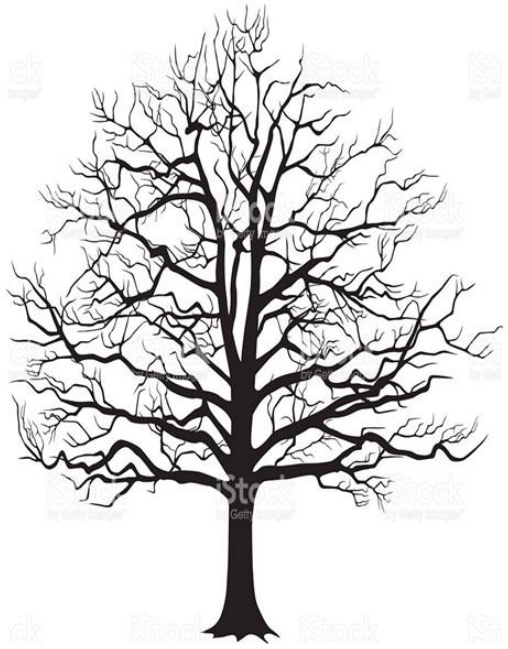
03. Index (색인)

Index



1. Index 개요

▣ 용도

1) 빠른 탐색(Quick Search)

* Quick Select (?)

Select	원하는 데이터를 빨리 탐색해서 조회
Delete	원하는 데이터를 빨리 탐색해서 삭제
Update	원하는 데이터를 빨리 탐색해서 수정

2) 고유성(Uniqueness)

- Primary key, Unique Key 제약사항 정의시 Unique Index 자동 생성

```
①SELECT   INDEX_NAME,UNIQUENESS,LEAF_BLOCKS,DISTINCT_KEYS,CLUSTERING_FACTOR,
          NUM_ROWS,SAMPLE_SIZE,LAST_ANALYZED
  FROM    USER_INDEXES
  WHERE   TABLE_NAME IN ('CUSTOMER')
  ORDER   BY TABLE_NAME;

② SELECT INDEX_NAME,COLUMN_POSITION,COLUMN_NAME
   FROM   USER_IND_COLUMNS
   WHERE  TABLE_NAME IN ('CUSTOMER')
   ORDER  BY TABLE_NAME,INDEX_NAME,COLUMN_POSITION;
```

1. Index 개요

▣ 생성

1) 빠른 탐색(Quick Search)

```
CREATE INDEX CUSTOMER_NAME_IDX ON CUSTOMER(NAME)
NOLOGGING  PCTFREE 2                    // PCTFREE 2  15,322 Blocks, PCTFREE 20  18,861  Blocks
TABLESPACE USERS;                       // drop index CUSTOMER_NAME_IDX ;


CREATE  INDEX  CUSTOMER_ACCOUNT_MGR_EMAIL_IDX
PCTFREE 10                              // default 10
ON CUSTOMER( ACCOUNT_MGR , EMAIL )
TABLESPACE USERS;


  CREATE UNIQUE INDEX CUSTOMER_EMAIL_UIDX        // Unique Index 수동 생성 가능한가? 의미는?
  ON CUSTOMER(EMAIL) TABLESPACE USERS;
```

[과제] 생성한 Index를 user_segments , user_extents에서 검색하여 할당된 block개수 확인

2) 고유성(Uniqueness)

// Automatically when Using Primary Key, Unique Key

① ALTER TABLE CUSTOMER DROP CONSTRAINT CUSTOMER_MOBILE_NO_UK;

② ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_MOBILE_NO_UK UNIQUE(MOBILE_NO);

①② 실행하면서 조회

```
SELECT INDEX_NAME,PCT_FREE,LOGGING,BLEVEL,LEAF_BLOCKS,DISTINCT_KEYS,NUM_ROWS
FROM  USER_INDEXES;                    // BLEVEL: Branch LEVEL (Branch + Root) , >4 Index Rebuild
```

1. Index 개요

▣ 관리

테이블에 DML 연산시 발생시 DBMS Server가 관련 인덱스를 자동으로 갱신

▣ 구성

- 테이블로 부터 논리적으로 물리적으로 독립된 별도의 객체(OBJECT).
- They can be created or dropped at any time and have no effect on the base tables or other indexes

- Index Entry(Record) := INDEXED COLUMNS + ROWID

EX) EMPNO ROWID (SINGLE COLUMN INDEX)

EMPNO	ROWID
7369	AAAR5RAAHAAAAFeAAA
7499	AAAR5RAAHAAAAFeAAB
7521	AAAR5RAAHAAAAFeAAC
7566	AAAR5RAAHAAAAFeAAD

LOGICAL
POINT OF DATA

항상
정렬된 순서
유지

EMPNO DEPT ROWID (COMPOSIT COLUMN INDEX)

JOB	EMPNO	ROWID
ANALYST	7788	AAAR5RAAHAAAAFeAAH
ANALYST	7902	AAAR5RAAHAAAAFeAAM
CLERK	7369	AAAR5RAAHAAAAFeAAA
CLERK	7876	AAAR5RAAHAAAAFeAAK
CLERK	7900	AAAR5RAAHAAAAFeAAL
CLERK	7934	AAAR5RAAHAAAAFeAAN
MANAGER	7566	AAAR5RAAHAAAAFeAAD
MANAGER	7698	AAAR5RAAHAAAAFeAAF

2. Index 유형

▣ 유형

구조	종류	Transaction
B * Tree	- **B * Tree Index** - IOT (Index Organized Table) - Descending Index - Reverse Key Index - FBI (Function Based Index)	OLTP
Bitmap	- **Bitmap Index** - Bitmap Join Index - FBI (Function Based Index)	DSS= OLAP = Batch Proecesing

사용되는 컬럼의 개수

- COMPOSITE 컬럼 ①대표를 잘뽑아야 (가장 제한 적인, 가장 자주 사용되는 컬럼을 대표로)
②별볼일 없는 놈들끼리 묶어서 잘해보자
③한번만 찾아보자

걸그룹리더의 기준

ⓐ 조건절에서 해당 컬럼이 항상 사용 되는가
ⓑ 조건절에서 항상 '=' 로 사용 되는가
ⓒ 컬럼의 분포도가 좋은가
ⓓ 정렬(Sort)를 대신 할수 있는가

- SINGLE 컬럼

3. Index 구조 – B * Tree (Balanced Star Tree)

▣ Index 항상 좋은가(빠른가) ?

INDEX 사용이 유리한 상황	① WHERE clause, JOIN condition에서 자주 사용되는 경우 ② 컬럼 데이터 분포도(RANGE OF VALUES)가 넓은 경우(중복이 적은경우) ③ 컬럼이 NULL값을 많이 가지는 경우 ④ TABLE이 크고, 일반적으로 SELECT되는 ROWS가 전체 ROWS의 2~4%이하인 경우 **인덱스 손익 분기점) 10~15% ,10~5%,2~4%**
INDEX 사용이 불리한 상황	① TABLE이 작은 경우 ② WHERE clause, JOIN condition에서 자주 사용되지 않는 경우 ③ 일반적으로 SELECT되는 ROWS가 전체 ROWS의 2~4%이상인경우 ④ TABLE에 빈번한 갱신이 발생하는 경우 - 테이블에 Row가 INSERT, DELETE, UPDATE될때 관련된 INDEX에 INSERT, 삭제표시, DELETE표시 + INSERT 연산발생 - 1개의 테이블에 20개의 index 생성시 , 빈번한 DELETE, UPDATE 연산시

▣ INDEX STRUCTURE 조회

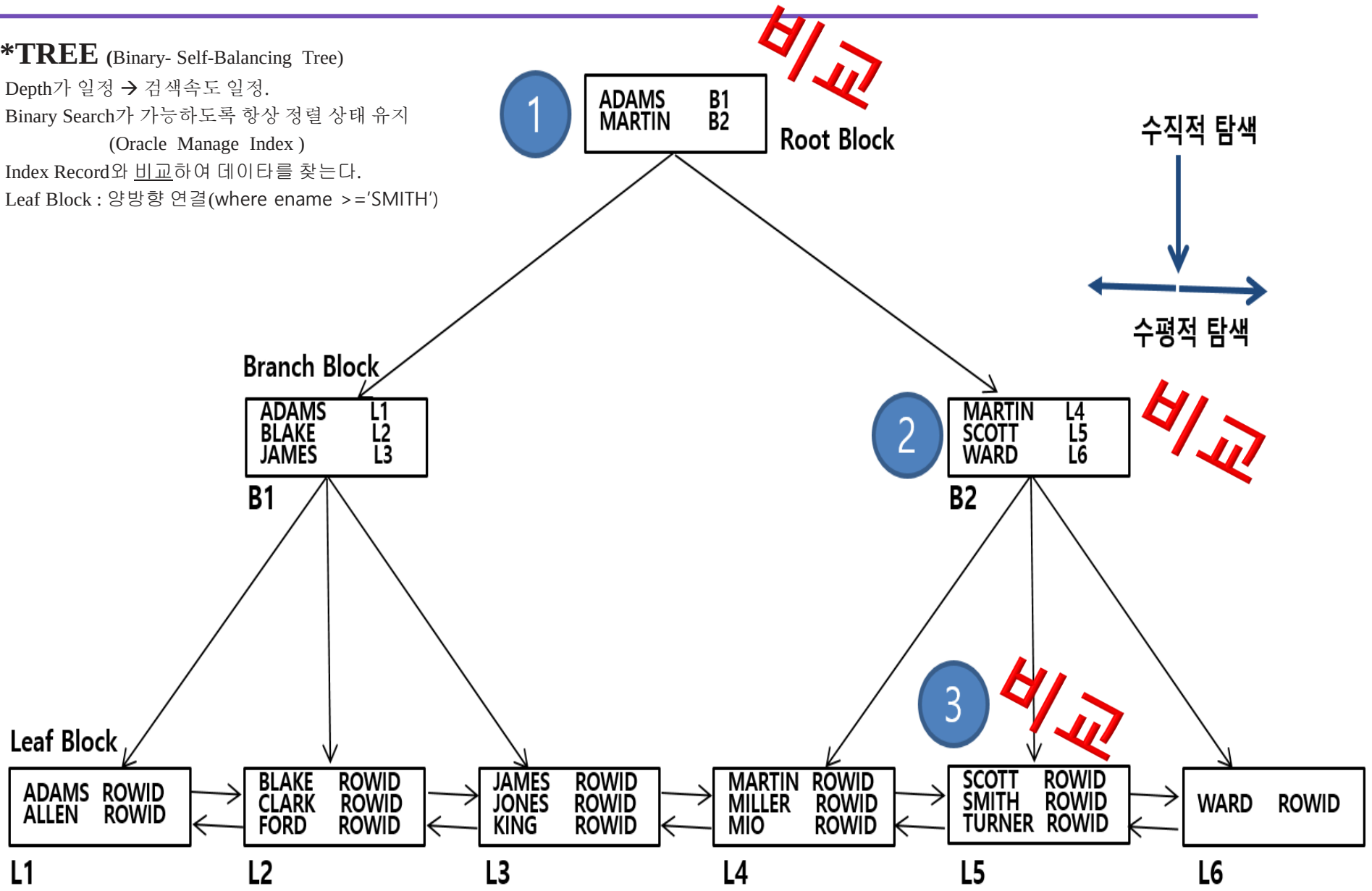
```
ANALYZE INDEX EMP_LARGE_EMPNO_IDX VALIDATE STRUCTURE;

SELECT * FROM INDEX_STATS;

SELECT  NAME,HEIGHT,LF_BLKS,LF_ROWS,BR_BLKS,BR_ROWS,DISTINCT_KEYS,DEL_LF_ROWS
        FROM INDEX_STATS;
```

* Height = Blevel + 1

3. Index 구조 – B * Tree (Balanced Star Tree)

B*TREE (Binary- Self-Balancing Tree)

- Depth가 일정 → 검색속도 일정.
- Binary Search가 가능하도록 항상 정렬 상태 유지
  (Oracle Manage Index)
- Index Record와 비교하여 데이타를 찾는다.
- Leaf Block : 양방향 연결(where ename >='SMITH')



3. Index 구조 – B * Tree (Balanced Star Tree)

▣ Rowid

```
SELECT ENAME,ROWID  FROM EMP ORDER BY ENAME;
 SELECT ROWID,
       SUBSTR(ROWID,1,6)    AS OBJECT_ID,                    -- 6자리
       SUBSTR(ROWID,7,3)    AS DATAFILE_ID,                  -- 3자리
       SUBSTR(ROWID,10,6)   AS BLOCK_ID,                     -- 6자리
       SUBSTR(ROWID,16,3)   AS ROW_NUM                       -- 3자리
 FROM EMP;
 * BASE 64 인코딩(Encoding)을 사용하여 출력 0~9 , a~z, A~Z , + , /

SELECT
          DBMS_ROWID.ROWID_OBJECT(ROWID)             AS OBJECT_ID,          -- Rowid decoding
          DBMS_ROWID.ROWID_RELATIVE_FNO(ROWID)       AS DATAFILE_ID,
          DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)       AS BLOCK_ID,
          DBMS_ROWID.ROWID_ROW_NUMBER(ROWID)         AS ROW_NUM
FROM    EMP;

SELECT  * FROM  DBA_OBJECTS WHERE OBJECT_ID = ? ;                  -- DBA계정에서
SELECT  * FROM  DBA_DATA_FILES WHERE FILE_ID = ? ;
SELECT  SEGMENT_NAME,EXTENT_ID,BLOCK_ID,BLOCKS,BYTES   FROM DBA_EXTENTS
WHERE  SEGMENT_NAME='EMP_LARGE';
```

4. Index 사용 원리 및 특징

▣ INDEX를 통해서 데이터를 찾는 원리

B* Tree 인덱스는 Root 노드 에서 Branch 노드를 거쳐서 목적지인 Leaf 노드에 도달(**수직적 탐색**)할 때 까지 계속해서 값을 **비교**해 가면서 목적지에 (Leaf 노드/블록) 도달 한다. 목적지 블럭내/블럭간을 **수평적 탐색**을 하며 Index record 를 비교하여 원하는 **Rowid를 얻는 과정.**

* **B* Tree 인덱스에서 데이터를 찾는 원리는 비교 .**

비교가 불가능한 경우 인덱스를 사용할수 없다

- 값을 비교하기 전에 변형이 발생하면 비교를 할수 없기 때문에 index 사용(X)
- 부정형(NOT,<>,!=,<>)인 경우 직접 비교할대상이 없기 때문에 index사용(X)
- NULL인 경우 index에 저장되지 않기 때문에 NULL로 비교하는 경우 index 사용(X)

ex)

```
where   substr(job,1,4)  =  'CLER';
where   job||dept  =  'CLERK10';
where   telno      =  23456789;          // 암시적 형변환 : telno(문자)  <  23456789(숫자)
where   telno      like '2%';            // 암시적 형변환 : to_char(telno) like '2%'
where   to_number(telno) =  23456789;    // 명시적 형변환 :to_number(telno) =  23456789 ;
                                         //  Lvalue vs Rvalue 간의 data type이 일치해야 변형이 발생하지 않는다.
```

4. Index 사용 원리 및 특징

▣ INDEX를 사용하지 못하는 경우

◎ INDEX 사용이 비효율적인 경우 (누가? 어떤 기준으로?)

◎ 비교가 불가능한 경우

① 인덱스가 걸려있는 컬럼에 변형(명시적,암시적) 이 발생한 경우 비교하기 전에 인덱스가 걸려 있는 컬럼에 변형이 발생한다면 비교를 할수 없기 때문에 인덱스를 사용할수 없다.

② NULL 비교

NULL은 비교연산자(ex = , >,<, <=, <= ..)를 사용해서 비교가 불가능 하고 인덱스 컬럼에 NULL은 저장되지 않는다. 따라서 NULL 은 인덱스를 통해서 검색 할 수 없다.

③ 부정형 비교

```
WHERE ID <> '05652222';
```

4. Index 사용 원리 및 특징

▣ 데이터 접근 경로

데이터를 읽는 작업을 스캔(SCAN)이라 하고 스캔을 수행하는 방식을 접근경로 (Access Path)라 한다. 접근경로는 Row Source에서 데이터를 추출하는 방식(경로)을 의미

FULL TABLE SCAN	테이블의 전체 데이터를 읽어 조건에 맞는 데이터를 추출 - 데이타 BLOCK만을 읽는다 - **다중 BLOCK I/O** (DB_FILE_MULTIBLOCK_READ_COUNT) **& Sequential Access** - PARALLEL QUERY 사용 - SELECT되는 ROW의 수가 많은 경우에 유리
INDEX SCAN - Index Unique Scan - Index Range Scan - Index Full Scan - Index Fast Full Scan - Index Skip scan	인덱스 스캔을 통해 얻은 Rowid를 사용하여 테이블에서 조건에 맞는 데이터를 추출 - INDEX BLOCK과 데이타 BLOCK을 읽는다 - **단일 BLOCK I/O & Random Access** - SELECT되는 ROW의 수가 적은 경우에 유리
ROWID SCAN	ROWID를 직접 사용하여 데이터를 추출

4. Index 사용 원리 및 특징

▣ INDEX 손익분기점

7.X 버전	전체 데이터중 15% 이하 접근
8.X 버전	전체 데이터중 10% 이하 접근
9.X 버전	전체 데이터중 5% 이하 접근

?? Index는 소량의 데이터 접근시 효율적이다. 소량의 기준은?

5억건 데이터의 5%는 2,500만건

▣ INDEX SCAN vs FULL TABLE SCAN

	INDEX SCAN	FULL TABLE SCAN
Access	Random Access	Sequential Access
Block I/O	1 Block I/O	Multi Block I/O
Path	2 Path (Index → Table)	1 Path (Table)
Transaction Type	OLTP	DSS (=Batch)
Join Type	Nested Loop Join	Sort Merge Join

4. Index 사용 원리 및 특징

	INDEX FULL SCAN	INDEX FAST FULL SCAN
Block Read	논리적 정렬된 순서	물리적 저장된 순서
Block IO	1 Block IO	Multi Block IO
출력결과	정렬된 결과	정렬되지 않는 결과
사용 컬럼 제약	인덱스에 포함되지 않는 컬럼 사용 가능	인덱스에 포함된 컬럼만 사용 가능

5. Index 와 DML 연산

테이블 데이터에 DML 연산 발생시 DBMS가 관련 인덱스 관리를 자동.

```
SELECT    TABLE_NAME, INDEX_NAME
FROM      USER_INDEXES   WHERE   TABLE_NAME IN ('CUSTOMER','EMP_LARGE')    ORDER    BY 1;
```

CUSTOMER 테이블에
- 1 Row INSERT시 4개의 인덱스에 각각 인덱스 레코드가 정렬된 순서로 INSERT.
- 1 Row DELETE시 4개의 인덱스에 각각 인덱스 레코드가 DELETE 표시
- 1 Row UPDATE시 관련 인덱스에 인덱스 레코드가 DELETE 표시후 새로운 인덱스 레코드를 INSERT .

인덱스 관리 및 생성과 관련해서 이서 이해 필요한 2가지 사항.

① 인덱스 레코드는 삭제 되지 않고 삭제 표시만 한다. 테이블에 빈번한 DELETE, UPDATE 연산이 발생하면 삭제 표시가 된 인덱스 레코드가 늘어난다. 인덱스 블록내에 사용하지 않는 공간이 증가 함에 따라 인덱스 스캔의 효율성이 떨어지게 된다. 때로는 인덱스가 테이블 보다 더많은 저장 공간(Tablespace)을 차지 한다. 빈번한 DML연산이 발생하는 테이블에 생성된 인덱스는 주기적인 사용 현황 모니터링을 통해 인덱스 재구성(Reorganization) 필요 재구성은 DBA의 역할이지만 튜닝을 하는 개발자 입장에서도 재구성의 필요성 이해.

② 인덱스 갯수

1개 테이블에 인덱스가 20개 있으면 DML 연산 발생시 어떤일이 발생할까?
- 1 Row INSERT시 20개의 인덱스에 각각 인덱스 레코드가 정렬된 순서로 INSERT된다.
- 1 Row DELETE시 20개의 인덱스에 각각 인덱스 레코드가 DELETE 표시 된다.
- 1 Row UPDATE시 관련 인덱스에 인덱스 레코드가 DELETE 표시후 새로운 인덱스 레코드를 INSERT 한다.

** 인덱스를 테이블에 접근하는 각 어플리케이션의 개별 요구에 의해서 만들게 되면 수십개의 인덱스를 각각 생성하고 트랜잭션이 빈번하게 많이 발생할때 응답시간이 느려지게 된다. 전체 관점에서 인덱스 요구 사항을 파악해서 최소한의 인덱스를 생성하도록 인덱스 설계 전략을 가져야 한다.

6. Index Scan 실습 1/2

실행계획 실습시

```
* INDEX UNIQUE SCAN
SELECT * FROM CUSTOMER WHERE  ID = '05333333';

* INDEX RANGE SCAN
SELECT * FROM CUSTOMER WHERE  NAME = '김민준';
SELECT * FROM CUSTOMER WHERE  ID >= '05652222';

* INDEX FULL SCAN  OR INDEX RANGE SCAN
SELECT ID FROM CUSTOMER WHERE  ID >= '05652222';

* SORT를 대신하는 INDEX SCAN
SELECT * FROM CUSTOMER ORDER BY ID;
SELECT ID FROM CUSTOMER ORDER BY ID;

* INDEX FAST FULL SCAN
SELECT COUNT(ID) FROM  CUSTOMER;

* ROWID SCAN
SELECT ROWID,ID,NAME,EMAIL FROM CUSTOMER   WHERE  ROWNUM < 10;
SELECT * FROM   CUSTOMER WHERE ROWID = 'AAASO8AAHAAAzuzAAE';
```

6. Index Scan 실습 1/2

```
** INDEX 활용
SELECT MIN(ID) FROM CUSTOMER;
SELECT MAX(ID) FROM CUSTOMER;
SELECT MAX(ID),MIN(ID) FROM CUSTOMER;
SELECT * FROM CUSTOMER WHERE  ID <> '05333333';
SELECT * FROM CUSTOMER WHERE  ID < '05333333';
SELECT * FROM CUSTOMER WHERE  ID LIKE '0533333%';
SELECT * FROM CUSTOMER WHERE  ID LIKE '%0533333%';
SELECT * FROM CUSTOMER WHERE  SUBSTR(ID,1,6) ='053333';
SELECT * FROM CUSTOMER WHERE  ID IS NULL;
SELECT * FROM CUSTOMER WHERE  ID IS NULL;
```

6. Index Scan 실습 2/2

▣ 데이터딕셔니리에 있는 객체 통계정보(Object Statistics)

```
SELECT NUM_ROWS,BLOCKS,EMPTY_BLOCKS,AVG_SPACE,AVG_ROW_LEN,SAMPLE_SIZE,LAST_ANALYZED
FROM USER_TABLES WHERE TABLE_NAME='CUSTOMER';

SELECT * FROM USER_TAB_COLUMNS WHERE TABLE_NAME ='CUSTOMER';

SELECT COLUMN_NAME,DATA_TYPE,DATA_LENGTH,NULLABLE,NUM_DISTINCT,LOW_VALUE,HIGH_VALUE,DENSITY,
NUM_NULLS,LAST_ANALYZED,SAMPLE_SIZE
FROM    USER_TAB_COLUMNS
WHERE   TABLE_NAME ='CUSTOMER';

SELECT TABLE_NAME,COLUMN_NAME,ENDPOINT_NUMBER,ENDPOINT_ACTUAL_VALUE FROM USER_HISTOGRAMS
WHERE TABLE_NAME ='CUSTOMER'
ORDER BY COLUMN_NAME,ENDPOINT_NUMBER;

SELECT TABLE_NAME,COLUMN_NAME,ENDPOINT_NUMBER,ENDPOINT_ACTUAL_VALUE FROM USER_HISTOGRAMS
WHERE TABLE_NAME ='CUSTOMER' AND COLUMN_NAME='ID'
ORDER BY COLUMN_NAME,ENDPOINT_NUMBER;

SELECT TABLE_NAME,INDEX_NAME,UNIQUENESS,CLUSTERING_FACTOR,NUM_ROWS,SAMPLE_SIZE,LAST_ANALYZED
FROM USER_INDEXES
WHERE TABLE_NAME='CUSTOMER';
```