SQL과 실행계획을 이용한 튜팅

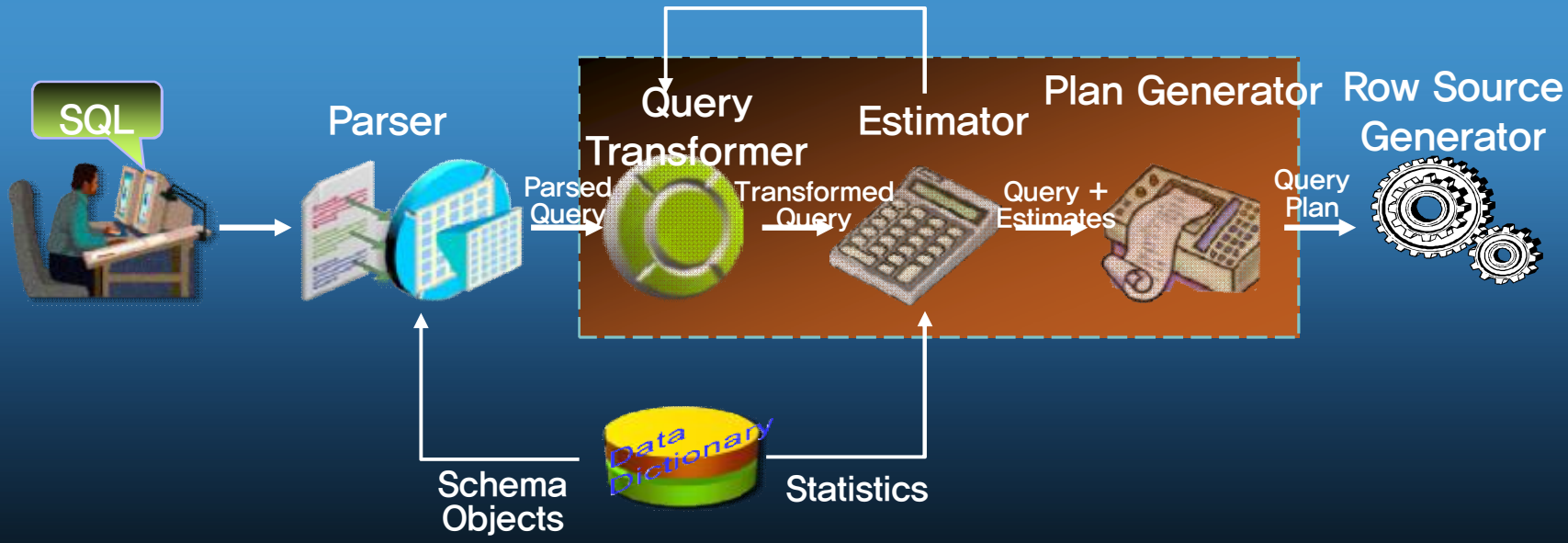
엔코아 컨설팅
컨설팅 사업본부 본부장
김동훈 이사

CONTENTS

- SQL의 개념
- 실행계획 패턴
- 실행계획의 최적화

SQL의 개념 – 수행단계

SQL은 데이터처리 방법을 기술한 것이 아니라 단지 필요한 데이터를 요구한 것임

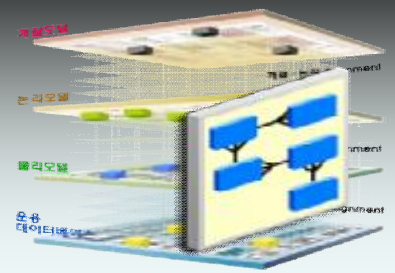


- 사용자가 실행한 SQL은 데이터 딕셔너리를 참조하여 파싱을 수행.
- 옵티마이져는 파싱 결과를 이용해 논리적으로 적용 가능한 실행계획 형태를 선택하고, 힌트를 감안하여 일차적으로 잠정적인 실행계획들을 생성
- 데이터 딕셔너리의 통계정보(데이터의 분포도, 테이블 저장구조, 인덱스 구조, 파티션 형태, 비교연산자)등을 감안하여 각 실행계획의 비용을 계산
- 실행계획들의 산출된 비용을 비교하여 가장 최소의 비용을 가진 실행계획을 선택 (최저가 입찰 방식이므로 항상 최적의 결정이라고만 할 수는 없음)

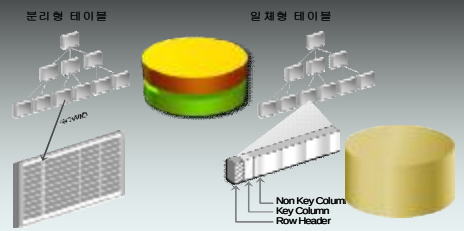
SQL의 개념 – 데이터처리 영향 요소

SQL 작성시 데이터 처리 효율에 영향을 미치는 요소를 항상 생각

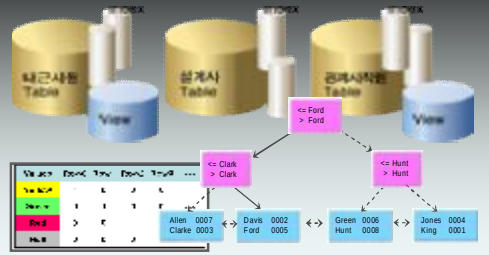
데이터 모델



데이터 저장형태



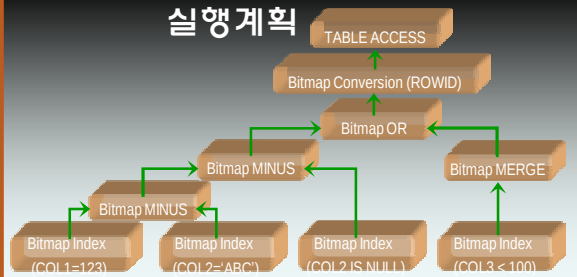
인덱스 형태 및 구조



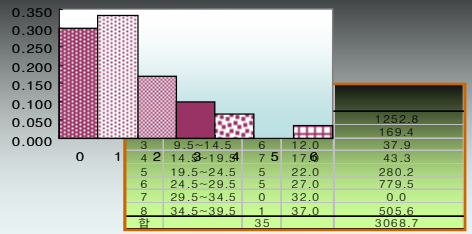
SQL의 형태


```
SELECT * FROM table1
WHERE COL1 = 123
AND COL2 <> 'ABC'
OR  COL3 < 100;
```

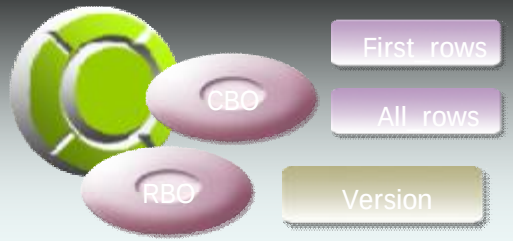
실행계획



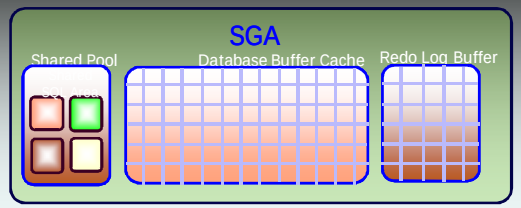
통계 정보



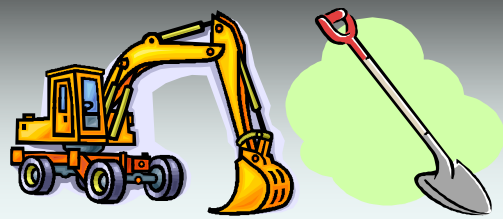
옵티마이저



메모리의 활용



클러스터링 팩터



SQL의 개념 – 집합 개념 필수

SQL은 집합을 처리하기 위한 도구

원 집합

번호	시작일시	종료일시	고객
123	20070112	20070130	이순신
123	20070130	20070204	김유신
123	20070204	20070228	김유신
123	20070228	20070501	이순신
123	20070501	99991231	을지문득
125	20070301	20070317	강감찬
125	20070317	20070412	강감찬
125	20070412	20070510	계백
125	20070510	99991231	홍길동
.	.	.	.

중간 필요 집합

번호	시작일시	종료일시	SW	고객
123	20070112	20070130	1	이순신
123	20070130	20070204	2	김유신
123	20070204	20070228	2	김유신
123	20070228	20070501	3	이순신
123	20070501	99991231	4	을지문득
125	20070301	20070317	1	강감찬
125	20070317	20070412	1	강감찬
125	20070412	20070510	2	계백
125	20070510	99991231	3	홍길동
.	.	.		.

결과집합

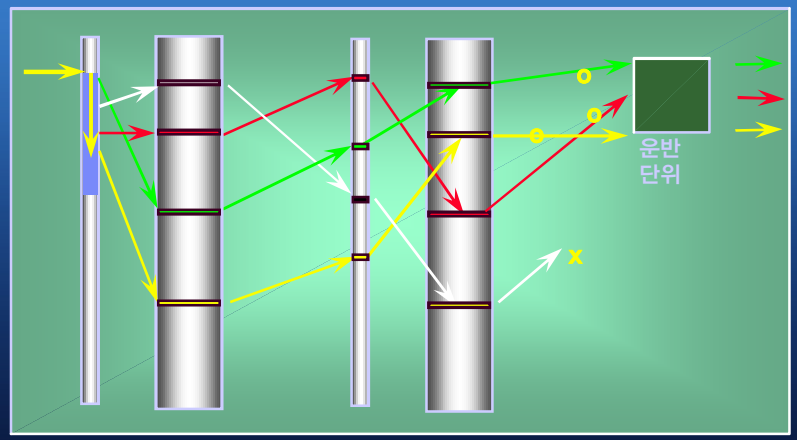
전화번호	시작일시	종료일시	고객
123	20070112	20070130	이순신
123	20070130	20070228	김유신
123	20070228	20070501	이순신
123	20070501	99991231	을지문득
125	20070301	20070412	강감찬
125	20070412	20070510	계백
125	20070510	99991231	홍길동
.	.	.	.

- 원 집합에서 결과 집합을 얻기 위해서는
- 반드시 중간 집합이 필요함
- 중간 집합은 Analytic function READ, SUM을 사용하여 구할 수 있음.

SQL의 개념 – 집합 연결 방법

두 개 이상의 집합을 연결하는 방법은 4가지 형태만 존재

조인의 활용



UNION, GROUP BY

CD	SQ	AT
A	1	150
A	2	200
B	1	100
B	2	120
…	…	…

CD	SQ	BT
A	10	300
A	11	100
B	11	150
B	15	100
…	…	…

UNION ALL è

CD	SQ	AT	BT
A	1	150	
A	2	200	
B	1	100	
B	2	120	
…	…	…	
A	10		300
A	11		100
B	11		150
B	15		100
…	…		

GROUP BY è

CD	AT	BT
A	350 400	
B	220	250
…	..	….

저장형 함수 및 Scalar Subquery 활용

```
CREATE FUNCTION  FUNC1
      (@v_empno     varchar(10),
       RETURN     varchar2  is
BEGIN
 declare V_avg_amt int
       SELECT avg(급여총액)
            into v_avg_amt
       FROM  급여
       WHERE 사번 = v_empno
            and 년월 like  '199803%'  ;
       RETURN  v_avg_amt;
END FUNC1 ;
```

```
SELECT 사번, 성명,
                직급, 직책,
                ……,

FUNC1(empno)
FROM     사원
WHERE 부서 =  '1100'  ;
```

서브쿼리의 활용

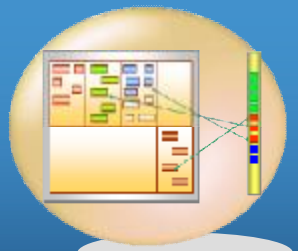
```
SELECT  x.자재코드, x.자재명, x.안전재고, y.재고수량
 FROM    자재 x,  자재일일재고 y
WHERE y.자재코드 = x.자재코드
    and  y.년월일  = convert(varchar(8), getdate, 112)
    and  x.자재코드 IN (SELECT  자재코드
                        FROM   구매의뢰
                        WHERE 진행상태 =  '발주중'
                          and  출고일자 like '199807%' )
```

실행계획 패턴 – 실행계획 수립에 영향을 미치는 요소

SQL 형태

SELECT …
FROM TAB
WHERE …

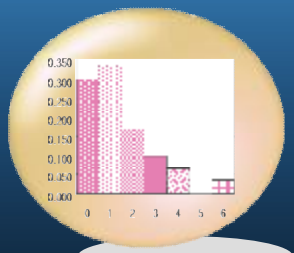
인덱스, 테이블 구조



사용 컬럼,
연산자 형태

=, LIKE, …
A = '상수',
A = :변수…

통계 및 Parameter 정보



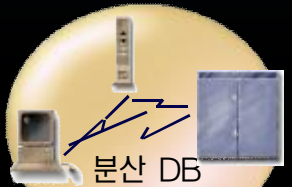
힌트 사용

SELECT …
/*+ FULL */
FROM …

옵티마이저 모드

RULE
All_rows
First_rows

시스템 및
네트워크 상태



DBMS 버전

실행계획 패턴 – 테이블 및 조인 액세스

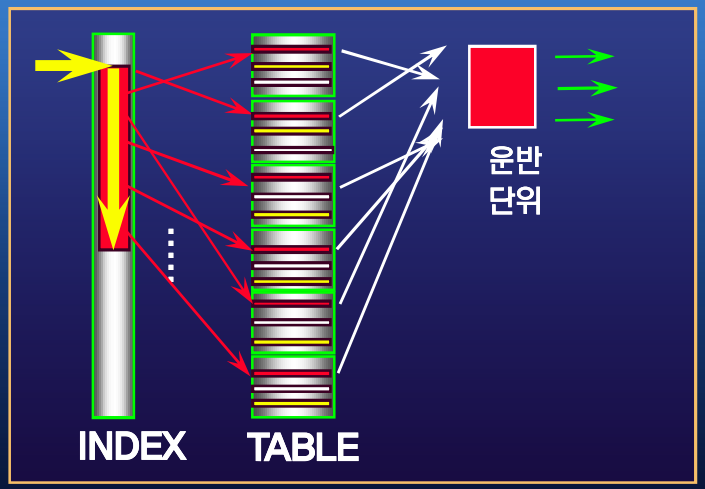
옵티마이저는 대부분 두 개의 패턴을 통해 실행계획을 작성

TABLE 액세스

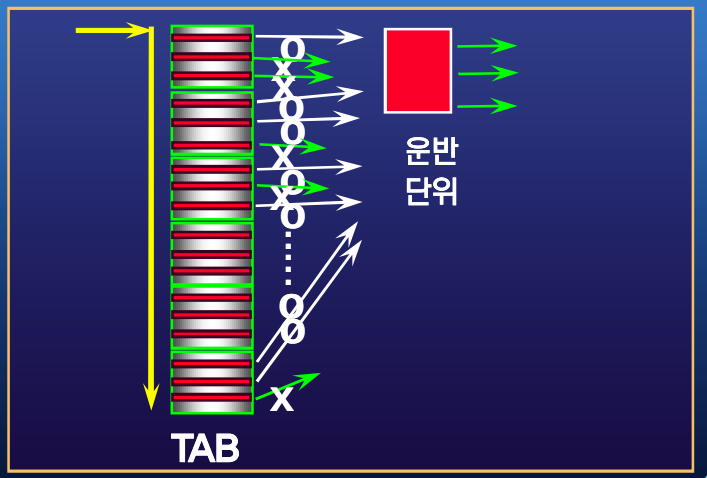
실행계획 패턴

(두 가지 방법만 존재)

Index Range Scan & Table Scan

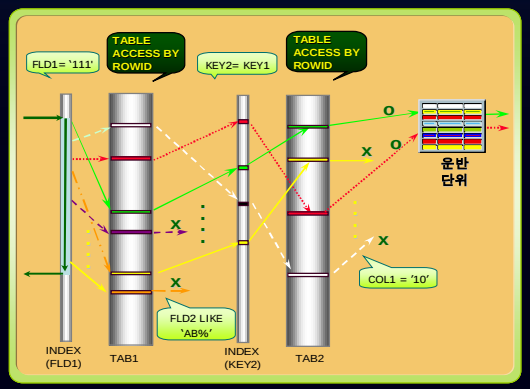


Full Table Scan

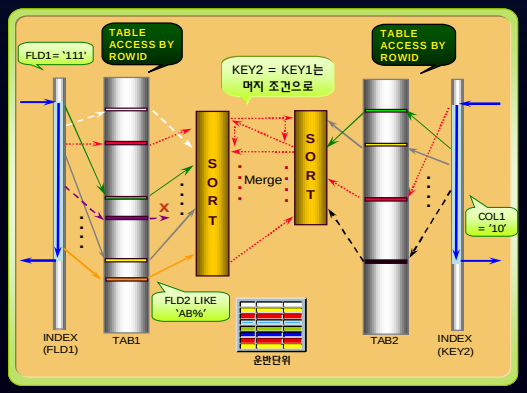


조 인

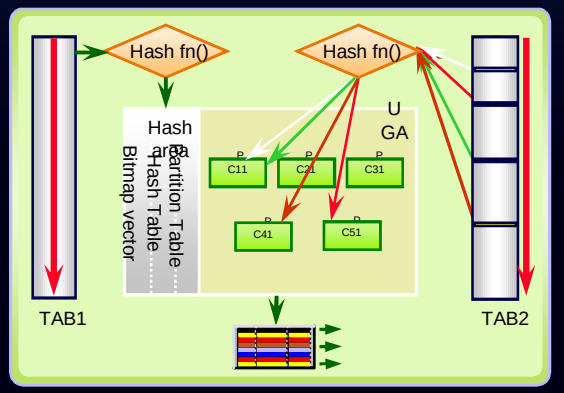
Nested Loop



Sort Merge



Hash



실행계획 패턴 – 실행 계획 패턴

옵티마이져의 거의 대부분은 다음의 실행계획 패턴으로 수행

- Nested Loop 조인 패턴

```
NESTED LOOPS
  NESTED LOOPS
    TABLE ACCESS (BY INDEX ROWID) OF 'SDAMASTER.SAM_ORG'
      INDEX (UNIQUE SCAN) OF 'SDAMASTER.PK1_SAM_ORG'
    TABLE ACCESS (BY INDEX ROWID) OF 'SDAMASTER.SAM_STF'
      INDEX (RANGE SCAN) OF 'SDAMASTER.I01_SAM_STF'
  TABLE ACCESS (BY INDEX ROWID) OF 'SDAMASTER.INS_INS_PL'
    INDEX (SKIP SCAN) OF 'SDAMASTER.I02_INS_INS_PL'
```

실행계획 패턴 – 실행 계획 패턴

- Sort Merge 조인 패턴

```
MERGE JOIN
  SORT(JOIN)
    TABLE ACCESS FULL OF '사원'
  SORT(JOIN)
    VIEW
      SORT(UINQUE)
        TABLE ACCESS BY ROWID OF '근태'
          INDEX RANGE SCAN OF '유형_일자_IDX'
```

- Hash 조인 패턴

```
HASH JOIN
  TABLE ACCESS (BY ROWID) OF ORDER
    INDEX (RANGE SCAN) OF ORDDATE_INDEX (NON-UNIQUE)
  TABLE ACCESS (FULL) OF DEPT
```

- 조인은 반드시 성공한 결과가 다음 조인에 참여
- 조인 테이블이 무수히 많아도 테이블을 액세스하는 방법은 2가지만 존재

실행계획 패턴 – 복잡한 실행 계획 패턴

복잡하지만 단위별로 분석하면 앞에 설명한 기본적인 패턴을 벗어나지 못함

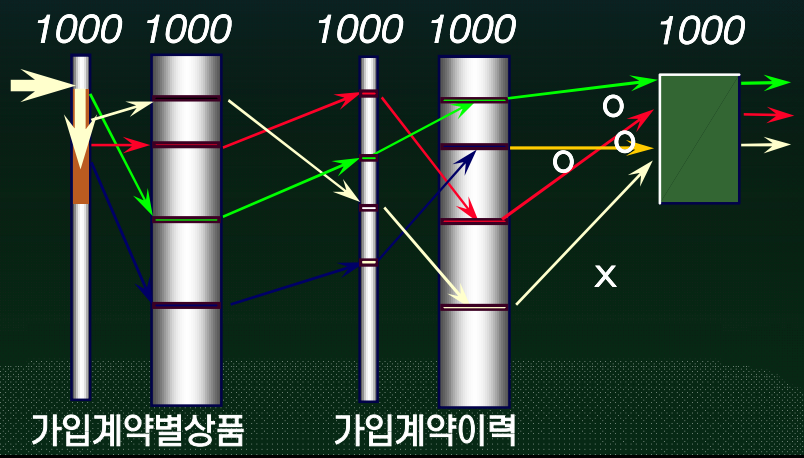
```
Rows       Execution Plan
-------    ----------------------------------------------------
      0  SELECT STATEMENT
   1181   SORT (GROUP BY)
   1181    NESTED LOOPS
   1514     VIEW
 256854      SORT (GROUP BY)
 256854       HASH JOIN (OUTER)
 256854        VIEW
 256854         UNION-ALL
 256852          NESTED LOOPS
      0           INLIST ITERATOR (CONCATENATED)
 256856             TABLE ACCESS (BY INDEX ROWID) OF `DPACCB'
 256908               INDEX (RANGE SCAN) OF 'DPACCB_IDX1' (NON-UNIQUE)
24915032          TABLE ACCESS (BY INDEX ROWID) OF 'BR_INFO'
40840104            INDEX (RANGE SCAN) OF `BR_INFO_IDX2' (NON-UNIQUE)
      0          INLIST ITERATOR (CONCATENATED)
 422116            TABLE ACCESS (BY INDEX ROWID) OF 'DPACCB'
 422118              INDEX (RANGE SCAN) OF 'DPACCB_IDX2' (NON-UNIQUE)
     15        VIEW
     15         SORT (GROUP BY)
     15          TABLE ACCESS (BY INDEX ROWID) OF `DPDDBS'
16080201            INDEX (RANGE SCAN) OF `DPDDBS_IDX1' (NON-UNIQUE)
   1688     TABLE ACCESS (BY INDEX ROWID) OF 'DPGDPF'
   3202       INDEX (RANGE SCAN) OF 'PK_DPGDPF' (UNIQUE)
```

실행계획의 최적화 – 최적화 대상

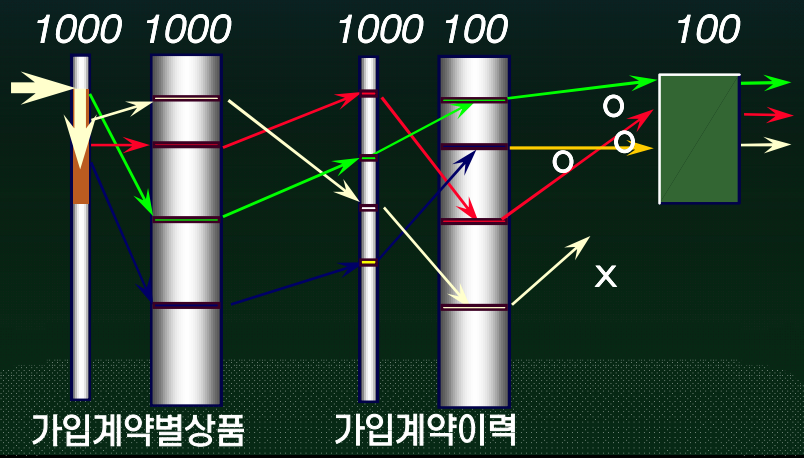
실행계획의 최적화란 비효율을 제거하는 것임

- 최초 입력이 1000건이고 출력이 1000건이면 비효율은 없음
- 입력이 1000인데 출력이 100이면 900건의 비효율이 발생한 것임
- 비효율이 발생하는 경우 항상 비효율을 발생시킨 원인이 존재
- 이 원인을 분석하여 해결하는 것이 실행계획 최적화

```
SELECT 가입계약번호, 전화번호, Y.상품
  FROM 가입계약이력  X, 가입계약별상품 Y
WHERE X.가입계약 = Y.가입계약
  AND Y.상품코드 ='FF001'
```




```
SELECT 가입계약번호, 전화번호, Y.상품
  FROM 가입계약이력  X, 가입계약별상품 Y
WHERE X.가입계약 = Y.가입계약
  AND Y.상품코드 ='FF001'
  AND X.상태 ='정지'
```



실행계획의 최적화 – 테이블 특성

테이블 특성에 따라서 전략을 수립하여 실행계획을 최적화

① 적은 데이터를 가진 소형 테이블
- DB_FILE_MULTIBLOCK_READ_COUNT 값 이하 블록 크기의 테이블
- 한번의 멀티블럭I/O에 의해 인덱스 없이 전체테이블 스캔 가능
- 다른 테이블들과 연결될 경우 인덱스 유무는 옵티마이져에 영향
- 특히 조인에서 내측루프로 수행되면 작은 테이블이라도 인덱스 없으면 큰 영향

② 주로 참조되는(Referenced) 역할을 하는 중대형 테이블
- 트랜잭션 데이터의 행위 주체, 목적이 되는 개체들로 구성된 테이블(키 엔티티 집합, '고객')
- 데이터 증감 없고, 검색 위주 액세스 발생, 검색 조건 형태 고정
- 블록 내에 최대한 조밀한 인덱스 공간을 위해 PCTFREE 를 0에 가깝게 부여

실행계획의 최적화 – 테이블 특성

③ 업무적 구체적인 행위를 관리하는 중대형 테이블
- '매출정보' 와 같은 업무의 구체적인 수행 내용을 담고 있는 트랜잭션 테이블
- 결합인덱스, B-Tree, 결합인덱스 컬럼 구성 순서에 따른 효율성 차이
- 최소의 인덱스, 최대의 액세스 유형, 기존 인덱스 형태 및 예상 감안한 인덱스 구성 전략 필요

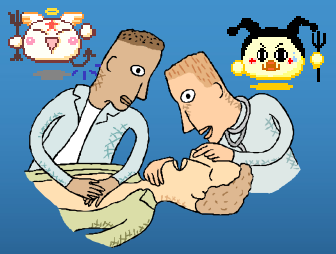
④ 저장용(Log性) 대형 테이블
- 로그성 데이터 관리 목적의 테이블
- 저장 우선, 갱신 거의 없으므로 PCTFREE 여유공간 불필요, PRIMARY KEY 제약조건 불필요
- 식별자 키 필요하면 UNIQUE INDEX 생성, 파티션 사용 고려

실행계획의 최적화 – 대용량 데이터 처리 최적화

실행계획 제어를 통한 조인의 횟수 감소를 통한 최적화

소량의 균은

치료약이 되지만…



대량의 균은

생명을 위태롭게 한다.

0.01초 차이만 나더라도…

0.01 * 10000 = 100초

0.01 * 1억 = 277 시간 = 12일

반복!!

è 대량 데이터 처리를 위한 대책이 필요

- 병렬처리(Parallel Processing)
- 파티션(Static Partition)
- Merge(Array Processing)
- 전략적인 인덱스 구성
- 고급 SQL 구사
- 실행계획 최적화
- 해쉬 조인(Hash Join)
- ……

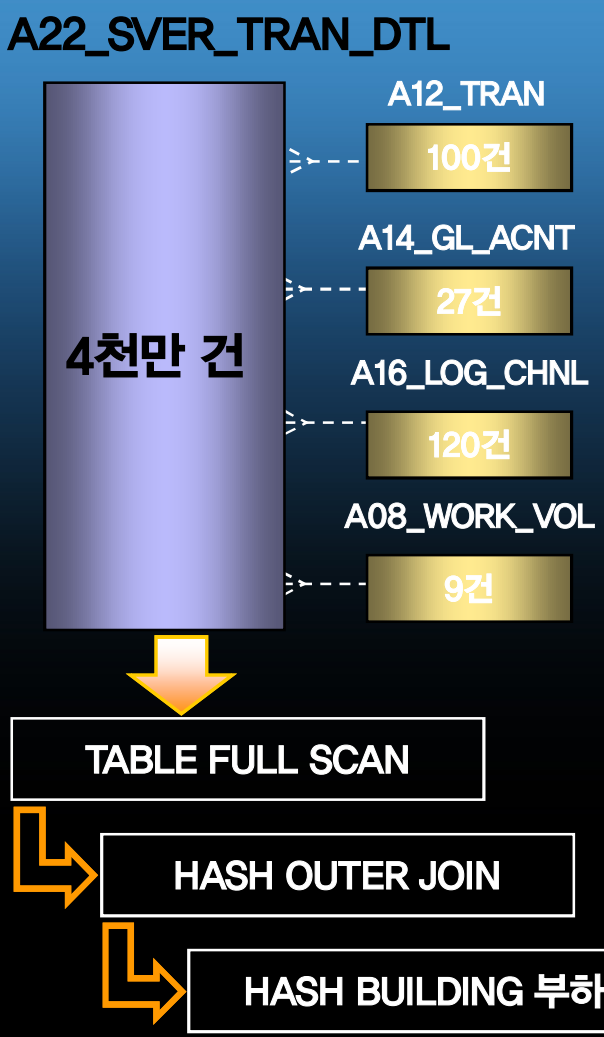
대량의 INSERT가 발생한다면

분리형이 가장 적당…

적당한 파티션 전략이 필요함

실행계획의 최적화 – 조인 횟수 감소

실행계획 제어를 통한 조인의 횟수 감소를 통한 최적화




```
SELECT  ...
  FROM A22_SVER_TRAN_DTL A22  X,
       (SELECT ..., ROWNUM
        FROM A12_TRAN A12, A14_GL_ACNT A14,
             A16_LOG_CHNL A16,
            (SELECT DISTINCT WORK_BR_CD ACNT_BR_CD
            FROM A08_WORK_VOL
           WHERE ASIN_SBU_CD = '30'
             AND BASE_YM = :IN_DATE  ) A99) A12
 WHERE A22.SVER_TYPE_CD    = A12.SVER_TYPE_CD
   AND A22.UPMU_CATE_CD    = A12.UPMU_CATE_CD
   AND A22.GL_ACCOUNT_ID   = A12.GL_ACCOUNT_ID
   AND A22.ACNT_BR_CD = A12.WORK_BR_CD
```

- 핵심은 조인 횟수의 감소
- 기존의 조인 방식보다 훨씬 빠른 속도 보장

실행계획의 최적화 – 비절차형 처리

입력, 변경을 동시에 처리하여 대부분의 절차형 데이터 처리를 SQL 하나로 처리

```
MERGE  /*+ USE_HASH(Y X) PARALLEL(Y 20) FULL(X) PARALLEL(X 20) */  INTO DW.KCF_CONT_MASTER  X
USING   INFRA.ECT_KCF_CONT_MASTER_02   Y
ON       (X.I_NCN=Y.I_NCN    AND X.I_YYYYMM='999912' )
WHEN MATCHED THEN
  UPDATE
  SET      X.I_CAN_GUBUN=Y.I_CAN_GUBUN,
           X.I_RESALE = NVL(Y.I_RESALE,X.I_RESALE),
           X.I_PREFIX = NVL(DECODE(Y.I_PREFIX,'X',X.I_PREFIX,Y.I_PREFIX),X.I_PREFIX),
           X.I_AGE = DECODE(SUBSTR(Y.I_AGE,1,1), '–', '*', 'X', 'X', '*', '*', LPAD(SUBSTR(Y.I_AGE,1,3),3,'0')),
           X.I_CUST_GRADE = NVL(Y.I_CUST_GRADE,X.I_CUST_GRADE)
WHEN NOT MATCHED THEN
  INSERT
  VALUES ( Y.I_NCN                    , '999912'                ,
             Y.I_CAN_GUBUN                , Y.I_RESALE                  ,
             Y.I_PREFIX                  , Y.I_CN                      ,
             Y.I_BAN                     , Y.I_CUSTOMER                ,
             Y.I_CTN                     , Y.MODEL_SERIAL_NO           ,
             DECODE(SUBSTR(Y.I_AGE,1,1), '–', '*', 'X', 'X', '*', '*', LPAD(SUBSTR(Y.I_AGE,1,3),3,'0')) ,
             Y.I_GENDER                  ,
             Y.I_FORMER                  , Y.I_S_PRICEPLAN             ,
             Y.I_ZIPCODE                 , Y.I_MODEL                   ,
             Y.I_DEALER                  , Y.I_JATASA_GUBUN            ,
                ..          );
```

퀴즈

번호별상태

번호	상태
123	사용
124	사용
125	미사용
126	사용
127	사용
128	미사용
129	사용
.	.

번호별 상품상태

번호	단말기	상태
123	S01	미사용
124	S02	사용
125	S03	미사용
126	S04	사용
127	S05	사용
128	S06	미사용
129	S07	사용
.	.	.

번호별 단말기상태

번호	단말기	상태
123	D01	사용
124	D02	미사용
125	D03	미사용
126	D04	사용
127	D05	미사용
128	D06	사용
129	D07	사용
.	.	.

```
SELECT  번호,
        COUNT(DECODE(SW,1,1))  번호별상태,
        COUNT(DECODE(SW,2,1))  번호별상품상태,
        COUNT(DECODE(SW,3,1))  번호별단말기상태
FROM ( SELECT 번호, SW,
              COUNT(*) OVER (PARTITION BY 번호) CNT
       FROM ( SELECT 번호,1 SW
              FROM  번호별상태
              WHERE  상태= '사용'
              UNION ALL
              SELECT 번호, 2 SW
              FROM  번호별상품상태
              WHERE  상태= '사용'
              UNION ALL
              SELECT 번호, 3 SW
              FROM  번호별단말기상태
              WHERE  상태= '사용' ) X ) A
WHERE CNT <> 3
GROUP BY 번호
```

Q & A

감사합니다.

ESI 교육센타 7월 교육 안내(www.en-core.com)

Oracle 사용자를 위한 **SQL** 활용

기간 : **2007**년 **07**월 **02**일 ~ **2007**년 **07**월 **04**일

새로 쓴 대용량 데이터베이스 솔루션 **I**

기간 : **2007**년 **07**월 **09**일 ~ **2007**년 **07**월 **13**일

관계 형 데이터베이스 모델링 **I (**개론**)**

기간 : **2007**년 **07**월 **18**일 ~ **2007**년 **07**월 **20**일

새로 쓴 대용량 데이터베이스 솔루션 **for MS-SQL**

기간 : **2007**년 **07**월 **23**일 ~ **2007**년 **07**월 **26**일