파티션 테이블, 파티션 인덱스

Version	Date	Author	Description
1.0	2014-09-12	강아름	최초 작성
1.1	2025-12-10	이지영	

TMAXTibero

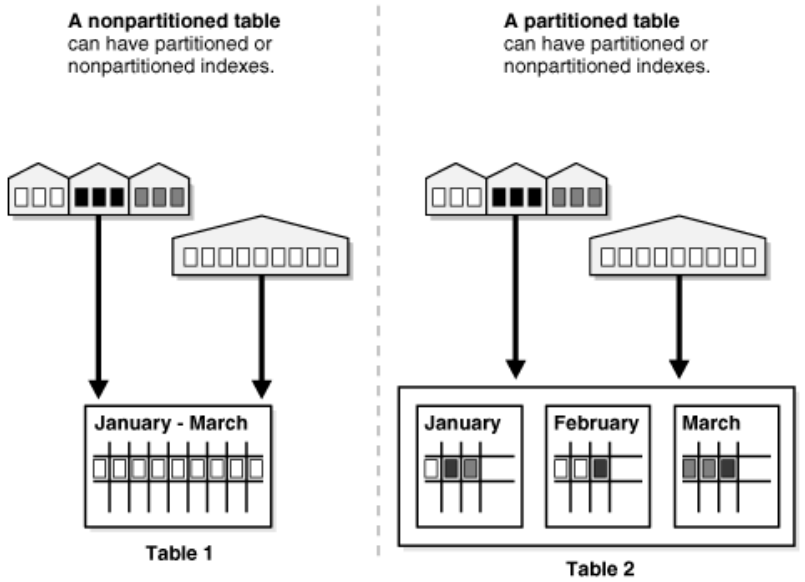
Contents

01 파티셔닝 정의

파티셔닝이란?

- 파티셔닝이란 테이블, 인덱스를 보다 작은 단위로 분할

- 한층 세분화된 레벨의 데이터베이스 오브젝트 관리 및 접근을 할 수 있도록 해주는 기능



TMAXTibero

02 파티션 테이블

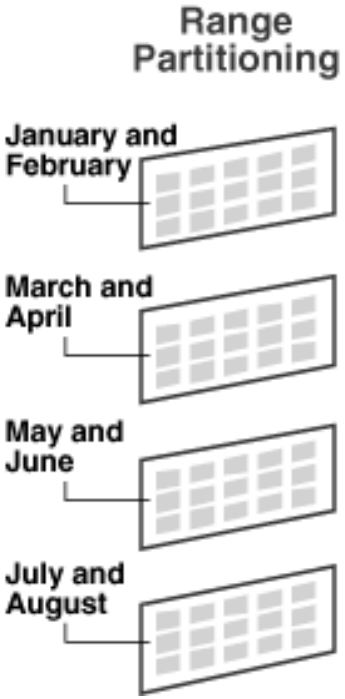
1. Range 파티션
2. Hash 파티션
3. List 파티션
4. Composite 파티션

1. 범위 파티션 (1/2)

[특징]

- 테이블은 여러 개의 세그먼트로 구성되며 각 세그먼트는 특정 컬럼 값이 동일한 범위 안에 들어 오는 로우로 구성
- 범위를 잘못 판단하면 데이터 비대칭 적재 발생
- Partition Pruning 가능

Partition Pruning : 쿼리 실행 시 조건절에 따라 필요한 파티션만 접근하고 불필요한 파티션은 자동으로 제외하는 최적화 기법

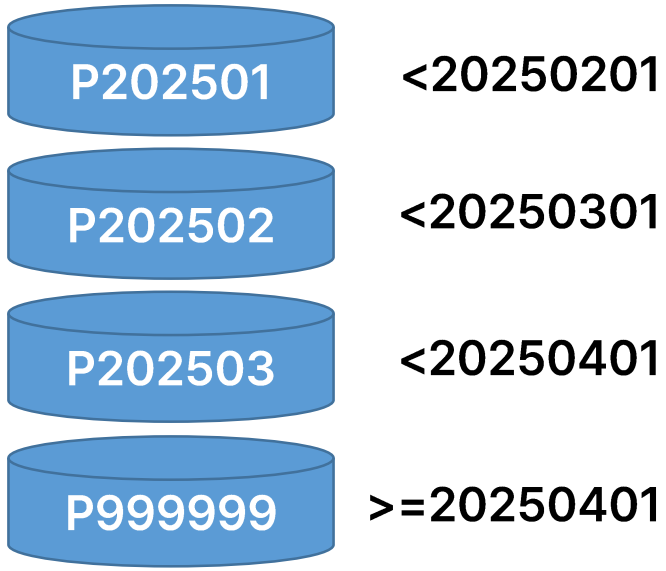


[활용 방법]

- 파티션 키 : 실적년월, 등록년월 등
- Range 조건으로 파티션 제거 (Partition Elimination) 효과를 얻고자 하거나 과거 데이터를 Purge 할 때 활용
- 시계열성 데이터에 대한 액세스 및 관리

1. 범위 파티션 (2/2)

```
CREATE TABLE range_table (
col1 DATE,
col2 VARCHAR2(8),
col3 VARCHAR(5)
)
PARTITION BY RANGE(col1)(
PARTITION p202501 VALUES LESS THAN('20250201')
,PARTITION p202502 VALUES LESS THAN('20250301')
,PARTITION p202503 VALUES LESS THAN('20250401')
,PARTITION p999999 VALUES LESS THAN(MAXVALUE)
);
SELECT * FROM range_table WHERE col1 BETWEEN 20250301 AND 20250331;

Execution Plan
----------------------------------------------------------------------------
1 PARTITION RANGE (SINGLE PART) (Cost:26, %CPU:0, Rows:1) (PS:3, PE:3)
2 TABLE ACCESS (FULL): RANGE_TABLE (Cost:26, %CPU:0, Rows:1)
```



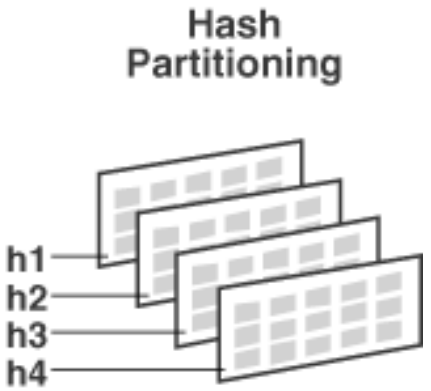
2. 해시 파티션 (1/2)

[특징]

- 명시된 파티션 모두에 데이터 **고르게 분산**
- 로우들은 파티션 키로 **해쉬 값**을 취해 각 파티션으로 매핑
- **파티션 수는 2의 n배수로 설정**

(2의 n배수가 아닌 경우, 파티션 수 변경 시 각 파티션 간 크기의 균형이 깨질 수 있다)

- 파티션 키 : 고객번호,주문번호,청약번호 등



[활용 방법]

- 병렬처리의 I/O 분산
- Static Partitioning에 의한 병렬 처리
- 주어진 Range에 어느 정도의 Data가 Mapping 될지 예측하기 어려운 환경이나 Range별 Data량의 차이가 많이 나는 업무에 적당

2. 해시 파티션 (2/2)

```
CREATE TABLE hash_table1 (
col1 VARCHAR2(8),
col2 VARCHAR2(8),
col3 VARCHAR(5)
)PARTITION BY HASH(col1) PARTITIONS 4;
```

[100만개 입력 후 파티션별 count]

_PPT_P845400	_PPT_P845500	_PPT_P845600	_PPT_P845700
249553	250471	249553	250506

[Tibero7]

SYS_P1011	SYS_P1012	SYS_P1013	SYS_P1014
249680	250356	250384	249580

[Oracle19c]

```
CREATE TABLE hash_table2 (
col1 VARCHAR2(8),
col2 VARCHAR2(8),
col3 VARCHAR(5)
)PARTITION BY HASH(col1) PARTITIONS 5;
```

[100만개 입력 후 파티션별 count]

_PPT_P845900	_PPT_P846000	_PPT_P846100	_PPT_P846200	_PPT_P846300
125432	248933	250091	250818	124726

[Tibero7]

SYS_P1015	SYS_P1016	SYS_P1017	SYS_P1018	SYS_P1019
125369	249444	249316	250529	125342

[Oracle19c]

3. 리스트 파티션 (1/2)

[특징]

- **연속성이 없는 컬럼값**을 각 파티션으로 분할
- 데이터들에 순서가 없고 관련성이 없더라도 그룹핑 가능
- 하지만 반드시 하나의 컬럼만 파티션 키로 지정이 가능(다중 파티션 키 지정 불가)
- IOT는 LIST PARTITION 생성 불가
- MAX VALUE와 같은 UPPER-BOUND 정의 불가

[활용 방법]

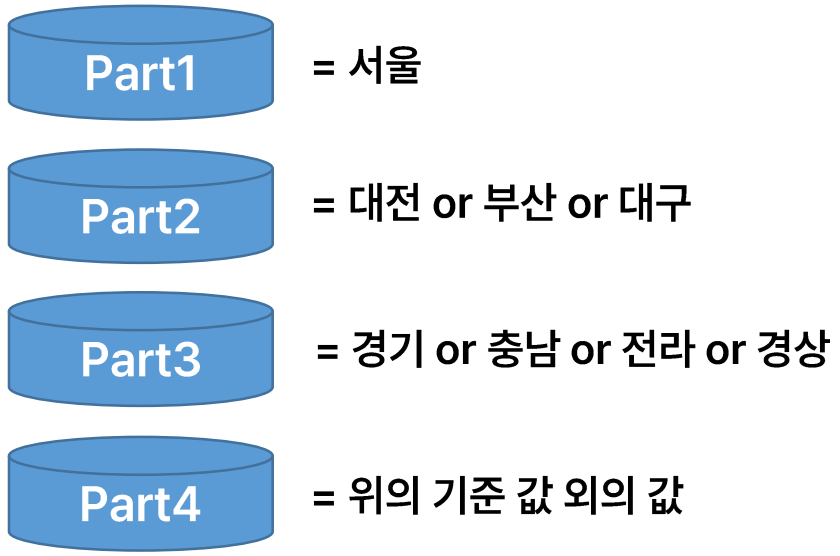
- 일정한 순서가 없는 데이터를 인위적으로

그룹핑 하고자 하는 경우 활용

- BUSINESS 친화도에 따른 이상 데이터의 그룹핑

3. 리스트 파티션 (2/2)

```
CREATE TABLE list_table (
col1 VARCHAR2(8),
col2 VARCHAR2(8),
col3 VARCHAR(5)
)
PARTITION BY LIST(col1)(
PARTITION part1 VALUES ('서울') TABLESPACE T1
,PARTITION part2 VALUES ('대전', '부산','대구') TABLESPACE T2
,PARTITION part3 VALUES ('경기','충남','전라','경상') TABLESPACE T3
,PARTITION part4 VALUES (DEFAULT)
);
```



4. 복합 파티션 (1/3)

[특징]

■ 일차적으로 Partitioning을 한 후

각 파티션 내에서 partitioning으로

sub-partition 생성

■ Historical data 와 disk-striping 모두를 적용

할 수 있는 이상적인 방법

[활용방법]

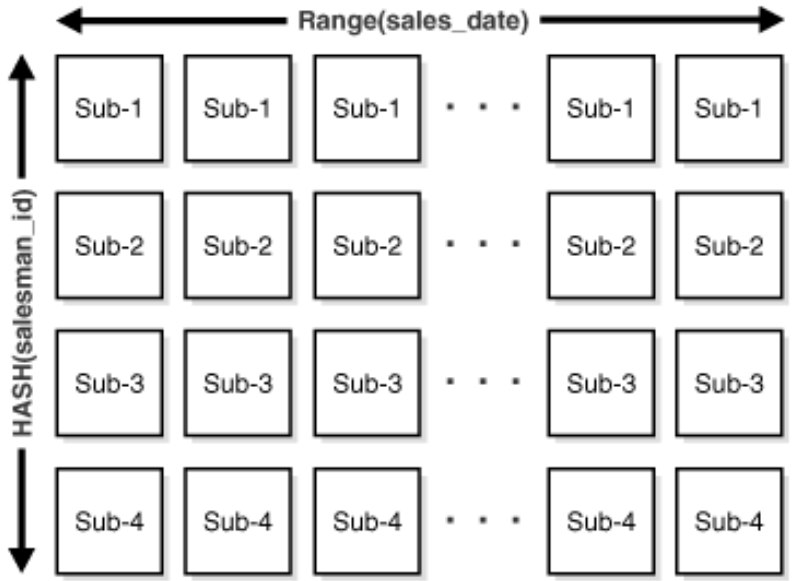
■ 주문일자로 Range partitioning 하고,

주문번호로 Hash Sub-partitioning

■ 비연속적인 데이터를 시계 열로 관리하며

주기적인 DB 관리 작업이 목적의 경우

RANGE + LIST PARTITION 고려



4. 복합 파티션 (2/3)

```
CREATE TABLE sales_composite
(salesman_id NUMBER(5) ,
salesman_name VARCHAR2(30) ,
sales_amount NUMBER(10) ,
sales_date DATE)
PARTITION BY RANGE(sales_date)
SUBPARTITION BY HASH(salesman_id)
SUBPARTITION TEMPLATE(
SUBPARTITION sp1 TABLESPACE ts1 ,
SUBPARTITION sp2 TABLESPACE ts2 ,
SUBPARTITION sp3 TABLESPACE ts3 ,
SUBPARTITION sp4 TABLESPACE ts4)
(PARTITION sales_jan2025 VALUES LESS THAN(TO_DATE('02/01/2025','MM/DD/YYYY'))
PARTITION sales_feb2025 VALUES LESS THAN(TO_DATE('03/01/2025','MM/DD/YYYY'))
PARTITION sales_mar2025 VALUES LESS THAN(TO_DATE('04/01/2025','MM/DD/YYYY'))
PARTITION sales_apr2025 VALUES LESS THAN(TO_DATE('05/01/2025','MM/DD/YYYY'))
PARTITION sales_may2025 VALUES LESS THAN(TO_DATE('06/01/2025','MM/DD/YYYY')));
```

4. 복합 파티션 (3/3)

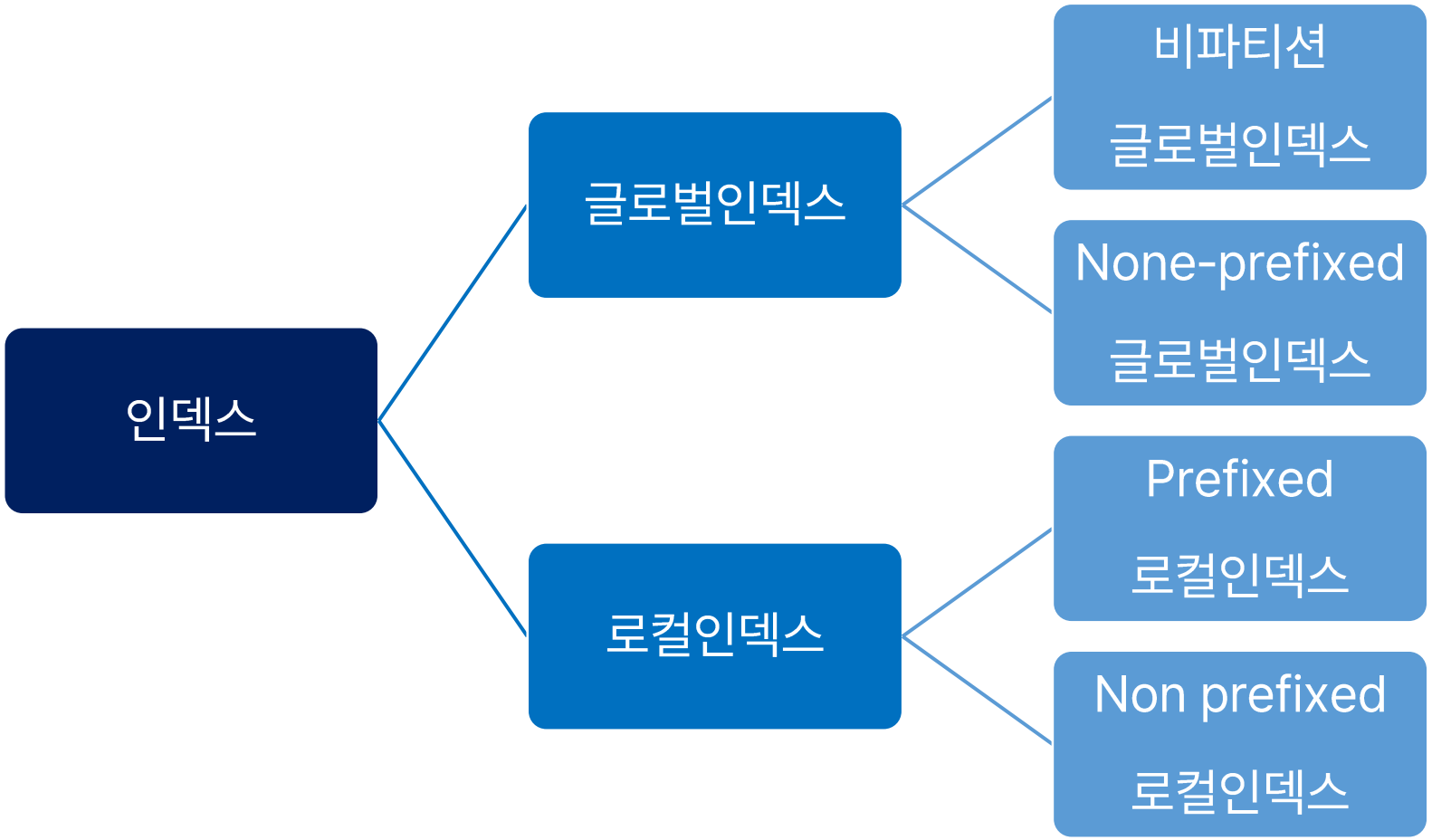
Main \ SUB	Range	List	Hash
Range	O	O	O
List	O	O	O
Hash	O	O	O

TMAXTibero

03 파티션 인덱스

1. 인덱스 종류
2. 글로벌 인덱스
3. 로컬 인덱스

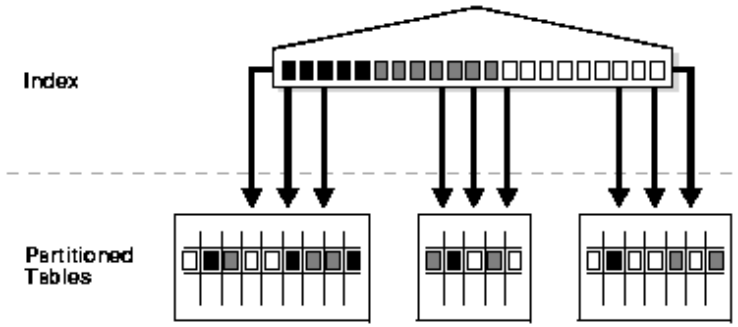
1. 인덱스 종류

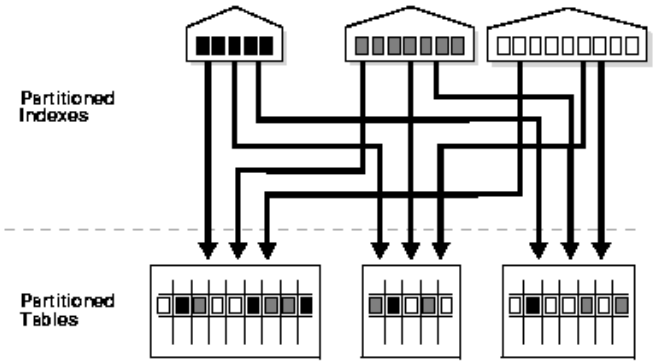

인덱스
글로벌인덱스
로컬인덱스
비파티션 글로벌인덱스
None-prefixed 글로벌인덱스
Prefixed 로컬인덱스
Non prefixed 로컬인덱스

2. 글로벌 인덱스

[정의]
- 테이블과 다르게 파티션 되는 인덱스

[특징]
- 비파티션 테이블과 파티션 테이블에 생성 가능
- 테이블 전체를 기준으로 정렬
- 파티션 기준 테이블의 파티션 구성에 변경(drop, exchange, split등)이 생길 때마다 인덱스가 unusable 상태로 바뀌고 그때마다 인덱스를 재 생성해야 한다.

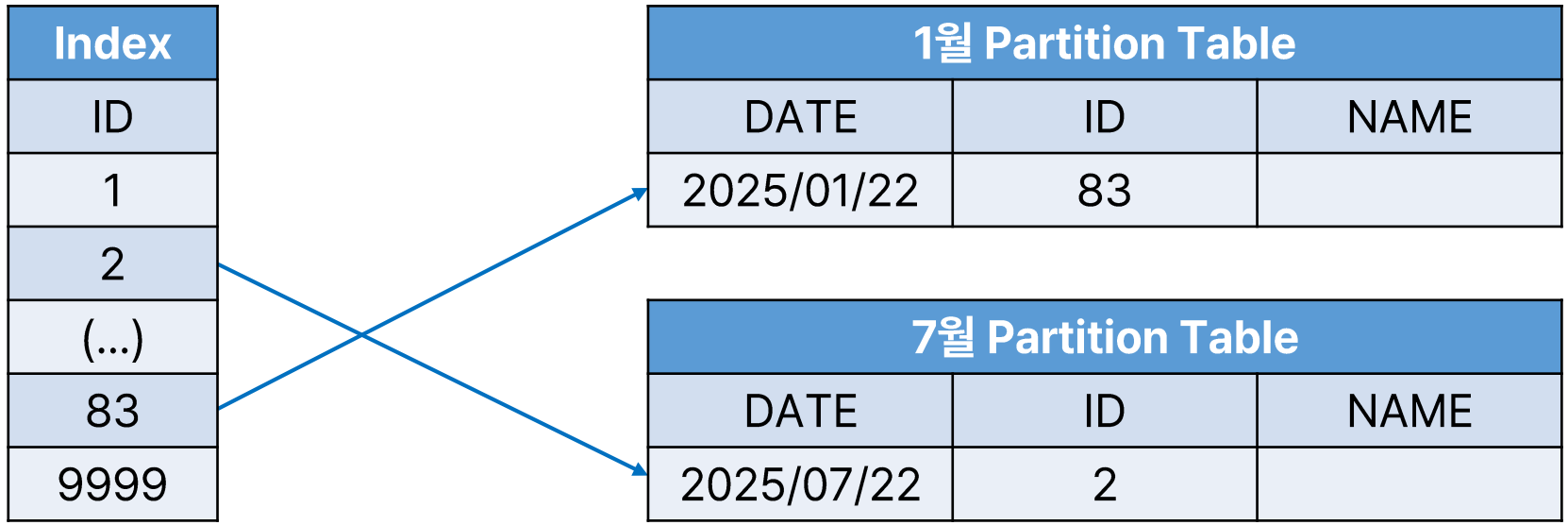




2. 글로벌 인덱스-글로벌 비파티션 인덱스(1/2)

[정의]
-파티셔닝하지 않은 인덱스

[특징] - Index : Table = 1:M 관계
하나의 인덱스 세그먼트가 여러 테이블 파티션 세그먼트와 관계



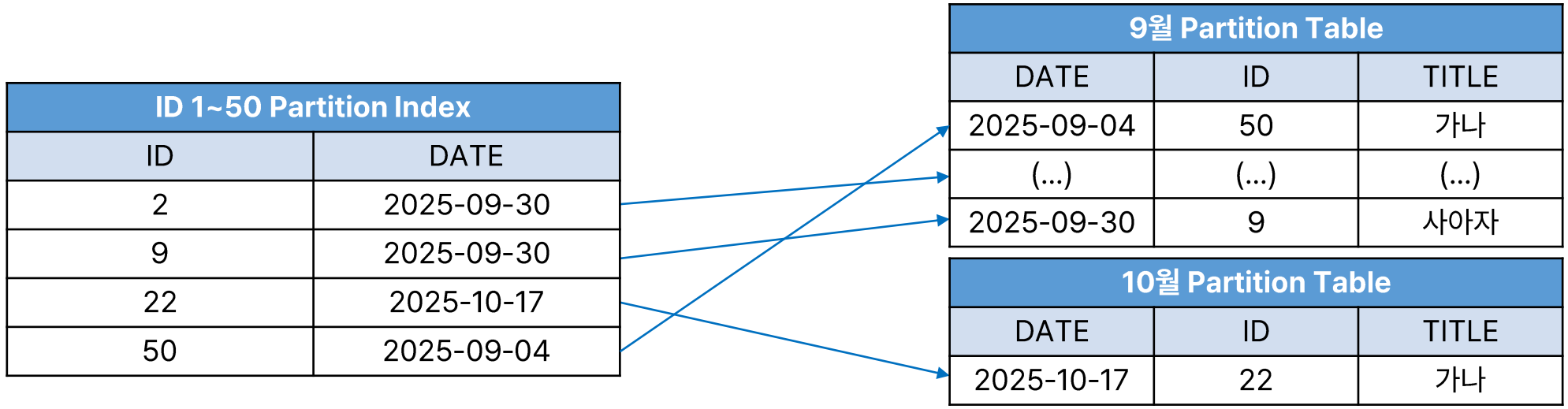
2. 글로벌 인덱스-글로벌 비파티션 인덱스(2/2)

```
CREATE TABLE range_table (
col1 NUMBER,
col2 NUMBER
)
PARTITION BY RANGE (col1) (
PARTITION p1 VALUES LESS THAN(100)
, PARTITION p2 VALUES LESS THAN(200)
, PARTITION p3 VALUES LESS THAN(300)
, PARTITION p4 VALUES LESS THAN(maxvalue)
)
create index t_idx3 on range_table (col2) ;
select index_name, partitioned, uniqueness, status
from dba_indexes
where index_name ='T_IDX3';

INDEX_NAME                     PARTITIONED  UNIQUENESS   STATUS
------------------------------ ------------ ------------ --------
T_IDX3                         NO           NONUNIQUE    VALID
```

2. 글로벌 인덱스-글로벌 prefixed 인덱스(1/3)

[특징]
- 테이블 파티션과 독립적인 구성을 갖도록 파티셔닝
- M:M 관계

[활용방법]
- OLTP 환경에서 여러 파티션에 걸쳐 엑세스가 빈번한 경우
- 파티션 키가 아닌 컬럼 들로 Unique Index가 필요한 경우
- 업무 특성이나 엑세스 형태 등을 고려하여 파티션 인덱스를 생성하고자 하나 Local 인덱스를 만들 수 없는 경우
- 관리 상 파티션 Drop을 자주 한다면 Global Index 적용 시 주의



ID 1~50 Partition Index	
ID	DATE
2	2025-09-30
9	2025-09-30
22	2025-10-17
50	2025-09-04

9월 Partition Table		
DATE	ID	TITLE
2025-09-04	50	가나
(...)	(...)	(...)
2025-09-30	9	사아자

10월 Partition Table		
DATE	ID	TITLE
2025-10-17	22	가나

2. 글로벌 인덱스-글로벌 prefixed 인덱스(2/3)

```
CREATE TABLE range_table (
ID NUMBER,
AGE NUMBER
)
PARTITION BY RANGE (ID) (
PARTITION p1 VALUES LESS THAN(100)
, PARTITION p2 VALUES LESS THAN(200)
, PARTITION p3 VALUES LESS THAN(300)
, PARTITION p4 VALUES LESS THAN(maxvalue)
)
create index t_idx5 on range_table(AGE,ID) GLOBAL
partition by range(AGE) (
PARTITION p1 VALUES LESS THAN(20)
, PARTITION p2 VALUES LESS THAN(30)
, PARTITION p3 VALUES LESS THAN(40)
, PARTITION p4 VALUES LESS THAN(50)
, PARTITION p5 VALUES LESS THAN(maxvalue)
) ;
```

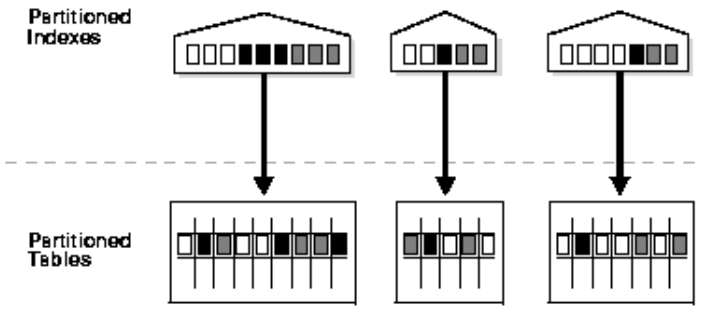
2. 글로벌 인덱스-글로벌 prefixed 인덱스(3/3)

```
create index t_idx5 on range_table(AGE,ID) GLOBAL
partition by range(AGE) (
PARTITION p1 VALUES LESS THAN(20) ,
PARTITION p2 VALUES LESS THAN(30) ,
PARTITION p3 VALUES LESS THAN(40) ,
PARTITION p4 VALUES LESS THAN(50) ,
PARTITION p5 VALUES LESS THAN(maxvalue) ) ;

SELECT index_name, partitioned, uniqueness, status
FROM dba_indexes
WHERE index_name ='t_idx5' ;
INDEX_NAME                       PARTITIONED  UNIQUENESS  STATUS
-------------------------------- ------------ ----------- --------
T_IDX5                           YES          NONUNIQUE   VALID

SELECT index_name,locality
FROM dba_part_indexes
WHERE index_name='T_IDX5';
INDEX_NAME                       LOCALITY
-------------------------------- -----------
T_IDX5                           GLOBAL
```

3. 로컬 인덱스

[정의] 테이블과 동일하게 파티션 되는 인덱스

[특징]
- 파티션 테이블에 생성 가능
- 파티션 단위로의 정렬을 보장
- 파티션 테이블의 로컬 인덱스의 경우 테이블에 구조 변경하여도 DBMS가 자동적으로 rebuild하여 usable상태로 변경



3. 로컬 인덱스-로컬 prefixed 인덱스 (1/3)

[특징]
-파티션 인덱스를 생성할 때, 파티션 키 컬럼을 인덱스 키 컬럼 왼쪽 선두에 두는 것

[활용방법]
-OLTP 환경에서 특정 파티션 Access가 많고 조인이 빈번한 경우
-파티션키 컬럼을 선두로 하는 인덱스가 필요한 경우
-성능,가용성,관리의 용이성 등에서 다른 타입의 인덱스보다 우수

9월 Partition Index	
DATE	TITLE
2025-09-01	가나
2025-09-02	가나나
2025-09-03	다라마

9월 Partition Table		
DATE	ID	TITLE
2025-09-01	55	가나
2025-09-02	399	가나나
2025-09-03	2	다라마

3. 로컬 인덱스-로컬 prefixed 인덱스 (2/3)

```
CREATE TABLE range_table (
col1 NUMBER,
col2 NUMBER
)
PARTITION BY RANGE (col1) (
PARTITION p1 VALUES LESS THAN(100) , PARTITION p2
VALUES LESS THAN(200) , PARTITION p3 VALUES LESS
THAN(300) , PARTITION p4 VALUES LESS
THAN(maxvalue)
)
```

```
CREATE INDEX T_IDX3 ON range_table (
col1 ASC,
col2 ASC
)
LOCAL
(
PARTITION P1
,PARTITION P2
,PARTITION P3
,PARTITION P4
);

CREATE INDEX t_idx3 ON range_table (
col1, col2
) LOCAL ;
```

3. 로컬 인덱스-로컬 prefixed 인덱스 (3/3)

```
SELECT index_name, partitioned, uniqueness, status
FROM dba_indexes
WHERE index_name ='T_IDX3'
```

INDEX_NAME	PARTITIONED	UNIQUENESS	STATUS
T_IDX3	**YES**	NONUNIQUE	VALID

```
SELECT index_name,locality,alignment
FROM dba_part_indexes
WHERE index_name='T_IDX3';
```

INDEX_NAME	LOCALITY	ALIGNMENT
T_IDX3	**LOCAL**	**PREFIXED**

3. 로컬 인덱스-로컬 non prefixed 인덱스 (1/3)

[특징]
-파티션 인덱스를 생성할 때, 파티션 키 컬럼을 인덱스 키 컬럼 왼쪽 선두에 주지 않는 것.
-파티션 키가 인덱스 컬럼에 아예 속하지 않을 때 (Unique Index 생성 불가)

[활용방법]
-DW와 같은 대용량 Ad Hoc Query 중심 환경에서 다른 테이블과의 조인이 적고 좁은 범위의 Partition-wide access가 많은 경우
-파티션 키 이외의 컬럼으로 조건 검색 시 파티션 별 처리가 가능하여 Historical Data 검색에 유리
-관리 상 파티션 Drop을 자주 하면서도 인덱스의 가용성과 관리의 용이성을 유지하고자 하는 경우

9월 Partition Index	
ID	DATE
2	2025-09-03
55	2025-09-04
399	2025-09-05

9월 Partition Table		
DATE	ID	TITLE
2025-09-01	55	가나
2025-09-02	399	가나나
2025-09-03	2	다라마

3. 로컬 인덱스-로컬 non prefixed 인덱스 (2/3)

```
CREATE TABLE range_table (
col1 NUMBER,
col2 NUMBER
)
PARTITION BY RANGE (col1) (
PARTITION p1 VALUES LESS THAN(100)
, PARTITION p2 VALUES LESS THAN(200)
, PARTITION p3 VALUES LESS THAN(300)
, PARTITION p4 VALUES LESS THAN(maxvalue)
)
```

```
CREATE INDEX t_idx3 ON range_table (
col2 ASC, col1 ASC
)
LOCAL
(
PARTITION P1,
PARTITION P2,
PARTITION P3,
PARTITION P4
);

CREATE INDEX t_idx3 ON range_table (col2,col1)
LOCAL ;
```

3. 로컬 인덱스-로컬 non prefixed 인덱스 (3/3)

```
SELECT index_name, partitioned, uniqueness, status
FROM dba_indexes
WHERE index_name ='T_IDX3'
```

INDEX_NAME	PARTITIONED	UNIQUENESS	STATUS
T_IDX3	**YES**	NONUNIQUE	VALID

```
SELECT INDEX_NAME,LOCALITY,ALIGNMENT
FROM dba_part_indexes
WHERE index_name='T_IDX3‘;
```

INDEX_NAME	LOCALITY	ALIGNMENT
T_IDX3	**LOCAL**	**NON_PREFIXED**

--파티션 키가 인덱스 컬럼에 아예 속하지 않을 때 (Unique Index 생성 불가)
SQL> CREATE UNIQUE INDEX t_idx3 ON range_table (**col2**) LOCAL;
TBR-7314: Partitioning columns must be contained in index key columns

TMAXTibero

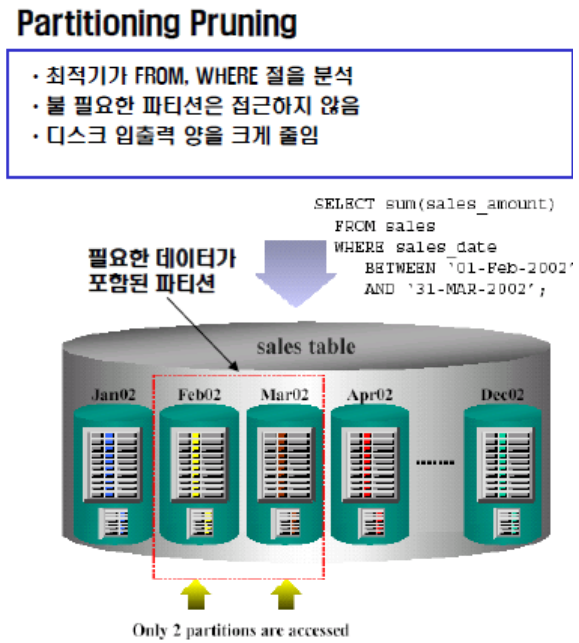
03 파티셔닝 장점

1. 가용성, 성능향상
2. 관리비용 감소

1. 가용성 (1/4)

- 옵티마이저가 참조하지 않아도 될 파티션을 처리 대상 제외

- 디스크 장애 시, 가용 디스크의 데이터를 사용하는 쿼리는 정상적으로 처리



1. 가용성 (2/4)

```
CREATE TABLE EMP(
NAME VARCHAR(12 BYTE),
JOIN_DATE DATE NOT NULL,
CONSTRAINT EMP_PK PRIMARY KEY(JOIN_DATE)
)
PCTFREE 10
INITRANS 2
LOGGING
NOPARALLEL
PARTITION BY RANGE("JOIN_DATE")

(
PARTITION "P_2024" VALUES LESS THAN ('20250101') TABLESPACE T1,
PARTITION "P_2025" VALUES LESS THAN ('20260101') TABLESPACE T2,
PARTITION "P_2026" VALUES LESS THAN ('20270101') TABLESPACE T2
);
```

1. 가용성 (3/4)

```
SQL> SELECT * FROM emp;

NAME        JOIN_DATE
----------- ------------------
LEE          2024/01/10
PARK         2025/03/17
BAEK         2026/04/23

3 rows selected.

SQL> ALTER TABLESPACE t2 OFFLINE;

SQL> SELECT * FROM emp WHERE JOIN_DATE <'2024/12/01';

NAME        JOIN_DATE
----------- ------------------------------
LEE          2024/01/10

SQL> SELECT * FROM emp WHERE JOIN_DATE <'2025/12/01';
TBR-24035: Unable to read file 8 at this time.
```

1. 가용성 (4/4)

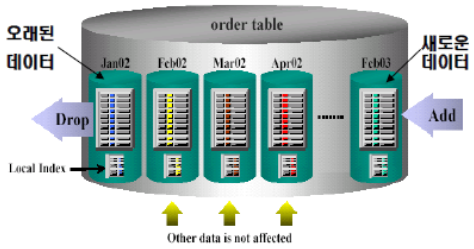
```
SQL> ALTER TABLESPACE t2 ONLINE;

SQL> CREATE INDEX TIBERO.EMP_IDX ON TIBERO.EMP (
NAME ASC,JOIN_DATE ASC
)
LOGGING
LOCAL
(
PARTITION P_2024 TABLESPACE T2,
PARTITION P_2025 TABLESPACE T1,
PARTITION P_2026 TABLESPACE T1
);

SQL> ALTER TABLESPACE t2 OFFLINE;

SQL> SELECT * FROM tibero.emp WHERE JOIN_DATE >= '2025/01/01';

NAME                      JOIN_DATE
------------------------- --------------------------------
PARK                      2025/03/17
BAEK                      2026/04/23
```

2. 관리비용감소 (1/2)

Partition은 독립적으로
ADD, DROP, TRUNCATE 가능

Rolling Window Operation

- 새로운 데이터를 로딩
- 오래된 데이터는 제거(purge)
- 데이터 웨어하우스에 주기적으로 최신의 정보를 유지



2. 관리비용감소 (2/2)

구분	일반 테이블	파티션 테이블
과거 데이터 삭제	대량 DELETE(HWM 유지)	DROP PARTITION
소요 시간	매우 김	매우 짧음
Redo/Undo	많음	거의 없음
락 영향	큼	최소
복구 범위	전체	파티션 단위
인덱스 관리	전체 단위	파티션 단위

Thank You

TMAXTibero