('A'),('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '가전외' ELSE
NULL END = '냉장고')) THEN '냉장고' WHEN ((LEVEL = 1) AND (CASE WHEN ("S"."GOODS_CD" = 'A') THEN '냉장
고 ' WHEN (("S"."GOODS_CD") IN (('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '냉장고외가전'
WHEN (("S"."GOODS_CD") NOT IN (('A'),('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '가전외'
ELSE NULL END = '가전외')) THEN '가전외' WHEN ((LEVEL = 2) AND ((CASE WHEN ("S"."GOODS_CD" = 'A') THEN
'냉장고' WHEN (("S"."GOODS_CD") IN (('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '냉장고외가
전' WHEN (("S"."GOODS_CD") NOT IN (('A'),('B'),('C'),('D'),('E'),('F'),('F'),('H'),('I'))) THEN '가전
외' ELSE NULL END) IN (('냉장고'),('냉장고외가전')))) THEN '가전' WHEN (LEVEL = 3) THEN '전체'
ELSE '불필요' END <> '불필요') (0.990)
   9 - access: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-07-15 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
```

실기

5.반복 제거 – 문제2

Group By 월,고객유형(함수) 과 Group By 일,고객유형(함수)로 판매수량 합계를 조회한다.

5. 반복 제거

문제2	Group By 월,고객유형(함수) 과 Group By 일,고객유형(함수)로 판매수량 합계를 조회한다.

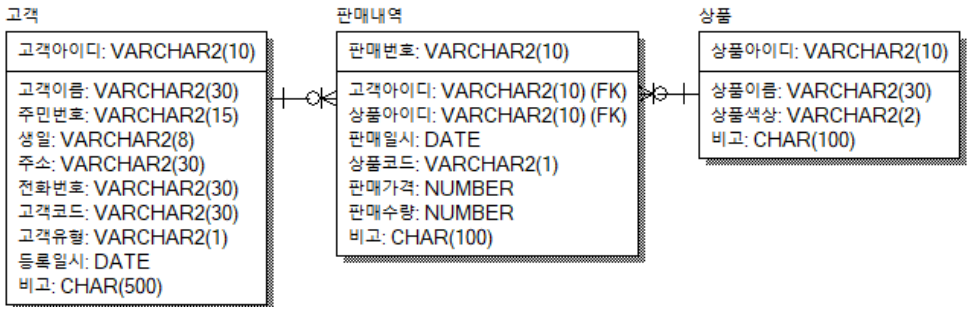
문제설명

- A 온라인 쇼핑몰의 관리 담당자는 2022년 3사 분기 동안 판매한 판매 정보에 대해 월, 고객유형 별로 Group By를 하고 일, 고객유형 별로 Group By를 해서 판매수량 합계를 조회하는 쿼리를 사용하고 있다.

GUBUN	SALE_DT	CUST_TYPE	SQLE_QY_SUM
월별	202207	A	5497410
월별	202207	B	5684300
월별	202207	C	5685200
월별	202207	D	182490
월별	202208	A	5490380
월별	202208	B	5723090
월별	202208	C	5652580
월별	202208	D	175200
월별	202209	A	5356620
월별	202209	B	5499810
월별	202209	C	5496130
월별	202209	D	172760
:			
일별	20220930	A	180550
일별	20220930	B	184940
일별	20220930	C	182010
일별	20220930	D	4960

- 판매일시 조회는 최대 3개월이며, 분기별 조회를 자주 한다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고, 고객코드(z0005, z0006)를 가지며 전체 고객의 0.5%는 z0005 값을 , 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33% 씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);
- 판매내역 테이블은 월30만 건, 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);
- 상품 테이블은 총 1만 개 상품에 대한 정보를 저장한다.

5. 반복 제거

문제2	Group By 월,고객유형(함수) 과 Group By 일,고객유형(함수)로 판매수량 합계를 조회한다.

SQL

```
CREATE OR REPLACE FUNCTION F_CUST_TYPE( P_CUST_ID IN VARCHAR2 ) RETURN VARCHAR2
IS
  V_CUST_TY   VARCHAR2(1);
BEGIN
  SELECT CUST_TY INTO V_CUST_TY
    FROM T_CUST
   WHERE CUST_ID = P_CUST_ID;

  RETURN  V_CUST_TY;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
/


SELECT /*+ FULL(S) */
       '월별'                          AS GUBUN,
       TO_CHAR(S.SALE_DT,'YYYYMM')     AS SALE_DT,
       F_CUST_TYPE(S.CUST_ID)          AS CUST_TYPE,
       SUM(S.SALE_QY)                  AS SQLE_QY_SUM
  FROM T_SALE_LIST   S    -- 월 30만 건, 전체 12개월
 WHERE S.SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD') AND TO_DATE('2022/09/30','YYYY/MM/DD')
 GROUP BY TO_CHAR(S.SALE_DT, 'YYYYMM'), F_CUST_TYPE(S.CUST_ID)
UNION ALL
SELECT /*+ FULL(S) */
       '일별'                          AS GUBUN,
       TO_CHAR(S.SALE_DT, 'YYYYMMDD')  AS SALE_DT,
       F_CUST_TYPE(S.CUST_ID)          AS CUST_TYPE,
       SUM(S.SALE_QY)                  AS SQLE_QY_SUM
  FROM T_SALE_LIST   S    -- 월 30만 건, 전체 12개월
 WHERE S.SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD') AND TO_DATE('2022/09/30','YYYY/MM/DD')
 GROUP BY TO_CHAR(S.SALE_DT,'YYYYMMDD'), F_CUST_TYPE(S.CUST_ID)
 ORDER BY 1,2,3;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation              | Name        | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)        |             |      380 |00:00:00.0003 |        0 |        1 |
|   2 |  UNION ALL             |             |      380 |00:00:00.0000 |        0 |        1 |
|   3 |   GROUP BY (HASH)      |             |       12 |00:00:00.4323 |        0 |        1 |
|   4 |    TABLE ACCESS (FULL) | T_SALE_LIST |     920K |00:00:27.1587 |    2839K |        1 |
|   5 |   GROUP BY (HASH)      |             |      368 |00:00:00.5309 |        0 |        1 |
|   6 |    TABLE ACCESS (FULL) | T_SALE_LIST |     920K |00:00:26.1839 |    2839K |        1 |
-------------------------------------------------------------------------------------------------
   4 - filter: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-09-30 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
   6 - filter: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-09-30 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
```

5. 반복 제거

문제2	Group By 월,고객유형(함수) 과 Group By 일,고객유형(함수)로 판매수량 합계를 조회한다.

분석

① Union All의 첫 번째 쿼리 블록에서, 판매내역 테이블을 Full Scan 해서, 2022년 3사 분기 조건에 맞는 92만 건을 찾아 월, 고객유형으로 Group By 해서 판매수량 합계를 추출한다. 이때 Group By 절의 F_CUST_TYPE 함수를 92만 번 호출하며, 여기서 구해진 고객유형 값이 Select절의 F_CUST_TYPE 함수를 다시 호출하지 않고 재사용 된다.

② Union All의 두 번째 쿼리 블록에서, 판매내역 테이블을 Full Scan 해서, 2022년 3사 분기 조건에 맞는 92만 건을 찾아 일, 고객유형으로 Group By 해서 판매수량 합계를 추출한다. 이때 Group By 절의 F_CUST_TYPE 함수를 92만 번 호출하며, 여기서 구해진 고객유형 값이 Select절의 F_CUST_TYPE 함수를 다시 호출하지 않고 재사용 된다.

③ Union All의 각 쿼리 블록 결과를 합쳐서 정렬한다.

④ ①과 ②에서 고객유형을 구하는 F_CUST_TYPE 함수 호출(총 184만 번)이 많아, 성능 저하가 발생한다.
- 함수에서 엑세스 한 블록 량 = (280만 블록 – 8만 블록) * 2 = 540만 블록
- 함수 184만 번 호출 시 사용한 CPU

5. 반복 제거

문제2 – 튜닝 1단계	F_CUST_TYPE 함수 호출을 고객과 조인하는 것으로 변경한다.

튜닝내역

1. 고객유형을 구하는 F_CUST_TYPE 함수 호출(총 360만 번)이 많아 성능 저하가 발생한다. 따라서 F_CUST_TYPE 함수 호출을 고객과 조인하는 것으로 변경한다.
   - 540만 블록(함수에서 액세스 한 블록 량) → 724 블록
   - 함수 360만 번 호출 시 사용한 CPU → 0

SQL

```
SELECT /*+ FULL(S) */
       '월별'                          AS GUBUN,
       TO_CHAR(S.SALE_DT,'YYYYMM')     AS SALE_DT,
       F_CUST_TYPE(S.CUST_ID)          AS CUST_TYPE,
       C.CUST_TY                       AS CUST_TYPE,
       SUM(S.SALE_QY)                  AS SQLE_QY_SUM
  FROM T_SALE_LIST   S,   -- 월30만 건, 전체 12개월
       T_CUST        C    -- 추가
 WHERE S.SALE_DT BETWEEN TO_DATE( '2022/07/01' ,'YYYY/MM/DD') AND TO_DATE('2022/09/30','YYYY/MM/DD')
AND S.CUST_ID = C.CUST_ID  -- 추가
GROUP BY TO_CHAR(S.SALE_DT, 'YYYYMM'), F_CUST_TYPE(S.CUST_ID)  C.CUST_TY
UNION ALL
SELECT  /*+ FULL(S) */
       '일별'                          AS GUBUN,
       TO_CHAR(S.SALE_DT,'YYYYMMDD')   AS SALE_DT,
       F_CUST_TYPE(S.CUST_ID)          AS CUST_TYPE,
       C.CUST_TY                       AS CUST_TYPE,
       SUM(S.SALE_QY)                  AS SQLE_QY_SUM
  FROM T_SALE_LIST   S,   -- 월30만 건, 전체 12개월
       T_CUST        C    -- 추가
 WHERE S.SALE_DT BETWEEN TO_DATE( '2022/07/01' ,'YYYY/MM/DD') AND TO_DATE('2022/09/30','YYYY/MM/DD')
   AND S.CUST_ID = C.CUST_ID  -- 추가
  GROUP BY TO_CHAR(S.SALE_DT,'YYYYMMDD'), F_CUST_TYPE(S.CUST_ID)  C.CUST_TY
ORDER BY 1,2,3;
```

5. 반복 제거

문제2 – 튜닝 1단계	F_CUST_TYPE 함수 호출을 고객과 조인하는 것으로 변경한다.

TRACE

```
-------------------------------------------------------------------------------------------------
| ID | Operation                    | Name          | Rows     | Elaps. Time  | CR Gets | Starts  |
-------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)              |               |      380 |00:00:00.0002 |       0 |       1 |
|  2 |  UNION ALL                   |               |      380 |00:00:00.0000 |       0 |       1 |
|  3 |   GROUP BY (HASH)            |               |       12 |00:00:00.5580 |       0 |       1 |
|  4 |    HASH JOIN                 |               |      920K|00:00:00.4306 |       0 |       1 |
|  5 |     HASH JOIN                |               |    70000 |00:00:00.0550 |       0 |       1 |
|  6 |      INDEX (FAST FULL SCAN)  | IX_T_CUST_TY  |    70000 |00:00:00.0006 |     131 |       1 |
|  7 |      INDEX (FAST FULL SCAN)  | PK_T_CUST     |    70000 |00:00:00.0007 |     231 |       1 |
|  8 |     TABLE ACCESS (FULL)      | T_SALE_LIST   |      920K|00:00:00.3468 |   79473 |       1 |
|  9 |   GROUP BY (HASH)            |               |      368 |00:00:00.6897 |       0 |       1 |
| 10 |    HASH JOIN                 |               |      920K|00:00:00.4450 |       0 |       1 |
| 11 |     HASH JOIN                |               |    70000 |00:00:00.0481 |       0 |       1 |
| 12 |      INDEX (FAST FULL SCAN)  | IX_T_CUST_TY  |    70000 |00:00:00.0004 |     131 |       1 |
| 13 |      INDEX (FAST FULL SCAN)  | PK_T_CUST     |    70000 |00:00:00.0006 |     231 |       1 |
| 14 |     TABLE ACCESS (FULL)      | T_SALE_LIST   |      920K|00:00:00.3382 |   79473 |       1 |
-------------------------------------------------------------------------------------------------

   4 - access: ("C"."CUST_ID" = "S"."CUST_ID") (0.000)
   8 - filter: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-09-30 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
  10 - access: ("C"."CUST_ID" = "S"."CUST_ID") (0.000)
  14 - filter: ("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-09-30 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
```

참고

- IX_T_CUST_TY 와 PK_T_CUST 를 사용한 Index Join을 고객 테이블 Full Scan으로 변경하면, 1번 Full Scan(5858 블록) 시 액세스하는 블록 양이 Index Join(362 블록) 보다 많다. 물론 Full Scan 은 Multi Block I/O이고 Index Join은 랜덤 I/O 라는 차이는 있지만...

```
-------------------------------------------------------------------------------------------------
| ID | Operation                    | Name          | Rows     | Elaps. Time  | CR Gets | Starts  |
-------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)              |               |      380 |00:00:00.0003 |       0 |       1 |
|  2 |  UNION ALL                   |               |      380 |00:00:00.0000 |       0 |       1 |
|  3 |   GROUP BY (HASH)            |               |       12 |00:00:00.5839 |       0 |       1 |
|  4 |    HASH JOIN                 |               |      920K|00:00:00.4746 |       0 |       1 |
|  5 |     TABLE ACCESS (FULL)      | T_CUST        |    70000 |00:00:00.1687 |    5858 |       1 |
|  6 |     TABLE ACCESS (FULL)      | T_SALE_LIST   |      920K|00:00:00.3644 |   79473 |       1 |
|  7 |   GROUP BY (HASH)            |               |      368 |00:00:00.6320 |       0 |       1 |
|  8 |    HASH JOIN                 |               |      920K|00:00:00.4209 |       0 |       1 |
|  9 |     TABLE ACCESS (FULL)      | T_CUST        |    70000 |00:00:00.0151 |    5858 |       1 |
| 10 |     TABLE ACCESS (FULL)      | T_SALE_LIST   |      920K|00:00:00.3521 |   79473 |       1 |
-------------------------------------------------------------------------------------------------
```

5. 반복 제거

문제2 – 튜닝 2단계	Grouping Sets을 사용하여 판매내역을 1번 만 읽는다.

튜닝내역

1. Union All을 사용하여 판매내역 테이블을 2번 읽는 것을, Grouping Sets을 사용하여 판매내역을 1번 만 읽는다.

SQL

```
SELECT  CASE WHEN SALE_DT1 IS NOT NULL THEN '월별'      ELSE '일별'      END  AS GUBUN,
        CASE WHEN SALE_DT1 IS NOT NULL THEN SALE_DT1   ELSE SALE_DT2   END  AS SALE_DT,
        CUST_TYPE,
        SQLE_QY_SUM
  FROM (
         SELECT /*+ FULL(S) */
                TO_CHAR(S.SALE_DT,'YYYYMM')     AS SALE_DT1,
                TO_CHAR(S.SALE_DT,'YYYYMMDD')   AS SALE_DT2,
                C.CUST_TY                       AS CUST_TYPE,
                SUM(S.SALE_QY)                  AS SQLE_QY_SUM
           FROM T_SALE_LIST   S,
                T_CUST        C
          WHERE SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD')
                            AND TO_DATE('2022/09/30','YYYY/MM/DD')
            AND S.CUST_ID = C.CUST_ID
          GROUP BY GROUPING SETS(  (TO_CHAR(S.SALE_DT,'YYYYMM'),  C.CUST_TY),
                                   (TO_CHAR(S.SALE_DT,'YYYYMMDD'),C.CUST_TY)
                                )
       )
 ORDER BY 1,2,3;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID | Operation                    | Name          | Rows     | Elaps. Time  | CR Gets | Starts  |
-------------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)              |               |      380 |00:00:00.0002 |       0 |       1 |
|  2 |  GROUP BY (HASH)             |               |      380 |00:00:00.0035 |       0 |       1 |
|  3 |   CUBE                       |               |      736 |00:00:00.0000 |       0 |       1 |
|  4 |    GROUP BY (HASH)           |               |      368 |00:00:00.9395 |       0 |       1 |
|  5 |     HASH JOIN                |               |     920K |00:00:00.3789 |       0 |       1 |
|  6 |      HASH JOIN               |               |    70000 |00:00:00.0697 |       0 |       1 |
|  7 |       INDEX (FAST FULL SCAN) | IX_T_CUST_TY  |    70000 |00:00:00.0009 |     131 |       1 |
|  8 |       INDEX (FAST FULL SCAN) | PK_T_CUST     |    70000 |00:00:00.0009 |     231 |       1 |
|  9 |      TABLE ACCESS (FULL)     | T_SALE_LIST   |     920K |00:00:00.3480 |   79473 |       1 |
-------------------------------------------------------------------------------------------------------

   5 - access("C"."CUST_ID" = "S"."CUST_ID") (0.000)
   9 - filter("S"."SALE_DT" >= TO_DATE('2022-07-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-09-30 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.506 * 1.000)
```

5. 반복 제거

문제2 – 튜닝 2단계	Grouping Sets을 사용하여 판매내역을 1번 만 읽는다.

참고

- Grouping Sets 을 내부적으로 처리되는 방법은 티베로와 오라클이 약간 다르다.
- 티베로는 CUBE 아래의 Group By에서 Grouping Sets에서 사용한 Group By Key를 모두 사용해서 중간집합(368건)을 만들고, CUBE 위의 Group By에서 중간집합에서 만들어진 것 중 필요한 것을 고르고(368건), 다시 월, 고객유형으로 Group By 해서 생성된 12건과 합쳐 380건을 추출한다.

```
-------------------------------------------------------------------------------------------------------
| ID | Operation                    | Name          | Rows     | Elaps. Time  | CR Gets | Starts |
-------------------------------------------------------------------------------------------------------
|  1 | ORDER BY (SORT)              |               |      380 |00:00:00.0002 |       0 |      1 |
|  2 |  GROUP BY (HASH)             |               |      380 |00:00:00.0035 |       0 |      1 |
|  3 |   CUBE                       |               |      736 |00:00:00.0000 |       0 |      1 |
|  4 |    GROUP BY (HASH)           |               |      368 |00:00:00.9395 |       0 |      1
|  5 |     HASH JOIN                |               |     920K |00:00:00.3789 |       0 |      1
|  6 |      HASH JOIN               |               |    70000 |00:00:00.0697 |       0 |      1
|  7 |       INDEX (FAST FULL SCAN) | IX_T_CUST_TY  |    70000 |00:00:00.0009 |     131 |      1
|  8 |       INDEX (FAST FULL SCAN) | PK_T_CUST     |    70000 |00:00:00.0009 |     231 |      1
|  9 |      TABLE ACCESS (FULL)     | T_SALE_LIST   |     920K |00:00:00.3480 |   79473 |      1
-------------------------------------------------------------------------------------------------------
```

- 오라클은 TEMP TABLE TRANSFORMATION 아래에서 고객과 판매내역을 조인해서 중간집합(90만건)을 만들고, 중간집합으로 일,고객유형으로 Group By 해서 생성된 12건과, 중간집합으로 월, 고객유형으로 Group By 해서 생성된 368건을 합쳐 388건을 추출한다. 오라클의 중간 집합이 더 크기 때문에 사용된 전체 블록 양이 더 많다.

```
-----------------------------------------------------------------------------------------------------------
|Id | Operation                    | Name                    |Start|A-Rows |   A-Time   |Buffers |
-----------------------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT             |                         |   1 |   380 |00:00:01.97 |  89911 |
| 1 |  SORT ORDER BY               |                         |   1 |   380 |00:00:01.97 |  89911 |
| 2 |   VIEW                       |                         |   1 |   380 |00:00:01.97 |  89911 |
| 3 |    TEMP TABLE TRANSFORMATION |                         |   1 |   380 |00:00:01.97 |  89911 |
| 4 |     LOAD AS SELECT           |                         |   1 |     0 |00:00:01.53 |  83251 |
| 5 |      HASH JOIN               |                         |   1 |  920K|00:00:00.95 |  79888 |
| 6 |       VIEW                   |index$_join$_005         |   1 | 70000 |00:00:00.17 |    426 |
| 7 |        HASH JOIN             |                         |   1 | 70000 |00:00:00.16 |    426 |
| 8 |         INDEX FAST FULL SCAN |IX_T_CUST_TY             |   1 | 70000 |00:00:00.02 |    197 |
| 9 |         INDEX FAST FULL SCAN |PK_T_CUST                |   1 | 70000 |00:00:00.09 |    229 |
|10 |       TABLE ACCESS FULL      |T_SALE_LIST              |   1 |  920K|00:00:00.35 |  79462 |
|11 |     LOAD AS SELECT           |                         |   1 |     0 |00:00:00.27 |   3322 |
|12 |      HASH GROUP BY           |                         |   1 |    12 |00:00:00.27 |   3318 |
|13 |       TABLE ACCESS FULL      |SYS_TEMP_0FD9D6606_583FBC|   1 |  920K|00:00:00.13 |   3318 |
|14 |     LOAD AS SELECT           |                         |   1 |     0 |00:00:00.17 |   3321 |
|15 |      HASH GROUP BY           |                         |   1 |   368 |00:00:00.17 |   3315 |
|16 |       TABLE ACCESS FULL      |SYS_TEMP_0FD9D6606_583FBC|   1 |  920K|00:00:00.09 |   3315 |
|17 |     VIEW                     |                         |   1 |   380 |00:00:00.01 |      8 |
|18 |      TABLE ACCESS FULL       |SYS_TEMP_0FD9D6607_583FBC|   1 |   380 |00:00:00.01 |      8 |
-----------------------------------------------------------------------------------------------------------
```

5. 반복 제거

문제2 – 튜닝 3단계	Grouping Sets을 WITH 문으로 변경한다.

튜닝내역

1. Grouping Sets은 중간 집합을 만들고, 생성 된 중간 집합을 가지고 필요한 추가 가공(Group By 등)을 해서 원하는 값을 추출한다. 티베로는 일, 고객유형으로 Group By 된 중간 집합을 만들었고, 오라클은 Group By 하지 않은 테이블 수준의 중간 집합을 만들어 사용한다. 중간 집합은 WITH 문으로 구현할 수 있으므로, 오라클은 Grouping Sets을 일, 고객유형으로 Group By한 WITH 문을 사용하는 것이 더 좋다.

SQL

```
WITH
W_DAY_SUM AS  -- 일별로 GROUP BY
(   SELECT /*+ MATERIALIZE  FULL(S)   */
           TO_CHAR(S.SALE_DT,'YYYYMMDD') AS SALE_DT2,
           C.CUST_TY                     AS CUST_TY,
           SUM(S.SALE_QY)                AS SALE_QY_SUM
      FROM T_SALE_LIST   S,
           T_CUST        C
     WHERE S.SALE_DT BETWEEN TO_DATE('2022/07/01','YYYY/MM/DD') AND TO_DATE('2022/09/30','YYYY/MM/DD')
       AND S.CUST_ID = C.CUST_ID
     GROUP BY TO_CHAR(S.SALE_DT,'YYYYMMDD'), C.CUST_TY
)
SELECT'일별'                         AS GUBUN,
       SALE_DT2                      AS SALE_DT,
       CUST_TY                       AS CUST_TYPE,
       SALE_QY_SUM                   AS SQLE_QY_SUM
  FROM W_DAY_SUM
UNION ALL
SELECT'월별'                         AS GUBUN,
       SUBSTR(SALE_DT2,1,6)          AS SALE_DT,
       CUST_TY                       AS CUST_TYPE,
       SUM(SALE_QY_SUM)              AS SQLE_QY_SUM
  FROM W_DAY_SUM
 GROUP BY SUBSTR(SALE_DT2,1,6), CUST_TY
 ORDER BY 1,2,3;
```

TRACE

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	BUFF TRANSFORMATION		380	00:00:01.4706	79835	1
2	GROUP BY (HASH)		368	00:00:00.6653	0	1
3	HASH JOIN		920K	00:00:00.4148	0	1
4	HASH JOIN		70000	00:00:00.0553	0	1
5	INDEX (FAST FULL SCAN)	IX_T_CUST_TY	70000	00:00:00.0008	131	1
6	INDEX (FAST FULL SCAN)	PK_T_CUST	70000	00:00:00.0006	231	1
7	TABLE ACCESS (FULL)	T_SALE_LIST	920K	00:00:00.3312	79473	1
8	ORDER BY (SORT)		380	00:00:00.0002	0	1
9	UNION ALL		380	00:0		
10	COLUMN PROJECTION		368	00:0		
11	BUFF		368	00:00:00.0000		
12	GROUP BY (SORT)		12	00:00:00.0003	0	1
13	BUFF		368	00:00:00.0000	0	1

WITH 문을 사용하면 상단에 블록 누적 값이 나온다.

실기

5. 반복 제거 – 문제3

과목 성적이 90점 이상이면, 과목 성적을 고득점 테이블에 입력한다.

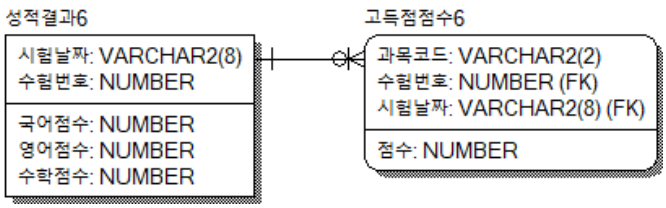
5. 반복제거

문제3	과목 성적이 90점 이상이면, 과목 성적을 고득점 테이블에 입력한다.

문제설명

- 학사 담당자 A씨는 성적결과를 읽어 과목 성적이 90점 이상인 고득점 과목을 따로 고득점 테이블에 입력하는 쿼리가 필요하다.
- 성적결과를 1번 만 읽고, 페이지 처리를 하지 않는다.
- 신규 인덱스 생성 또는 인덱스 수정은 필요 없다.

ERD



- 성적결과 테이블은 2024/1/1에 1만 명 수험생이 치른 국어, 영어, 수학 과목의 성적을 저장한다.
- 고득점점수 테이블은 특정 시험날짜에 치른 국어, 영어, 수학 과목의 성적 중 90점 이상 인 과목의 성적을 저장한다. 과목코드는 국어는 01, 영어는 02, 수학은 03 값을 가진다.

5. 반복제거

문제3 – 튜닝 1단계	행을 열로 바꾸기 위해, 중간집합(7451건)을 만들어 3번 복제한다.

튜닝내역

1. 2024/1/1 성적 만 있으므로, FULL 힌트로 성적결과 테이블을 Full Scan 한다.
2. 행으로 된 국어, 영어, 수학 점수에서 90점을 만족하는 과목 만 입력하기 위해, 행을 열로 바꾸기 위해 복제를 한다.
   ① 성적결과를 시험일시와 90점 이상 조건으로 1번 만 읽고 중간 집합(7451건)을 만든다.
   ② 행(1개 행에 국어, 영어, 수학이 존재)을 열로 바꾸기 위해 3번 복제(1은 국어, 2는 영어, 3은 수학)하고 90점 이상을 추출해서 입력한다.

SQL

```
INSERT INTO T_HIGH_SCORE6 (TEST_DY, EXAM_NO, SUBJECT_CD, SCORE)
SELECT TEST_DY,
       EXAM_NO,
       CASE WHEN FLAG=1 THEN '01'
            WHEN FLAG=2 THEN '02'
            WHEN FLAG=3 THEN '03'
       END AS SUBJECT_CD,
       CASE WHEN FLAG=1 THEN KOREAN
            WHEN FLAG=2 THEN ENGLISH
            WHEN FLAG=3 THEN MATHS
       END AS SCORE
  FROM (SELECT /*+ FULL(S) */  TEST_DY, EXAM_NO, KOREAN, ENGLISH, MATHS
          FROM T_SCORE_RESULT6  S
         WHERE TEST_DY = '20240101'
           AND (   KOREAN  >= 90
                OR ENGLISH >= 90
                OR MATHS   >= 90
               )  --7451 ROWS
       ) A,
       (SELECT LEVEL FLAG FROM DUAL  CONNECT BY LEVEL <= 3 ) B
 WHERE (FLAG = 1 AND KOREAN  >= 90 )
    OR (FLAG = 2 AND ENGLISH >= 90 )
    OR (FLAG = 3 AND MATHS   >= 90 );
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation              | Name           | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | INSERT                 | T_HIGH_SCORE6  |        0 |00:00:00.0264 |        4 |        1 |
|   2 |  NESTED LOOPS          |                |    11016 |00:00:00.0073 |        0 |        1 |
|   3 |   CONNECT BY (STACK)   |                |        3 |00:00:00.0000 |        0 |        1 |
|   4 |    DPV                 | _VT_DUAL       |        1 |00:00:00.0000 |        0 |        1 |
|   5 |    BUFF                |                |        0 |00:00:00.0000 |        0 |        0 |
|   6 |     DPV                | _VT_DUAL       |        0 |00:00:00.0000 |        0 |        0 |
|   7 |   TABLE ACCESS (FULL)  | T_SCORE_RESULT6|     7451 |00:00:00.0074 |       44 |        1 |
-------------------------------------------------------------------------------------------------

   2 - access: ((((LEVEL = 1) AND ("S"."KOREAN" >= 90)) OR ((LEVEL = 2) AND ("S"."ENGLISH" >= 90))) OR
((LEVEL = 3) AND ("S"."MATHS" >= 90))) (0.011)
   7 - filter: ((("S"."KOREAN" >= 90) OR ("S"."ENGLISH" >= 90)) OR ("S"."MATHS" >= 90)) AND
("S"."TEST_DY" = '20240101') (0.747 * 1.000)
```

5. 반복 제거

문제3 – 튜닝 2단계	행을 열로 바꾸기 위해 중간 집합(7451건)을 만들어 UNPIVOT 기능을 사용한다.

튜닝내역

1. 행으로 된 국어, 영어, 수학 점수 중에서 90점을 만족하는 과목 만 입력하기 위해, 행을 열로 바꾸기 위해 UNPIVOT 기능을 사용한다.

   ① 성적결과를 시험일시와 90점 이상 조건으로 1번 만 읽고 중간 집합(7451건)을 만든다.

   ② 행(1개 행에 국어, 영어, 수학 존재)을 열로 바꾸기 위해, UNPIVOT 기능을 사용하여 SUBJECT_CD로 01(국어), 02(영어), 03(수학)을 추출하고, SCORE로 과목 점수를 추출한다.

SQL

```
INSERT INTO T_HIGH_SCORE6 (TEST_DY, EXAM_NO, SUBJECT_CD, SCORE)
SELECT TEST_DY, EXAM_NO, SUBJECT_CD, SCORE
  FROM (  SELECT TEST_DY,
                 EXAM_NO,
                 SUBJECT_CD,  -- UNPIVOT절 안에 있는 컬럼
                 SCORE        -- UNPIVOT절 안에 있는 컬럼
            FROM ( SELECT /*+ FULL(S) */
                          TEST_DY,  EXAM_NO,
                          KOREAN,    → UNPIVOT IN 안으로
                          ENGLISH,   → UNPIVOT IN 안으로
                          MATHS      → UNPIVOT IN 안으로
                     FROM T_SCORE_RESULT6  S
                    WHERE TEST_DY =  '20240101'
                      AND (    KOREAN  >= 90
                            OR ENGLISH >= 90
                            OR MATHS   >= 90  ---7451 ROWS
                          )
                 )
                  UNPIVOT ( SCORE  FOR SUBJECT_CD  IN ( KOREAN    AS '01',
                                                        ENGLISH  AS '02',
                                                        MATHS    AS '03'
                                                      )
                          )
       )
 WHERE SCORE_X  >=90;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------
| ID | Operation                    | Name                | Rows     | Elaps. Time  | CR Gets  | Starts  |
-------------------------------------------------------------------------------------------------------------
| 1 | INSERT                        | T_HIGH_SCORE6       |        0 |00:00:00.0234 |        4 |       1 |
| 2 |  FILTER                       |                     |    11016 |00:00:00.0020 |        0 |       1 |
| 3 |   UNPIVOT                     |                     |    22353 |00:00:00.0017 |        0 |       1
| 4 |    TABLE ACCESS (ROWID)       | T_SCORE_RESULT6     |     7451 |00:00:00.0037 |       41 |       1
| 5 |     INDEX (FAST FULL SCAN)    | PK_T_SCORE_RESULT6  |    10000 |00:00:00.0006 |       44 |       1
-------------------------------------------------------------------------------------------------------------

   4 - filter: (((“T_SCORE_RESULT6”.“KOREAN” >= 90) OR (“T_SCORE_RESULT6”.“ENGLISH” >= 90)) OR
(“T_SCORE_RESULT6”.“MATHS” >= 90)) (0.747)
   5 - filter: (“T_SCORE_RESULT6”.“TEST_DY” = '20240101') (1.000)
```

5. 반복 제거

정리

1. Union All을 사용하여 동일 테이블을 여러 번 읽는 경우, 테이블을 1번 만 읽고 중간 집합을 만들어 복제해서 사용한다.
2. 중간 집합을 만들어 복제를 할 때, 중간 집합의 크기가 작을 수록 좋다.
3. 쿼리에서 함수를 사용하는 것 보다, 함수 안에서 사용된 테이블을 쿼리에서 사용(주로 Hash조인이 될 듯)하는 것이 좋다. 쿼리에서 함수를 사용하면 함수 호출 비용(CPU), 함수 안의 쿼리 파싱 비용(CPU), 실행 시 블록 엑세스 량(함수 안에서 건건이 테이블을 엑세스)이 증가한다.
4. 동일 테이블을 여러 차원으로 Group By 할 경우, Grouping Sets을 사용하여 테이블을 1번 만 읽으면 된다.
5. Grouping Sets 을 내부적으로 처리하는 방법은 티베로와 오라클이 약간 다르다. 티베로는 Grouping Sets 에서 사용한 Group By Key를 모두 사용해서 중간 집합을 만들고, 오라클은 Group By 직전 수준으로 중간 집합을 만든다. 따라서 오라클의 중간 집합이 더 크기 때문에 사용된 전체 블록 량이 더 많다.
6. Grouping Sets은 중간 집합을 만들고, 생성 된 중간 집합을 가지고 필요한 추가 가공(Group By 등)을 해서 원하는 값을 추출한다. 티베로는 일, 고객유형으로 Group By 된 중간 집합을 만들었고, 오라클은 Group By 하지 않은 테이블 수준의 중간 집합을 만들어 사용한다. 중간 집합은 WITH 문으로 구현할 수 있으므로, 오라클은 Grouping Sets을 일 ,고객유형 별로 Group By 한 WITH 문을 사용하는 것이 더 좋다.
7. 행을 열로 바꾸기 위해, 복제 또는 UNPIVOT 기능을 사용한다. UNPIVOT 기능을 사용하여 값(점수) 과 구분(과목구분)을 추출한다.

실기

6.분석 함수

6. 분석 함수

문제1	사원을 1번 만 읽고, 최소/최대 사원수를 가지는 부서 아이디를 조회한다.
튜닝 1단계	중간집합에서 부서의 사원수, 최소/최대 사원수를 가지는 부서를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.
문제2	판매내역을 1번 만 읽고, 판매내역 정보와 상품코드의 판매가격 평균/합계를 조회한다.
분석	판매내역을 2번 읽고, 판매내역 정보와 상품코드의 판매가격 평균/합계를 조회한다.
튜닝 1단계	중간집합에서 상품코드의 판매가격 평균/합계를 AVG 와 SUM 분석함수로 구한다.
문제3	판매내역을 1번 만 읽고, 상품코드의 마지막 판매번호에 해당하는 판매일시/판매수량을 조회한다.
분석	판매내역을 2번 읽고, 상품코드의 마지막 판매번호에 해당하는 판매일시와 판매수량을 조회한다.
튜닝 1단계	중간집합에서 상품코드의 마지막 판매번호를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.
튜닝 2단계	FIRST_VALUE와 LAST_VALUE 분석함수는 메모리 사용량이 많아 MAX 그룹함수로 변경한다.
문제4	성적결과를 1번 만 읽고 총점 평균, 최고 총점, 최고 총점 인원수를 성적집계에 입력한다.
튜닝 1단계	중간집합에서 최고 총점을 FIRST_VALUE 분석함수로 구한다.

실기

6.분석 함수 – 문제1

사원을 1번 만 읽고, 최소/최대 사원수를 가지는 부서 아이디를 조회한다.

6. 분석 함수

문제1	사원을 1번 만 읽고, 최소/최대 사원수를 가지는 부서 아이디를 조회한다.

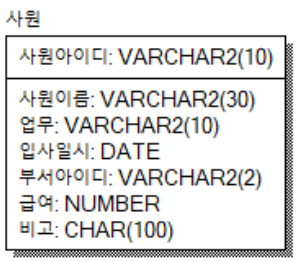
문제설명

- A 회사의 인사 담당자는 사원 테이블을 읽고 전체 사원수, 부서 개수, 부서의 평균 사원수, 부서의 최소 사원수, 부서의 최소/최대 사원수를 가진 부서 아이디를 조회하는 쿼리가 필요하다.

전체사원수	부서개수	부서의평균사원수	부서의최소사원수	부서의최대사원수	최소사원수의부서	최대사원수의부서
20300	7	2900	2817	2998	40	50

- 사원을 1번 만 읽고, 필요한 정보를 조회하는 쿼리를 작성하라.
- 페이지 처리를 하지 않으며, 신규 인덱스 생성 또는 인덱스 수정은 필요 없다.

ERD



- 사원 테이블은 2만 명 사원에 대한 정보를 저장한다. 사원은 업무(MANAGER, CLERK, SALESMAN, DEVELOPER, DESIGNER, PROGRAMMER, DBA) 와 부서아이디(01 ~ 07)를 골고루 가진다. 입사일시은 Date Type으로 2022/1 ~ 2023/12 사이 값을 가진다.

6. 분석 함수

문제1 – 튜닝 1단계	중간 집합에서 부서의 사원수, 최소/최대 사원수를 가지는 부서를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

튜닝내역

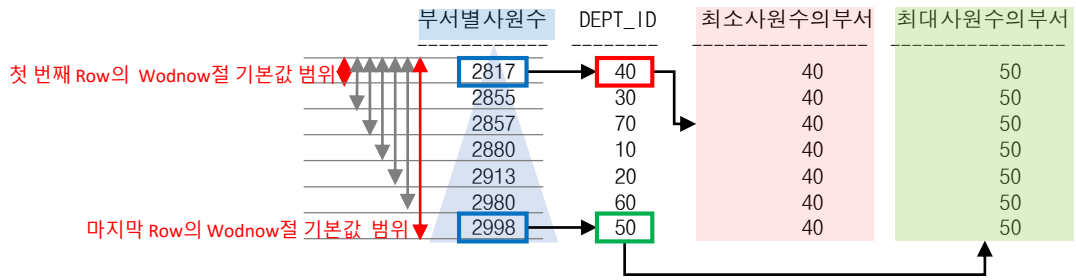
1. 판매내역을 1번 만 읽고, 원하는 값을 조회하는 쿼리를 작성하기 위해 다음 순서로 진행한다.

① 최종 결과를 어떤 단위로 조회 하느냐? 테이블 단위, Group By 단위, 1건 중 어떤 단위인가?
→ 1건 단위

② 최종 결과를 얻기 위해 필요한 중간 집합이 무엇인가?
→ 중간 집합(E 인라인뷰)에서 부서의 사원수, 최소/최대 사원수를 가지는 부서를 FIRST_VALUE 또는 LAST_VALUE로 구한다.

FIRST_VALUE 분석함수를 사용하여 Count(*)로 오름차순 정렬하여, 제일 먼저 나오는 Row의 DEPT_ID(40)를 구하고, LAST_VALUE 분석함수를 사용하여 Count(*)로 오름차순 정렬하여, 제일 마지막에 나오는 Row의 DEPT_ID(50)를 구한다.

분석함수(매개변수) Over(Partition By절 Order By절 Windows절)
- Partition By절 : 파티션 그룹을 지정한다.
- Order By절 : 파티션 그룹 안에서 순서를 지정한다
- Windows절 : 파티션 그룹 안에서 더 상세한 그룹으로 분할할 때 지정한다.
  기본값은 Rows Between Unbounded Preceding And Current Row(첫 번째 Row 부터 현재 Row까지)이며, LAST_VALUE를 사용할 경우, ~ And Current Row(현재 Row까지)를 ~ And Unbounded Following(마지막 Row까지)으로 변경해야 원하는 결과가 나온다.



③ 메인 쿼리에서 E 인라인뷰를 가지고 전체 사원수, 부서 개수, 부서의 평균 사원수, 부서의 최소 사원수, 부서의 최대 사원수, 최소/최대 사원수를 가진 부서 아이디를 조회한다.

SQL

```
SELECT  SUM("부서의사원수")          AS "전체사원수"
       ,COUNT(*)                    AS "부서개수"
       ,AVG("부서의사원수")          AS "부서의평균사원수"
       ,MIN("부서의사원수")          AS "부서의최소사원수"
       ,MAX("부서의사원수")          AS "부서의최대사원수"
       ,MAX("최소사원수의부서")      AS "최소사원수의부서"
       ,MAX("최대사원수의부서")      AS "최대사원수의부서"
  FROM (
    SELECT DEPT_ID
          ,COUNT(*)                                          AS "부서의사원수"
          ,FIRST_VALUE(DEPT_ID) OVER(ORDER BY COUNT(*))      AS "최소사원수의부서 "
          ,LAST_VALUE(DEPT_ID)  OVER(ORDER BY COUNT(*) ROWS BETWEEN UNBOUNDED PRECEDING
                                AND UNBOUNDED FOLLOWING)     AS " 최대사원수의부서"
      FROM T_EMP
     GROUP BY DEPT_ID
  ) E;
```

중간 집합

6. 분석함수

문제1 – 튜닝 1단계	중간집합에서 부서의 사원수, 최소/최대 사원수를 가지는 부서를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

TRACE

```
-------------------------------------------------------------------------------------------
| ID  | Operation                | Name  | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION        |       |        1 |00:00:00.0000 |       0  |       1  |
|   2 |  SORT AGGR               |       |        1 |00:00:00.0000 |       0  |       1  |
|   3 |   WINDOW                 |       |        7 |00:00:00.0000 |       0  |       1  |
|   4 |    ORDER BY (SORT)       |       |        7 |00:00:00.0000 |       0  |       1  |
|   5 |     GROUP BY (SORT)      |       |        7 |00:00:00.0029 |       0  |       1  |
|   6 |      TABLE ACCESS (FULL) | T_EMP |    20300 |00:00:00.0039 |     433  |       1  |
-------------------------------------------------------------------------------------------
```

중간 집합

참고

- 중간 집합(E 인라인뷰)에서 최소/최대 사원수를 가지는 부서를 구하기 위해서 FIRST_VALUE 와 LAST_VALUE 분석함수 중 어떤 것을 사용해도 된다. 아래 쿼리는 최대 사원수를 가지는 부서를 구할 때 Windows절 기본값을 수정하기 싫어서 FIRST_VALUE를 사용했다.

```
SELECT  SUM("부서의사원수")          AS "전체사원수"
       ,COUNT(*)                    AS "부서개수"
       ,AVG("부서의사원수")          AS "부서의평균사원수"
       ,MIN("부서의사원수")          AS "부서의최소사원수"
       ,MAX("부서의사원수")          AS "부서의최대사원수"
       ,MAX("최소사원수의부서")      AS "최소사원수의부서"
       ,MAX("최대사원수의부서")      AS "최대사원수의부서"
  FROM (
        SELECT DEPT_ID
              ,COUNT(*)                                          AS "부서의사원수"
              ,FIRST_VALUE(DEPT_ID) OVER(ORDER BY COUNT(*))      AS "최소사원수의부서 "
              ,FIRST_VALUE(DEPT_ID) OVER(ORDER BY COUNT(*) DESC) AS " 최대사원수의부서"
          FROM T_EMP
         GROUP BY DEPT_ID
       ) E;
```

실기

6.분석 함수 – 문제2

판매내역을 1번 만 읽고, 판매내역 정보와 상품코드의 판매가격 평균/합계를 조회한다.

6. 분석함수

문제2	판매내역을 1번 만 읽고, 판매내역 정보와 상품코드의 판매가격 평균/합계를 조회한다.

문제설명

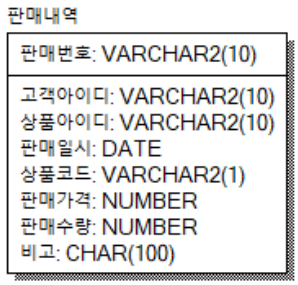
- A 회사의 인사 담당자는 판매내역 테이블을 읽고 판매번호, 상품코드, 상품코드의 평균 판매금액과 판매금액 합계를 조회하는 쿼리를 사용하고 있다. 판매코드와 상품코드로 정렬한다.

```
SALE_NO     GOODS_CD  SALE_PRICE  SALE_PRICE_AVG  SALE_PRICE_SUM
----------  --------  ----------  --------------  --------------
0000000001         A        9000            5499      6021593000
0000000002         A        2000            5499      6021593000
0000000003         A        8000            5499      6021593000
0000000004         A        1100            5499      6021593000
0000000005         A        1500            5499      6021593000
  :
0000003001         B        8000            5495      1002860000
0000003002         B        8000            5495      1002860000
0000003003         B        8000            5495      1002860000
  :
0003649800         X        4000            5491       400856000
0003649899         X        7000            5491       400856000
0003649900         X        3000            5491       400856000
0003649999         X        2000            5491       400856000
0003650000         X        1000            5491       400856000

3650000 rows selected.
```

해당 상품코드의 평균 판매가격과 판매가격 합계

- 판매내역을 1번 만 읽고, 페이지 처리를 하지 않는다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 판매내역 테이블은 월 30만 건(일 1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);

6. 분석함수

문제2	판매내역을 1번 만 읽고, 판매내역 정보와 상품코드의 판매가격 평균/합계를 조회한다.

SQL

```
SELECT S.SALE_NO,
       S.GOODS_CD,
       S.SALE_PRICE,
       ROUND(A.SALE_PRICE_AVG)    AS  SAL_PRICE_AVG,
       A.SALE_PRICE_SUM           AS  SAL_PRICE_SUM
  FROM (
         SELECT GOODS_CD,
                AVG(SALE_PRICE)   AS SALE_PRICE_AVG,
                SUM(SALE_PRICE)   AS SALE_PRICE_SUM
           FROM T_SALE_LIST        -- 총 365만 건
          GROUP BY GOODS_CD
       ) A,
       T_SALE_LIST  S
 WHERE A.GOODS_CD = S.GOODS_CD
 ORDER BY S.SALE_NO, S.GOODS_CD;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                | Name         | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)          |              |    3650K |00:00:02.3736 |       0  |       1  |
|   2 |  HASH JOIN               |              |    3650K |00:00:00.4562 |       0  |       1  |
|   3 |   GROUP BY (HASH)        |              |      24  |00:00:00.3749 |       0  |       1  |
|   4 |    TABLE ACCESS (FULL)   | T_SALE_LIST  |    3650K |00:00:00.2862 |   79473  |       1  |
|   5 |   TABLE ACCESS (FULL)    | T_SALE_LIST  |    3650K |00:00:00.3157 |   79473  |       1  |
-------------------------------------------------------------------------------------------------
   2 - access: ("T_SALE_LIST"."GOODS_CD" = "B"."GOODS_CD") (0.042)
```

분석

① A 인라인뷰에서 판매내역 테이블을 Full Scan 하고, 상품코드로 Group By 해서 상품코드의 판매가격 평균/합계를 구한다.

② 판매내역 테이블을 다시 Full Scan 하고, A 인라인뷰와 상품코드로 조인해서 판매번호, 상품코드, 상품코드의 판매금액 평균/합계를 조회한다. 판매내역 테이블을 총 2번 읽는다.

6. 분석함수

문제2 – 튜닝 1단계	중간집합에서 상품코드의 평균 판매가격과 판매가격 합계를 AVG 와 SUM 분석함수로 구한다.

튜닝내역

1. 테이블을 1번 만 읽고, 원하는 값을 조회하는 쿼리를 작성하기 위해 다음 순서로 진행한다.

   ① 최종 결과를 어떤 단위로 조회 하나? 테이블 단위, Group By 단위, 1건 중 어떤 단위인가?
   → 판매내역 테이블 단위

   ② 최종 결과를 얻기 위해 필요한 중간 집합은 무엇인가?
   → 중간 집합(S 인라인뷰)에서 상품코드의 평균 판매가격과 합계를 AVG 와 SUM 분석함수로 구한다.

   ③ 메인 쿼리에서 S 인라인뷰를 가지고 판매번호, 상품코드, 상품코드의 평균 판매금액과 판매금액 합계를 조회한다.

SQL

중간 집합

```
SELECT SALE_NO,
       GOODS_CD,
       SALE_PRICE_AVG,
       SALE_PRICE_SUM
  FROM (
        SELECT SALE_NO,
               GOODS_CD,
               ROUND(AVG(SALE_PRICE) OVER(PARTITION BY GOODS_CD))    AS SALE_PRICE_AVG,
               ROUND(SUM(SALE_PRICE) OVER(PARTITION BY GOODS_CD))    AS SALE_PRICE_SUM
          FROM T_SALE_LIST
       ) S
 ORDER BY SALES_NO, GOODS_CD;
```

→ S 인라인뷰를 제거해서 아래 쿼리로 변경이 가능하다.

```
SELECT SALE_NO,
       GOODS_CD,
       ROUND(AVG(SALE_PRICE) OVER(PARTITION BY GOODS_CD))      AS SALE_PRICE_AVG,
       ROUND(SUM(SALE_PRICE) OVER(PARTITION BY GOODS_CD))      AS SALE_PRICE_SUM
  FROM T_SALE_LIST
 ORDER BY SALES_NO, GOODS_CD;
```

TRACE

중간 집합

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                | Rows     | Elaps. Time | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)          |    3650K |00:00:02.3631 |       0  |       1  |
|   2 |  WINDOW (SIMPLE)         |    3650K |00:00:01.1124 |       0  |       1  |
|   3 |   GROUP BY               |    3650K |00:00:00.0827 |       0  |       1  |
|   4 |    ORDER BY (SORT)       |    3650K |00:00:01.2134 |       0  |       1  |
|   5 |     TABLE ACCESS (FULL)  | T_SALE_LIST |    3650K |00:00:00.3071 |   79473  |       1  |
-------------------------------------------------------------------------------------------------
```

실기

6.분석 함수 – 문제3

판매내역을 1번 만 읽고, 상품코드의 마지막 판매번호에 해당하는 판매일시와 판매수량을 조회한다.

6. 분석함수

문제3	판매내역을 1번 만 읽고, 상품코드의 마지막 판매번호에 해당하는 판매일시/판매수량을 조회한다.

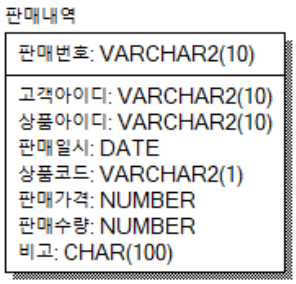
문제설명

- A 회사의 판매 담당자는 판매내역 테이블을 읽고 상품코드, 상품코드의 마지막 판매번호에 해당하는 판매일시/판매수량을 조회하는 쿼리를 사용하고 있다.

```
GOODS_CD SALE_NO_MAX SALE_DT          SALE_QY
-------- ----------- ---------- ----------
A        0003649930  2022/12/31         30
B        0003649935  2022/12/31         20
C        0003649940  2022/12/31         20
D        0003649945  2022/12/31         60
E        0003649950  2022/12/31         40
F        0003649955  2022/12/31         80
G        0003649960  2022/12/31        100
H        0003649965  2022/12/31         20
I        0003649970  2022/12/31         20
J        0003649972  2022/12/31         90
K        0003649974  2022/12/31         50
L        0003649976  2022/12/31         50
M        0003649978  2022/12/31         50
N        0003649980  2022/12/31        100
O        0003649982  2022/12/31         80
P        0003649984  2022/12/31         30
Q        0003649986  2022/12/31         10
R        0003649988  2022/12/31         10
S        0003649990  2022/12/31         10
T        0003649992  2022/12/31        100
U        0003649994  2022/12/31         60
V        0003649996  2022/12/31         70
W        0003649998  2022/12/31         40
X        0003650000  2022/12/31         80

24 rows selected.
```

- 판매내역을 1번 만 읽고, 페이지 처리를 하지 않는다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 판매내역 테이블은 월30만 건(일1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);

6. 분석함수

문제3	판매내역을 1번 만 읽고, 상품코드의 마지막 판매번호에 해당하는 판매일시/판매수량을 조회한다.

SQL

```
SELECT S.GOODS_CD,
       S.SALE_NO     AS  SALE_NO_MAX,
       S.SALE_DT     AS  SALE_DT,
       S.SALE_QY     AS  SALE_QY
  FROM (
         SELECT GOODS_CD         AS  GOODS_CD,
                MAX(SALE_NO)     AS  MAX_SALE_NO
           FROM T_SALE_LIST
          GROUP BY GOODS_CD
       ) A,
       T_SALE_LIST  S
 WHERE A.MAX_SALE_NO = S.SALE_NO  -- PK
 ORDER BY S.GOODS_CD;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation              | Name         | Rows     | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)        |              |       24 |00:00:00.0000 |        0 |        1 |
|   2 |  HASH JOIN             |              |       24 |00:00:00.1409 |        0 |        1 |
|   3 |   GROUP BY (HASH)      |              |       24 |00:00:00.2928 |        0 |        1 |
|   4 |    TABLE ACCESS (FULL) | T_SALE_LIST  |     3650K|00:00:01.4846 |    79473 |        1 |
|   5 |   TABLE ACCESS (FULL)  | T_SALE_LIST  |     3650K|00:00:00.3004 |    79473 |        1 |
-------------------------------------------------------------------------------------------------
   2 - access: (MAX("T_SALE_LIST"."SALE_NO") = "B"."SALE_NO") (0.000)
```

분석

① A 인라인뷰에서 판매내역 테이블을 Full Scan 하고, 상품코드로 Group By 해서 상품코드의 마지막 판매번호를 구한다.

② 판매내역 테이블을 다시 Full Scan 하고, 판매번호(PK)를 A 인라인뷰의 마지막 판매번호와 조인해서 상품코드, 마지막 판매번호의 판매일시/판매수량을 조회한다. 판매내역 테이블을 총 2번 읽는다.

6. 분석함수

문제3 – 튜닝 1단계	중간집합에서 상품코드의 마지막 판매번호를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

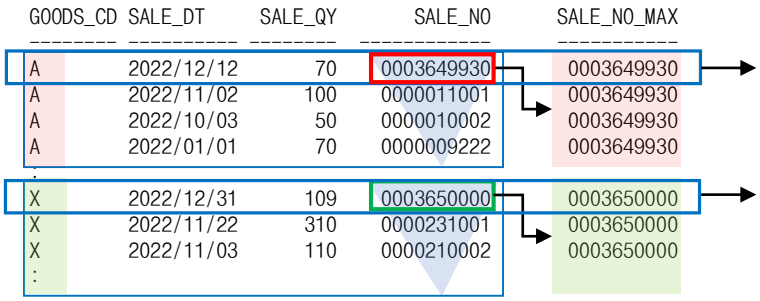
튜닝내역

1. 판매내역을 1번 만 읽고, 원하는 값을 조회하는 쿼리를 작성하기 위해 다음 순서로 진행한다.

   ① 최종 결과를 어떤 단위로 조회 하나? 테이블 단위, Group By 단위, 1건 중 어떤 단위인가?
   → Group By 상품코드

   ② 최종 결과를 얻기 위해 필요한 중간 집합은 무엇인가?
   → 중간 집합(S 인라인뷰)에서 판매내역 정보와 상품코드의 마지막 판매번호를 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

   ③ 메인 쿼리에서 S 인라인뷰를 가지고 판매번호가 마지막 판매번호 인 것을 찾아, 상품코드, 상품코드의 마지막 판매번호에 해당하는 판매일시와 판매수량을 조회한다.



GOODS_CD	SALE_DT	SALE_QY	SALE_NO	SALE_NO_MAX
A	2022/12/12	70	0003649930	0003649930
A	2022/11/02	100	0000011001	0003649930
A	2022/10/03	50	0000010002	0003649930
A	2022/01/01	70	0000009222	0003649930
:				
X	2022/12/31	109	0003650000	0003650000
X	2022/11/22	310	0000231001	0003650000
X	2022/11/03	110	0000210002	0003650000
:				

SQL

```
SELECT S.GOODS_CD,
       S.SALE_NO  AS SALE_NO_MAX,
       S.SALE_DT  AS SALE_DT,
       S.SALE_QY  AS SALE_QY
  FROM (                                                    -- 중간 집합
        SELECT GOODS_CD,
               SALE_DT,
               SALE_QY,
               SALE_NO,
               FIRST_VALUE(SALE_NO) OVER(PARTITION BY GOODS_CD ORDER BY SALE_NO DESC) SALE_NO_MAX
          FROM T_SALE_LIST
       ) S
 WHERE S.SALE_NO = S.SALE_NO_MAX
 ORDER BY GOODS_CD;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation          | Name        | Rows     | Elaps. Time | CR Gets  | Used Mem |
-------------------------------------------------------------------------------------------------
|   1 | FILTER             |             |       24 |00:00:00.1656 |        0 |       0K |
|   2 |  WINDOW (SIMPLE)   |             |    3650K |00:00:00.9935 |        0 |    264K |
|   3 |   GROUP BY         |             |    3650K |00:00:00.0960 |        0 |       0K |
|   4 |    ORDER BY (SORT) |             |    3650K |00:00:02.1707 |        0 |     53M |
|   5 |     TABLE ACCESS (FULL) | T_SALE_LIST |    3650K |00:00:00.3971 |    79473 |       0K |
-------------------------------------------------------------------------------------------------
```

중간 집합

```
   1 - filter: ("T_SALE_LIST"."SALE_NO" = FIRST_VALUE("T_SALE_LIST"."SALE_NO") OVER (PARTITION BY
"T_SALE_LIST"."GOODS_CD" ORDER BY "T_SALE_LIST"."SALE_NO" DESC)) (0.000)
```

6. 분석함수

문제3 – 튜닝 2단계	FIRST_VALUE와 LAST_VALUE 분석함수는 메모리 사용량이 많아 MAX 그룹함수로 변경한다.

튜닝내역

1. 중간집합에서 판매내역 정보와 상품코드의 마지막 판매번호를 구하기 위해, FIRST_VALUE 또는 LAST_VALUE 분석함수를 사용하면 메모리 사용량이 많아, MAX 그룹함수로 변경한다.

   ① MAX 그룹함수를 사용하여 상품코드의 마지막 판매번호를 구한다. 이때 필요한 다른 컬럼을 고정 길이로 연결(||)해서 같이 구한다.

   ② 메인에서 S 인라인뷰를 가지고 SALE_NO_MAX를 Substr으로 짤라서 조회한다.

SQL

```
SELECT GOODS_CD,
       SUBSTR(SALE_NO_MAX,1,10)          AS SALE_NO_MAX,
       SUBSTR(SALE_NO_MAX,11,8)          AS SALE_DT,
       TO_NUMBER(SUBSTR(SALE_NO_MAX,19)) AS SALE_QTY
  FROM (
        SELECT GOODS_CD,
               MAX ( SALE_NO || TO_CHAR(SALE_DT,'YYYYMMDD') || SALE_QY )   AS SALE_NO_MAX
          FROM T_SALE_LIST
         GROUP BY GOODS_CD
       ) S
 ORDER BY GOODS_CD;
```

TRACE

FIRST_VALUE 사용한 경우

```
| ID  | Operation                | Name        | Rows     | Elaps. Time  | CR Gets    | Used Mem |
-------------------------------------------------------------------------------------------------
|   1 | FILTER                   |             |       24 |00:00:00.1656 |          0 |       0K |
|   2 |  WINDOW (SIMPLE)         |             |     3650K|00:00:00.9935 |          0 |     264K |
|   3 |   GROUP BY               |             |     3650K|00:00:00.0960 |          0 |       0K |
|   4 |    ORDER BY (SORT)       |             |     3650K|00:00:02.1707 |          0 |      53M |
|   5 |     TABLE ACCESS (FULL)  | T_SALE_LIST |     3650K|00:00:00.3971 |      79473 |       0K |
-------------------------------------------------------------------------------------------------
```

MAX 사용한 경우

```
| ID  | Operation                | Name        | Rows     | Elaps. Time  | CR Gets    | Used Mem |
-------------------------------------------------------------------------------------------------
|   1 | ORDER BY (SORT)          |             |       24 |00:00:00.0000 |          0 |     145K |
|   2 |  GROUP BY (HASH)         |             |       24 |00:00:02.4179 |          0 |    3113K |
|   3 |   TABLE ACCESS (FULL)    | T_SALE_LIST |     3650K|00:00:00.3856 |      79473 |       0K |
-------------------------------------------------------------------------------------------------
```

6. 분석함수

문제3 – 튜닝 2단계	FIRST_VALUE와 LAST_VALUE 분석함수는 메모리 사용량이 많아 MAX 그룹함수로 변경한다.

참고

- 오라클도 FIRST_VALUE 와 LAST_VALUE 분석함수를 사용하면 메모리 사용량이, MAX 그룹함수 보다 많다.

FIRST_VALUE 사용한 경우

```
|Id |Operation           | Name       |A-Rows |   A-Time   |Buffers | OMem |1Mem |Used-Mem|Used-Tmp|
-------------------------------------------------------------------------------------------------
| 0 |SELECT STATEMENT    |            |    24 |00:00:03.37 |  79470 |      |     |        |        |
| 1 | VIEW               |            |    24 |00:00:03.37 |  79470 |      |     |        |        |
| 2 |  WINDOW SORT       |            |  3650K|00:00:07.44 |  79470 | 127M|3826K|  97M(1)|    115K|
| 3 |   TABLE ACCESS FULL| T_SALE_LIST|  3650K|00:00:00.52 |  79462 |      |     |        |        |
-------------------------------------------------------------------------------------------------
```

MAX 사용한 경우

```
|Id | Operation          | Name       | A-Rows |   A-Time   |Buffers |  OMem |  1Mem | Used-Mem
-----------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT   |            |     24 |00:00:02.68 |  79462 |       |       |
| 1 |  SORT GROUP BY     |            |     24 |00:00:02.68 |  79462 |  4096 |  4096 | 4096  (0)
| 2 |   TABLE ACCESS FULL| T_SALE_LIST|   3650K|00:00:00.54 |  79462 |       |       |
-----------------------------------------------------------------------------------------------
```

실기

6.분석 함수 – 문제4

성적결과를 1번만 읽고 총점 평균, 최고 총점, 최고 총점 인원수를 성적집계에 입력한다.

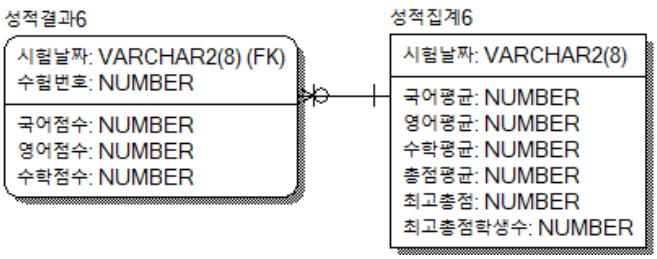
6. 분석함수

문제4	성적결과를 읽어 총점 평균, 최고 총점, 최고 총점자 수를 성적집계에 입력한다.

문제설명

- 학사 담당자 A씨는 2024/1/1에 치른 시험 성적결과를 읽어 국어 평균, 영어 평균, 수학 평균, 총점 평균, 최고 총점, 최고 총점자 수를 성적집계에 입력하는 쿼리가 필요하다.
- 성적결과를 1번 만 읽고, 페이지 처리를 하지 않는다.
- 신규 인덱스 생성 또는 인덱스 수정은 필요 없다.

ERD



- 성적결과 테이블은 2024/1/1에 1만 명 수험생이 치른 국어, 영어, 수학 과목의 성적을 저장한다.
- 성적집계 테이블은 2024/1/1에 치른 시험에 대해 국어 평균, 영어 평균, 수학 평균, 총점 평균, 최고 총점, 최고 총점자 수를 저장한다.

6. 분석함수

문제4 – 튜닝 1단계	중간집합에서 최고 총점을 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

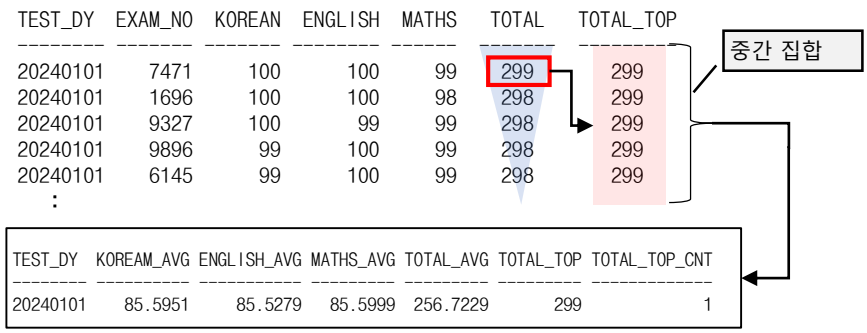
튜닝내역

1. 성적결과를 1번 만 읽고, 원하는 값을 조회하는 쿼리를 작성하기 위해 다음 순서로 진행한다.

   ① 최종 결과를 어떤 단위로 조회 하나? 테이블 단위, Group By 단위, 1건 중 어떤 단위인가?
   → 1건 단위

   ② 최종 결과를 얻기 위해 필요한 중간 집합은 무엇인가?
   → 중간 집합(S 인라인뷰)에서 성적결과 정보와 최고 총점(국어+영어+수학)을 FIRST_VALUE 또는 LAST_VALUE 분석함수로 구한다.

   ③ 메인 쿼리에서 S 인라인뷰를 가지고 시험 날짜, 국어 평균, 영어 평균, 수학 평균, 총점평균, 최고 총점, 최고 총점자 수를 조회한다.



SQL

```
INSERT INTO T_SCORE_SM6(TEST_DY, KOREAM_AVG, ENGLISH_AVG, MATHS_AVG, TOT_AVG, TOT_TOP, TOT_TOP_CNT)
SELECT TEST_DY,
       AVG(KOREAN)      AS KOREA_AVG,
       AVG(ENGLISH)     AS ENGLISH_AVG,
       AVG(MATHS)       AS MATHS_AVG,
       AVG(TOTAL)       AS TOTAL_AVG,
       AVG(TOTAL_TOP)   AS TOTAL_TOP,
       SUM(CASE WHEN TOTAL = TOTAL_TOP THEN 1  ELSE 0 END)   AS TOTAL_TOP_CNT
  FROM ( SELECT TEST_DY, KOREAN, ENGLISH, MATHS,
                KOREAN+ENGLISH+MATHS   AS TOTAL,
                FIRST_VALUE( KOREAN+ENGLISH+MATHS) OVER( ORDER BY KOREAN+ENGLISH+MATHS DESC ) AS TOTAL_TOP
           FROM T_SCORE_RESULT6
          WHERE TEST_DY =  '20240101'
       ) S
 GROUP BY TEST_DY;
```

중간 집합

TRACE

```
-----------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name               | Rows     | Elaps. Time  | CR Gets | Starts  |
-----------------------------------------------------------------------------------------------------------
| 1 | INSERT                       | T_SCORE_SM6        |        0 |00:00:00.0002 |       0 |       1 |
| 2 |  SORT AGGR                   |                    |        1 |00:00:00.0077 |       0 |       1 |
| 3 |   WINDOW (SIMPLE)            |                    |    10000 |00:00:00.0058 |       0 |       1 |
| 4 |    ORDER BY (SORT)           |                    |    10000 |00:00:00.0072 |       0 |       1 |
| 5 |     TABLE ACCESS (ROWID)     | T_SCORE_RESULT6    |    10000 |00:00:00.0041 |      41 |       1 |
| 6 |      INDEX (FAST FULL SCAN)  | PK_T_SCORE_RESULT6 |    10000 |00:00:00.0010 |      44 |       1 |
-----------------------------------------------------------------------------------------------------------
   6 - filter: ("T_SCORE_RESULT6"."TEST_DY"='20240101') (1.000)
```

중간 집합

6. 분석함수

정리

1. 테이블을 1번 만 읽고, 원하는 값을 조회하는 쿼리를 작성하기 위해 다음 순서로 진행한다.
   ① 최종 결과를 어떤 단위로 조회 하나? 테이블 단위, Group By 단위, 1건 중 어떤 단위인가?
   ② 최종 결과를 얻기 위해 필요한 중간 집합은 무엇인가?
   중간 집합을 만들기 위해 주로 FIRST_VALUE, LAST_VALUE, AVG, SUM 분석 함수를 사용한다.
   ③ 메인 쿼리에서 중간 집합을 가지고 원하는 값을 조회한다.
2. 분석함수의 구성은 다음과 같다.
   분석함수(매개변수) Over(Partition By절 Order By절 Windows절)
   - Partition By절 : 파티션 그룹을 지정한다.
   - Order By절 : 파티션 그룹 안에서 순서를 지정한다
   - Windows절 : 파티션 그룹 안에서 더 상세한 그룹으로 분할할 때 지정한다.
     기본값은 Rows Between Unbounded Preceding And Current Row(첫 번째 Row에서 현재 Row까지)이다.
3. FIRST_VALUE 분석함수는 Order By절로 정렬하여, 첫 번째 Row부터 현재 Row 중에서, 첫 번째 Row의 매개변수 값을 구한다. LAST_VALUE 분석함수는 Order By절로 정렬하여, 첫 번째 Row부터 현재 Row 중에서 가장 나중에 나오는 Row(즉 현재 Row)의 매개변수 값을 구한다.
4. FIRST_VALUE 대신 LAST_VALUE를 사용할 경우, Windows절의 기본값(Rows Between Unbounded Preceding And Current Rows)에서 ~ And Current Rows(현재 Row까지)를 ~ And Unbounded Following(마지막 Row까지)으로 변경해야 원하는 결과가 나온다.
5. FIRST_VALUE 와 LAST_VALUE 분석함수는 메모리 사용량이 많아, 가능한 경우 MAX 그룹함수로 변경한다.

실기

7. 페이지 처리

7. 페이지 처리

문제1	검색 조건과 Order By에 맞게 페이지 처리를 하는 쿼리를 작성한다.
튜닝 1단계	화면 출력하는 페이지의 Row 식별을 위해 인라인뷰를 사용하여, Order By 후에 Rownum을 적용한다.
튜닝 2단계	S1 인라인뷰 안은 Order By 때문에 전체 범위로 처리되므로, Index Only Scan으로 처리하여 테이블 블록 액세스를 피한다.
문제2	상품 테이블과 조인해서 찾은 상품 이름으로 Order By 한 후, 페이지 처리해서 조회한다.
분석	Order By를 위해 필요한 상품 이름을 찾기 위해, 판매내역→상품으로 NL 조인 시, 상품에 액세스하는 횟수가 많으면 상품의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	상품과 NL 조인을 Hash 조인으로 변경한다.
문제3	상품 테이블과 조인해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.
분석	판매내역에서 판매일시 조건을 만족하는 31만 건이 모두 상품과 조인해서 성능 저하가 발생한다.
튜닝 1단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 페이지 처리해서 조회에만 사용되면 화면에서 조회하는 건만 상품과 NL 아웃터 조인한다.
튜닝 2단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 페이지 처리해서 조회에만 사용되면 화면에서 조회하는 건만 스칼라 서브쿼리로 처리한다.
문제4	함수를 호출해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.
분석	판매내역에서 판매일시 조건을 만족하는 31만 건이 모두 함수를 호출해서 성능 저하가 발생한다.
튜닝 1단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 호출한다.
튜닝 2단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 스칼라 서브쿼리로 처리한다.
문제5	C1 인라인뷰 안의 Order By 때문에 고객과 스칼라 서브쿼리가 전체 범위로 처리된다.
분석	C1 인라인뷰 안은 Order By 때문에 고객에서 고객유형를 만족하는 22만 건이 모두 추출된다. 이때 오라클의 경우, C1 인라인뷰 안의 LAST_LOGIN_DATE 스칼라 서브쿼리가 22만 번 처리된다.
튜닝 1단계	오라클의 경우, 마지막 로그인이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 스칼라 서브쿼리를 화면에서 조회하는 건만 처리한다.
튜닝 2단계	C1 인라인뷰 안에서 인덱스로 검색 조건과 Order By를 같이 처리할 수 있으면, 고객에서 20건(2페이지 * 10건) 만 추출되고 멈춘다.
튜닝 3단계	C1 인라인뷰 안에서 인덱스 만으로 고객을 액세스하고, 추가로 필요한 컬럼은 ROWID를 사용하여 고객 테이블을 다시 액세스해서 고객 테이블 블록을 랜덤 엑세스하는 양을 줄인다.

실기

7.페이지 처리 – 문제1

검색 조건과 Order By에 맞게 페이지 처리를 하는 쿼리를 작성한다.

7. 페이지 처리

문제1	검색 조건과 Order By에 맞게 페이지 처리를 하는 쿼리를 작성한다.

문제설명

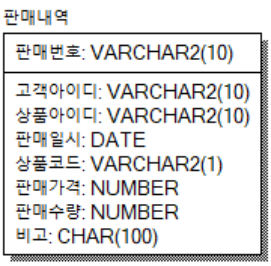
- A 회사의 판매 담당자는 검색 조건과 Order By를 입력하여 판매일시, 고객 아이디, 판매가격, 판매수량를 조회하는 쿼리가 필요하다.
- 검색 조건은 V_MON 변수에 문자 Type으로 '202201'이 입력되고, Order By는 V_ORD 변수에 'ORDER_PRICE' 또는 'ORDER_QY' 중 하나가 입력된다.
  - ORDER_PRICE : order by SALE_DT asc, CUST_ID asc, SALE_PRICE desc
  - ORDER_QY : order by SALE_DT asc, CUST_ID asc, SALE_QY desc
- 페이지 처리를 위해 V_PAGE 변수에 페이지 번호(2페이지)와 V_LIST 변수에 조회되는 건수(10건)가 입력된다.
- 필요한 정보를 조회하는 쿼리를 작성하고, 힌트를 사용하여 실행계획을 고정한다. 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

판매년월 202201 | 내림차순 정렬 ◉ 판매가격 ○ 판매수량 | 조회

번호	판매일시	고객아이디	판매가격	판매수량
11	2022/01/01	0000000062	9000	60
12	2022/01/01	0000000062	7000	50
13	2022/01/01	0000000062	5000	40
14	2022/01/01	0000000095	10000	40
15	2022/01/01	0000000095	4000	100
16	2022/01/01	0000000109	9000	70
17	2022/01/01	0000000109	6000	90
18	2022/01/01	0000000118	6000	10
19	2022/01/01	0000000118	4000	40
20	2022/01/01	0000000118	2000	70

1 2 3 4 5 >

ERD



- 판매내역 테이블은 월 30만 건(일 1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);

7. 페이지 처리

문제1 – 튜닝 1단계	인라인뷰를 사용하여, Order By 후에 Rownum을 적용한다.

튜닝내역

1. 처리 순서가 From → Where → Select의 Rownum → Order By 이므로, 인라인뷰를 사용하여 Order By 후에 Rownum을 적용한다.
   ① S1 인라인뷰 안에서 검색 조건과 Order By를 만족하는 31만 건을 찾는다.
   ② S2 인라인뷰 안에서 찾은 31만 건에 Rownum으로 번호를 부여한다.
   ③ 메인쿼리에서 RN과 ROWNUM을 사용하여 원하는 페이지를 조회한다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_ORD VARCHAR2(30);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202201';
EXEC :V_ORD := 'ORDER_PRICE';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, SALE_DT, CUST_ID, SALE_PRICE, SALE_QY
  FROM ( SELECT SALE_DT, CUST_ID, SALE_PRICE, SALE_QY,
                ROWNUM  AS RN
           FROM ( SELECT /*+ INDEX(S IX_T_SALE_LIST_DT) */
                         SALE_DT, CUST_ID, SALE_PRICE, SALE_QY
                    FROM T_SALE_LIST  S   -- 월 31만 건
                   WHERE SALE_DT BETWEEN TO_DATE(:V_MON || '01','YYYYMMDD')
                                     AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                   ORDER BY SALE_DT ASC,
                            CUST_ID ASC,
                            (CASE WHEN :V_ORD ='ORDER_PRICE' THEN SALE_PRICE
                                  WHEN :V_ORD ='ORDER_QY'    THEN SALE_QY
                             END ) DESC
                ) S1
       ) S2
 WHERE RN BETWEEN (:V_PAGE-1)*:V_LIST + 1 AND :V_PAGE * :V_LIST -- 2 페이지의 RN은 11부터 20까지
   AND ROWNUM <= :V_LIST; -- 10개가 추출되면 메모리에 있는 30만 건에서 RN 조건을 검사하지 않는다.
```

3. 원하는 페이지 조회

2. ROWNUM 으로 번호 부여

1. 조건에 맞는 것을 가져와서 정렬

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID| Operation                | Name              | Rows  | Elaps. Time  | CR Gets  | Starts |
-------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)        |                   |   10  |00:00:00.0000 |          |        |
| 2 |  FILTER                  |                   |   10  |00:00:00.0000 |        0 |      1 |
| 3 |   ORDER BY (SORT) TOP-N  |                   |   20  |00:00:00.0711 |        0 |      1 |
| 4 |    TABLE ACCESS (ROWID)  | T_SALE_LIST       |  310K |00:00:00.0666 |     6796 |      1 |
| 5 |     INDEX (RANGE SCAN)   | IX_T_SALE_LIST_DT |  310K |00:00:00.1339 |      879 |      1 |
-------------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :8) (0.100)
   2 - filter: (ROWNUM >= (((:4 - 1) * :5) + 1)) (0.100)
   5 - access: ("S"."SALE_DT" >= TO_DATE(CONCAT(:0,'01'),'YYYYMMDD')) AND ("S"."SALE_DT" <=
LAST_DAY(TO_DATE(:1,'YYYYMM'))) (1.000 * 0.088)
```

전체 처리(index→table)

7. 페이지 처리

문제1 – 튜닝 2단계	S1 인라인뷰는 Order By 때문에 전체 범위로 처리되므로, Index Only Scan으로 처리하여 테이블 블록 액세스를 피하다.

튜닝내역

1. S1 인라인뷰 안에서 판매일시 인덱스로 추출한 31만 건이 판매내역 테이블 블록을 랜덤 엑세스하여 테이블에서 31만 건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(6796 블록)이 많아 성능 저하가 발생한다. 따라서 Index Only Scan으로 처리하여 판매내역 테이블 블록을 액세스 하지 않도록 한다.

SQL

```
CREATE INDEX IX_T_SALE_LIST_DT_CUST_PRC_QY ON T_SALE_LIST( SALE_DT, CUST_ID, SALE_PRICE, SALE_QY );

VARIABLE V_MON VARCHAR2(8);
VARIABLE V_ORD VARCHAR2(30);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202201';
EXEC :V_ORD := 'ORDER_PRICE';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, SALE_DT, CUST_ID, SALE_PRICE, SALE_QY
  FROM ( SELECT SALE_DT, CUST_ID, SALE_PRICE, SALE_QY,
                ROWNUM  AS RN
           FROM ( SELECT /*+ INDEX(S IX_T_SALE_LIST_DT_CUST_PRC_QY) */
                         SALE_DT, CUST_ID, SALE_PRICE, SALE_QY
                    FROM T_SALE_LIST  S   -- 월 31만 건
                   WHERE SALE_DT BETWEEN TO_DATE(:V_MON || '01','YYYYMMDD')
                                     AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                   ORDER BY SALE_DT ASC,
                            CUST_ID ASC,
                            (CASE WHEN :V_ORD ='ORDER_PRICE' THEN SALE_PRICE
                                  WHEN :V_ORD ='ORDER_QY'    THEN SALE_QY
                             END ) DESC
                ) S1
       ) S2
 WHERE RN BETWEEN (:V_PAGE-1)*:V_LIST + 1 AND :V_PAGE * :V_LIST --2 페이지의 RN은 11부터 20까지
   AND ROWNUM <= :V_LIST;-- 10개가 추출되면 메모리에 있는 30만 건에서 RN 조건을 검사하지 않는다.

DROP INDEX IX_T_SALE_LIST_DT_CUST_PRC_QY;
```

3. 원하는 페이지 조회

2. ROWNUM 으로 번호 부여

1. 조건에 맞는 것을 가져와서 정렬

TRACE

```
-------------------------------------------------------------------------------------------------------------
| ID| Operation                | Name                        | Rows | Elaps. Time | CR Gets| Starts |
-------------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)        |                             |   10 |00:00:00.0000|      0|      1 |
| 2 |  FILTER                  |                             |   10 |00:00:00.0000|      0|      1 |
| 3 |   ORDER BY (SORT) TOP-N  |                             |   20 |00:00:00.0667|      0|      1
| 4 |    INDEX (RANGE SCAN)    |IX_T_SALE_LIST_DT_CUST_PRC_QY| 310K |00:00:00.2393|   1629|      1
-------------------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :8) (0.100)
   2 - filter: (ROWNUM >= (((:4 - 1) * :5) + 1)) (0.100)
   4 - access: ("S"."SALE_DT" >= TO_DATE(CONCAT(:0,'01'),'YYYYMMDD')) AND ("S"."SALE_DT" <=
LAST_DAY(TO_DATE(:1,'YYYYMM'))) (1.000 * 0.088)
```

전체 처리(index only scan)

실기

7.페이지 처리 – 문제2

상품 테이블과 조인해서 찾은 상품 이름으로 Order By 한 후, 페이지 처리해서 조회한다.

7. 페이지 처리

문제2	상품 테이블과 조인해서 찾은 상품 이름으로 Order By 한 후, 페이지 처리해서 조회한다.

문제설명

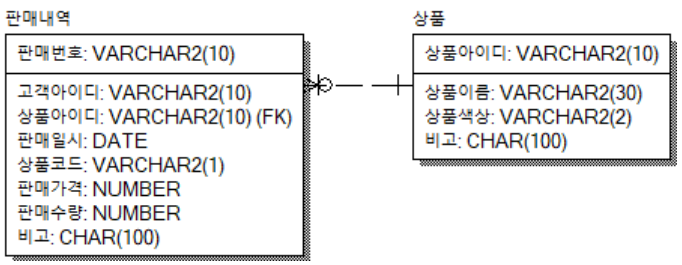
- A 회사의 판매 담당자는 검색 조건을 입력하여 고객 아이디, **상품 이름**, 판매금액으로 Order By하여 조회하는 화면을 사용하고 있다.
- 검색 조건은 V_MON 변수에 문자 Type으로 '202205'가 입력되고, 페이지 처리를 위해 V_PAGE 변수에 페이지 번호(2페이지)와 V_LIST 변수에 조회되는 건수(10건)가 입력된다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

판매년월 202205 조회

번호	고객아이디	상품이름	판매금액	판매일시
11	0000000004	PROD-0000000035	300000	2022/05/05
12	0000000004	PROD-0000000063	80000	2022/05/21
13	0000000004	PROD-0000000084	60000	2022/05/14
14	0000000004	PROD-0000000100	100000	2022/05/10
15	0000000005	PROD-0000000065	560000	2022/05/23
16	0000000005	PROD-0000000066	810000	2022/05/18
17	0000000005	PROD-0000000075	300000	2022/05/17
18	0000000006	PROD-0000000020	120000	2022/05/29
19	0000000006	PROD-0000000020	500000	2022/05/17
20	0000000006	PROD-0000000020	810000	2022/05/15

1 2 3 4 5 >

ERD



- 판매내역 테이블은 월 31만 건(일 1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);
- 상품 테이블은 총 1만 개 상품에 대한 정보를 저장한다.

7. 페이지 처리

문제2	상품 테이블과 조인해서 찾은 상품 이름으로 Order By 한 후, 페이지 처리해서 조회한다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+ LEADING(S G) INDEX(S IX_T_SALE_LIST_DT) USE_NL(G) */
                       S.CUST_ID, G.GOODS_NM, (S.SALE_PRICE * S.SALE_QY) TOTAL_PRICE, S.SALE_DT
                  FROM T_SALE_LIST  S              -- 월 31만 건
                       LEFT OUTER JOIN T_GOODS G  -- 총 1만 건
                                    ON (S.GOODS_ID = G.GOODS_ID)
                 WHERE S.SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                     AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY S.CUST_ID, G.GOODS_NM, (S.SALE_PRICE * S.SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name              | Rows  | Elaps. Time  | CR Gets  | Starts   |
-------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)            |                   |   10  |00:00:00.0000 |       0  |       1  |
| 2 |  FILTER                      |                   |   10  |00:00:00.0000 |       0  |       1  |
| 3 |   ORDER BY (SORT) TOP-N      |                   |   20  |00:00:00.0831 |       0  |       1  |
| 4 |    INDEX JOIN (LEFT OUTER)   |                   |  310K |00:00:00.0083 |       0  |       1  |
| 5 |     TABLE ACCESS (ROWID)     | T_SALE_LIST       |  310K |00:00:00.0587 |    6803  |       1  |
| 6 |      INDEX (RANGE SCAN)      | IX_T_SALE_LIST_DT |  310K |00:00:00.1798 |     879  |       1  |
| 7 |     TABLE ACCESS (ROWID)     | T_GOODS           |  310K |00:00:00.0151 |    310K  |    310K  |
| 8 |      INDEX (UNIQUE SCAN)     | PK_T_GOODS        |  310K |00:00:00.0933 |    620K  |    310K  |
-------------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :6) (0.100)
   2 - filter: (ROWNUM >= (((:2 - 1) * :3) + 1)) (0.100)
   6 - access: ("S"."SALE_DT" >= TO_DATE(:0,'YYYYMM')) AND ("S"."SALE_DT" <=
LAST_DAY(TO_DATE(:1,'YYYYMM'))) (1.000 * 0.418)
   8 - access: ("G"."GOODS_ID" = "S"."GOODS_ID") (0.000)
```

분석

① S1 인라인뷰 안에서 인덱스로 판매일시에 해당하는 31만 건을 찾아, 상품 이름을 찾기 위해 판매내역→상품으로 NL 아웃터 조인한다.

② 상품의 인덱스를 31만 번 액세스 한 후, 31만 건을 추출한다. 이때 인덱스 블록을 랜덤 액세스하는 양(62만 블록)이 많아 성능 저하가 발생한다.

③ 인덱스로 추출한 31만 건이 테이블 블록을 랜덤 액세스해서 31만 건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(31만 블록)이 많아 성능 저하가 발생한다.

7. 페이지 처리

문제2 – 튜닝 1단계	상품과 NL 조인을 Hash 조인으로 변경한다.

튜닝내역

1. 판매내역→상품으로 NL 아웃터 조인 시, 상품으로 액세스 횟수가 많아 상품의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 상품 테이블 크기 보다 많아 성능 저하가 발생한다. 따라서 상품 테이블을 Full Scan 해서 Hash 조인한다.

   ① LEADING, INDEX, FULL 과 USE_HASH 힌트로 지정한 인덱스를 사용해서 판매내역→상품으로 Hash 조인한다. 이때 먼저 액세스 된 판매내역으로 Hash Table이 생성된다.

   ② 그러나 SWAP_JOIN_INPUTS 힌트를 추가로 사용하여 상품으로 Hash Table을 만든다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+ LEADING(S G) INDEX(S IX_T_SALE_LIST_DT) FULL(G)
                           USE_HASH(G) SWAP_JOIN_INPUTS(G) */
                       S.CUST_ID, G.GOODS_NM, (S.SALE_PRICE * S.SALE_QY) TOTAL_PRICE, S.SALE_DT
                  FROM T_SALE_LIST  S              -- 월 31만 건
                       LEFT OUTER JOIN T_GOODS G  -- 총 1만 건
                                    ON (S.GOODS_ID = G.GOODS_ID)
                 WHERE S.SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                     AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY S.CUST_ID, G.GOODS_NM, (S.SALE_PRICE * S.SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID| Operation                      | Name              | Rows    | Elaps. Time  |CR Gets | Starts |
-------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)              |                   |      10 |00:00:00.0001 |      0 |      1 |
| 2 |  FILTER                        |                   |      10 |00:00:00.0000 |      0 |      1 |
| 3 |   ORDER BY (SORT) TOP-N        |                   |      20 |00:00:00.0962 |      0 |      1 |
| 4 |    HASH JOIN (REVERSE LEFT OUTER)|                 |    310K |00:00:00.0549 |      0 |      1 |
| 5 |     TABLE ACCESS (FULL)        | T_GOODS Hash Table|   10000 |00:00:00.0013 |    213 |      1 |
| 6 |     TABLE ACCESS (ROWID)       | T_SALE_LIST       |    310K |00:00:00.0725 |   6803 |      1 |
| 7 |      INDEX (RANGE SCAN)        | IX_T_SALE_LIST_DT |    310K |00:00:00.0029 |    879 |      1 |
-------------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :4) (0.100)
   2 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   4 - access: ("G"."GOODS_ID" = "S"."GOODS_ID") (0.000)
   7 - access: ("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.418)
```

실기

7.페이지 처리 – 문제3

상품 테이블과 조인해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

7. 페이지 처리

문제3	상품 테이블과 조인해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

문제설명

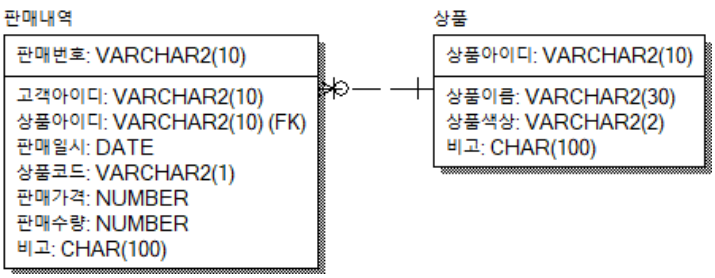
- A 회사의 판매 담당자는 검색 조건을 입력하여 판매정보를 고객 아이디, 상품 이름, 상품 아이디, 판매금액으로 Order By하여 조회하는 화면을 사용하고 있다.
- 검색 조건은 V_MON 변수에 문자 Type으로 '202205'가 입력되고, 페이지 처리를 위해 V_PAGE 변수에 페이지 번호(2페이지)와 V_LIST 변수에 조회되는 건수(10건)가 입력된다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

판매년월 202205 조회

번호	고객아이디	상품이름	판매금액	판매일시
11	0000000004	PROD-0000000035	300000	2022/05/05
12	0000000004	PROD-0000000063	80000	2022/05/21
13	0000000004	PROD-0000000084	60000	2022/05/14
14	0000000004	PROD-0000000100	100000	2022/05/10
15	0000000005	PROD-0000000065	560000	2022/05/23
16	0000000005	PROD-0000000066	810000	2022/05/18
17	0000000005	PROD-0000000075	300000	2022/05/17
18	0000000006	PROD-0000000020	120000	2022/05/29
19	0000000006	PROD-0000000020	500000	2022/05/17
20	0000000006	PROD-0000000020	810000	2022/05/15

1 2 3 4 5 >

ERD



- 판매내역 테이블은 월 31만 건(일 1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);
- 상품 테이블은 총 1만 개 상품에 대한 정보를 저장한다.

7. 페이지 처리

문제3	상품 테이블과 조인해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+ LEADING(S G) INDEX(S IX_T_SALE_LIST_DT) FULL(G)
                       USE_HASH(G) SWAP_JOIN_INPUTS(G) */
                       S.CUST_ID, G.GOODS_NM, (S.SALE_PRICE * S.SALE_QY) TOTAL_PRICE, S.SALE_DT
                  FROM T_SALE_LIST  S              -- 월 31만 건
                       LEFT OUTER JOIN T_GOODS G  -- 총 1만 건
                                    ON (S.GOODS_ID = G.GOODS_ID)
                 WHERE S.SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                     AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY S.CUST_ID, G.GOODS_NM, S.GOODS_ID, (S.SALE_PRICE * S.SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
| ID| Operation                          | Name             | Rows  | Elaps. Time  |CR Gets |Starts |
-------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)                  |                  |    10 |00:00:00.0000 |      0 |     1 |
| 2 |  FILTER                            |                  |    10 |00:00:00.0000 |      0 |     1 |
| 3 |   ORDER BY (SORT) TOP-N            |                  |    20 |00:00:00.0996 |      0 |     1 |
| 4 |    HASH JOIN (REVERSE LEFT OUTER)  |                  |   310K|00:00:00.0614 |      0 |     1 |
| 5 |     TABLE ACCESS (FULL)            | T_GOODS          | 10000 |00:00:00.0017 |    213 |     1 |
| 6 |     TABLE ACCESS (ROWID)           | T_SALE_LIST      |   310K|00:00:00.0738 |   6803 |     1 |
| 7 |      INDEX (RANGE SCAN)            | IX_T_SALE_LIST_DT|   310K|00:00:00.0030 |    879 |     1 |
-------------------------------------------------------------------------------------------------------
   1 - filter: (ROWNUM <= :4) (0.100)
   2 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   4 - access: ("G"."GOODS_ID" = "S"."GOODS_ID") (0.000)
   7 - access: ("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.418)
```

분석

① 판매내역에서 판매일시 조건을 만족하는 31만 건이 모두 상품과 조인을 한다.

② 페이지 처리에서 상품 이름이 검색 조건이나 Order By에 사용되지 않고, 단순히 상품 이름 조회에만 사용된다면, 판매내역에서 추출된 31만 건이 모두 상품 이름이 필요하지 않다.

7. 페이지 처리

문제3 – 튜닝 1단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 상품과 NL 아웃터 조인한다.

튜닝내역

1. 페이지 처리에서 상품 이름이 검색 조건이나 Order By에 사용되지 않고, 단순히 상품 이름 조회에만 사용되면, 판매내역에서 추출된 31만 건이 모두 상품 이름이 필요하지 않다. 화면에서 조회하는 10건 만 상품과 NL 아웃터 조인한다. 상품의 PK컬럼과 아웃터 조인을 하기 때문에 조인 후 건수가 변하지 않는다.

   ① S2 인라인뷰 안에서 검색 조건과 Order By를 만족하는 31만 건을 추출하고, Rownum으로 번호를 부여한다.

   ② LEADING 과 USE_NL 힌트로 S2 인라인뷰→상품으로 NL 아웃터 조인한다. 이때 화면에서 조회하는 10건 만 상품과 NL 아웃터 조인된다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT /*+ LEADING(S2 G) USE_NL(G) */
       RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_ID, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+ INDEX(S IX_T_SALE_LIST_DT)   */
                       SALE_DT, CUST_ID, GOODS_ID, (SALE_PRICE * SALE_QY) AS TOTAL_PRICE
                  FROM T_SALE_LIST  S            -- 월 31만 건
                 WHERE SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                   AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY CUST_ID, G.GOODS_NM, GOODS_ID, (SALE_PRICE * SALE_QY)
                ) S1
       ) S2
       LEFT OUTER JOIN T_GOODS G
                                ON (S2.GOODS_ID = G.GOODS_ID)   -- 최대 10건 만 NL 조인
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

```
-------------------------------------------------------------------------------------------
| ID  | Operation                | Name               | Rows | Elaps. Time  |CR Gets| Starts  |
-------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)          |                    |   10 |00:00:00.0000 |    0  |       1 |
| 2 |  INDEX JOIN (LEFT OUTER)   |                    |   10 |00:00:00.0000 |    0  |       1 |
| 3 |   FILTER                   |                    |   10 |00:00:00.0000 |    0  |       1 |
| 4 |    ORDER BY (SORT) TOP-N   |                    |   20 |00:00:00.0954 |    0  |       1 |
| 5 |     TABLE ACCESS (ROWID)   | T_SALE_LIST        | 310K |00:00:00.0773 | 6803  |       1 |
| 6 |      INDEX (RANGE SCAN)    | IX_T_SALE_LIST_DT  | 310K |00:00:00.0029 |  879  |       1 |
| 7 |   TABLE ACCESS (ROWID)     | T_GOODS            |   10 |00:00:00.0000 |   10  |      10 |
| 8 |    INDEX (UNIQUE SCAN)     | PK_T_GOODS         |   10 |00:00:00.0000 |   20  |      10 |
-------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :4) (0.100)
   3 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   6 - access: ("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.418)
   8 - access: ("G"."GOODS_ID" = "S"."GOODS_ID") (0.000)
```

부분범위처리(31만 건이 아닌 10건 처리)

7. 페이지 처리

문제3 – 튜닝 2단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 스칼라 서브쿼리로 처리한다.

튜닝내역

1. 페이지 처리에서 상품 이름이 검색 조건이나 Order By에 사용되지 않고, 단순히 상품 이름 조회에만 사용된다면, 판매내역에서 추출된 31만 건이 모두 상품 이름이 필요하지 않다. 화면에서 조회하는 10건 만 스칼라 서브쿼리로 상품 이름을 찾는다.
   ① S2 인라인뷰 안에서 검색 조건과 Order By를 만족하는 31만 건을 추출하여 Rownum으로 번호를 부여한다.
   ② 화면에서 조회하는 10건 만 스칼라 서브쿼리로 처리한다. 스칼라 서브쿼리의 캐싱 기능을 사용하여 실제로 상품은 8건 만 조인 된다. 중복된 상품이름은 캐싱 된 메모리에서 가져온다.

SQL

```
VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, CUST_ID,
       ( SELECT GOODS_NM
           FROM T_GOODS  G
          WHERE G.GOODS_ID = S2.GOODS_ID ) GOODS_NM, -- 최대 10건 만 스칼라 서브쿼리 처리
       TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_ID, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+ INDEX(S IX_T_SALE_LIST_DT)   */
                       CUST_ID, GOODS_ID, (SALE_PRICE * SALE_QY) TOTAL_PRICE, SALE_DT
                  FROM T_SALE_LIST  S            -- 월 31만 건
                 WHERE SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM') AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY CUST_ID, G.GOODS_NM, GOODS_ID, (SALE_PRICE * SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation              | Name        | Rows    | Elaps. Time  | CR Gets  | Starts  |
-------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)        |             |      10 |00:00:00.0001 |       24 |       1 |
| 2 |  FILTER                  |             |      10 |00:00:00.0000 |        0 |       1 |
| 3 |   ORDER BY (SORT) TOP-N  |             |      20 |00:00:00.0921 |        0 |       1 |
| 4 |    TABLE ACCESS (FULL)   | T_SALE_LIST |    310K |00:00:00.2656 |    79473 |       1 |
| 5 |  CACHE                   |             |       8 |00:00:00.0000 |        0 |       8 |
| 6 |   TABLE ACCESS (ROWID)   | T_GOODS     |       8 |00:00:00.0000 |        8 |       8 |
| 7 |    INDEX (UNIQUE SCAN)   | PK_T_GOODS  |       8 |00:00:00.0000 |       16 |       8 |
-------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :4) (0.100)
   2 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   4 - filter: ("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) (0.418 * 1.000)
   7 - access: ("G"."GOODS_ID" = :5) (0.000)
```

부분범위처리(31만 건이 아닌 10건 처리하며, 캐싱기능을 사용하여 8건 만 상품을 액세스)

실기

7.페이지 처리 – 문제4

함수를 호출해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

7. 페이지 처리

문제4	함수를 호출해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

문제설명

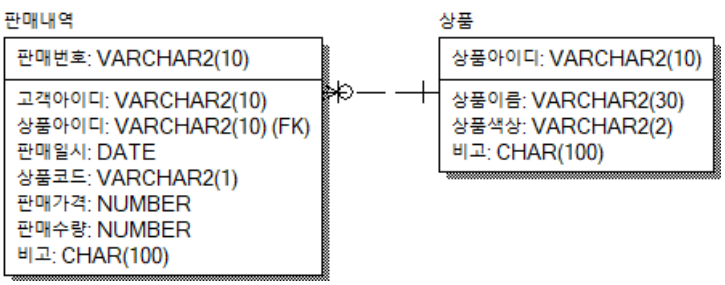
- A 회사의 판매 담당자는 검색 조건을 입력하여 판매정보를 고객 아이디, 상품 이름, 상품 아이디, 판매금액으로 Order By하여 조회하는 화면을 사용하고 있다.
- 검색 조건은 V_MON 변수에 문자 Type으로 '202205'가 입력되고, 페이지 처리를 위해 V_PAGE 변수에 페이지 번호(2페이지)와 V_LIST 변수에 조회되는 건수(10건)가 입력된다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

판매년월 202205 조회

번호	고객아이디	상품이름	판매금액	판매일시
11	0000000004	PROD-0000000035	300000	2022/05/05
12	0000000004	PROD-0000000063	80000	2022/05/21
13	0000000004	PROD-0000000084	60000	2022/05/14
14	0000000004	PROD-0000000100	100000	2022/05/10
15	0000000005	PROD-0000000065	560000	2022/05/23
16	0000000005	PROD-0000000066	810000	2022/05/18
17	0000000005	PROD-0000000075	300000	2022/05/17
18	0000000006	PROD-0000000020	120000	2022/05/29
19	0000000006	PROD-0000000020	500000	2022/05/17
20	0000000006	PROD-0000000020	810000	2022/05/15

1 2 3 4 5 >

ERD



- 판매내역 테이블은 월 31만 건(일 1만 건), 12개월(2022/1 ~ 2022/12) 동안 총 365만 건 판매내역 정보를 저장한다. 판매일시는 Date Type이다. 판매 된 상품은 상품코드(A ~ X, 총 24개)를 가지며, A ~ I는 가전으로 전체 판매의 70%를 차지하고, J ~ X는 가전외 로 전체 판매의 30%를 차지한다. A는 가전 중에서 큰 비중을 차지하는 냉장고로 전체 판매의 30%를 차지한다.
  판매일시 컬럼에 인덱스가 있다.
  CREATE INDEX IX_T_SALE_LIST_DT ON T_SALE_LIST (SALE_DT);
- 상품 테이블은 총 1만 개 상품에 대한 정보를 저장한다.

7. 페이지 처리

문제4	함수를 호출해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

SQL

```
CREATE OR REPLACE FUNCTION F_GOODS_NM( P_GOODS_ID IN VARCHAR2 ) RETURN VARCHAR2
IS
  V_GOODS_NM   VARCHAR2(30);
BEGIN
  SELECT GOODS_NM INTO V_GOODS_NM
    FROM T_GOODS
   WHERE GOODS_ID = P_GOODS_ID;

  RETURN  V_GOODS_NM;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
/

VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;


SELECT RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_NM, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+  INDEX(S IX_T_SALE_LIST_DT)  */
                       CUST_ID,
                       F_GOODS_NM(S.GOODS_ID) GOODS_NM,
                       (SALE_PRICE * SALE_QY) TOTAL_PRICE,
                       SALE_DT
                  FROM T_SALE_LIST  S
                 WHERE SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                   AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY CUST_ID, G.GOODS_NM, GOODS_ID, (SALE_PRICE * SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

7. 페이지 처리

문제4	함수를 호출해서 찾은 상품 이름이 Order By에 사용되지 않고, 페이지 처리해서 조회하는데 사용된다.

TRACE

```
-------------------------------------------------------------------------------------------------
| ID  | Operation                | Name              | Rows  | Elaps. Time  |CR Gets |Starts  |
-------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)          |                   |   10  |00:00:00.0000 |     0  |     1  |
| 2 |  COLUMN PROJECTION         |                   |   10  |00:00:00.0000 |     0  |     1  |
| 3 |   FILTER                   |                   |   10  |00:00:00.0000 |     0  |     1  |
| 4 |    ORDER BY (SORT) TOP-N   |                   |   20  |00:00:00.0966 |     0  |     1  |
| 5 |     TABLE ACCESS (ROWID)   | T_SALE_LIST       | 310K  |00:00:11.3575 |   936K |     1  |
| 6 |      INDEX (RANGE SCAN)    | IX_T_SALE_LIST_DT |  310K |00:00:00.0963 |   879  |     1  |
-------------------------------------------------------------------------------------------------
   1 - filter: (ROWNUM <= :6) (0.100)
   3 - filter: (ROWNUM >= (((:2 - 1) * :3) + 1)) (0.100)
   6 - access: ("S"."SALE_DT" >= TO_DATE(:0,'YYYYMM')) AND ("S"."SALE_DT" <=
LAST_DAY(TO_DATE(:1,'YYYYMM'))) (1.000 * 0.418)
```

분석

① S1 인라인뷰 안에서 인덱스로 판매일시에 해당하는 31만 건을 추출해서, 함수를 31만 번 호출하여 상품 이름을 찾는다. 함수를 31만 번 호출하는데 발생한 테이블 블록 액세스 양은 929K (936K – 6803) 블록이다.

7. 페이지 처리

문제4 – 튜닝 1단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 호출한다.

튜닝내역

1. 페이지 처리에서 상품 이름이 검색 조건이나 Order By에 사용되지 않고, 단순히 상품 이름 출력에만 사용된다면, 판매내역에서 추출된 31만 건이 모두 상품 이름이 필요하지 않다. 화면에 출력하는 건만 함수를 호출한다.

   ① S2 인라인뷰 안에서 검색 조건과 Order By를 만족하는 31만 건을 추출하고, Rownum으로 번호를 부여한다.

2. 화면에서 조회하는 10건 만 함수를 호출한다. 함수를 10번 호출하는데 발생한 테이블 블록 액세스 양은 30 블록이다.

SQL

```
CREATE OR REPLACE FUNCTION F_GOODS_NM( P_GOODS_ID IN VARCHAR2 ) RETURN VARCHAR2
IS
  V_GOODS_NM   VARCHAR2(30);
BEGIN
  SELECT GOODS_NM INTO V_GOODS_NM
    FROM T_GOODS
   WHERE GOODS_ID = P_GOODS_ID;

  RETURN  V_GOODS_NM;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
/

VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;


SELECT RN,
       CUST_ID,
       F_GOODS_NM(S2.GOODS_ID) GOODS_NM   -- 화면에 출력하는 건만 함수 호출
       TOTAL_PRICE,
       SALE_DT
  FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_ID, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+  INDEX(S IX_T_SALE_LIST_DT)  */
                       CUST_ID, (SALE_PRICE * SALE_QY) TOTAL_PRICE, SALE_DT
                  FROM T_SALE_LIST  S
                 WHERE SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                   AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY CUST_ID, G.GOODS_NM, GOODS_ID, (SALE_PRICE * SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

7. 페이지 처리

문제4 – 튜닝 1단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 호출한다.

TRACE

```
-------------------------------------------------------------------------------------------------
| ID| Operation                  | Name              | Rows  | Elaps. Time  |CR Gets | Starts|
-------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)          |                   |   10  |00:00:00.0004 |    30  |     1 |
| 2 |  FILTER                    |                   |   10  |00:00:00.0000 |     0  |     1 |
| 3 |   ORDER BY (SORT) TOP-N    |                   |   20  |00:00:00.0989 |     0  |     1 |
| 4 |    TABLE ACCESS (ROWID)    | T_SALE_LIST       |  310K |00:00:00.0619 |  6803  |     1 |
| 5 |     INDEX (RANGE SCAN)     | IX_T_SALE_LIST_DT |  310K |00:00:00.0024 |   879  |     1 |
-------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :4) (0.100)
   2 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   5 - access: ("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.418)
```

7. 페이지 처리

문제4 – 튜닝 2단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 스칼라 서브쿼리로 처리한다.

튜닝내역

1. 페이지 처리에서 상품 이름이 검색 조건이나 Order By에 사용되지 않고, 단순히 상품 이름 조회에만 사용된다면, 판매내역에서 추출된 31만 건이 모두 상품 이름이 필요하지 않다. 화면에서 조회하는 건만 함수를 서브쿼리로 처리한다.

   ① S2 인라인뷰 안에서 검색 조건과 Order By를 만족하는 31만 건을 추출하고, Rownum으로 번호를 부여한다.

   ② 화면에서 조회하는 10건 만 함수를 서브쿼리로 처리한다. 스칼라 서브쿼리는 캐싱 기능을 사용하여 중복된 상품 아이디는 캐싱 된 상품 이름을 반환하여, 실제로 함수는 1번 만 호출된다.

SQL

```
CREATE OR REPLACE FUNCTION F_GOODS_NM( P_GOODS_ID IN VARCHAR2 ) RETURN VARCHAR2
IS
  V_GOODS_NM   VARCHAR2(30);
BEGIN
  SELECT GOODS_NM INTO V_GOODS_NM
    FROM T_GOODS
   WHERE GOODS_ID = P_GOODS_ID;

  RETURN  V_GOODS_NM;
EXCEPTION
  WHEN OTHERS THEN
    RETURN NULL;
END;
/

VARIABLE V_MON VARCHAR2(8);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_MON := '202205';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;


SELECT RN, CUST_ID, TOTAL_PRICE, SALE_DT
       (SELECT F_GOODS_NM(S2.GOODS_ID) FROM DUAL) GOODS_NM,--화면 출력 건만 스칼라 서브쿼리로 함수호출
FROM (SELECT ROWNUM AS RN, CUST_ID, GOODS_ID, TOTAL_PRICE, SALE_DT
          FROM (SELECT /*+  INDEX(S IX_T_SALE_LIST_DT)  */
                       CUST_ID, (SALE_PRICE * SALE_QY) TOTAL_PRICE, SALE_DT
                  FROM T_SALE_LIST  S
                 WHERE SALE_DT BETWEEN TO_DATE(:V_MON,'YYYYMM')
                                   AND LAST_DAY(TO_DATE(:V_MON,'YYYYMM'))
                 ORDER BY CUST_ID, G.GOODS_NM, GOODS_ID, (SALE_PRICE * SALE_QY)
                ) S1
       ) S2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

7. 페이지 처리

문제4 – 튜닝 2단계	상품 이름이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 화면에서 조회하는 건만 함수를 스칼라 서브쿼리로 처리한다.

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID | Operation                    | Name              | Rows   | Elaps. Time  |CR Gets | Starts |
-------------------------------------------------------------------------------------------------------
|1  | ORDER BY (SORT)              |                   |    10  |00:00:00.0001 |     3  |      1|
|2  |  COUNT (STOP NODE)           |                   |    10  |00:00:00.0000 |     0  |      1|
|3  |   FILTER                     |                   |    10  |00:00:00.0000 |     0  |      1|
|4  |    ORDER BY (SORT) TOP-N     |                   |    20  |00:00:00.0775 |     0  |      1|
|5  |     TABLE ACCESS (ROWID)     | T_SALE_LIST       |   310K |00:00:00.0601 |  6803  |      1|
|6  |      INDEX (RANGE SCAN)      | IX_T_SALE_LIST_DT |   310K |00:00:00.0025 |   879  |      1|
|7  | CACHE                        |                   |     1  |00:00:00.0000 |     0  |      1|
|8  |  DPV                         | _VT_DUAL          |     1  |00:00:00.0001 |     3  |      1|
--- -------------------------------------------------------------------------------------------------------
   2 - filter: (ROWNUM <= :4) (0.100)
   3 - filter: (ROWNUM >= (((:0 - 1) * :1) + 1)) (0.100)
   6 - access: ("S"."SALE_DT" >= TO_DATE('2022-05-01 00:00:00','YYYY-MM-DD HH24:MI:SS')) AND
("S"."SALE_DT" <= TO_DATE('2022-05-31 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000 * 0.418)
```

실기

7.페이지 처리 – 문제5

C1 인라인뷰 안은 Order By 때문에 고객과 스칼라 서브쿼리(오라클)가 전체 범위로 처리된다.

7. 페이지 처리

문제5	C1 인라인뷰 안의 Order By 때문에 고객과 스칼라 서브쿼리(오라클)가 전체 범위로 처리된다.

문제설명

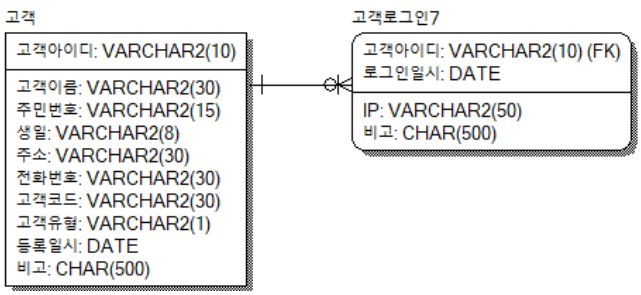
- A 온라인 쇼핑의 고객 담당자는 고객유형을 검색 조건을 입력하여, 고객 정보를 등록일시 내림차순, 고객아이디으로 Order By해서 조회하는 쿼리를 사용하고 있다. 최근 1달 이내의 로그인 정보가 있으면 마지막 로그인 일시를 출력하고, 없으면 NULL을 출력한다.
- 검색 조건은 V_CUST_TY 변수에 문자 Type으로 'A'가 입력되고, 페이지 처리를 위해 V_PAGE 변수에 페이지 번호(2페이지)와 V_LIST 변수에 조회되는 건수(10건)가 입력된다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리의 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

고객유형 [A] [조회]

번호	등록일시	고객아이디	고객이름	전화번호	마지막 로그인
11	2023/06/24	0000069631	KIM C WU	010-1234-9630	
12	2023/06/24	0000069637	KIM I UG	010-1234-9636	2024/01/20
13	2023/06/24	0000069639	KIM K BK	010-1234-9638	
14	2023/06/24	0000069642	KIM N VM	010-1234-9641	2024/01/20
15	2023/06/24	0000069650	KIM V MN	010-1234-9649	2024/01/19
16	2023/06/24	0000069654	KIM Z IE	010-1234-9653	2024/01/16
17	2023/06/24	0000069661	KIM G DK	010-1234-9660	2024/01/12
18	2023/06/24	0000069663	KIM I RK	010-1234-9662	2023/12/31
19	2023/06/23	0000069661	KIM L NP	010-1234-9665	2024/01/19
20	2023/06/23	0000069662	KIM M BJ	010-1234-9666	2024/01/20

1 2 3 4 5 >

ERD



- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);
- 고객로그인 테이블은 8개월(2023/6 ~ 2024/1) 동안, 총 70만 건 로그인 정보를 저장한다. 고객 1명이 평균 10번 로그인을 한다.

7. 페이지 처리

문제5	C1 인라인뷰 안의 Order By 때문에 C1 인라인뷰 안의 고객과 스칼라 서브쿼리(오라클)가 전체 범위로 처리된다.

SQL

```
VARIABLE V_CUST_TY VARCHAR2(1);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_CUST_TY := 'A';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT  RN, RGS_DT, CUST_ID, CUST_NM, TELNO, LAST_LOGIN_DATE
  FROM (SELECT  CUST_ID, CUST_NM, TELNO, RGS_DT, LAST_LOGIN_DATE, ROWNUM RN
          FROM ( SELECT /*+ INDEX(C IX_T_CUST_TY) */
                        CUST_ID, CUST_NM, TELNO, RGS_DT,
                        (SELECT CASE WHEN ADD_MONTHS(TO_DATE('2024/02/29','YYYY/MM/DD'),-1)
                                         <= MAX(L.LOGIN_DT)  THEN MAX(L.LOGIN_DT)
                                    ELSE NULL
                                END LAST_LOGIN_DATE
                           FROM T_CUST_LOGIN7   L
                          WHERE L.CUST_ID = C.CUST_ID
                        ) LAST_LOGIN_DATE
                   FROM T_CUST    C              -- 7만 건
                  WHERE CUST_TY = :V_CUST_TY   -- A,B,C는 각 33%, D는 1%
                  ORDER BY RGS_DT DESC, CUST_ID
                ) C1
       ) C2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

테스트를 위해 오늘을 2024/2/29 로 가정한다.

7. 페이지 처리

문제5	C1 인라인뷰 안은 Order By 때문에 고객과 스칼라 서브쿼리(오라클)가 전체 범위로 처리된다.

TRACE

티베로

```
-------------------------------------------------------------------------------------------------------
|ID | Operation                          | Name             | Rows  | Elaps. Time   |CR Gets |Starts  |
-------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)                  |                  |    10 |00:00:00.0001 |    30  |      1 |
| 2 |  FILTER                            |                  |    10 |00:00:00.0000 |     0  |      1 |
| 3 |   ORDER BY (SORT) TOP-N            |                  |    20 |00:00:00.0040 |     0  |      1
| 4 |    TABLE ACCESS (ROWID)            | T_CUST           | 22580 |00:00:00.0123 |  5780  |      1
| 5 |     INDEX (RANGE SCAN)             | IX_T_CUST_TY     | 22580 |00:00:00.0003 |    45  |      1
| 6 |  CACHE                             |                  |    10 |00:00:00.0000 |     0  |     10 |
| 7 |   COLUMN PROJECTION                |                  |    10 |00:00:00.0000 |     0  |     10 |
| 8 |    SORT AGGR                       |                  |    10 |00:00:00.0000 |     0  |     10
| 9 |     STOP LIMIT2                    |                  |    10  00:00:00.0000 |     0  |     10
|10 |      INDEX (RANGE SCAN) DESCENDING| PK_T_CUST_LOGIN7 |   100 |00:00:00.0000 |    30  |     10
-------------------------------------------------------------------------------------------------------

   1 - filter: (ROWNUM <= :5) (0.100)
   2 - filter: (ROWNUM >= (((:1 - 1) * :2) + 1)) (0.100)
   5 - access: ("C"."CUST_TY" = :0) (0.326)
  10 - access: ("L"."CUST_ID" = :6) (0.000)
```

Order By 때문에 전체범위처리

부분범위처리(22만 건이 아닌 10건 처리하며, 캐싱 기능이 있으나 중복이 없어 10건 처리됨)

오라클

```
-------------------------------------------------------------------------------------------------------
|Id | Operation                          | Name             | Starts | A-Rows |   A-Time    |Buffers |
-------------------------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT                   |                  |      1 |     10 |00:00:00.08 |  49678 |
| 1 |  SORT AGGREGATE                    |                  |  22564 |  22564 |00:00:00.05 |  43848
| 2 |   FIRST ROW                        |                  |  22564 |  22564 |00:00:00.04 |  43848
| 3 |    INDEX RANGE SCAN (MIN/MAX)      | PK_T_CUST_LOGIN7 |  22564 |  22564 |00:00:00.03 |  43848
| 4 |  SORT ORDER BY                     |                  |      1 |     10 |00:00:00.08 |  49678 |
| 5 |   COUNT STOPKEY                    |                  |      1 |     10 |00:00:00.08 |  49678 |
| 6 |    FILTER                          |                  |      1 |     10 |00:00:00.08 |  49678 |
| 7 |     VIEW                           |                  |      1 |     10 |00:00:00.08 |  49678 |
| 8 |      COUNT                         |                  |      1 |     20 |00:00:00.08 |  49678 |
| 9 |       VIEW                         |                  |      1 |     20 |00:00:00.08 |  49678 |
|10 |        SORT ORDER BY               |                  |      1 |     20 |00:00:00.08 |  49678
|11 |         TABLE ACCESS BY INDEX ROWID| T_CUST           |      1 |  22564 |00:00:00.02 |   5830
|12 |          INDEX RANGE SCAN          | IX_T_CUST_TY     |      1 |  22564 |00:00:00.01 |     48
-------------------------------------------------------------------------------------------------------

   3 - access("L"."CUST_ID"=:B1)
   5 - filter(ROWNUM<=:V_LIST)
   6 - filter((:V_PAGE-1)*:V_LIST+1<=:V_PAGE*:V_LIST)
   7 - filter(("RN"<=:V_PAGE*:V_LIST AND "RN">=(:V_PAGE-1)*:V_LIST+1))
  12 - access("C"."CUST_TY"=:CUST_TY)
```

전체범위처리(22만 건 처리하며, 캐싱 기능이 있으나 중복이 없어 22만 번 처리됨)

Order By 때문에 전체범위처리

분석

① C1 인라인뷰 안의 Order By 때문에 고객에서 고객유형를 만족하는 22만 건이 전부 추출된다.

② 이때 티베로의 경우, 경우 LAST_LOGIN_DATE 스칼라 서브쿼리가 화면에서 조회하는 10건에 대해서 만 처리되지만, 오라클의 추출된 22만 건 전부에 대해 처리되었다.

7. 페이지 처리

문제5 – 튜닝 1단계	오라클의 경우, 마지막 로그인이 검색 조건이나 Order By에 사용되지 않고, 조회에만 사용되면 스칼라 서브쿼리를 화면 조회 건만 처리한다.

튜닝내역

1. C1 인라인뷰 안에서 마지막 로그인이 검색조건이나 Order By에 사용되지 않고, 단순히 마지막 로그인 조회에만 사용되면, 고객에서 추출된 22만 건 모두 마지막 로그인이 필요하지 않다. 화면에서 조회하는 10건만 스칼라 서브쿼리로 마지막 로그인을 찾는다.
   ① C2 인라인뷰 안에서 검색 조건과 정렬 조건을 만족하는 22만 건을 추출하고, Rownum으로 번호를 부여한다.
   ② 화면에 출력하는 10건 만 스칼라 서브쿼리로 처리한다. 스칼라 서브쿼리는 캐싱 기능을 사용하여 중복 된 고객 아이디는 캐싱 된 마지막 로그인을 반환하나, 중복된 고객 아이디가 없어 고객로그인은 10번 엑세스 된다.

SQL

```
VARIABLE V_CUST_TY VARCHAR2(1);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_CUST_TY := 'A';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, RGS_DT, CUST_ID, CUST_NM, TELNO,
       ( SELECT CASE WHEN ADD_MONTHS(TO_DATE('2024/02/29','YYYY/MM/DD'),-2)
                         <= MAX(L.LOGIN_DT) THEN MAX(L.LOGIN_DT)
                     ELSE NULL
                END LAST_LOGIN_DATE
           FROM T_CUST_LOGIN7  L
          WHERE L.CUST_ID = C2.CUST_ID
       ) LAST_LOGIN_DATE
  FROM (SELECT  CUST_ID, CUST_NM, TELNO, RGS_DT, LAST_LOGIN_DATE, ROWNUM RN
          FROM ( SELECT /*+ INDEX(C IX_T_CUST_TY) */  CUST_ID, CUST_NM, TELNO, RGS_DT
                   FROM T_CUST    C              -- 7만 건
                  WHERE CUST_TY = :V_CUST_TY    -- A,B,C는 각 33%, D는 1%
                  ORDER BY RGS_DT DESC, CUST_ID
                ) C1
       ) C2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;
```

TRACE

부분범위처리(22만 건이 아닌 10건 처리하며, 캐싱 기능이 있으나 중복이 없어 10번 처리됨)

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation                      | Name                | Rows  | Time         | Gets | Starts |
-------------------------------------------------------------------------------------------------------
|  1 | COUNT (STOP NODE)              |                     |    10 |00:00:00.0002 |   30 |      1 |
|  2 |  FILTER                        |                     |    10 |00:00:00.0000 |    0 |      1 |
|  3 |   ORDER BY (SORT) TOP-N        |                     |    20 |00:00:00.0075 |    0 |      1 |
|  4 |    TABLE ACCESS (ROWID)        | T_CUST              | 22580 |00:00:00.0198 | 5780 |      1 |
|  5 |     INDEX (RANGE SCAN)         | IX_T_CUST_TY        | 22580 |00:00:00.0004 |   45 |      1 |
|  6 |  CACHE                         |                     |    10 |00:00:00.0000 |    0 |     10 |
|  7 |   COLUMN PROJECTION            |                     |    10 |00:00:00.0000 |    0 |     10 |
|  8 |    SORT AGGR                   |                     |    10 |00:00:00.0000 |    0 |     10 |
|  9 |     COLUMN PROJECTION          |                     |   100 |00:00:00.0000 |    0 |     10 |
| 10 |      INDEX (RANGE SCAN) DESCENDING| PK_T_CUST_LOGIN7 |   100 |00:00:00.0001 |   30 |     10 |
-------------------------------------------------------------------------------------------------------
```

7. 페이지 처리

문제5 – 튜닝 2단계	C1 인라인뷰 안에서 인덱스로 검색 조건과 Order By를 같이 처리할 수 있으면, 고객에서 20건(2페이지 * 10건) 만 추출되고 멈춘다.

튜닝내역

1. C1 인라인뷰 안에서 Order By 때문에 고객에서 고객유형을 만족하는 22만 건을 전부 추출해서 Order By한다. 그러나 검색 조건과 Order By를 같이 처리할 수 있는 인덱스가 있으면 22만 건을 전부 추출하지 않고 20건 만 추출하고 멈춘다. 따라서 실행계획에 정렬 작업이 없다.
   ① 검색 조건과 Order By를 같이 처리하기 위해, CUST_TY + RGS_DT DESC + CUST_ID 으로 인덱스를 생성한다.

SQL

```
CREATE INDEX IX_T_CUST_TY_RGSDT_ID ON T_CUST ( CUST_TY, RGS_DT DESC, CUST_ID);

VARIABLE V_CUST_TY VARCHAR2(1);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;

EXEC :V_CUST_TY := 'A';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT RN, RGS_DT, CUST_ID, CUST_NM, TELNO,
       ( SELECT CASE WHEN ADD_MONTHS(TO_DATE('2024/02/29','YYYY/MM/DD'),-2)
                         <= MAX(L.LOGIN_DT) THEN MAX(L.LOGIN_DT)
                     ELSE NULL
                END LAST_LOGIN_DATE
           FROM T_CUST_LOGIN7  L
          WHERE L.CUST_ID = C2.CUST_ID
       ) LAST_LOGIN_DATE
  FROM (SELECT  CUST_ID, CUST_NM, TELNO, RGS_DT, ROWNUM RN
          FROM ( SELECT /*+ INDEX(C IX_T_CUST_TY_RGSDT_ID) */  CUST_ID, CUST_NM, TELNO, RGS_DT
                   FROM T_CUST    C              -- 7만 건
                  WHERE CUST_TY = :V_CUST_TY     -- A,B,C는 각 33%, D는 1%
                  ORDER BY RGS_DT DESC, CUST_ID
                ) C1
       ) C2
 WHERE RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;

DROP INDEX IX_T_CUST_TY_RGSDT_ID;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------
|ID | Operation                      | Name                 | Rows  | Elaps. Time  |CR Gets |Starts|
-------------------------------------------------------------------------------------------------------------
| 1 | COUNT (STOP NODE)              |                      |    10 |00:00:00.0027 |     30 |    1 |
| 2 |  FILTER                        |                      |    10 |00:00:00.0000 |      0 |    1 |
| 3 |   COUNT (STOP NODE)            |                      |    20 |00:00:00.0000 |      0 |    1 |
| 4 |    TABLE ACCESS (ROWID)        | T_CUST               |    20 |00:00:00.0000 |     20 |    1 |
| 5 |     INDEX (RANGE SCAN)         | IX_T_CUST_TY_RGSDT_ID|   208 |00:00:00.0006 |      6 |    1 |
| 6 |  CACHE                         |                      |    10 |00:00:00.0000 |      0 |   10 |
| 7 |   COLUMN PROJECTION            |                      |    10 |00:00:00.0000 |      0 |   10 |
| 8 |    SORT AGGR                   |                      |    10 |00:00:00.0000 |      0 |   10 |
| 9 |     COLUMN PROJECTION          |                      |   100 |00:00:00.0000 |      0 |   10 |
|10 |      INDEX (RANGE SCAN) DESCENDING| PK_T_CUST_LOGIN7  |   100 |00:00:00.0025 |     30 |   10 |
-------------------------------------------------------------------------------------------------------------
```

20건이 추출되는 순간 멈춘다

7. 페이지 처리

문제5 – 튜닝 3단계	C1 인라인뷰 안에서 인덱스 만으로 고객을 액세스하고, 추가로 필요한 컬럼은 ROWID를 사용하여 고객 테이블을 다시 액세스한다.

튜닝내역

1. C1 인라인뷰 안에서 인덱스 만으로 검색 조건과 Order By를 처리하고, 조회에 필요한 컬럼은 ROWID를 사용하여 고객을 다시 액세스해서, 고객 테이블 블록을 랜덤 엑세스하는 양을 줄인다.

SQL

```
CREATE INDEX IX_T_CUST_TY_RGSDT_ID ON T_CUST ( CUST_TY, RGS_DT DESC, CUST_ID);

VARIABLE V_CUST_TY VARCHAR2(1);
VARIABLE V_PAGE NUMBER;
VARIABLE V_LIST NUMBER;
EXEC :V_CUST_TY := 'A';
EXEC :V_PAGE := 2;
EXEC :V_LIST := 10;

SELECT /*+ LEAING(C2 C3) USE_NL(C2) */ RN, C3.RGS_DT, C3.CUST_ID, C3.CUST_NM, C3.TELNO,
       ( SELECT CASE WHEN ADD_MONTHS(TO_DATE('2024/02/29','YYYY/MM/DD'),-2)
                         <= MAX(L.LOGIN_DT) THEN MAX(L.LOGIN_DT)
                     ELSE NULL
                END LAST_LOGIN_DATE
           FROM T_CUST_LOGIN7  L
          WHERE L.CUST_ID = C2.CUST_ID
       ) LAST_LOGIN_DATE
  FROM (SELECT  CUST_ID, CUST_NM, TELNO, RGS_DT, ROWNUM RN, RID
          FROM ( SELECT /*+ INDEX(C IX_T_CUST_TY_RGSDT_ID) */
                        CUST_ID, CUST_NM, TELNO, RGS_DT, ROWID AS RID
                   FROM T_CUST    C               -- 7만 건
                  WHERE CUST_TY = :V_CUST_TY    -- A,B,C는 각 33%, D는 1%
                  ORDER BY RGS_DT DESC, CUST_ID
                ) C1
       ) C2,
       T_CUST C3
 WHERE C2.RID = C3.ROWID
   AND RN BETWEEN ((:V_PAGE-1) * :V_LIST) + 1 AND :V_PAGE * :V_LIST
   AND ROWNUM <= :V_LIST;

DROP INDEX IX_T_CUST_TY_RGSDT_ID;
```

TRACE

20건이 추출되는 순간 멈춘다

ID	Operation	Name	Rows	Elaps. Time	CR Gets	Starts
1	COUNT (STOP NODE)		10	00:00:00.0001	30	1
2	INDEX JOIN		10	00:00:00.0000	0	1
3	FILTER		10	00:00:00.0000	0	1
4	COUNT (STOP NODE)		20	00:00:00.0000	0	1
5	COLUMN PROJECTION		208	00:00:00.0000	0	1
6	INDEX (RANGE SCAN)	IX_T_CUST_TY_RGSDT_ID	208	00:00:00.0000	6	1
7	TABLE ACCESS (ROWID)	T_CUST	10	00:00:00.0001	10	10
8	CACHE		10	00:00:00.0000	0	10
9	COLUMN PROJECTION		10	00:00:00.0000	0	10
10	SORT AGGR		10	00:00:00.0000	0	10
11	COLUMN PROJECTION		100	00:00:00.0000	0	10
12	INDEX (RANGE SCAN) DESCENDING	PK_T_CUST_LOGIN7	100	00:00:00.0000	30	10

7. 페이지 처리

정리

1. SQL은 From → Where → Select의 Rownum → Order By 순으로 처리되므로, 페이지 처리를 할 때는 인라인뷰를 사용하여 Order By 후에 Rownum을 적용한다.(문제1)
2. 페이지 처리를 할 때, 인라인뷰 안은 Order By 때문에 전체 범위로 처리된다. 그래서 Index Only Scan으로 처리하면 테이블 블록을 랜덤 액세스 하지 않아 성능이 향상된다.(문제1)
3. 페이지 처리를 할 때, 인라인뷰 안에서 검색 조건을 만족하는 모든 건이 Order By에 사용되는 상품 이름을 찾기 위해 상품과 조인을 할 경우, 상품 테이블의 크기가 작다면 NL 조인 보다 Hash 조인이 성능에 좋다.(문제2)
4. 페이지 처리를 할 때, 인라인뷰 안에서 검색 조건을 만족하는 모든 건이 상품과 조인해서 찾은 상품 이름이 검색 조건이나 Order By에 사용되지 않으면, 인라인뷰 밖에서 화면에서 조회하는 건만 상품과 NL 아웃터 조인 또는 스칼라 서브쿼리로 처리하는 것이 성능에 좋다.(문제3)
5. 페이지 처리를 할 때, 인라인뷰 안에서 함수를 호출하여 찾은 상품 이름이 검색조건이나 Order By에 사용되지 않으면, 인라인뷰 밖에서 화면에서 조회하는 건만 함수를 호출하는 것이 성능에 좋다. 이때 함수 호출을 스칼라 서브쿼리로 처리하면 캐싱 기능이 사용되어 중복된 상품 아이디는 캐싱 된 상품 이름을 반환하여 실제 함수 호출 횟수가 줄어든다.(문제4)
6. 페이지 처리를 할 때, 인라인뷰 안에서 처리된 스칼라 서브쿼리의 값이 검색 조건이나 Order By에 사용되지 않으면, 인라인뷰 밖에서 화면에서 조회하는 건만 스칼라 서브쿼리로 처리한다. (문제5)
7. 페이지 처리를 할 때, 인라인뷰 안은 Order By 때문에 전체 범위로 처리된다. 그러나 인덱스로 검색 조건과 Order By를 같이 처리할 수 있으면, 인라인뷰 안은 화면에 출력하는 건만 추출되고 멈춘다.(문제6)
8. 페이지 처리를 할 때, 인라인뷰 안에서 테이블 블록을 랜덤 엑세스하는 양을 줄여야 하는데, 이때 Index Only Scan이 불가능한 경우(필요한 컬럼 개수가 많거나, 기존 인덱스 개수가 많아 신규 인덱스 생성이 불가능 등), 검색 조건(Order By까지 만족하면 더 좋음)을 만족하는 인덱스로 해당 테이블을 먼저 엑세스하고, 추가로 필요한 컬럼은 ROWID를 사용하여 해당 테이블을 다시 액세스한다.(문제6)

실기

8.파티션

8. 파티션

문제1	모든 고객이, 가공된 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 Hash 조인한다.
분석	주문 테이블의 주문일시(파티션 키) 컬럼 가공으로 Partition Pruning이 되지 않아, 주문 테이블을 Full Scan 한다.
튜닝 1단계	Partition Pruning이 되도록 주문 테이블의 주문일시(파티션 키)컬럼 가공을 제거한다.
문제2	소수의 고객이, 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Local Prefixed 인덱스(범위, =) 사용한다.
분석	IX_P_ORDER8_DT_CUSTID 인덱스의 첫 번째 컬럼 인 ORDER_DT가 범위(Between) 조건으로 사용되어, CUST_ID가 = 조건이지만 확인으로만 사용되고 ORDER_DT 검색 조건에만 해당하는 인덱스 블록을 액세스한다.
튜닝 1단계	Local Prefixed 인덱스(범위,=)를 Local Non-Prefixed 인덱스(=,범위)로 변경한다.
튜닝 2단계	Local Non-Prefixed 인덱스(=,범위)를 Non-Partitioned(=,범위) 인덱스로 변경한다.
문제3	소수의 고객이, 아주 넓은(6개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Non-Partitioned 인덱스를 사용한다.
분석	주문일시가 매우 넓어(6개월), 주문 테이블 블록을 랜덤 액세스하는 양(5741 블록)이 많아 성능 저하가 발생한다.
튜닝 1단계	인덱스를 Index Only Scan으로 처리하여, 주문 테이블 블록을 액세스하지 않는다. (참고) Index Only Scan이 아닌, ROWID를 사용하여 주문 테이블을 다시 액세스
문제4	VIP 고객이, 특정 주문 기간 동안 주문(월 파티션) 및 결재(월 파티션)와 조인한다.
분석	Exists 서브쿼리를 FILTER로 처리하여 고객등급의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 주문일시(파티션 키) 컬럼 가공으로 Partition Pruning이 되지 않아, 주문 테이블을 Full Scan 한다. 스칼라 서브쿼리(4개)에서 결재의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.
튜닝 1단계	스칼라 서브쿼리(4개)를 조인으로, Exists 서브쿼리의 FILTER를 조인으로, 결재의 결재일시(파티션 키) 컬럼으로 Partition Pruning 사용.
문제5	최근 3개월 동안 주문유형이 A, B, C가 아닌 주문에 대해, 주문번호 별로 총판매가격을 자주 조회하는 쿼리.
분석	NOT IN(부정형) 조건 사용으로 고객 테이블을 Full Scan 한다. 주문상세에 조건이 없어 주문상세 테이블을 Full Scan 한다.
튜닝 1단계	Partition Pruning으로 찾은 3개의 테이블 파티션을 Full Scan 해서 Hash 조인한다.
튜닝 2단계	Partition Pruning으로 찾은 3개의 Local Non-Prefixed 인덱스(=,범위)로 NL 조인한다.
튜닝 3단계	Non-Partitioned 인덱스(=,범위)를 사용하여 NL 조인한다.

실기

8.파티션 – 문제1

모든 고객이, 가공된 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 Hash 조인한다.

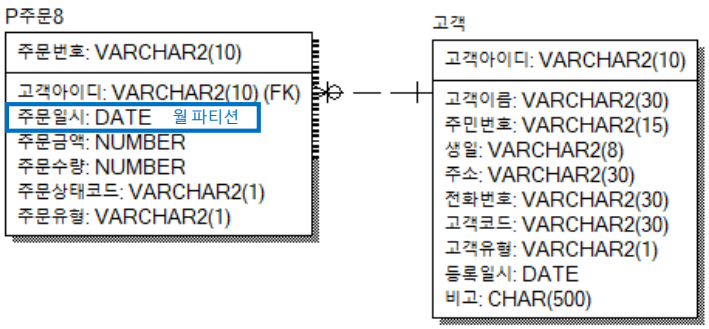
8. 파티션

문제1	모든 고객이, 가공된 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 Hash 조인한다.

문제설명

- A 온라인 쇼핑몰의 담당자는 고객이 2000/3/1 ~ 2000/05/31 동안 주문한 주문 정보(고객 이름, 주소, 전화번호, 고객코드, 주문유형, 주문수량, 주문가격)를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문 테이블은 월 90만 건, 12개월(2000/1 ~ 2000/12) 동안 총 1100만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장된다. 판매일시는 Date Type이고, 월별로 Range 파티션 되어 있다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있다. 고객은 고객코드(z0005, z0006)와 고객유형(A, B, C, D)을 가진다. 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객유형 A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다. 고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);

SQL

```
SELECT C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND TO_CHAR(O.ORDER_DT,'YYYY/MM/DD') BETWEEN '2000/03/01' AND '2000/05/31';--파티션 컬럼 가공
```

8. 파티션

문제1	모든 고객이, 가공된 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 Hash 조인한다.

TRACE

```
-------------------------------------------------------------------------------------------------------------
|ID | Operation                      | Name     | Rows    | Elaps. Time  |Pstart|Pend| CR Gets | Starts |
-------------------------------------------------------------------------------------------------------------
| 1 | HASH JOIN                      |          |  2760K |00:00:01.2256 |      |    |       0 |      1  |
| 2 |  TABLE ACCESS (FULL)           | T_CUST   | 70000  |00:00:00.0136 |      |    |    5858 |      1  |
| 3 |  PARTITION RANGE (ALL PART)    |          |  2760K |00:00:00.0001 |     1|  13|       0 |      1  |
| 4 |   TABLE ACCESS (FULL)          | P_ORDER8 |  2760K |00:00:06.4767 |      |    |   72232 |     13  |
-------------------------------------------------------------------------------------------------------------
   1 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   4 - filter: (TO_CHAR("O"."ORDER_DT",'YYYY/MM/DD') >= '2000/03/01') AND
(TO_CHAR("O"."ORDER_DT",'YYYY/MM/DD') <= '2000/05/31') (0.100 * 1.000)
```

분석

① 고객→주문 순서로 Hash 조인한다.

② 고객 테이블을 Full Scan 해서 7만 건을 가지고 Hash Table을 만든다.(Build 과정)

③ 주문 테이블의 주문일시(파티션 키) 컬럼 가공으로 Partition Pruning이 되지 않아, 주문 테이블을 Full Scan 해서 주문일자를 만족하는 276만 건을 가지고 276만 번 Hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다(Probe과정)

8. 파티션

문제1 – 튜닝 1단계	주문 테이블의 주문일시(파티션 키)컬럼으로 Partition Pruning이 되도록 컬럼 가공을 제거한다.

튜닝내역

1. 주문 테이블의 주문일시(파티션 키) 컬럼이 가공으로 Partition Pruning이 되지 않아 주문 테이블을 Full Scan 한다. 따라서 주문일시(파티션 키) 컬럼으로 Partition Pruning이 되도록 컬럼 가공을 제거한다.

SQL

```
SELECT /*+ LEADING(C O)  USE_HASH(O)  */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/05/31','YYYY/MM/DD');
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID | Operation                      | Name     | Rows   | Elaps. Time  |Pstart|Pend| CR Gets | Starts |
-------------------------------------------------------------------------------------------------------
| 1 | HASH JOIN                      |          |  2760K |00:00:01.2799 |      |    |       0 |      1 |
| 2 |  TABLE ACCESS (FULL)           | T_CUST   | 70000  |00:00:00.0160 |      |    |    5858 |      1 |
| 3 |  PARTITION RANGE (SUBSET PART) |          |  2760K |00:00:00.0000 |    3 |  5 |       0 |      1 |
| 4 |   TABLE ACCESS (FULL)          | P_ORDER8 |  2760K |00:00:00.3763 |      |    |   18193 |      3 |
-------------------------------------------------------------------------------------------------------
   1 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   4 - filter: ("O"."ORDER_DT" <= TO_DATE(' 2000-05-31 00:00:00','YYYY-MM-DD HH
```

파티션 3개만 액세스한다.

실기

8.파티션 – 문제2

소수의 고객이, 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Local Prefixed 인덱스(범위, =) 사용한다.

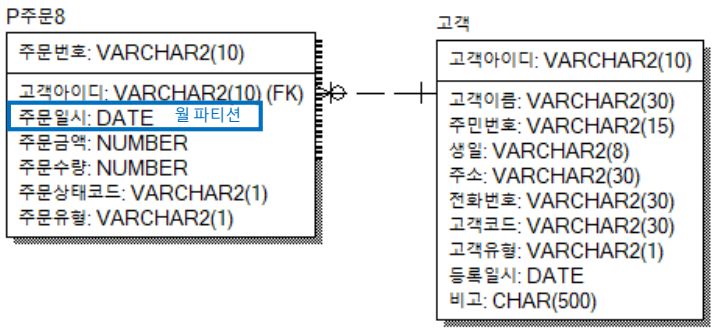
8. 파티션

문제2	소수의 고객이, 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Local Prefixed 인덱스(범위, =) 사용한다.

문제설명

- A 온라인 쇼핑몰의 담당자는 특정 전화번호와 주소를 가지는 소수의 고객이 2000/3/1 ~ 2000/05/31 동안 주문한 주문 정보(고객 이름, 주소, 전화번호, 고객코드, 주문유형, 주문수량, 주문가격)를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문 테이블은 월 90만 건, 12개월(2000/1 ~ 2000/12) 동안 총 1100만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장된다. 판매일시는 Date Type이고, 월별로 Range 파티션 되어 있다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있다. 고객은 고객코드(z0005, z0006)와 고객유형(A, B, C, D)을 가진다. 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객유형 A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다. 고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);

SQL

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX IX_P_ORDER8_DT_CUSTID ON P_ORDER8 ( ORDER_DT, CUST_ID ) LOCAL;-- Local Prefix 인덱스


SELECT /*+ LEADING(C O) USE_NL(O) INDEX_SS(O IX_P_ORDER8_DT_CUSTID) */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/05/31','YYYY/MM/DD')
   AND C.TELNO || C.ADDRESS LIKE '010-1234-4099R City%' ;  --컬럼 가공


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX IX_P_ORDER8_DT_CUSTID ON P_ORDER8;
```

8. 파티션

문제2	소수의 고객이, 넓은(3개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Local Prefixed 인덱스(범위, =) 사용한다.

TRACE

```
-----------------------------------------------------------------------------------------------------------
|ID| Operation                    | Name                 |Rows|Elaps. Time  |Pstart|Pend|CR Gets|Starts|
-----------------------------------------------------------------------------------------------------------
| 1|TABLE ACCESS (ROWID)          |P_ORDER8              |3221|00:00:00.0042|      |    | 2813 |    1 |
| 2| INDEX JOIN                   |                      |3221|00:00:00.0004|      |    |    0 |    1 |
| 3|  TABLE ACCESS (FULL)         |T_CUST                |   1|00:00:00.0202|      |    | 5858 |    1 |
| 4|  PARTITION RANGE (SUBSET PART)|                     |3221|00:00:00.0000|    3 |  5 |    0 |    1 |
| 5|   INDEX (SKIP SCAN)          |IX_P_ORDER8_DT_CUSTID |3221|00:00:00.0010|      |    |  335 |    3 |
-----------------------------------------------------------------------------------------------------------

   3 - filter: (CONCAT("C"."TELNO","C"."ADDRESS") LIKE [illegible]) (0.000)
   5 - access: ("O"."ORDER_DT" >= TO_DATE('2000-03-01 [illegible]HH24:MI:SS')) AND
("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" <= TO_DATE('2000-05-31 00:00:00','YYYY-MM-DD
HH24:MI:SS')) (1.000 * 0.000 * 0.420)
```

Local Prefixed 인덱스

티베로는 조인 시 인덱스를 Skip Scan 하지만, 오라클은 Range Scan 한다.

분석

① 고객 테이블을 Full Scan 해서 전화번호와 주소에 해당하는 1건을 찾아, 고객→주문으로 NL 조인 한다.

② 주문의 IX_P_ORDER8_DT_CUSTID 인덱스(Local Prefixed 인덱스)는 13개의 인덱스 파티션(1월 ~ 12월 파티션, Max 파티션)으로 구성되어, 3개의 인덱스 파티션(3월 ~ 5월 파티션)을 액세스하여, 3221건을 추출한다. IX_P_ORDER8_DT_CUSTID 인덱스의 첫 번째 컬럼(ORDER_DT)이 범위(Between) 조건으로 사용되면,
- 티베로의 경우, 조인 조건으로 =이 사용되야 하므로 IX_P_ORDER8_DT_CUSTID 인덱스의 첫 번째 컬럼(ORDER_DT) 조건(Between)을 Skip 하고 두 번째 컬럼(CUST_ID) 조건(=)으로 IX_P_ORDER8_DT_CUSTID 인덱스를 Skip Scan해서 335 블록을 읽는다.
- 오라클의 경우, IX_P_ORDER8_DT_CUSTID 인덱스의 두 번째 컬럼(CUST_ID) 조건(=)과 상관 없이, IX_P_ORDER8_DT_CUSTID 인덱스의 첫 번째 컬럼(ORDER_DT) 조건(Between)으로 IX_P_ORDER8_DT_CUSTID 인덱스를 Range Scan한다.

③ 인덱스로 추출한 3221건이 테이블 블록을 액세스해서 테이블에서 3221건을 추출한다.

* Prefixed 인덱스 와 Non-Prefixed 인덱스

- 테이블 파티션 키 컬럼과 상관없이, 인덱스를 파티션 하는 키 컬럼이 인덱스를 구성하는 컬럼의 왼쪽 선두에 위치하면 Prefixed 인덱스이고, 그렇지 않으면 Non-Prefixed 인덱스가 된다.
- 예로 주문 테이블의 01 인덱스가 주문일시(파티션 키) 컬럼으로 월별로 Range 파티션 되고, 주문일시 + 고객아이디 로 구성되면 Prefixed 인덱스이고, 02 인덱스가 주문일시(파티션 키) 컬럼으로 월별 Range 파티션 되고, 고객아이디 + 주문일시 로 구성되면 Non-Prefixed 인덱스가 된다.

* Local 인덱스

- 인덱스를 파티션 하는 키 컬럼과 파티션 단위가, 테이블을 파티션 하는 키 컬럼과 파티션 단위와 동일한 인덱스이다.

- 예로 주문 테이블이 판매일시 컬럼으로 월별 Range 파티션되어 있고, 01 인덱스도 판매일시 컬럼으로 월별 Range 파티션되어 있으면 01 인덱스는 Local 인덱스이다.

8. 파티션

문제2 – 튜닝 1단계	Local Prefixed 인덱스(범위,=)를 Local Non-Prefixed 인덱스(=,범위)로 변경한다.

튜닝내역

1. 주문의 IX_P_ORDER8_DT_CUSTID 인덱스(Local Prefixed 인덱스)의 첫 번째 컬럼(ORDER_DT)이 범위(Between) 조건으로 사용되면, 두 번째 컬럼(CUST_ID)의 조건과 상관 없이 첫 번째 컬럼(ORDER_DT) 조건을 만족하는 인덱스 블록(335 블록)을 액세스 한다. 이 경우, IX_P_ORDER8_CUSTID_DT 인덱스(Local Non-Prefixed 인덱스)를 첫 번째 컬럼이 = 조건이 되고 두 번째 컬럼이 범위(Between) 조건이 되도록 CUST_ID + ORDER_DT로 생성하면, CUST_ID = 조건과 ORDER_DT Between 조건을 모두 만족하는 인덱스 블록(25 블록)을 액세스한다.

* 인덱스 블록을 액세스하는 범위
범위 조건이 처음으로 나올 때 까지 사용된 모든 조건을 만족하는 인덱스 블록을 액세스한다. 예로 01 인덱스가 C1, C2, C3, C4로 구성되고 C1 = 조건, C2 = 조건, C3 Between 조건, C4 = 조건으로 사용되면, 범위 조건이 처음으로 나올 때 까지 사용된 C1 = 조건과 C2 = 조건과 C3 Between 조건을 모두 만족하는 인덱스 블록을 액세스한다.

SQL

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX IX_P_ORDER8_CUSTID_DT ON P_ORDER8 ( CUST_ID, ORDER_DT ) LOCAL; -- Local Non-Prefixed 인덱스


SELECT /*+ LEADING(C O) USE_NL(O) INDEX(C IX_T_CUST_TELNO) INDEX(O IX_P_ORDER8_CUSTID_DT) */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/05/31','YYYY/MM/DD' )
   AND C.TELNO = '010-1234-4099'
   AND C.ADDRESS LIKE 'R City';


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX IX_P_CUST_ID_ORDER8_DT;
```

TRACE

```
-----------------------------------------------------------------------------------------------------------------------
|ID| Operation                      |Name                 | Rows |Elaps. Time  |Pstart|Pend|CR Gets|Starts|
-----------------------------------------------------------------------------------------------------------------------
|1 |INDEX JOIN                      |                     | 3221 |00:00:00.0003|      |    |    0  |   1  |
|2 | TABLE ACCESS (ROWID)           | T_CUST              |    1 |00:00:00.0000|      |    |    7  |   1  |
|3 |  INDEX (RANGE SCAN)            | IX_T_CUST_TELNO     |    7 |00:00:00.0000|      |    |    2  |   1  |
|4 | PARTITION RANGE (SUBSET PART)  |                     | 3221 |00:00:00.0000|    3 |  5 |    0  |   1  |
|5 |  TABLE ACCESS (ROWID) LOCAL    | P_ORDER8            | 3221 |00:00:00.0036|      |    | 2874  |   3  |
|6 |   INDEX (RANGE SCAN)           | IX_P_ORDER8_CUSTID_DT| 3221 |00:00:00.0053|      |    |   25  |   3  |
-----------------------------------------------------------------------------------------------------------------------
   2 - filter: ("C"."ADDRESS" = 'R City') (1.000
   3 - access: ("C"."TELNO" = '010-1234-4099') (0.000)
   6 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" >= TO_DATE('2000-03-01 00:00:00','YYYY-
MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2000-05-31 00:00:00','YYYY-MM-DD HH24:MI:S
S')) (0.000 * 1.000 * 0.419)
```

Local Non-Prefixed 인덱스

8. 파티션

문제2 – 튜닝 2단계	Local Non-Prefixed 인덱스(=,범위)를 Non-Partitioned(=,범위) 인덱스로 변경한다.

튜닝내역

1. 주문의 인덱스를 첫 번째 컬럼이 = 조건이 되고, 두 번째 컬럼이 범위(Between) 조건이 되도록, CUST_ID + ORDER_DT로 IX_P_ORDER8_CUSTID_DT 인덱스(Local Non-Prefixed 인덱스)를 생성하면 CUST_ID = 조건과 ORDER_DT Between 조건을 모두 만족하는 인덱스 블록(25 블록) 을 액세스한다. 이때 Local Non-Prefixed 인덱스는 3개의 인덱스 파티션(3월 ~ 5월 파티션)을 액세스하므로 고객에서 1건이 IX_P_ORDER8_CUSTID_DT 인덱스를 사용하여 조인할 때 인덱스 수직 탐색이 3번 발생한다. Local Non-Prefixed 인덱스를 Non-Partitioned 인덱스로 변경하면 1개의 인덱스를 사용하므로 고객에서 1건이 인덱스를 사용하여 조인할 때 인덱스 수직 탐색이 1번만 발생하여 인덱스 블록을 액세스하는 양(21 블록)이 더 작아진다.

SQL

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX NP_IX_P_ORDER8_CUSTID_DT ON P_ORDER8 ( CUST_ID, ORDER_DT ); -- Non-Partitioned 인덱스


SELECT /*+ LEADING(C O) USE_NL(O) INDEX(C IX_T_CUST_TELNO) INDEX(O NP_IX_P_ORDER8_CUSTID_DT) */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/05/31','YYYY/MM/DD' )
   AND C.TELNO = '010-1234-4099'
   AND C.ADDRESS LIKE 'R City';


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX NP_IX_P_ORDER8_CUSTID_DT;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------
|ID| Operation                    |Name                    | Rows |Elaps. Time  |CR Gets | Starts|
-------------------------------------------------------------------------------------------------------------
|1 | INDEX JOIN                   |                        | 3221 |00:00:00.0003|      0 |     1 |
|2 |  TABLE ACCESS (ROWID)        | T_CUST                 |    1 |00:00:00.0000|      7 |     1 |
|3 |   INDEX (RANGE SCAN)         | IX_T_CUST_TELNO        |    7 |00:00:00.0000|      2 |     1 |
|4 |  TABLE ACCESS (ROWID) GLOBAL| P_ORDER8               | 3221 |00:00:00.0036|   2874 |     1
|5 |   INDEX (RANGE SCAN)         | NP_IX_P_ORDER8_CUSTID_DT| 3221 |00:00:00.0025|     21 |     1
-------------------------------------------------------------------------------------------------------------
   2 - filter: ("C"."ADDRESS" = 'R City') (1.000)
   3 - access: ("C"."TELNO" = '010-1234-4099') (0.000)
   5 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" >= TO_DATE('2000-03-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2000-05-31 00:00:00','YYYY-MM-DD
HH24:MI:SS')) (0.000 * 1.000 * 0.419)
```

Non-Partitioned 인덱스

실기

8.파티션 – 문제3

소수의 고객이, 아주 넓은(6개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Non-Partitioned 인덱스를 사용한다.

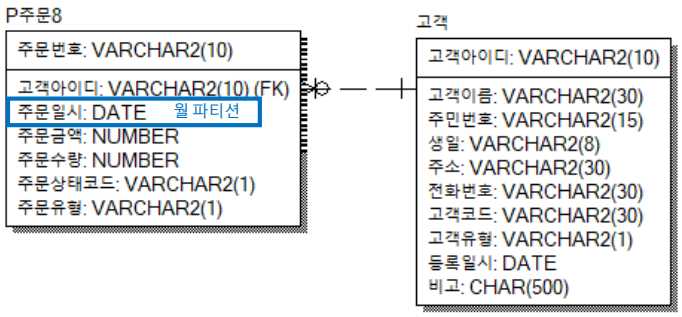
8. 파티션

문제3	소수의 고객이, 아주 넓은(6개월) 주문일시(파티션 키)를 가진 주문과 조인 시, Non-Partitioned 인덱스를 사용한다.

문제설명

- A 온라인 쇼핑몰의 담당자는 특정 전화번호와 주소를 가지는 소수의 고객이 2000/3/1 ~ 2000/08/31 동안 주문한 주문 정보(고객 이름, 주소, 전화번호, 고객코드, 주문유형, 주문수량, 주문가격)를 조회하는 쿼리를 사용하고 있다.
- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문 테이블은 월 90만 건, 12개월(2000/1 ~ 2000/12) 동안 총 1100만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장된다. 판매일시는 Date Type이고, 월별로 Range 파티션 되어 있다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있다. 고객은 고객코드(z0005, z0006)와 고객유형(A, B, C, D)을 가진다. 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객유형 A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다. 고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);

SQL

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX NP_IX_P_ORDER8_CUSTID_DT ON P_ORDER8 ( CUST_ID, ORDER_DT ); — Non-Partitioned 인덱스


SELECT /*+ LEADING(C O) USE_NL(O) INDEX(C IX_T_CUST_TELNO)INDEX(O NP_IX_P_ORDER8_CUSTID_DT) */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/08/31','YYYY/MM/DD')  -- 아주 넓은 주문일시
   AND C.TELNO = '010-1234-4099'
   AND C.ADDRESS LIKE 'R City%';


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX NP_IX_P_ORDER8_CUSTID_DT;
```

8. 파티션

문제3	특정 고객의, Non-Partitioned 인덱스를 사용하여 아주 넓은(6개월) 주문 일시(파티션 키)를 가진 주문과 조인한다.

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID| Operation                    |Name                  | Rows |Elaps. Time  |CR Gets | Starts|
-------------------------------------------------------------------------------------------------------
| 1 |INDEX JOIN                   |                      | 6463 |00:00:00.0006|     0  |    1  |
| 2 | TABLE ACCESS (ROWID)        | T_CUST               |    1 |00:00:00.0000|     7  |    1  |
| 3 |  INDEX (RANGE SCAN)         | IX_T_CUST_TELNO      |    7 |00:00:00.0000|     2  |    1  |
| 4 | TABLE ACCESS (ROWID) GLOBAL| P_ORDER8             | 6463 |00:00:00.0074|  5741  |    1  | ③
| 5 |  INDEX (RANGE SCAN)         | NP_IX_P_ORDER8_CUSTID_DT| 6463 |00:00:00.0120|    37  |    1  | ②
-------------------------------------------------------------------------------------------------------
   2 - filter: ("C"."ADDRESS" = 'R City') (1.000)
   3 - access: ("C"."TELNO" = '010-1234-4099') (0.000)
   5 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" >= TO_DATE('2000-03-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2000-08-31 00:00:00','YYYY-MM-DD
HH24:MI:SS')) (0.000 * 1.000 * 0.667)
```

Non-Partitioned 인덱스

분석

① 고객 테이블을 Index Range Scan으로 전화번호와 주소에 해당하는 1건을 찾아, 고객→주문으로 NL 조인한다.

② 고객에서 1건만 주문의 NP_IX_P_ORDER8_CUSTID_DT 인덱스(Non-Partitioned 인덱스)를 액세스하여 NL 조인하나, 주문의 주문일시 범위가 아주 넓어(6개월) 인덱스에서 많은 건(6463건)을 추출한다.

③ 인덱스로 추출한 6463건이 주문 테이블 블록을 랜덤 액세스해서 테이블에서 6463건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(5741 블록)이 많아 성능 저하가 발생한다.

8. 파티션

문제3 – 튜닝 1단계	인덱스를 Index Only Scan으로 처리하여, 주문 테이블 블록을 액세스하지 않는다.

튜닝내역

1. 주문의 NP_IX_P_ORDER8_CUSTID_DT 인덱스(Non-Partitioned 인덱스)에서 많은 건(6463건)이 추출되고, 많은 건(6463건)이 주문 테이블 블록을 랜덤 액세스하여 테이블에서 6463건을 추출한다. 이때 테이블 블록을 랜덤 액세스하는 양(5741 블록)이 많아 성능 저하가 발생한다. 따라서 주문 테이블 블록을 액세스하지 않도록 Index Only Scan으로 처리한다.

SQL

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX NP_IX_P_ORDER8_ID_DT_TY_QY_AMT ON P_ORDER8
( CUST_ID, ORDER_DT,  ORDER_TY, ORDER_QY, ORDER_AMT ); -- Non-Partitioned 인덱스를 Index Only Scan 처리


SELECT /*+ LEADING(C O) USE_NL(O) INDEX(C IX_T_CUST_TELNO)
           INDEX(O NP_IX_P_ORDER8_ID_DT_TY_QY_AMT) */
       C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ORDER_TY, O.ORDER_QY, O.ORDER_AMT
  FROM T_CUST    C,
       P_ORDER8  O  -- 월 90만 건, 12개월
 WHERE C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                      AND TO_DATE('2000/08/31','YYYY/MM/DD') -- 아주 넓은 주문일시
   AND C.TELNO = '010-1234-4099'
   AND C.ADDRESS LIKE 'R City%';


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX NP_IX_P_ORDER8_ID_DT_TY_QY_AMT;
```

TRACE

```
-----------------------------------------------------------------------------------------------------------------------
|ID | Operation            | Name                          | Rows    | Elaps. Time  | CR Gets| Starts |
-----------------------------------------------------------------------------------------------------------------------
| 1 | INDEX JOIN           |                               |  6463   |00:00:00.0006 |     0  |     1  |
| 2 |  TABLE ACCESS (ROWID)| T_CUST                        |     1   |00:00:00.0000 |     7  |     1  |
| 3 |   INDEX (RANGE SCAN) | IX_T_CUST_TELNO               |     7   |00:00:00.0000 |     2  |     1  |
| 4 | INDEX (RANGE SCAN)   | NP_IX_P_ORDER8_ID_DT_TY_QY_AMT|  6463   |00:00:00.0205 |    46  |     1  |
-----------------------------------------------------------------------------------------------------------------------
   2 - filter: ("C"."ADDRESS" = 'R City') (1
   3 - access: ("C"."TELNO" = '010-1234-4099'
   4 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" >= TO_DATE('2000-03-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2000-08-31 00:00:00','YYYY-MM-DD
HH24:MI:SS')) (0.000 * 1.000 * 0.667)
```

Index Only Scan

8. 파티션

문제3 – 튜닝 1단계	인덱스를 Index Only Scan으로 처리하여, 주문 테이블 블록을 액세스하지 않는다.

참고

- 만약에 Non-Partitioned 인덱스를 Index Only Scan으로 처리할 수 없으면, 페이지 처리(문제5 – 튜닝 3단계)에서 처럼 ROWID를 사용하여 주문 테이블을 다시 액세스해서, 주문 테이블 블록을 랜덤 액세스하는 양을 줄이면 어떨까?
- 페이지 처리는 전체 건이 아닌, 화면에서 조회하는 건만 ROWID로 다시 주문 테이블을 액세스하기 때문에 ROWID로 주문 테이블을 다시 액세스하는 블록 양이 많지 않다. 그러나 페이지 처리가 아닌 경우, 전체 건이 ROWID로 다시 주문 테이블을 액세스하면 주문 테이블 블록을 랜덤 액세스하는 양이 많아진다

```
CREATE INDEX IX_T_CUST_TELNO ON T_CUST ( TELNO );
CREATE INDEX NP_IX_P_ORDER8_CUSTID_DT ON P_ORDER8 ( CUST_ID, ORDER_DT );


SELECT /*+ LEAD ING(CO O2)  USE_NL(O2)*/
       CO.CUST_NM, CO.ADDRESS, CO.TELNO, CO.CUST_CD, O2.ORDER_TY, O2.ORDER_QY, O2.ORDER_AMT
  FROM ( SELECT /*+ LEADING(C O) USE_NL(O) INDEX(C IX_T_CUST_TELNO)
                    INDEX(O NP_IX_P_ORDER8_CUSTID_DT) */
                C.CUST_NM, C.ADDRESS, C.TELNO, C.CUST_CD, O.ROWID AS O_RID
           FROM T_CUST    C,
                P_ORDER8  O
          WHERE C.CUST_ID = O.CUST_ID
            AND O.ORDER_DT BETWEEN TO_DATE('2000/03/01','YYYY/MM/DD')
                               AND TO_DATE('2000/08/31','YYYY/MM/DD')
            AND C.TELNO = '010-1234-4099'
            AND C.ADDRESS LIKE 'R City%'
       ) CO,
       P_ORDER8 O2
 WHERE CO.O_RID = O2.ROWID;


DROP INDEX IX_T_CUST_TELNO ON T_CUST;
DROP INDEX NP_IX_P_ORDER8_CUSTID_DT;
```

```
-------------------------------------------------------------------------------------------------
|ID  | Operation           | Name                    | Rows | Elaps. Time | CR Gets|Starts |
-------------------------------------------------------------------------------------------------
|1 | INDEX JOIN            |                         | 6463 |00:00:00.001 |     0  |     1 |
|2 |  INDEX JOIN           |                         | 6463 |00:00:00.001 |     0  |     1 |
|3 |   TABLE ACCESS (ROWID)|T_CUST                   |    1 |00:00:00.000 |     7  |     1 |
|4 |    INDEX (RANGE SCAN) |IX_T_CUST_TELNO          |    7 |00:00:00.000 |     2  |     1 |
|5 |   INDEX (RANGE SCAN)  |NP_IX_P_ORDER8_CUSTID_DT | 6463 |00:00:00.000 |    37  |     1 |
|6 |  TABLE ACCESS (ROWID) |P_ORDER8                 | 6463 |00:00:00.010 |  6463  |  6463 |
-------------------------------------------------------------------------------------------------
   3 - filter: ("C"."ADDRESS" LIKE 'R City%') (1.000)
   4 - access: ("C"."TELNO" = '010-1234-4099') (0.000)
   5 - access: ("O"."CUST_ID" = "C"."CUST_ID") AND ("O"."ORDER_DT" >= TO_DATE('2000-03-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2000-08-31 00:00:00','YYYY-
MM-DD HH24:MI:SS')) (0.000 * 1.000 * 0.667)
```

실기

8.파티션 – 문제4

VIP 고객이, 특정 주문 기간 동안 주문(월 파티션) 및 결재(월 파티션)와 조인한다.

8. 파티션

문제4	VIP 고객이, 특정 주문 기간 동안 주문(월 파티션) 및 결재(월 파티션)와 조인한다.

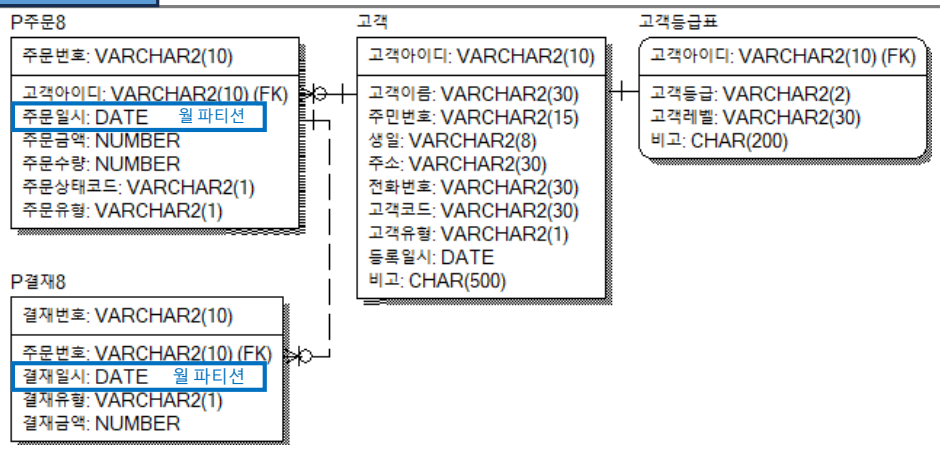
문제설명

- A 온라인 쇼핑몰의 담당자는 VIP 고객이 2000/10/01 ~ 2000/10/25 동안 주문한 주문 정보 및 결재 정보(고객 아이디, 고객 이름, 주문 횟수, 카드결재금액, 은행결재금액, 현금결재금액, 지로결재금액)를 고객 아이디와 고객 이름으로 정렬하여 조회하는 쿼리를 사용하고 있다.

```
CUST_ID    CUST_NM     ORD_CNT   CARD_AMT   BANK_AMT   CASH_AMT   JIRO_AMT
---------- --------- --------- ---------- ---------- ---------- ----------
0000000100 KIM V XG        903     113100     118700     109400     121500
0000000200 KIM R PP        870      97900     114300     112400     103500
0000000300 KIM N YG        887     116700     114900     105600     104600
0000000400 KIM J GI        902     119200     111700     112300     109200
  :
0000069300 KIM J YR          1          0          0          0          0

687 rows selected.
```

- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문 테이블은 월 90만 건, 12개월(2000/1 ~ 2000/12) 동안 총 1100만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장되며, 전체 주문의 70%는 VIP 고객의 주문이다. 주문일시는 Date Type이고, 월 단위로 Range 파티션 되어 있다. 주문은 모두 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
- 결재 테이블은 주문에 대한 결재 정보를 저장한다. 결재 오류 또는 주문자 변심 등의 이유로 결재 건수는 주문 건수의 약 90%(990만 건)가 된다. 결재일시는 Date Type이고, 월별로 Range 파티션 되어 있다. 결재유형은 A, B, C, D 값을 골고루 가진다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);
- 고객등급표 테이블은 7만명 고객에 대한 고객등급과 고객레벨을 저장한다. 고객등급은 01 ~ 11 값을 가지며, VIP 고객은 고객레벨에 VIP 값을 가지며, 전체 고객의 1%를 차지한다.

8. 파티션

문제4	VIP 고객이, 특정 주문 기간 동안 주문(월 파티션) 및 결재(월 파티션)와 조인한다.

SQL

```
CREATE INDEX IX_P_PAY8_ORDNO_TY ON P_PAY8 ( ORDER_NO, PAY_TY ) LOCAL;  -- Local Non-Prefixed 인덱스


SELECT CUST_ID, CUST_NM, COUNT(*) ORD_CNT,
       NVL(SUM(CARD_AMT),0) AS CARD_AMT,
       NVL(SUM(BANK_AMT),0) AS BANK_AMT,
       NVL(SUM(CASH_AMT),0) AS CASH_AMT,
       NVL(SUM(JIRO_AMT),0) AS JIRO_AMT
  FROM (SELECT /*+ LEADING(C O) */
               C.CUST_ID, C.CUST_NM,  O.ORDER_NO,
               ( SELECT P.PAY_AMT FROM P_PAY8 P  WHERE P.ORDER_NO = O.ORDER_NO AND P.PAY_TY = 'A') ACARD_AMT,  -- 월 80만 건, 12개월
               ( SELECT P.PAY_AMT FROM P_PAY8 P  WHERE P.ORDER_NO = O.ORDER_NO AND P.PAY_TY = 'B') BANK_AMT,
               ( SELECT P.PAY_AMT FROM P_PAY8 P  WHERE P.ORDER_NO = O.ORDER_NO AND P.PAY_TY = 'C') CASH_AMT,
               ( SELECT P.PAY_AMT FROM P_PAY8 P  WHERE P.ORDER_NO = O.ORDER_NO AND P.PAY_TY = 'D') JIRO_AMT
          FROM T_CUST    C,
               P_ORDER8  O  -- 월 90만 건, 12개월
         WHERE EXISTS ( SELECT /*+ NO_UNNEST */  1
                          FROM T_CUST_GRAD  G
                         WHERE G.CUST_ID = O.CUST_ID
                           AND G.ST_LV = 'VIP'          -- 전체 고객의 1%
                           AND ROWNUM <=1
                      )
           AND C.CUST_ID = O.CUST_ID
           AND TO_CHAR(O.ORDER_DT,'YYYY/MM/DD') BETWEEN '2000/10/01' AND '2000/10/25'
       )
 GROUP BY CUST_ID, CUST_NM
 ORDER BY CUST_ID, CUST_NM;


DROP INDEX IX_P_PAY8_ORDNO_TY;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------
|ID|Operation                          |Name             |Rows  | Elaps. Time |Pstart|Pend|CR Gets|Starts|
-------------------------------------------------------------------------------------------------------------
| 1|ORDER BY (SORT)                    |                 |  687 |00:00:00.0003|      |    |     0 |     1 |
| 2| GROUP BY (HASH)                   |                 |  687 |00:00:00.0689|      |    |     0 |     1 |
| 3|  HASH JOIN                        |                 | 527K|00:01:28.2834|      |    |    78M|     1 |
| 4|   FILTER                          |                 |  693 |00:00:00.2831|      |    |   210K|     1 |
| 5|    TABLE ACCESS (FULL)            |T_CUST           |70000 |00:00:00.0059|      |    |  5858 |     1 |
| 6|    CACHE                          |                 |70000 |00:00:00.0076|      |    |     0 | 70000 | ①
| 7|     COUNT (STOP NODE)(STOP LIMIT2)|                 |  693 |00:00:00.0029|      |    |     0 | 70000 |
| 8|      TABLE ACCESS (ROWID)         |T_CUST_GRAD      |  693 |00:00:00.0112|      |    | 70000 | 70000 |
| 9|       INDEX (UNIQUE SCAN)         |PK_T_CUST_GRAD   |70000 |00:00:00.0721|      |    |  140K| 70000 |
|10|   PARTITION RANGE (ALL PART)      |                 |  750K|00:00:00.0000|    1|  13|     0 |     1 | ②
|11|    TABLE ACCESS (FULL)            |P_ORDER8         |  750K|00:00:05.5300|      |    | 72232 |    13 |
  :
```

8. 파티션

문제4	VIP 고객이, 특정 주문 기간 동안 주문(월 파티션) 및 결재(월 파티션)와 조인한다.

TRACE

```
-------------------------------------------------------------------------------------------------------------------
|ID|Operation                          |Name              |Rows  | Elaps. Time |Pstart|Pend|CR Gets|Starts|
-------------------------------------------------------------------------------------------------------------------
 :
|12|  CACHE                            |                  |  115K|00:00:00.0337|      |    |     0 |  527K| ③
|13|   PARTITION RANGE (ALL PART)      |                  |  115K|00:00:00.1050|     1|  13|     0 |  527K|
|14|    TABLE ACCESS (ROWID) LOCAL     |P_PAY8            |  115K|00:00:00.2928|      |    |  115K|  6854K|
|15|     INDEX (RANGE SCAN)            |IX_P_PAY8_ORDNO_TY|  115K|00:00:13.5185|      |    |   19M|  6854K|
|16|  CACHE                            |                  |  121K|00:00:00.0288|      |    |     0 |  527K|
|17|   PARTITION RANGE (ALL PART)      |                  |  121K|00:00:00.1069|     1|  13|     0 |  527K|
|18|    TABLE ACCESS (ROWID) LOCAL     |P_PAY8            |  121K|00:00:00.3520|      |    |  121K|  6854K|
|19|     INDEX (RANGE SCAN)            |IX_P_PAY8_ORDNO_TY|  121K|00:00:12.6731|      |    |   19M|  6854K|
|20|  CACHE                            |                  |  121K|00:00:00.0286|      |    |     0 |  527K|
|21|   PARTITION RANGE (ALL PART)      |                  |  121K|00:00:00.1047|     1|  13|     0 |  527K|
|22|    TABLE ACCESS (ROWID) LOCAL     |P_PAY8            |  121K|00:00:00.3232|      |    |  121K|  6854K|
|23|     INDEX (RANGE SCAN)            |IX_P_PAY8_ORDNO_TY|  121K|00:00:12.8714|      |    |   19M|  6854K|
|24|  CACHE                            |                  |  116K|00:00:00.0283|      |    |     0 |  527K|
|25|   PARTITION RANGE (ALL PART)      |                  |  116K|00:00:00.1035|     1|  13|     0 |  527K|
|26|    TABLE ACCESS (ROWID) LOCAL     |P_PAY8            |  116K|00:00:00.3283|      |    |  116K|  6854K|
|27|     INDEX (RANGE SCAN)            |IX_P_PAY8_ORDNO_TY|  116K|00:00:12.6673|      |    |   19M|  6854K|
-------------------------------------------------------------------------------------------------------------------

   3 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   4 - filter: EXISTS ( SELECT /*+ NO_UNNEST */  1
                          FROM T_CUST_GRAD  G
                         WHERE G.CUST_ID = O.CUST_ID
                           AND CUST_LV = 'VIP'
                           AND ROWNUM <=1
                       ) (1.000)
   7 - filter: (ROWNUM = 1) (0.010)
   8 - filter: ("G"."CUST_LV" = 'VIP') (1.000)
   9 - access: ("G"."CUST_ID" = :0) (0.000)
  11 - filter: (TO_CHAR("O"."ORDER_DT",'YYYY/MM/DD') >= '2000/10/01') AND (TO_CHAR("O"."ORDER_DT",'YYYY/MM/DD')
<= '2000/10/25') (0.100 * 1.000)
  15 - access: ("P"."ORDER_NO" = :1) AND ("P"."PAY_TY" = 'A') (0.000 * 1.000)
  19 - access: ("P"."ORDER_NO" = :1) AND ("P"."PAY_TY" = 'B') (0.000 * 1.000)
  23 - access: ("P"."ORDER_NO" = :1) AND ("P"."PAY_TY" = 'C') (0.000 * 1.000)
  27 - access: ("P"."ORDER_NO" = :1) AND ("P"."PAY_TY" = 'D') (0.000 * 1.000)
```

분석

① 고객 테이블을 Full Scan 해서, Exists 서브쿼리를 FILTER 처리로 VIP 고객을 찾은 중간 집합(693건)으로 Hash Table을 만든다.(Build과정) 이때 고객등급의 인덱스와 테이블의 블록을 랜덤 엑세스하는 양(14만 + 7만=21만 블록)이 많아 성능 저하가 발생한다.

② 주문 테이블의 주문일시(파티션 키) 컬럼 가공으로 Partition Pruning이 되지 않아, 주문 테이블을 Full Scan 해서 주문일자를 만족하는 75만 건을 가지고, 75만 번 생성된 hash Table을 탐색하여 조인 조건을 만족하는 건을 찾는다.(Probe 과정)

③ 결재유형 별로 금액을 구하는 스칼라 서브쿼리(4개)에서 결재의 인덱스와 테이블의 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다.

8. 파티션

문제4 – 튜닝 1단계	스칼라 서브쿼리(4개)를 조인으로, Exists 서브쿼리의 FILTER를 조인으로, 결재의 결재일시(파티션 키) 컬럼으로 Partition Pruning 사용.

튜닝내역

1. Exists 서브쿼리에 UNNEST 힌트 사용 및 Rownum 제거로 고객등급→고객으로 NL 조인한다.
2. 주문의 주문일시(파티션 키) 컬럼으로 Partition Pruning이 되도록 컬럼 가공을 제거한다.
3. 스칼라 서브쿼리의 결재를 From절로 이동 및 결재일시(파티션 키) 조건 추가(결재일시는 항상 주문일시 보다 같거나 크다)로 Partition Pruning 하여 Hash 조인한다.

SQL

```
SELECT /*+ LEADING(G C O P) USE_NL(C) USE_HASH(O)  USE_HASH(P) */
       C.CUST_ID, C.CUST_NM, COUNT(O.ORDER_NO) ORD_CNT,
       NVL(SUM(CASE WHEN P.PAY_TY = 'A' THEN P.PAY_AMT ELSE NULL END),0) AS CARD_AMT,
       NVL(SUM(CASE WHEN P.PAY_TY = 'B' THEN P.PAY_AMT ELSE NULL END),0) AS BANK_AMT,
       NVL(SUM(CASE WHEN P.PAY_TY = 'C' THEN P.PAY_AMT ELSE NULL END),0) AS CASH_AMT,
       NVL(SUM(CASE WHEN P.PAY_TY = 'D' THEN P.PAY_AMT ELSE NULL END),0) AS JIRO_AMT
  FROM T_CUST    C,
       P_ORDER8  O, -- 월 90만 건, 12개월
       P_PAY8    P -- 월 80만 건, 12개월   스칼라 서브쿼리를 From절 아웃터 조인으로 변경
 WHERE EXISTS ( SELECT /*+ UNNEST */ 1
                  FROM T_CUST_GRAD  G          -- 고객등급 테이블이 작으므로 Full Scan해서 Hash 조인
                 WHERE G.CUST_ID = C.CUST_ID
                   AND G.CUST_LV = 'VIP'       -- 전체 조건의 1%
                   AND ROWNUM <=1
              )
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_DT BETWEEN TO_DATE('2000/10/01','YYYY/MM/DD') AND TO_DATE('2000/10/25','YYYY/MM/DD')
   AND O.ORDER_NO = P.ORDER_NO(+)
   AND P.PAY_DT(+) >= TO_DATE('2000/10/01','YYYY/MM/DD')  -- 조건 추가로 Partition Pruning
GROUP BY C.CUST_ID, C.CUST_NM
ORDER BY C.CUST_ID, C.CUST_NM;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------------
|ID| Operation                      | Name         | Rows  |Elaps. Time  |Pstart|Pend| CR Gets | Starts  |
-------------------------------------------------------------------------------------------------------------------
| 1| COLUMN PROJECTION              |              |  687  |00:00:00.0001|      |    |       0 |       1 |
| 2|  GROUP BY (SORT)               |              |  687  |00:00:00.1708|      |    |       0 |       1 |
| 3|   HASH JOIN (LEFT OUTER)       |              |  527K |00:00:02.0351|      |    |       0 |       1 |
| 4|    HASH JOIN                   |              |  527K |00:00:00.0797|      |    |       0 |       1 |
| 5|     INDEX JOIN                 |              |  693  |00:00:00.0002|      |    |       0 |       1 |
| 6|      TABLE ACCESS (FULL)       | T_CUST_GRAD  |  693  |00:00:00.0160|      |    |    2199 |       1 |
| 7|      TABLE ACCESS (ROWID)      | T_CUST       |  693  |00:00:00.0013|      |    |     693 |     693 |
| 8|       INDEX (UNIQUE SCAN)      | PK_T_CUST    |  693  |00:00:00.0445|      |    |    1386 |     693 |
| 9|     PARTITION RANGE (SINGLE PART)|            |  750K |00:00:00.0000|  10  | 10 |       0 |       1 |
|10|      TABLE ACCESS (FULL)       | P_ORDER8     |  750K |00:00:00.0653|      |    |    6128 |       1 |
|11|    PARTITION RANGE (SUBSET PART)|            | 2457K |00:00:00.0000|  10  | 13 |       0 |       1 |
|12|     TABLE ACCESS (FULL)        | P_PAY8       | 2457K |00:00:00.1833|      |    |   13096 |       4 |
-------------------------------------------------------------------------------------------------------------------

   3 - access: ("O"."ORDER_NO" = "P"."ORDER_NO") (0.000)
   3 - filter: ("P"."PAY_DT" >= "O"."ORDER_DT") (1.000)
   4 - access: ("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   6 - filter: ("G"."CUST_LV" = 'VIP') (0.007)
   8 - access: ("C"."CUST_ID" = "G"."CUST_ID") (0.007)
  10 - filter: ("O"."ORDER_DT" <= TO_DATE('2000-10-25 00:00:00','YYYY-MM-DD HH24:MI:SS')) (1.000)
```

실기

8.파티션 – 문제5

최근 3개월 동안 주문유형이 A, B, C가 아닌 주문에 대해, 주문번호 별로 총판매가격을 아주 자주 조회하는 쿼리.

8. 파티션

문제5	최근 3개월 동안 주문유형이 A, B, C가 아닌 주문에 대해, 주문번호 별로 총판매가격을 아주 자주 조회하는 쿼리.

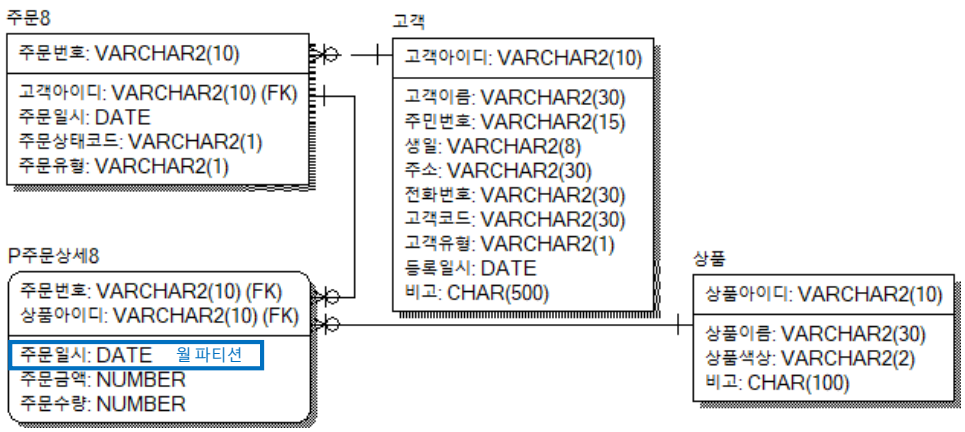
문제설명

- A 온라인 쇼핑몰의 담당자는 최근 3개월(2020/10/01 ~ 2020/12/31) 동안 주문한 주문 정보(고객 아이디, 고객 이름, 주문번호, 총판매가격)를 고객 아이디, 고객 이름과 주문번호로 정렬하여 조회하는 쿼리를 아주 자주 사용하고 있다.

```
CUST_ID    CUST_NM    ORDER_NO   TOTAL_PRICE
---------- ---------- ---------- -----------
0000060329 KIM I JL   0001494008     3590000
0000060355 KIM I YM   0001600546     1550000
0000060355 KIM I YM   0001654126     3360000
0000064175 KIM G PN   0001599752     2730000
0000064175 KIM G PN   0001801861     2210000
  :
0000069984 KIM R TH   0001753967     2880000

4487 rows selected.
```

- 아래 쿼리와 실행정보를 확인하고, 쿼리 성능을 개선하라.
- 튜닝 후 쿼리는 힌트를 사용하여 실행계획을 고정하며, 신규 인덱스를 생성할 경우 이유를 설명하라. 불필요한 인덱스를 생성하면 감점이 된다.

ERD



- 주문 테이블은 월 15만 건, 36개월(2020/1 ~ 2022/12) 동안 총 540만 건 주문 정보를 저장한다. 주문자는 고객아이디로 저장된다. 주문일시는 Date Type이다. 주문은 주문상태코드(1 ~ 5)와 주문유형(A ~ D)를 가진다. 주문상태코드는 대부분 5 값(주문확정)을 가지며, 주문유형 A, B, C 값은 주문의 33%씩 차지하며, 주문의 1%가 D 값을 가진다.
- 주문상세 테이블은 월 150만 건, 36개월(2020/1 ~ 2022/12) 동안 총 5400만 건 주문상세 정보를 저장한다. 하나의 주문은 여러 상품을 주문하는데, 1개 상품이 1개 주문상세에 해당한다. 1개 주문은 평균적으로 10개 상품을 주문한다. 주문 테이블의 주문일시를 반정규화하여 주문일시를 저장하며 Date Type이고 월별로 Range 파티션 되어 있다.
- 고객 테이블은 7만 명 고객에 대한 정보를 저장한다. 주민번호 뒷자리는 암호화되어 있고 고객코드(z0005, z0006)를 가지며, 전체 고객의 0.5%는 z0005 값을, 99.5%는 z0006 값을 가진다. 고객은 모두 고객유형(A, B, C, D)을 가지며, A, B, C 값은 고객의 33%씩 차지하며, 고객의 1%가 D 값을 가진다.
  고객유형 컬럼에 인덱스가 있다. CREATE INDEX IX_T_CUST_TY ON T_CONT (CUST_TY);
- 상품 테이블은 총1만 개 상품에 대한 정보를 저장한다.

8. 파티션

문제5	최근 3개월 동안 주문유형이 A, B, C가 아닌 주문에 대해, 주문번호 별로 총판매가격을 자주 조회하는 쿼리.

SQL

```
CREATE INDEX IX_T_ORDER8_TY_DT ON T_ORDER8 ( ORDER_TY, ORDER_DT );

SELECT C.CUST_ID, C.CUST_NM, O.ORDER_NO,  NVL(SUM(D.ORDER_AMT * D.ORDER_QY),0) TOTAL_PRICE
  FROM T_CUST      C,
       T_ORDER8    O,  -- 월 15만 건,  총 36개월
       P_ORDER_D8  D,  -- 월 150만 건, 총 36개월
       T_GOODS     G
 WHERE 1=1
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_TY NOT IN ('A','B','C')  -- 1%를 부정형으로 찾는다
   AND O.ORDER_DT BETWEEN TO_DATE('2020/10/01','YYYY/MM/DD') AND TO_DATE('2020/12/31','YYYY/MM/DD')
   AND O.ORDER_NO = D.ORDER_NO
   AND D.GOODS_ID = G.GOODS_ID
 GROUP BY C.CUST_ID, C.CUST_NM, O.ORDER_NO
 ORDER BY C.CUST_ID, C.CUST_NM, O.ORDER_NO;

DROP INDEX IX_T_ORDER8_TY_DT;
```

TRACE

```
-------------------------------------------------------------------------------------------------------
|ID|Operation                    | Name        | Rows   |Elaps. Time   |Pstart|Pend  |CR Gets | Starts |
-------------------------------------------------------------------------------------------------------
| 1|ORDER BY (SORT)              |             |  4564  |00:00:00.0026 |      |      |     0  |     1  |
| 2| GROUP BY (HASH)             |             |  4564  |00:00:00.0199 |      |      |     0  |     1  |
| 3|  HASH JOIN                  |             | 45640  |00:00:00.0178 |      |      |     0  |     1  |
| 4|   INDEX (FAST FULL SCAN)    | PK_T_GOODS  | 10000  |00:00:00.0061 |      |      |    44  |     1  |④
| 5|   HASH JOIN                 |             | 45640  |00:00:07.5042 |      |      |     0  |     1  |
| 6|    HASH JOIN                |             |  4564  |00:00:00.0143 |      |      |     0  |     1
| 7|     TABLE ACCESS (FULL)     | T_CUST      | 70000  |00:00:00.0150 |      |      |  5858  |     1  ①
| 8|     TABLE ACCESS (FULL)     | T_ORDER8    |  4564  |00:00:00.5970 |      |      | 30289  |     1  ②
| 9|    PARTITION RANGE (ALL PART)|            |    54M |00:00:00.0008 |    1 |   37 |     0  |     1
|10|     TABLE ACCESS (FULL)     | P_ORDER_D8  |    54M |00:00:21.4622 |      |      |  319K  |    37  ⑤
-------------------------------------------------------------------------------------------------------

   3 - access("G"."GOODS_ID" = "D"."GOODS_ID") (0.000)
   5 - access("O"."ORDER_NO" = "D"."ORDER_NO") (0.000)
   6 - access("C"."CUST_ID" = "O"."CUST_ID") (0.000)
   8 - filter(("O"."ORDER_TY") NOT IN ('A','B','C')) AND ("O"."ORDER_DT" >= TO_DATE('2020-10-01
00:00:00','YYYY-MM-DD HH24:MI:SS')) AND ("O"."ORDER_DT" <= TO_DATE('2020-12-31 00:00:00','YYYY-M
M-DD HH24:MI:SS')) (0.301 * 1.000 * 1.000)
```

분석

① 고객 테이블을 Full Scan 해서 추출한 7만 건으로 Hash Table을 만든다.

② 주문의 주문유형 조건이 부정형(NOT IN) 이므로 인덱스를 사용하지 못하고 주문 테이블을 Full Scan 해서 D 값에 해당하는 4564건(1%)을 찾아, 4564번 Hash Table을 탐색하여 조인 조건을 만족하는 중간 집합(4564)으로 Hash Table을 만든다.

③ 주문상세 테이블을 Full Scan 해서 추출한 5400만 건으로, 5400만 번 Hash Table을 탐색하여 조인 조건을 만족하는 45640건을 추출한다.

④ 상품의 PK 인덱스를 Full Scan 해서 Hash Table로 만들고 ③ 중간집합과 Hash 조인한다.

⑤ 추출한 45640건을 Group By 해서 4564건을 추출한다.

8. 파티션

문제5 – 튜닝 1단계	Partition Pruning으로 3개 파티션을 Full Scan 해서 Hash 조인한다.

튜닝내역

1. 주문유형 조건이 부정형(NOT IN) 이므로 인덱스를 사용하지 못하고 주문 테이블을 Full Scan 한다. 부정형(NOT IN)을 긍정형(IN)으로 변경하여 인덱스를 사용한다.
2. 주문상세에 조건이 없어 주문상세 테이블을 Full Scan 한다. 주문상세의 주문일시(파티션 키) 컬럼으로 Partition Pruning이 되도록 반정규화 된 주문일시 조건을 추가하여 주문상세의 3개 테이블 파티션(10월 ~ 12월 파티션)을 Full Scan 해서 460만 건을 찾아, 460만 번 Hash Table(고객과 주문이 조인 된 중간 집합)을 탐색하여 조인 조건을 만족하는 건을 찾는다.

SQL

```
CREATE INDEX IX_T_ORDER8_TY_DT ON T_ORDER8 ( ORDER_TY, ORDER_DT );


SELECT /*+  LEADING(O C D G) INDEX(O IX_T_ORDER8_TY_DT) FULL(C) FULL(D) FULL(G)
            USE_HASH(C) USE_HASH(D) USE_HASH(G) SWAP_JOIN_INPUTS(G) */
       C.CUST_ID, C.CUST_NM, O.ORDER_NO,  NVL(SUM(D.ORDER_AMT * D.ORDER_QY),0) TOTAL_PRICE
  FROM T_CUST      C,  --2
       T_ORDER8    O,  --1  월 15만 건,  총 36개월
       P_ORDER_D8  D,  --3  월 150만 건, 총 36개월
       T_GOODS     G
 WHERE 1=1
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_TY IN ('D') -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND O.ORDER_DT BETWEEN TO_DATE('2020/10/01','YYYY/MM/DD') AND TO_DATE('2020/12/31','YYYY/MM/DD')
   AND O.ORDER_NO = D.ORDER_NO
   AND O.ORDER_DT = D.ORDER_DT  -- 조건 추가로 Partition Pruning
   AND D.GOODS_ID = G.GOODS_ID
 GROUP BY C.CUST_ID, C.CUST_NM, O.ORDER_NO
 ORDER BY C.CUST_ID, C.CUST_NM, O.ORDER_NO;


DROP INDEX IX_T_ORDER8_TY_DT;
```

TRACE

```
-------------------------------------------------------------------------------------------------------------------------
|ID|Operation                        |Name              |Rows  |Elaps. Time  |Pstart|Pend|CR Gets|Starts|
-------------------------------------------------------------------------------------------------------------------------
| 1|ORDER BY (SORT)                  |                  | 4564 |             |      |    |       |      |
| 2| GROUP BY (HASH)                 |                  | 4564 |             |      |    |       |      |
| 3|  HASH JOIN                      |                  |45640 |00:00:00.0150|      |    |     0 |    1 |
| 4|   TABLE ACCESS (FULL)           |T_GOODS           |10000 |00:00:00.0010|      |    |   213 |    1 |
| 5|   HASH JOIN                     |                  |45640 |00:00:00.3377|      |    |     0 |    1 |
| 6|    HASH JOIN                    |                  | 4564 |00:00:00.0084|      |    |     0 |    1 |
| 7|     TABLE ACCESS (ROWID)        |T_ORDER8          | 4564 |00:00:00.0043|      |    |  2095 |    1 |
| 8|      INDEX (RANGE SCAN)         |IX_T_ORDER8_TY_DT | 4564 |00:00:00.0031|      |    |    21 |    1 |
| 9|     TABLE ACCESS (FULL)         |T_CUST            |70000 |00:00:00.0272|      |    |  5858 |    1 |
|10|    PARTITION RANGE (SUBSET PART)|                  | 4600K|00:00:00.0000|    10|  12|     0 |    1 |
|11|     TABLE ACCESS (FULL)         |P_ORDER_D8        | 4600K|00:00:00.1503|      |    | 26899 |    3 |
-------------------------------------------------------------------------------------------------------------------------
```

중간 집합

파티션 3개를 Full Scan 해서 460만 건을 추출한다. 실제 필요한 것은 1%이다.

8. 파티션

문제5 – 튜닝 2단계	Partition Pruning으로 찾은 파티션의 Local Non-Prefixed 인덱스(=,범위)를 사용하여 NL 조인한다.

튜닝내역

1. 주문상세의 반정규화 된 주문일시(파티션 키) 조건을 추가하여 Partition Pruning으로 파티션 3개(10월 ~ 12월 파티션)을 Full Scan 해서 중간 집합(고객과 주문의 조인 결과)과 Hash 조인하는 것을 NL 조인으로 변경한다.
2. NL 조인 시 사용할 컬럼으로 IX_P_ORDER_D8_NO_DT 인덱스(Local Non-Prefixed 인덱스) 를 ORDER_NO(=) + ORDER_DT(between) 컬럼으로 만든다. Hash 조인을 NL 조인으로 변경하는 이유는 최근 3개월 주문 정보를 자주 조회하기 때문에 데이터가 메모리에 존재할 가능성이 높고, 주문상세의 3개 테이블 파티션에서 모든 데이터가 아닌 1% 만 필요하기 때문이다.

* NL 조인 시 랜덤 엑세스 된 블록은 Hash 조인 시 Full Scan 된 블록 보다 오래 메모리에 존재함.

SQL

```
CREATE INDEX IX_T_ORDER8_TY_DT ON T_ORDER8 ( ORDER_TY, ORDER_DT );
CREATE INDEX IX_P_ORDER_D8_NO_DT ON P_ORDER_D8 ( ORDER_NO, ORDER_DT ) LOCAL; -- Local Non-Prefixed 인덱스

SELECT /*+ LEADING(O C D G) INDEX(O IX_T_ORDER8_TY_DT) FULL(C) INDEX(D IX_P_ORDER_D8_NO_DT) FULL(G)
           USE_HASH(C) USE_NL(D) USE_HASH(G) SWAP_JOIN_INPUTS(G) */
       C.CUST_ID, C.CUST_NM, O.ORDER_NO,  NVL(SUM(D.ORDER_AMT * D.ORDER_QY),0) TOTAL_PRICE
  FROM T_CUST      C,  --2
       T_ORDER8    O,  --1  월 15만 건,  총 36개월
       P_ORDER_D8  D,  --3  월 150만 건, 총 36개월
       T_GOODS     G
 WHERE 1=1
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_TY IN ('D')  -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND O.ORDER_DT BETWEEN TO_DATE('2020/10/01','YYYY/MM/DD') AND TO_DATE('2020/12/31','YYYY/MM/DD')
   AND O.ORDER_NO = D.ORDER_NO
   AND O.ORDER_DT = D.ORDER_DT  -- 조건 추가로 Partition Pruning
   AND D.GOODS_ID = G.GOODS_ID
 GROUP BY C.CUST_ID, C.CUST_NM, O.ORDER_NO
 ORDER BY C.CUST_ID, C.CUST_NM, O.ORDER_NO;

DROP INDEX IX_T_ORDER8_TY_DT;
DROP INDEX IX_P_ORDER_D8_NO_DT;
```

TRACE

```
-----------------------------------------------------------------------------------------------------------
|ID|Operation                       |Name              |Rows | Elaps. Time |Pstart|Pend|CR Gets|Starts|
-----------------------------------------------------------------------------------------------------------
| 1|ORDER BY (SORT)                 |                  | 4564|00:00:00.0023|      |    |      0|     1|
| 2| GROUP BY (HASH)                |                  | 4564|00:00:00.0134|      |    |      0|     1|
| 3|  HASH JOIN                     |                  |45640|00:00:00.0127|      |    |      0|     1|
| 4|   TABLE ACCESS (FULL)          |T_GOODS           |10000|00:00:00.0027|      |    |      0|     1|
| 5|   INDEX JOIN                   |                  |45640|00:00:00.0038|      |    |      0|     1|
| 6|    HASH JOIN                   |                  | 4564|00:00:00.0069|      |    |      0|     1|
| 7|     TABLE ACCESS (ROWID)       |T_ORDER8          | 4564|00:00:00.0032|      |    |   2095|     1|
| 8|      INDEX (RANGE SCAN)        |IX_T_ORDER8_TY_DT | 4564|00:00:00.0038|      |    |     21|     1|
| 9|     TABLE ACCESS (FULL)        |T_CUST            |70000|00:00:00.0138|      |    |   5858|     1|
|10|    PARTITION RANGE (SUBSET PART)|                 |45640|00:00:00.0005|    10|  12|      0| 4564|
|11|     TABLE ACCESS (ROWID)       |P_ORDER_D8        |45640|00:00:00.0302|      |    |   4815| 13692|
|12|      FILTER                    |                  |45640|00:00:00.0007|      |    |      0| 13692|
|13|       INDEX (RANGE SCAN)       |IX_P_ORDER_D8_NO_DT|45640|00:00:00.8312|      |    |  15486| 13692|
-----------------------------------------------------------------------------------------------------------
```

Hash 조인 → NL 조인으로 변경

중간 집합

Local Non-Prefix 인덱스를 사용하여 13692번 수직 탐색을 한다.

8. 파티션

문제5 – 튜닝 3단계	Non-Partitioned 인덱스(=,범위)를 사용하여 NL 조인한다.

튜닝내역

1. 중간 집합(고객과 주문의 조인 결과)→주문상세로 NL 조인을 할 때 사용되는 IX_P_ORDER_D8_NO_DT 인덱스(Local Non-Prefixed 인덱스)는 3개의 인덱스 파티션을(10월 ~ 12월 파티션)을 액세스하므로 고객에서 4564건 IX_P_ORDER_D8_NO_DT 인덱스를 사용하여 조인할 때 인덱스 수직 탐색이 13692번 발생한다. Local Non-Prefixed 인덱스를 Non-Partitioned 인덱스로 변경하면 1개의 인덱스를 사용하므로 중간집합에서 4564건이 인덱스를 사용하여 조인할 때 인덱스 수직 탐색이 4564번만 발생하여 인덱스 블록을 액세스하는 양이 더 작아진다.(15486→6807 블록)

SQL

```
CREATE INDEX IX_T_ORDER8_TY_DT ON T_ORDER8 ( ORDER_TY, ORDER_DT );
CREATE INDEX NP_IX_P_ORDER_D8_NO_DT ON P_ORDER_D8 ( ORDER_NO, ORDER_DT ); -- Non-Partitioned 인덱스


SELECT /*+ LEADING(O C D G) INDEX(O IX_T_ORDER8_TY_DT) FULL(C) INDEX(D NP_IX_P_ORDER_D8_NO_DT)
           FULL(G) USE_HASH(C) USE_NL(D) USE_HASH(G) SWAP_JOIN_INPUTS(G) */
       C.CUST_ID, C.CUST_NM, O.ORDER_NO,  NVL(SUM(D.ORDER_AMT * D.ORDER_QY),0) TOTAL_PRICE
  FROM T_CUST      C,  --2
       T_ORDER8    O,  --1
       P_ORDER_D8  D,  --3
       T_GOODS     G
 WHERE 1=1
   AND C.CUST_ID = O.CUST_ID
   AND O.ORDER_TY IN ('D')  -- 1% 찾는 것을 부정형에서 긍정형으로 변경
   AND O.ORDER_DT BETWEEN TO_DATE('2020/10/01','YYYY/MM/DD') AND TO_DATE('2020/12/31','YYYY/MM/DD')
   AND O.ORDER_NO = D.ORDER_NO
   AND O.ORDER_DT = D.ORDER_DT  -- 주문일시 조건 추가로 Partition Pruning
   AND D.GOODS_ID = G.GOODS_ID
 GROUP BY C.CUST_ID, C.CUST_NM, O.ORDER_NO
 ORDER BY C.CUST_ID, C.CUST_NM, O.ORDER_NO;


DROP INDEX IX_T_ORDER8_TY_DT;
DROP INDEX NP_IX_P_ORDER_D8_NO_DT;
```

TRACE

```
-----------------------------------------------------------------------------------------------------------
|ID| Operation                | Name                | Rows   | Elaps. Time  | CR Gets | Starts   |
-----------------------------------------------------------------------------------------------------------
| 1|ORDER BY (SORT)           |                     |   4564 |00:00:00.0023 |       0 |        1 |
| 2| GROUP BY (HASH)          |                     |   4564 |00:00:00.0137 |       0 |        1 |
| 3|  HASH JOIN               |                     |  45640 |00:00:00.0141 |       0 |        1 |
| 4|   TABLE ACCESS (FULL)    | T_GOODS             |  10000 |              |         |          |
| 5|   INDEX JOIN             |                     |  45640 |              |         |          |
| 6|    HASH JOIN             |                     |   4564 |00:00:00.0099 |       0 |        1 |
| 7|     TABLE ACCESS (ROWID) | T_ORDER8            |   4564 |00:00:00.0045 |    2095 |        1 |
| 8|      INDEX (RANGE SCAN)  | IX_T_ORDER8_TY_DT   |   4564 |00:00:00.0001 |      21 |        1 |
| 9|     TABLE ACCESS (FULL)  | T_CUST              |  70000 |00:00:00.0175 |    5858 |        1 |
|10|    TABLE ACCESS (ROWID)  | P_ORDER_D8          |  45640 |00:00:00.0345 |    4815 |     4564 |
|11|     FILTER               |                     |  45640 |00:00:00.0003 |       0 |     4564 |
|12|      INDEX (RANGE SCAN)  | NP_IX_P_ORDER_D8_NO_DT | 45640 |00:00:00.7922 |    6807 |     4564 |
-----------------------------------------------------------------------------------------------------------
```

중간 집합

Non-Partitioned 인덱스를 사용하여 4564번 수직 탐색을 한다.

8. 파티션

정리

1. 테이블의 파티션 키 컬럼이 조건에 가공되어 사용되면 Partition Pruning이 되지 않아, 모든 테이블 파티션을 Full Scan 한다. 또는 NL 조인 시, 모든 인덱스 파티션을 액세스한다.
2. 고객과 주문이 조인할 때, 모든 고객이 주문과 조인하면 Hash 조인이 좋고, 특정 고객이 주문과 조인하면 NL 조인이 좋다.
3. 인덱스 블록을 액세스하는 범위는 범위 조건이 처음으로 나올 때 까지 사용된 모든 조건을 만족하는 인덱스 블록을 액세스한다. 예로 01 인덱스가 C1, C2, C3, C4로 구성되고 C1 = 조건, C2 = 조건, C3 Between 조건, C4 = 조건으로 사용되면, 범위 조건이 처음으로 나올 때 까지 사용된 C1 = 조건과 C2 = 조건과 C3 Between 조건을 모두 만족하는 인덱스 블록을 액세스한다.
4. Local (파티션) 인덱스는 인덱스를 파티션 하는 키 컬럼과 파티션 단위가, 테이블을 파티션 하는 키 컬럼과 파티션 단위와 동일한 인덱스이다. 예로 주문 테이블이 판매일시 컬럼으로 월별 Range 파티션되어 있고, 01 인덱스도 판매일시 컬럼으로 월별 Range 파티션되어 있으면 01 인덱스는 Local 파티션 인덱스이다.
5. Prefixed (파티션) 인덱스는 테이블 파티션 키 컬럼과 상관없이, 인덱스를 파티션 하는 키 컬럼이 인덱스를 구성하는 컬럼의 왼쪽 선두에 위치하는 인덱스이다. 그렇지 않으면 Non-Prefixed 파티션 인덱스이다. 예로 주문 테이블의 01 인덱스가 주문일시(파티션 키) 컬럼으로 월별로 Range 파티션 되고, 주문일시 + 고객아이디 로 구성되면 Prefixed 인덱스이고, 02 인덱스가 주문일시(파티션 키) 컬럼으로 월별 Range 파티션 되고, 고객아이디 + 주문일시 로 구성되면 Non-Prefixed 파티션 인덱스가 된다.
6. Local Prefixed 인덱스와 Local Non-Prefixed 인덱스 모두, 인덱스 접근을 한번 시도할 때 마다 여러 번의 수직 탐색이 발생한다. 즉 고객에서 1건이 고객 → 주문(주문일시로 파티션)으로 NL 조인 시, N 개의 인덱스 파티션을 사용하므로 인덱스 수직 탐색이 N번 발생한다. 그래서 수직 탐색을 줄이기 위해 Non-Prefixed 인덱스를 Non-Partitioned 인덱스로 변경하면 인덱스 블록을 액세스 하는 양이 줄어든다.
7. Non-Partitioned 인덱스를 사용하여도 인덱스→테이블로 액세스 하는 건수가 많으면 테이블 블록을 랜덤 액세스하는 양이 많아 성능 저하가 발생한다. 이 경우 Index Only Scan 을 사용한다.
8. 인덱스 파티션 비교(테이블이 주문일시 컬럼으로 월 파티션 된 경우)

인덱스 종류	특징
Local Prefixed (주문일시 + 고객아이디)	• 인덱스가 파티션 되어 필요한 인덱스 파티션 개수 만큼 수직 탐색이 발생한다. • 주로 날짜는 범위, 고객아이디는 = 조건이므로 날짜 범위 조건에 해당하는 인덱스 블록을 액세스해서 인덱스 스캔 효율이 떨어진다.
Local Non-Prefixed (고객아이디 + 주문일시)	• 인덱스가 파티션 되어 필요한 인덱스 파티션 개수 만큼 수직 탐색이 발생한다. • 주로 고객아이디는 =, 날짜는 범위 조건이므로 고객아이디 = 과 날짜 범위 조건에 모두 해당하는 인덱스 블록을 액세스해서 인덱스 스캔 효율이 좋다.
Non-Partitioned 인덱스 (고객아이디 + 주문일시)	• 인덱스가 파티션되어 있지 않아 한번의 수직 탐색 만 발생한다. • 넓은 범위의 조건을 인덱스에서 읽는 경우, 테이블 블록을 랜덤 엑세스하는 양이 많아 성능 저하가 발생한다. 이때는 Index Only Scan을 검토한다.