

# ORACLE INDEX

Created by [Jongwon](#)

# INDEX?

An index is an optional **structure**,  
associated with a table or table cluster,  
that can sometimes **speed data access**

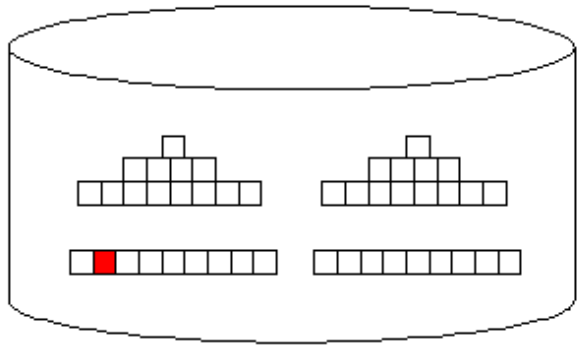
# 데이터가 매우 크다면?

200GB 정도 되는 인구 테이블

5억명 정도로 구성되어 있을 때

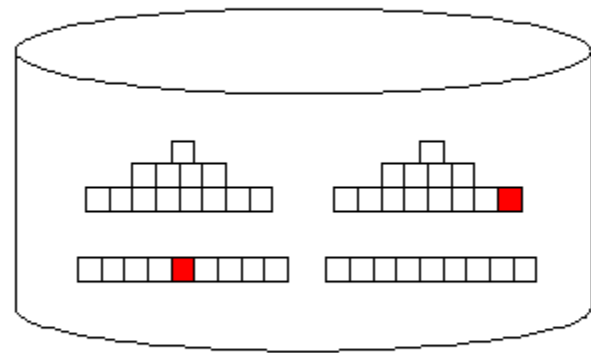
특정 사람들을 찾을 경우

# 데이터의 검색



Oracle Full-table scan

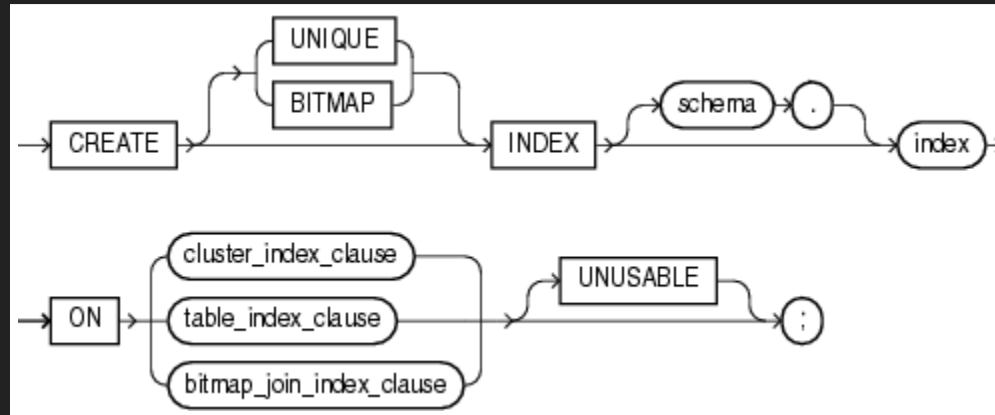
처음부터 하나하나 검색해야 한다.



Oracle Index access

Index를 통해 검색한다.

# CREATE INDEX



```
CREATE [ UNIQUE | BITMAP ] INDEX [ schema. ] index
  ON { cluster_index_clause
      | table_index_clause
      | bitmap_join_index_clause
      }
[ UNUSABLE ] ;
```

# KEY

```
-- HR Schema
CREATE INDEX emp_test_idx ON employees(employee_id);
                                ↑ KEY
```

**column** 또는 **expressions** 들의 집합

논리적인 개념으로 Index와는 다르다.

# ROWID

```
-- HR Schema  
SELECT ROWID, employee_id, last_name  
FROM employees;
```

테이블에 값이 실제로 저장되어 있지는 않은

pseudo column

# ROWID

데이터 오브젝트 번호, 데이터 파일 번호, 블록 번호, 로우 번호를 포함





# INDEX의 종류

- B\*Tree Indexes
- Bitmap Indexes
- Function-Based Indexes

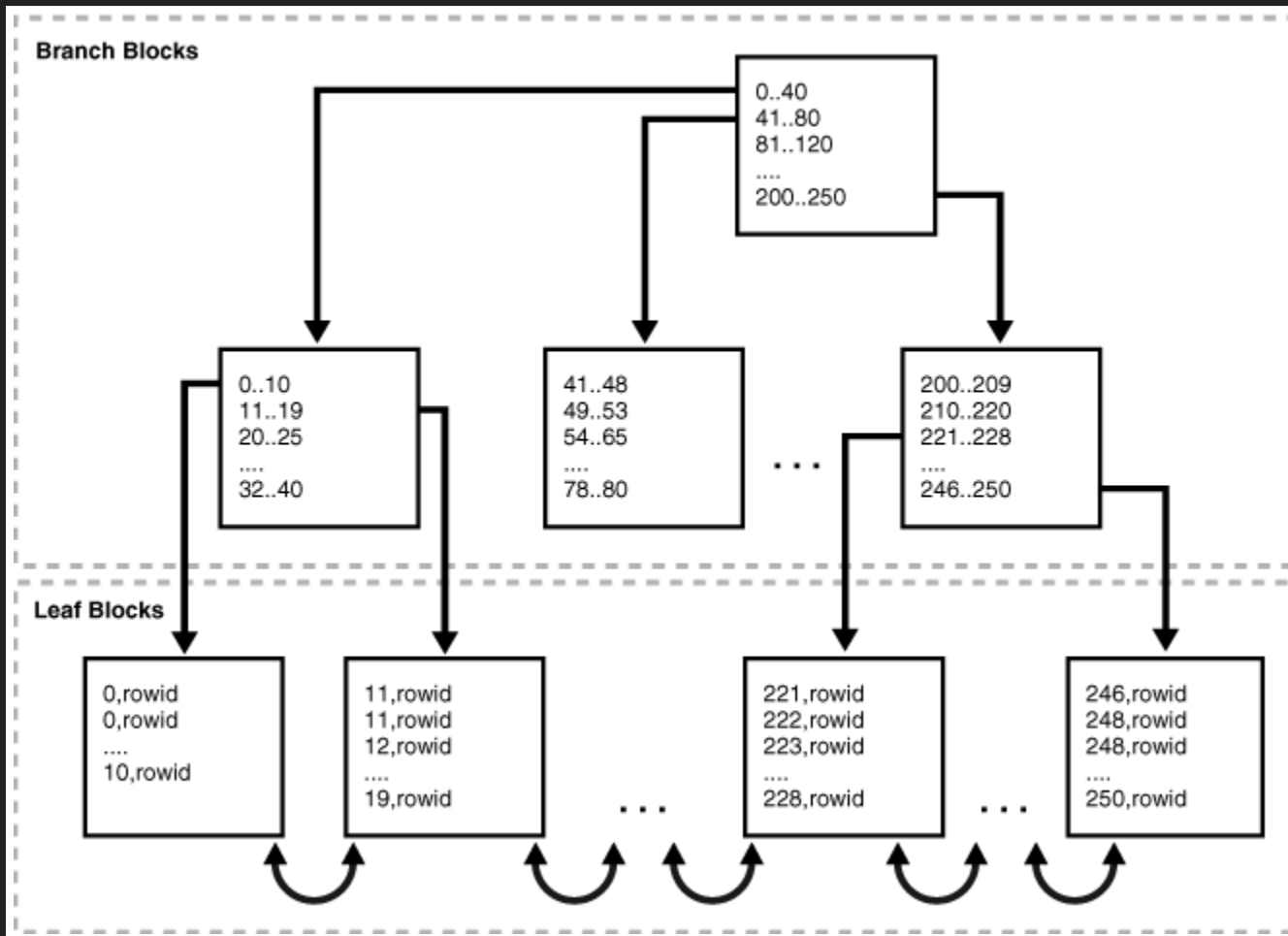
# B\*TREE INDEX

Binary Trees가 아닌 Balanced Trees

책 뒷면의 색인과 비슷한 방법

ROWID를 KEY 값과 함께 저장하는 인덱스 구조

# BALANCED TREES



# BRANCH BLOCKS

하위 노드 Block을 찾아가기 위한 Data Block Address 정보를 갖는다.

---

# LEAF BLOCKS

KEY 컬럼과 함께 해당 테이블 레코드를 찾아가기 위한 주소정보(ROWID)를 갖는다.

# B\*TREE INDEX SUBTYPES

- Index-organized tables
- Reverse key Indexes
- Descending Indexes
- B\*tree cluster Indexes

# BITMAP INDEX

```
-- SH Schema
SELECT cust_id, cust_last_name, cust_marital_status, cust_gender
FROM customers
WHERE ROWNUM < 100
ORDER BY cust_id;
```

Value	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

# BITMAP INDEX

Distinct Value의 수가 적거나 DML문이 거의 사용되지 않는 경우

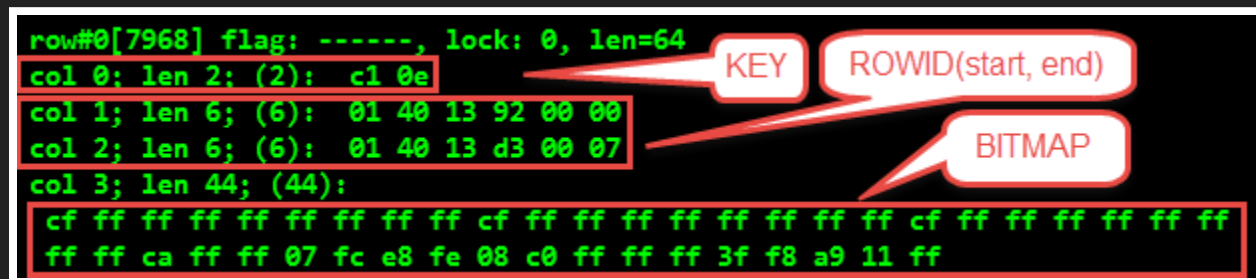
읽기 위주의 대용량 데이터 환경(Data Warehousing)에 적합

두개 이상의 Bitmap을 활용한 Bitwise 연산으로 여러 인덱스를 동시에 활용

# BITMAP INDEX STRUCTURE

아래 화면은 SH 스키마의 SALES\_PROD\_BIX 라는 BITMAP INDEX를 DUMP한 것입니다.

```
row#0[7968] flag: -----, lock: 0, len=64
col 0; len 2; (2): c1 0e
col 1; len 6; (6): 01 40 13 92 00 00
col 2; len 6; (6): 01 40 13 d3 00 07
col 3; len 44; (44):
cf ff ff ff ff ff ff ff cf ff ff ff ff ff ff ff cf ff ff ff ff ff ff
ff ff ca ff ff 07 fc e8 fe 08 c0 ff ff ff 3f f8 a9 11 ff
```





# FUNCTION BASED INDEX (FBI)

테이블에 Index를 생성할 때 하나 이상의 열에 함수나 expressions를 포함하는 것.

B\*Tree Index나 Bitmap Index가 될 수 있다.

# IMPLICIT 형변환 문제

```
-- SH Schema, SET AUTOTRACE ON
CREATE TABLE promotion_test AS SELECT * FROM PROMOTIONS;
ALTER TABLE promotion_test ADD v_category VARCHAR2(2);
UPDATE promotion_test SET v_category = promo_category_id;
CREATE INDEX before_test ON promotion_test(v_category);

SELECT promo_id, promo_name FROM promotion_test WHERE v_category = 10
```

## Execution Plan

Plan hash value: 2866056181

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		20	660	5 (0)	00:00:01
* 1	TABLE ACCESS FULL	PROMOTION_TEST	20	660	5 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter(TO\_NUMBER("V\_CATEGORY")=10)

# IMPLICIT 형변환 문제

```
DROP INDEX before_test;  
CREATE INDEX after_test ON promotion_test(to_number(v_category));  
SELECT promo_id, promo_name FROM promotion_test where v_category = 10
```

## Execution Plan

Plan hash value: 1895496721

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		20	860	4 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	PROMOTION_TEST	20	860	4 (0)	00:00:01
* 2	INDEX RANGE SCAN	AFTER_TEST	20		1 (0)	00:00:01

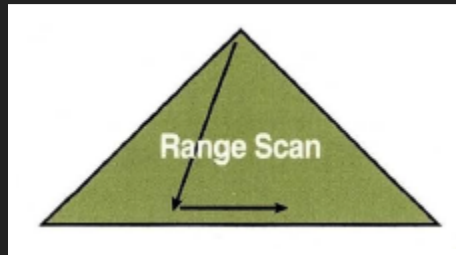
Predicate Information (identified by operation id):

2 - access(TO\_NUMBER("V\_CATEGORY")=10)

# INDEX SCAN의 종류

- Index Range Scan
- Index Range Scan Descending
- Index Full Scan
- Index Unique Scan
- Index Fast Full Scan
- Index Skip Scan

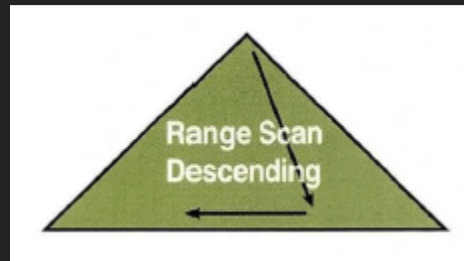
# INDEX RANGE SCAN



Branch block을 수직적으로 스캔한 후 Leaf block에서 Key값의 범위만큼을 스캔

**스캔 범위와 테이블 액세스 횟수가 속도의 관건**

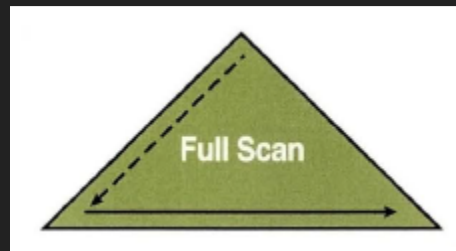
# INDEX RANGE SCAN DESCENDING



Index Range Scan과 기본적으로 동일하나 Index를 뒤에서부터 스캔

ORDER BY ~ DESC 를 사용했을 경우

# INDEX FULL SCAN

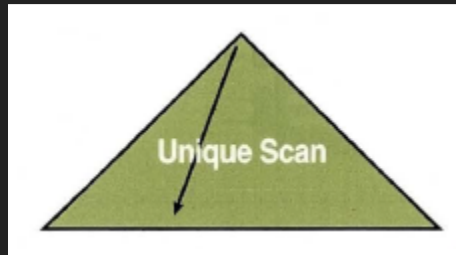


Leaf block을 처음부터 끝까지 수평적으로 스캔

테이블이 대용량인데 Key값을 이용한 **조건절로 필터링되는 데이터가 극히 일부**인 경우

Index Range Scan에서 Index를 구성하는 컬럼이 조건절에 사용되지 않았을 경우 발생

# INDEX UNIQUE SCAN

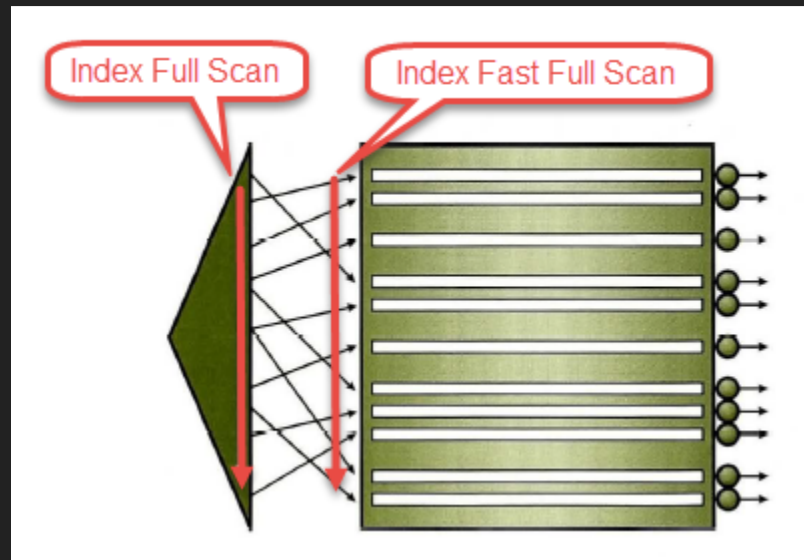


수직적 스캔만으로 데이터를 찾는 방식

Unique Key를 통해 **등호** 조건으로 탐색하는 경우



# INDEX FAST FULL SCAN



Index 트리 구조를 무시하고, 물리적으로 디스크에 저장된 순서대로 Leaf block을 스캔

# INDEX SKIP SCAN



Root 또는 Branch block에서 읽은 Key값 정보를 이용해서

조건에 맞는 레코드를 포함하는 Leaf Block을 골라서 액세스

Distinct Value가 적은 선행 Key가 조건절에서 누락됐고 후행 Key의 Distinct Value가 많을 때 효과적

# INDEX 생성 지침

- 조건절에 항상 사용되거나, 자주 등장하는 컬럼을 선정한다.
- 등호 조건으로 자주 조회되는 컬럼들을 앞쪽에 둔다.

# 왜 INDEX가 사용되지 않았나

- Optimizer가 인덱스를 사용하기에 효율적이지 않다고 판단할 경우
- Function Based Index를 사용하지 않고 인덱스 컬럼을 조건절에서 가공할 경우
- is (not) null 조건이나 <> 비교를 사용할 경우
- Column에 산술 연산자(mathematical operation)를 사용했을 경우
- Column에 연결 연산자(concatenate)를 사용했을 경우
- 데이터 타입의 묵시적(implicit) 형변환

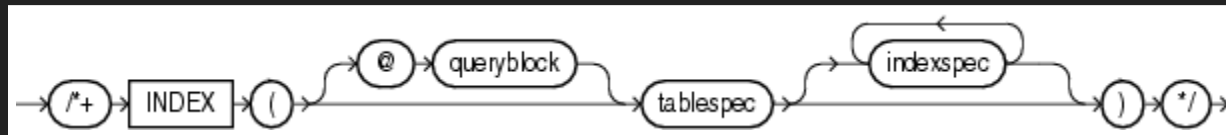
---

예외가 있을 수 있고, 정상적으로 인덱스 사용이 불가능할 뿐  
사용자체가 불가능한 것은 아니다.

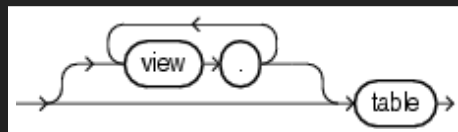
# HINT

Comment를 사용하여 Optimizer가 HINT를 사용해 Execution Plan을 선택하게 할 수 있다.

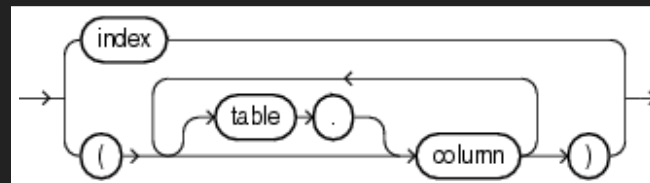
## INDEX Hint



`tablespec::=`



`indexspec::=`



# QUERY BLOCK

```
EXPLAIN PLAN FOR  
SELECT employee_id FROM employees;
```

```
1* SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY(NULL, NULL, 'ALL'))  
  
PLAN_TABLE_OUTPUT  
-----  
Plan hash value: 3788064173  
  
-----  
| Id | Operation          | Name          | Rows | Bytes | Cost (%CPU)| Time     |  
-----  
|  0 | SELECT STATEMENT   |               |    21 |    84 |    1   (0)| 00:00:01 |  
|  1 |  INDEX FULL SCAN   | EMP_EMP_ID_PK |    21 |    84 |    1   (0)| 00:00:01 |  
-----  
  
Query Block Name / Object Alias (identified by operation id):  
-----  
  
  1 - SEL$1 / EMPLOYEES@SEL$1  
  
Column Projection Information (identified by operation id):  
-----  
  
  1 - "EMPLOYEE_ID"[NUMBER,22]  
  
18 rows selected.
```

# INDEX 조회

Data Dictionary의 뷰로 조회할 수 있다.

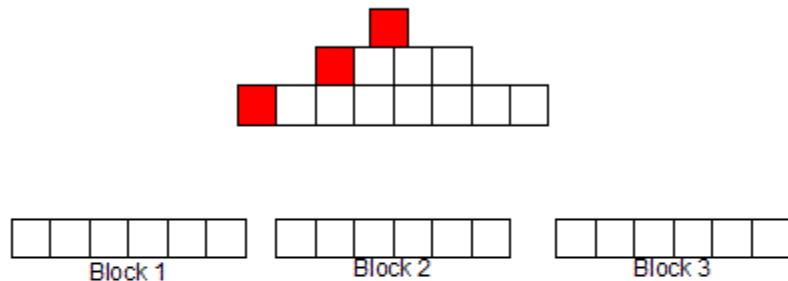
```
-- 현재 유저 내의 INDEX
DESC user_indexes
DESC user_ind_columns

-- 현재 유저의 권한으로 사용할 수 있는 모든 INDEX
DESC all_indexes
DESC all_ind_columns

-- 모든 유저의 INDEX
DESC dba_indexes
DESC dba_ind_columns
```

# CLUSTERING FACTOR

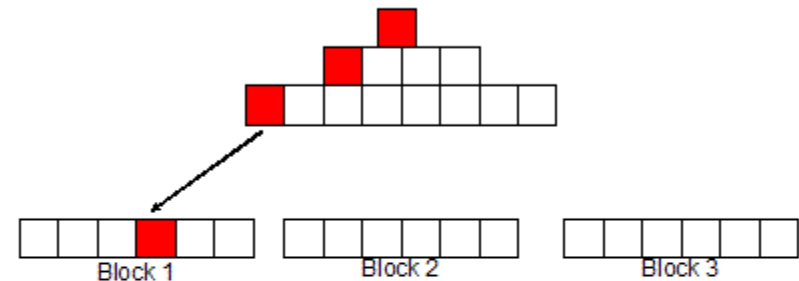
인접한 Leaf block들이 가리키는 Data block에 데이터가 밀집된 정도



Select order\_nbr, item\_name from ordor natural join item;

## Un-Clustered table rows

Clustering\_factor ~= num\_rows



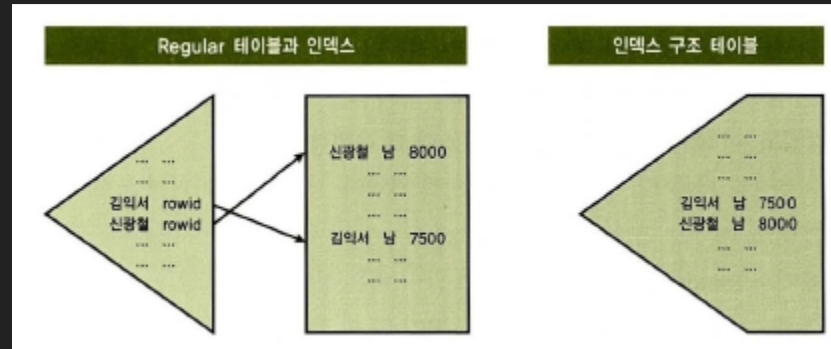
Select order\_nbr, item\_name from ordor natural join item;

## Clustered table rows

Clustering\_factor ~= blocks



# Heap-Organized Table vs Index-Organized Table



```
-- SH Schema
CREATE TABLE index_org_t (a PRIMARY KEY, b)
ORGANIZATION INDEX
AS SELECT promo_id, promo_name FROM promotions;

SELECT index_name, index_type, clustering_factor
FROM user_indexes
WHERE table_name IN ('PROMOTIONS', 'INDEX_ORG_T');
```

# HIGHLIGHT

- KEY, ROWID
  - Balanced Trees
  - Bitmap Index
  - Implicit Issue
  - Index Scan
  - Why Indexes Aren't Used
  - HINT
  - Clustering Factor
- 

# REFERENCE

- Oracle Concepts - Administering Oracle Indexes
- Index Column Order Does Matter
- Efficient SQL Statements
- Indexes and Index-Organized Tables
- 조시형. 『오라클 성능 고도화 원리와 해법 I, II』 비투엔컨설팅