삼성증권 일별 통계 집계 쿼리 튜닝 보고서

SQL ID

VX_QUEUE02_ABANDON_DAY (튜닝 대상 SQL은 VIEW를 구성하는 쿼리문이므로 VIEW 이름으로 대체)

튜닝 완료 일자

2023년 01월 17일 화요일

튜닝 작업자

- 플랫폼개발2팀 윤영우 선임
- AICC개발2팀 주인선 선임
- AICC개발2팀 최원준 선임

튜닝 전/후 쿼리 소요 시간 및 입출력 블록 수

튜닝 전	튜닝 후
1분 40초	3초

쿼리 튜닝 성공의 핵심 척도는 전체 버퍼 블록 입출력 블록 수이지만 Oracle이 아닌 MS-SQL의 SQL Trace 를 보는 것에 익숙하지 않아 확인하지 못 하였습니다.

원문 SQL

내용이 상당히 길어 핵심 내용만 기재하였습니다.

```
SELECT MSF.START_DATE_TIME_KEY AS Q_DATE_TIME_KEY,
       'DAY' AS Q_DAY,
       DT.CAL_DATE AS Q_CAL_DATE,
       DT.CAL_YEAR_NUM AS Q_CAL_YEAR_NUM
       ...
FROM MEDIATION_SEGMENT_FACT MSF
     INNER JOIN DATE_TIME TD ON (MSF.TECHNICAL_DESCRIPTOR_KEY
                                 = TD.TECHNICAL_DESCRIPTOR_KEY)
     ...
     INNER JOIN RESOURCE_ R ON (R.RESOURCE_KEY = MSF.RESOURCE_KEY
                                AND R.RESOURCE_TYPE_CODE = 'QUEUE')
WHERE ...
      MSF.START_DATE_TIME_KEY
      BETWEEN (SELECT ... CAL_DATE ⩽ '2022-10-13')
      AND (SELECT ... CAL_DATE ⩽ '2022-10-13')
      AND S_CNFR_CLSS_CD IN ('FM');
```

원문 SQL Trace

당시 MS-SQL 쿼리의 실제 Trace 결과가 텍스트가 아닌 이미지로 추출 되었습니다. 따라서 해당 이미지 내용을 토대로 하기 Oracle Trace 형태로 다시 작성하였습니다.

```
...
WHATEVER JOIN
    NESTED LOOPS
        TABLE ACCESS FULL  ⟶ RESOURCE_ 테이블 FULL SCAN 후 반환 Rows는 168행
        TABLE ACCESS FULL  ⟶ MEDIATION_SEGMENT_FACT 테이블을 168번 중첩 루프를
                             통해 FULL SCAN
    WHATEVER VIEW
...
```

문제점

일별 통계 집계 보고서를 생성하는 웹페이지에서 '2022-10-13' 날짜 같은 작년 말 특정 일자를 입력하면 1초에서 2초만에 보고서를 생성했지만 올해 1월 중 한 날짜를 입력하면 보고서 생성까지 1분 40초 정도 걸리는 문제가 있었습니다. '2022-10-13' 날짜를 입력했을 때 Trace 결과는 하기와 같았습니다.

```
...
WHATEVER JOIN
    HASH JOIN  ⟶ 조인을 완료하는데 걸리는 시간: 2초
        TABLE ACCESS FULL  ⟶ RESOURCE_ 테이블에서 추출한 168행의 Join Key를 이용하여
                             Hash Table 생성
        TABLE ACCESS FULL  ⟶ MEDIATION_SEGMENT_FACT 테이블은 Join Key를 이용하여
                             Hash Join 수행
    WHATEVER VIEW
...
```

'2023-01-06' 날짜를 입력했을 때 Trace 결과는 하기와 같았습니다.

```
...
WHATEVER JOIN
    NESTED LOOPS  ⟶ 조인을 완료하는데 걸리는 시간: 1분 40초
        TABLE ACCESS FULL  ⟶ RESOURCE_ 테이블 FULL SCAN 후 반환 Rows는 168행
        TABLE ACCESS FULL  ⟶ MEDIATION_SEGMENT_FACT 테이블을 168번 중첩 루프를
                             통해 FULL SCAN
    WHATEVER VIEW
...
```

'2022-10-13' 날짜를 입력했을 경우에는 옵티마이저가 Hash Join을 이용하는 실행 계획을 선택하여 쿼리 수행 시간이 대략 2초 밖에 걸리지 않았지만 **'2023-01-06' 날짜를 입력했을 경우에는 옵티마이저가 NL 조인을 이용**하는 실행 계획을 선택하여 쿼리 수행 시간이 1분 40초 소요 되었습니다. 시간이 오래 걸린 이유는 **MEDIATION_SEGMENT_FACT 테이블에 Join Key 컬럼인 RESOURCE_KEY 컬럼에 인덱스가 없었기 때문**입니다. 따라서 20만 건이 넘는 MEDIATION_SEGMENT_FACT 테이블을 매번 FULL SCAN 수행하여 불필요한 블록 입출력을 만들어내고 있었습니다. 또한 해당 쿼리에 힌트 지정을 하지 않았기에 **입력된 날짜에 따라서 옵티마이저가 매번 다른 실행 계획을 생성**해내는 것도 문제였습니다.

개선 SQL

모든 테이블의 선택도 및 카디널리티를 분석하여 필요한 인덱스 생성과 힌트 지정을 통해 매번 최적의 실행 계획을 수행할 수 있도록 하고 싶었지만 시간 부족 및 고객사의 압박으로 일단 **NL 조인을 통해 발생하는 병목 현상을 회피**하자는 전략으로 튜닝을 실행하였습니다. 따라서 어떠한 날짜가 입력 되더라도 똑같은 성능을 발휘할 수 있도록 RESOURCE_ 테이블 및 MEDIATION_SEGMENT_FACT 테이블이 Hash Join을 수행하도록 힌트를 지정하였습니다.

```
SELECT MSF.START_DATE_TIME_KEY AS Q_DATE_TIME_KEY,
       'DAY' AS Q_DAY,
       DT.CAL_DATE AS Q_CAL_DATE,
       DT.CAL_YEAR_NUM AS Q_CAL_YEAR_NUM
       ...
FROM MEDIATION_SEGMENT_FACT MSF
     INNER JOIN DATE_TIME TD ON (MSF.TECHNICAL_DESCRIPTOR_KEY
                                 = TD.TECHNICAL_DESCRIPTOR_KEY)
     ...
     INNER HASH JOIN RESOURCE_ R ON (R.RESOURCE_KEY = MSF.RESOURCE_KEY
                                     AND R.RESOURCE_TYPE_CODE = 'QUEUE')
WHERE ...
      MSF.START_DATE_TIME_KEY
      BETWEEN (SELECT ... CAL_DATE ⩽ '2022-10-13')
      AND (SELECT ... CAL_DATE ⩽ '2022-10-13')
      AND S_CNFR_CLSS_CD IN ('FM');
```

MEDIATION_SEGMENT_FACT 테이블에 RESOURCE_KEY 컬럼에 대한 인덱스를 생성하여 NL 조인으로 유도하지 않은 이유는 상기 쿼리가 다수의 사용자에 의해 실시간으로 실행되는 OLTP 쿼리도 아닐 뿐더러 운영 중의 인덱스 생성을 통한 영향도를 단기간에 파악할 수 없었기 때문입니다.

개선 SQL Trace

개선 SQL Trace의 이미지를 자세히 확인하지 못 하였지만 문제가 되었던 병목 구간인 NL 조인은 나타나지 않았고 대신 옵티마이저가 다른 실행 계획을 생성하여 이전과 다른 실행 계획으로 쿼리를 수행하고 있었습니다.

```
...
WHATEVER JOIN
    HASH JOIN
        TABLE ACCESS FULL  ⟶ RESOURCE_ (확실하지 않음)
        MERGE JOIN
            TABLE ACCESS FULL
            TABLE ACCESS FULL  ⟶ MEDIATION_SEGMENT_FACT (확실하지 않음)
    WHATEVER VIEW
...
```

마무리

한 가지 아쉬운 점은 입력된 날짜에 따라 극심한 시간 차이를 보여주는 쿼리문에 Hash Join 힌트를 통해 어떠한 날짜 입력 값에도 3초라는 일관된 성능을 보여줄 수 있도록 하였지만 튜닝 전 쿼리에서 옵티마이저가 최적의 실행 계획을 이용할 때는 특정 날짜에서 1초도 안 걸린다는 점입니다. 단순히 병목 구간을 해소하는 것뿐만이 아니라 조인하는 모든 테이블의 선택도 및 카디널리티를 분석하고 인덱스를 이용한다면 옵티마이저와 대등한 성능을 내겠지만 시간 부족으로 불가능 하였습니다. 또한 OLAP와 같은 환경에서 Hash Join의

쓸모는 대단하지만 자칫 이에 유혹되어 OLTP 환경에서도 남발하게 된다면 불필요한 CPU 및 메모리 사용으로 DBMS에 장애를 유발할 수도 있음을 상기해야 합니다. 결과적으로 운영자 입장에서는 시스템의 안정성이 최고의 가치이므로 산발적인 쿼리 성능보다 일관된 3초가 낫다고 생각합니다.