15장. 대용량 데이터 처리 및 관리

15.1 Table Partitioning

Partitioning 은 논리적으로 하나의 큰 테이블을 작은 물리적 조각으로 분할하는 것을 말합니다.
파티셔닝은 다음과 같은 여러 가지 이점을 제공할 수 있습니다.

- 적재된 데이터가 대용량 이더라도 데이터 접근 시 파티션 단위로 액세스 범위를 줄여 쿼리 성능이 향상됩니다.
- 파티션 별로 데이터가 분산 저장되므로 디스크 성능이 향상됩니다.
- 파티셔닝으로 인한 쿼리 수정이 없어 조회 테이블 관리를 위한 응용프로그램 개발이 필요 없습니다.
- 파티션 단위로 데이터를 옮기거나 인덱스를 재생성 할 수 있습니다.
- 대량 데이터 적재 및 삭제 작업시 개별 파티션 단위의 추가, 삭제, 분리 등의 DDL문을 통해 손쉽고 빠르게 수행할 수 있습니다.

15.1.1 지원하는 **Partitioning** 기법

- Range Partition

  테이블은 키 열 또는 열 집합에 의해 정의된 “RANGE(범위)”로 분할되며 데이터는 서로 다른 파티션에 할당된 값 범위 간에 겹치지 않습니다.
  예를 들어, date 값을 기준으로 매달 1일에서 말일까지 범위로 월별로 Partition 을 나누는 경우입니다.

- List Partition

  테이블은 각 분할 영역에 나타나는 키 값을 명시적으로 나열하여 분할됩니다.
  예를 들어, location 값을 기준으로 서울, 부산, 인천 등 Data 값들의 List 를 기준으로 Partition 을 나누는 경우입니다.

- Hash Partition

  테이블은 각 파티션에 대한 Hash 계수와 나머지를 지정하여 분할됩니다. 각 파티션에는 Hash 함수를 적용한 나머지 값을 활용하여 분할 저장 됩니다.
  Partition 을 나눌 명확한 방법이 없을 경우 유용합니다. 특히 향후 거대해질 것으로 예상되는데 초기단계에서 Partition 을 설정하기 어려울 때 더욱 유용합니다.
  예를 들어, 자동으로 증가하는 id 값을 기준으로 Partition 을 정의할 수 있습니다.

15.1.2 Syntax

- PARENT PARTITION TABLE 생성

```
CREATE TABLE table_name (
   column_name data_type
   [, ... ]
) PARTITION BY { RANGE | LIST | HASH } ( { column_name | ( expression ) } [, ... ] )
```

- CHILD PARTITION TABLE 생성

```
CREATE TABLE table_name PARTITION OF parent_table
{ FOR VALUES partition_bound_spec | DEFAULT }

partition_bound_spec is:

# RANGE PARTITION
FROM ( { partition_bound_expr | MINVALUE | MAXVALUE } [, ...] )
  TO ( { partition_bound_expr | MINVALUE | MAXVALUE } [, ...] )

# LIST PARTITION
IN ( partition_bound_expr [, ...] )

# HASH PARTITION
WITH ( MODULUS numeric_literal, REMAINDER numeric_literal )
```

- PARENT/CHILD PARTITION TABLE 삭제

```
DROP TABLE table_name
```

- CHILD PARTITION TABLE 분리

```
ALTER TABLE name
   DETACH PARTITION partition_name
```

- CHILD PARTITION TABLE 부착

```
ALTER TABLE name
   ATTACH PARTITION partition_name
{ FOR VALUES partition_bound_spec | DEFAULT }
```

15.1.3 Example

1) RANGE PARTITION

```
# RANGE PARTITION TABLE 생성(PARENT PARTITION)
CREATE TABLE measurement (
   id          serial,
   city_id      int not null,
   logdate      date not null,
   peaktemp     int,
   unitsales    int
) PARTITION BY RANGE (logdate); -- 파티션 키를 logdate 로 지정

# SUB PARTITION 생성
CREATE TABLE measurement_y2020 PARTITION OF measurement FOR VALUES FROM
('2020-01-01') TO ('2021-01-01');

CREATE TABLE measurement_y2021 PARTITION OF measurement FOR VALUES FROM
('2021-01-01') TO ('2022-01-01');

CREATE TABLE measurement_y2022 PARTITION OF measurement FOR VALUES FROM
('2022-01-01') TO ('2023-01-01');

# PARENT PARTITION에 DATA INSERT
- 부모 파티션에 데이터를 insert하면 SUB파티션 생성시 정의된 범위에 따라 실제로
  SUB파티션에 데이터가 저장됩니다.
INSERT INTO measurement (id, city_id, logdate, peaktemp, unitsales)  values
(1111, 1,'2020-01-05', 39, 50),
(1112, 2, '2021-01-05', 45, 50),
(1113, 3, '2022-01-05', 41, 50);

# PARTITION TABLE별 데이터 확인
```

```
SELECT * FROM measurement;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
 1112 |       2 | 2021-01-05 |       45 |        50
 1113 |       3 | 2022-01-05 |       41 |        50
(3 rows)

SELECT * FROM measurement_y2020;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
(1 row)

SELECT * FROM measurement_y2021;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1112 |       2 | 2021-01-05 |       45 |        50
(1 row)

SELECT * FROM measurement_y2022;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1113 |       3 | 2022-01-05 |       41 |        50
(1 row)
```

2) LIST PARTITION

```
# 기존 존재하는 TABLE 제거
DROP TABLE measurement;

# LIST PARTITION TABLE 생성
CREATE TABLE measurement (
   id           serial,
   city_id      int not null,
   logdate      date not null,
   peaktemp     int,
   unitsales    int
) PARTITION BY LIST (city_id); -- 파티션 키를 city_id 로 지정

# SUB PARTITION 생성
CREATE TABLE measurement_c1 PARTITION OF measurement FOR VALUES in(1);

CREATE TABLE measurement_c2 PARTITION OF measurement FOR VALUES in(2);

CREATE TABLE measurement_c3 PARTITION OF measurement FOR VALUES in(3);

# PARENT PARTITION에 DATA INSERT
INSERT INTO measurement (id, city_id, logdate, peaktemp, unitsales)  values
(1111, 1,'2020-01-05', 39, 50),
```

```
(1112, 2, '2021-01-05', 45, 50),
(1113, 3, '2022-01-05', 41, 50);

# PARTITION TABLE별 데이터 확인
SELECT * FROM measurement;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
 1112 |       2 | 2021-01-05 |       45 |        50
 1113 |       3 | 2022-01-05 |       41 |        50
(3 rows)

SELECT * FROM measurement_c1;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
(1 row)

SELECT * FROM measurement_c2;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1112 |       2 | 2021-01-05 |       45 |        50
(1 row)

SELECT * FROM measurement_c3;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1113 |       3 | 2022-01-05 |       41 |        50
(1 row)
```

3) HASH PARTITION

```
# 기존 존재하는 TABLE 제거
DROP TABLE measurement;

# HASH PARTITION TABLE 생성
CREATE TABLE measurement (
   id          serial,
   city_id       int not null,
   logdate       date not null,
   peaktemp      int,
   unitsales     int
) PARTITION BY HASH (id); -- 파티션 키를 id 로 지정

# SUB PARTITION 생성
CREATE TABLE measurement_id1 PARTITION OF measurement FOR VALUES WITH
(modulus 3, remainder 0);

CREATE TABLE measurement_id2 PARTITION OF measurement FOR VALUES WITH
(modulus 3, remainder 1);
```

```
CREATE TABLE measurement_id3 PARTITION OF measurement FOR VALUES WITH
(modulus 3, remainder 2);

# PARENT PARTITION에 DATA INSERT
INSERT INTO measurement (id, city_id, logdate, peaktemp, unitsales)  values
(1111, 1,'2020-01-05', 39, 50),
(1112, 2, '2021-01-05', 45, 50),
(1113, 3, '2022-01-05', 41, 50);

# PARTITION TABLE별 데이터 확인
SELECT * FROM measurement;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
 1112 |       2 | 2021-01-05 |       45 |        50
 1113 |       3 | 2022-01-05 |       41 |        50
(3 rows)

SELECT * FROM measurement_id1;
 id | city_id | logdate | peaktemp | unitsales
----+---------+---------+----------+-----------
(0 rows)

SELECT * FROM measurement_id2;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1111 |       1 | 2020-01-05 |       39 |        50
(1 row)

SELECT * FROM measurement_id3;
 id  | city_id |  logdate   | peaktemp | unitsales
------+---------+------------+----------+-----------
 1112 |       2 | 2021-01-05 |       45 |        50
 1113 |       3 | 2022-01-05 |       41 |        50
(2 rows)
```

4) Partition 제거**,** 분리**,** 부착

```
# PARTITION TABLE 제거
DROP TABLE measurement_id3;

# PARTITION TABLE 제거 확인
\d+ measurement;
postgres=# \d+ measurement;
                                Partitioned table "public.measurement"
 Column   | Type   | Collation | Nullable |              Default              | Storage | Stats
target | Description
----------+--------+-----------+----------+-----------------------------------+---------+--------------+---
----------
```

```
 id       | integer |          | not null | nextval('measurement_id_seq'::regclass) | plain   |
|
 city_id   | integer |          | not null |                                  | plain   |           |
 logdate   | date    |          | not null |                                  | plain   |           |
 peaktemp  | integer |          |          |                                  | plain   |           |
 unitsales | integer |          |          |                                | plain   |           |
Partition key: HASH (id)
Partitions: measurement_id1 FOR VALUES WITH (modulus 3, remainder 0),
            measurement_id2 FOR VALUES WITH (modulus 3, remainder 1)
```

 # PARTITION TABLE 분리

```
ALTER TABLE measurement DETACH PARTITION measurement_id1;
```

 # PARTITION TABLE 분리 확인

```
\d+ measurement;
                                Partitioned table "public.measurement"

 Column   |  Type   | Collation | Nullable |              Default              | Storage | Stats
target | Description
-----------+---------+-----------+----------+-----------------------------------+---------+--------------+---
----------
 id       | integer |          | not null | nextval('measurement_id_seq'::regclass) | plain   |
|
 city_id   | integer |          | not null |                                  | plain   |           |
 logdate   | date    |          | not null |                                  | plain   |           |
 peaktemp  | integer |          |          |                                  | plain   |           |
 unitsales | integer |          |          |                                | plain   |           |
Partition key: HASH (id)
Partitions: measurement_id2 FOR VALUES WITH (modulus 3, remainder 1)
```

 # PARTITION TABLE 부착

```
ALTER TABLE measurement ATTACH PARTITION measurement_id1 FOR VALUES WITH
(modulus 3, remainder 0);
```

 # PARTITION TABLE 부착 확인

```
\d+ measurement;
                                Partitioned table "public.measurement"
 Column   |  Type   | Collation | Nullable |              Default              | Storage | Stats
target | Description
-----------+---------+-----------+----------+-----------------------------------+---------+--------------+---
----------
 id       | integer |          | not null | nextval('measurement_id_seq'::regclass) | plain   |
|
 city_id   | integer |          | not null |                                  | plain   |           |
 logdate   | date    |          | not null |                                  | plain   |           |
 peaktemp  | integer |          |          |                                  | plain   |           |
 unitsales | integer |          |          |                                | plain   |           |
Partition key: HASH (id)
Partitions: measurement_id1 FOR VALUES WITH (modulus 3, remainder 0),
            measurement_id2 FOR VALUES WITH (modulus 3, remainder 1)
```

15.2 병렬 처리

AgensSQL 은 Query 응답속도를 더 빠르게 하기 위해 옵티마이저에서 병렬쿼리가 특정 쿼리에 대한 가장 빠른 실행계획이라 판단되면 여러개의 CPU 를 활용할 수 있는 Parallel Query 기능이 있습니다. 많은 양의 데이터에서 사용자에게 몇 개의 행만 반환하는 쿼리에서 일반적으로 가장 유용하며 빠르게 실행될 수 있습니다.

15.2.1 병렬 처리 조건

1) max_parallel_workers_per_gather 의 값을 0보다 큰 값으로 설정해야 합니다. 해당 파라미터의 기본 값이 2이므로 기본 설정인 경우 문제는 없습니다.
2) 시스템이 single user mode 에서 실행되지 않아야 합니다. 해당 모드에서는 전체 데이터베이스 시스템이 단일 프로세스에서 실행되므로 Background worker 를 사용할 수 없어서 parallel 동작을 할 수 없습니다.

15.2.2 examples

```
# Test Table 생성
create table people(
   id         integer not null
 , born       date not null
 , phone      text null
 , dept       text null
 , create_dt  date not null
);

# Test Data Insert
insert into people
select series as id,
(timestamp '1970-01-01' + random() * (timestamp '2002-12-31' - timestamp '1970-01-01'))::date
as born,
'010-'||LPAD(trunc(random()*9999)::text,4,'0')||'-'||LPAD(trunc(random()*9999)::text,4, '0') as
phone,
substr('서울대전대구부산', trunc(random() * 4)::integer*2 + 1, 2) as dept,
now() + series * INTERVAL '1 day' as create_dt
from generate_series(1, 500000) series;

# Parallel Query
explain select * from people where id=12345;

                          QUERY PLAN
------------------------------------------------------------------------------
 Gather  (cost=1000.00..7022.34 rows=1625 width=76)
   Workers Planned: 2
```

```
  ->  Parallel Seq Scan on people  (cost=0.00..5859.84 rows=677 width=76)
        Filter: (id = 12345)
(4 rows)
```

15.3 SQL Hint 기능 (pg_hint_plan)

AgensSQL의 옵티마이저는 각 쿼리에 대하여 실행 비용을 산정하여 최적의 실행계획을 선택하여 쿼리를 실행합니다. 그러나 옵티마이저의 실행계획의 경우에 따라서는 완벽하지 않습니다. 따라서 pg_hint_plan을 이용하여 특정 쿼리의 성능 향상을 위하여 인덱스를 지정하거나, 조인 순서 및 방법을 선택하여 지정할 수 있습니다. 그러나 잘못된 Hint는 비효율이 발생할 수 있습니다.

15.3.1 Hint Plan 유형

1) SCAN

Hint	설명
SeqScan	Table Full Scan 방식을 유도
IndexScan	Index Scan 방식을 유도
IndexOnlyScan	Index Only Scan 방식을 유도한다 Index Only Scan 불가시 Index Scan으로 동작
BitmapScan	Bitmap Scan 방식을 유도

2) JOIN

Hint	설명
NestLoop	NL Join으로 유도
HashJoin	Hash Join으로 유도
MergeJoin	Merge Join으로 유도

15.3.2 Syntax

pg_hint_plan의 경우 쿼리의 앞에 /*+ HINT */ 의 형식으로 사용합니다.

```
# SCAN
/*+
 { SeqScan | IndexScan | IndexOnlyScan | BitmapScan }( table_name [ Index_name ] )
*/

# JOIN
/*+
{ NestLoop | HashJoin | MergeJoin } ( table_name | ( expression ) )
*/
```

15.3.3 Example

```
# Database 접속
$ asql -U agens -d postgres

# pg_hint_plan 생성
create extension pg_hint_plan;

# Table 생성
create table tab1(
   id         integer not null
 , born     date not null
 , phone     text null
 , dept      text null
 , create_dt  date not null
);

create table tab2(
   id         integer not null
 , born     date not null
 , phone     text null
 , dept      text null
 , create_dt  date not null
);

# Test Data Insert
insert into tab1
select series as id,
(timestamp '1970-01-01' + random() * (timestamp '2002-12-31' - timestamp
'1970-01-01'))::date as born,
'010-'||LPAD(trunc(random()*9999)::text,4,'0')||'-'||LPAD(trunc(random()*9999)::text,4, '0')
as phone,
substr('서울대전대구부산', trunc(random() * 4)::integer*2 + 1, 2) as dept,
now() + series * INTERVAL '1 day' as create_dt
from generate_series(1, 500000) series;
```

```
insert into tab2
select series as id,
(timestamp '1990-01-01' + random() * (timestamp '2002-12-31' - timestamp
'1970-01-01'))::date as born,
'010-'||LPAD(trunc(random()*9999)::text,4,'0')||'-'||LPAD(trunc(random()*9999)::text,4, '0')
as phone,
substr('서울대전대구부산', trunc(random() * 4)::integer*2 + 1, 2) as dept,
now() + series * INTERVAL '1 day' as create_dt
from generate_series(1, 500000) series;

# Index 생성
create index idx_tab1 on tab1(phone);

# SCAN 활용
/*+
SeqScan (tab1)
*/
explain select * from tab1 where phone='010-7872-1770';

                          QUERY PLAN
------------------------------------------------------------------------------
 Gather  (cost=1000.00..8021.17 rows=2500 width=76)
   Workers Planned: 2
   ->  Parallel Seq Scan on tab1  (cost=0.00..6771.17 rows=1042 width=76)
         Filter: (phone = '010-7872-1770'::text)



/*+
IndexScan (tab1 idx_tab1)
*/
explain select * from tab1 where phone='010-7872-1770';

                          QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using idx_tab1 on tab1  (cost=0.42..7780.17 rows=2500 width=76)
   Index Cond: (phone = '010-7872-1770'::text)


/*+
IndexOnlyScan (tab1 idx_tab1)
*/
explain select phone from tab1 where phone='010-7872-1770';

                          QUERY PLAN
------------------------------------------------------------------------------
 Index Only Scan using idx_tab1 on tab1  (cost=0.42..7780.17 rows=2500 width=32)
   Index Cond: (phone = '010-7872-1770'::text)

# JOIN 활용
/*+
```

```
NestLoop(a b)
*/
explain select * from tab1 a, tab2 b where a.id=b.id;

                           QUERY PLAN
-------------------------------------------------------------------------------
 Gather  (cost=1000.00..2169156497.28 rows=812565000 width=152)
   Workers Planned: 2
   ->  Nested Loop  (cost=0.00..2087898997.28 rows=338568750 width=152)
         Join Filter: (a.id = b.id)
         ->  Parallel Seq Scan on tab2 b  (cost=0.00..5521.27 rows=135428 width=76)
         ->  Seq Scan on tab1 a  (cost=0.00..9167.00 rows=500000 width=76)


/*+
HashJoin(a b)
*/
explain select * from tab1 a, tab2 b where a.id=b.id;

                        QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=11480.09..28461922.09 rows=812565000 width=152)
   Hash Cond: (a.id = b.id)
   ->  Seq Scan on tab1 a  (cost=0.00..9167.00 rows=500000 width=76)
   ->  Hash  (cost=7417.26..7417.26 rows=325026 width=76)
         ->  Seq Scan on tab2 b  (cost=0.00..7417.26 rows=325026 width=76)

/*+
MergeJoin(a b)
*/
explain select * from tab1 a, tab2 b where a.id=b.id;

                           QUERY PLAN
-------------------------------------------------------------------------------
 Merge Join  (cost=130332.13..12321682.26 rows=812565000 width=152)
   Merge Cond: (b.id = a.id)
   ->  Sort  (cost=51618.21..52430.77 rows=325026 width=76)
         Sort Key: b.id
         ->  Seq Scan on tab2 b  (cost=0.00..7417.26 rows=325026 width=76)
   ->  Materialize  (cost=78713.92..81213.92 rows=500000 width=76)
         ->  Sort  (cost=78713.92..79963.92 rows=500000 width=76)
               Sort Key: a.id
               ->  Seq Scan on tab1 a  (cost=0.00..9167.00 rows=500000 width=76)
```

15.4 Partition Pruning 기능

Partition Pruning은 Partitioning Table에서 쿼리의 성능을 향상 시키기 위한 기능 입니다. Partitioning Table을 조회하는 쿼리의 경우 파티션 조건에 충족하는 행이 있는 Partition Table만을 대상으로 쿼리 하도록 합니다.

15.4.1 기능 활성화 조건

```
postgresql.conf 설정

enable_partition_pruning = on

또는 ALTER SYSTEM 명령을 통한 동적 반영

alter system set enable_partition_pruning = on ;

select pg_reload_conf() ;
```

15.4.2 Example

```
# Range Partition Table

explain select * from measurement where logdate = '2022-06-01';
                            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on measurement_y2022 measurement  (cost=0.00..31.25 rows=8 width=20)
   Filter: (logdate = '2022-06-01'::date)

# List Partition Table
 explain select * from measurement where city_id=3;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on measurement_c3 measurement  (cost=0.00..31.25 rows=8 width=20)
   Filter: (city_id = 3)

# Hash Partition Table
explain select * from measurement where id=3;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on measurement_id2 measurement  (cost=0.00..31.25 rows=8 width=20)
   Filter: (id = 3)
```

15.5 다양한 Index 활용

데이터베이스는 조건에 맞는 데이터를 검색하기 위하여 테이블의 모든 데이터를 검색 해야 합니다. 이를 효율적으로 검색하는 방법은 조건에 필요한 데이터 인덱스를 생성하면 조건에 맞추어 필요한 데이터를 더 효율적으로 검색할 수 있습니다.

15.5.1 Index 유형

AgensSQL은 B-tree, Hash, GIN, BRIN 등 여러 인덱스 유형을 제공합니다. 기본적으로 CREATE INDEX 명령으로 생성하는 경우 B-tree 인덱스로 생성 됩니다.

B-tree	순서로 데이터를 정렬하여 범위 조건 쿼리를 사용하는 순차적인 일반 데이터에 사용을 고려합니다.
Hash	Hash 값을 이용하여 정렬하여 불특정 단순 동등성 조건 쿼리를 사용하는 데이터에 사용을 고려 합니다.
GIN	GIN(Generalized Inverted Index)연산자를 사용하여 인덱스를 생성합니다. 특정 조건(ex. LIKE) 쿼리를 이용 데이터에 사용을 고려 합니다.
BRIN	데이터 블록 범위 인덱스로 데이터를 페이지 단위로 인덱스 하는 방법입니다. 대량의 데이터에서 특정 컬럼값이 저장 순서에 따라 범위 정렬되어 있는 경우 효율적 입니다.

15.5.2 Examples

```
# Test Table 생성
create table people(
   id         integer not null
  , born      date not null
  , phone     text null
  , dept      text null
  , create_dt  date not null
);

# Test Data Insert
insert into people
select series as id,
(timestamp '1970-01-01' + random() * (timestamp '2002-12-31' - timestamp '1970-01-01'))::date as born,
'010-'||LPAD(trunc(random()*9999)::text,4,'0')||'-'||LPAD(trunc(random()*9999)::text,4, '0') as phone,
substr('서울대전대구부산', trunc(random() * 4)::integer*2 + 1, 2) as dept,
now() + series * INTERVAL '1 day' as create_dt
```

```
from generate_series(1, 500000) series;

# B-tree 인덱스 생성
CREATE INDEX people_id_btree ON people (id);

# B-tree 인덱스 활용
EXPLAIN SELECT * FROM people where id='12345';
                QUERY PLAN
-----------------------------------------------------
 Index Scan using people_id_btree on people  (cost=0.42..8.44 rows=1 width=33)
   Index Cond: (id = 12345)

# Hash 인덱스 생성
CREATE INDEX people_born_hash ON people USING HASH (born);

# Hash 인덱스 활용
EXPLAIN SELECT * FROM people where born='1987-01-01';
               QUERY PLAN
-----------------------------------------------------
 Bitmap Heap Scan on people  (cost=4.33..160.20 rows=42 width=33)
   Recheck Cond: (born = '1987-01-01'::date)
   ->  Bitmap Index Scan on people_born_hash  (cost=0.00..4.32 rows=42 width=0)
         Index Cond: (born = '1987-01-01'::date)

# GIN 인덱스 생성 (pg_trgm 설치필요)
CREATE EXTENSION pg_trgm;
CREATE INDEX people_phone_gin ON people USING GIN(phone gin_trgm_ops);

# GIN 인덱스 활용
EXPLAIN SELECT * FROM people where phone LIKE '%1987%';

                            QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on people  (cost=55.00..4386.02 rows=4000 width=76)
   Recheck Cond: (phone ~~ '%1987%'::text)
   ->  Bitmap Index Scan on people_phone_gin  (cost=0.00..54.00 rows=4000 width=0)
         Index Cond: (phone ~~ '%1987%'::text)
(4 rows)

# BRIN 인덱스 생성
CREATE INDEX people_create_brin ON people USING BRIN(create_dt);

# BRIN 인덱스 활용
EXPLAIN SELECT * FROM people where create_dt < '2023-12-07';

                QUERY PLAN
-----------------------------------------------------
 Bitmap Heap Scan on people  (cost=13.27..4369.67 rows=4950 width=33)
   Recheck Cond: (create_dt < '2023-12-07'::date)
   ->  Bitmap Index Scan on people_create_brin  (cost=0.00..12.03 rows=15152 width=0)
         Index Cond: (create_dt < '2023-12-07'::date)
```