

### Задание

Из задания к курсовому проекту возьмем описание одной из таблиц – «... экологических катастрофах (предприятие, тип катастрофы, дата, количество пострадавших людей, последствия, сведения об участии ЭС в устранении ЭК ...». На основе этого описание создадим таблицу `eco_catastrophy` как секционированную таблицу с предложением *PARTITION BY*, указав метод разбиения (в нашем случае *RANGE*) и список столбцов, которые будут образовывать ключ разбиения (в нашем случае это один столбец – `data_catastrophe`).

### Создание партии по диапазону:

```
CREATE TABLE eco_catastrophy (  
  "id" SERIAL NOT NULL,  
  id_company int4 NOT NULL,  
  id_type_catastrophe int4 NOT NULL,  
  data_catastrophe TIMESTAMP NOT NULL,  
  number_victims int4 NOT NULL,  
  effects TEXT,  
  detriment DECIMAL,  
  id_department int4[]  
) PARTITION BY RANGE (data_catastrophe);
```

### Создание секций:

В определении каждой секции должны задаваться границы, соответствующие методу и ключу разбиения родительской таблицы. Секции, создаваемые таким образом, во всех отношениях являются обычными таблицами PostgreSQL. Указание границ, при котором множество значений новой секции пересекается со множеством значений в одной или нескольких существующих секциях, будет ошибочным. В нашем примере каждая секция должна содержать данные за один квартал.

```
CREATE TABLE eco_catastrophy_2020_3kv PARTITION OF eco_catastrophy  
  FOR VALUES FROM ('2020-07-01') TO ('2020-10-01');  
CREATE TABLE eco_catastrophy_2020_4kv PARTITION OF eco_catastrophy  
  FOR VALUES FROM ('2020-10-01') TO ('2021-01-01');  
CREATE TABLE eco_catastrophy_2021_1kv PARTITION OF eco_catastrophy  
  FOR VALUES FROM ('2021-01-01') TO ('2021-04-01');  
CREATE TABLE eco_catastrophy_default PARTITION OF eco_catastrophy DEFAULT;
```

### Создание индекса:

Создадим в секционируемой таблице индекс по ключевому столбцу – `data_catastrophe`. При этом автоматически будет создан соответствующий индекс в каждой секции и все секции, которые вы будете создавать или присоединять позднее, тоже будут содержать такой индекс. Индекс по ключу, строго говоря, создавать не обязательно, но в большинстве случаев он будет полезен

```
CREATE INDEX ON eco_catastrophy (data_catastrophe);
```

```
A-Z eco_catastrophy_2020_3kv_data_catastrophe_idx  
A-Z eco_catastrophy_2020_4kv_data_catastrophe_idx  
A-Z eco_catastrophy_2021_1kv_data_catastrophe_idx  
A-Z eco_catastrophy_data_catastrophe_idx  
A-Z eco_catastrophy_default_data_catastrophe_idx
```

### Добавление данных:

```
INSERT INTO eco_catastrophy (id_company, id_type_catastrophe, data_catastrophe,
number_victims, effects, detriment, id_department)
SELECT
    trunc(random() * 10),
    trunc(random() * 1000),
    now() - '3 month'::interval * random(),
    trunc(random() * 10000),
    md5(random()::text),
    (random() * 100000)::decimal,
    mas_if((trunc(random() * 10))::int4)
FROM
    generate_series(1, 100000);
```

Запускаем данную команду несколько раз, с разными параметрами интервала дат (*now()* - '8 month'::interval \* random(), *now()* - '6 month'::interval \* random(), *now()* - '4 month'::interval \* random(), *now()* - '2 month'::interval \* random(), *now()* + '2 month'::interval \* random()) и получаем 500 000 случайных записей. Функция *mas\_if(in\_cnt)* – это пользовательская функция, которая на вход принимает размер массива (*in\_cnt*), целое число, а на выходе возвращает массив случайных целых чисел.

### Результат добавления данных:

```
SELECT
    (SELECT count(*) FROM eco_catastrophy_2020_3kv) as eco_catastrophy_2020_3kv,
    (SELECT count(*) FROM eco_catastrophy_2020_4kv) as eco_catastrophy_2020_4kv,
    (SELECT count(*) FROM eco_catastrophy_2021_1kv) as eco_catastrophy_2021_1kv,
    (SELECT count(*) FROM eco_catastrophy_default) as eco_catastrophy_default,
    (SELECT count(*) FROM ONLY eco_catastrophy) as eco_catastrophy;
```

eco_catastrophy_2020_3kv	eco_catastrophy_2020_4kv	eco_catastrophy_2021_1kv	eco_catastrophy_default	eco_catastrophy
50326	134046	260650	54978	0

Создаем новую таблицу вне структуры секций и загружаем в нее данные из секции *eco\_catastrophy\_default*:

```
CREATE TABLE eco_catastrophy_2021_2kv
    (LIKE eco_catastrophy INCLUDING DEFAULTS INCLUDING CONSTRAINTS);
ALTER TABLE eco_catastrophy_2021_2kv ADD CONSTRAINT catastrophe_2021_2kv
    CHECK (data_catastrophe >= DATE '2021-04-01' AND
           data_catastrophe < DATE '2021-07-01');
INSERT INTO eco_catastrophy_2021_2kv SELECT * FROM eco_catastrophy_default;
```

С помощью предложения LIKE определяем таблицу, из которой в новую таблицу будут автоматически скопированы все имена столбцов, их типы данных и их ограничения на NULL. Дополняем командами что и как будем копировать: INCLUDING DEFAULTS – скопировать выражения значений по умолчанию в определениях копируемых столбцов, без этого указания выражения по умолчанию не копируются, и INCLUDING CONSTRAINTS – скопировать ограничения-проверки.

Создаем ограничение CHECK в присоединяемой таблице, соответствующее ожидаемому ограничению секции. Благодаря этому система сможет обойтись без сканирования, необходимого для проверки неявного ограничения секции.

Копируем все данные, которые хранятся в дефолтной секции в новую таблицу, которая пока еще не является секцией нашей партии – *eco\_catastrophy*.

```
INSERT INTO eco_catastrophy_2021_2kv SELECT * FROM eco_catastrophy_default
> Affected rows: 54978
> Time: 0,491s
```

Удаляем все данные из дефолтной секции:

```
DELETE FROM eco_catastrophy_default;
```

```
DELETE FROM eco_catastrophy_default
> Affected rows: 54978
> Time: 0,041s
```

Делаем таблицу eco\_catastrophy\_2021\_2kv секцией:

```
ALTER TABLE eco_catastrophy ATTACH PARTITION eco_catastrophy_2021_2kv
FOR VALUES FROM ('2021-04-01') TO ('2021-07-01');
```

```
ALTER TABLE eco_catastrophy ATTACH PARTITION eco_catastrophy_2021_2kv
FOR VALUES FROM ('2021-04-01') TO ('2021-07-01')
> OK
> Time: 0,083s
```

Результат добавление новой секции eco\_catastrophy\_2021\_2kv:

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy
WHERE data_catastrophe >= '2021-04-01' ORDER BY data_catastrophe
```

```
QUERY PLAN
Merge Append (cost=0.45..7756.52 rows=55568 width=110) (actual time=0.012..318.898 rows=54978 loops=1)
Sort Key: eco_catastrophy_2021_2kv.data_catastrophe
-> Index Scan using eco_catastrophy_2021_2kv_data_catastrophe_idx on eco_catastrophy_2021_2kv (cost=0.29..6270.35 rows=54978 width=110) (actual time=0.012..318.898 rows=54978 loops=1)
Index Cond: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
-> Index Scan using eco_catastrophy_default_data_catastrophe_idx on eco_catastrophy_default (cost=0.15..930.47 rows=590 width=109) (actual time=0.000..0.000 rows=0 loops=0)
Index Cond: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Planning Time: 10.213 ms
Execution Time: 320.335 ms
```

Исходя из построенного плана видим, что поиск идет по новой добавленной секции и когда мы делали операцию добавления новой секции, создался новый индекс (eco\_catastrophy\_2021\_2kv\_data\_catastrophe\_idx) по полю data\_catastrophe

Создание партиции по списку:

```
CREATE TABLE eco_catastrophy_list (
  "id" SERIAL NOT NULL,
  id_company int4 NOT NULL,
  id_type_catastrophe int4 NOT NULL,
  data_catastrophe TIMESTAMP NOT NULL,
  number_victims int4 NOT NULL,
  effects TEXT,
  detriment DECIMAL,
  id_department int4[]
) PARTITION BY LIST ("id_company");
```

### Создание секций:

```
CREATE TABLE eco_catastrophy_list_0 PARTITION OF eco_catastrophy_list
FOR VALUES IN (0);
CREATE TABLE eco_catastrophy_list_1_3 PARTITION OF eco_catastrophy_list
FOR VALUES IN (1,3,5);
CREATE TABLE eco_catastrophy_list_7_9 PARTITION OF eco_catastrophy_list
FOR VALUES IN (7,9);
CREATE TABLE eco_catastrophy_list_default PARTITION OF eco_catastrophy_list DEFAULT;
CREATE TABLE eco_catastrophy_list_2_4_6_8 PARTITION OF eco_catastrophy_list
FOR VALUES IN (2,4,6,8)
PARTITION BY RANGE (data_catastrophe);
```

### Создание вложенных секций:

```
CREATE TABLE eco_catastrophy_list_2_4_6_8_2020 PARTITION OF
eco_catastrophy_list_2_4_6_8 FOR VALUES FROM ('2020-01-01') TO ('2021-01-01');
CREATE TABLE eco_catastrophy_list_2_4_6_8_2021 PARTITION OF
eco_catastrophy_list_2_4_6_8 FOR VALUES FROM ('2021-01-01') TO ('2022-01-01');
```

### Создание индекса:

```
CREATE INDEX ON eco_catastrophy_list ("id_company");
```

```
A-Zeco_catastrophy_list_0_id_company_idx
A-Zeco_catastrophy_list_1_3_id_company_idx
A-Zeco_catastrophy_list_2_4_6_8_2020_id_company_idx
A-Zeco_catastrophy_list_2_4_6_8_2021_id_company_idx
A-Zeco_catastrophy_list_2_4_6_8_id_company_idx
A-Zeco_catastrophy_list_7_9_id_company_idx
A-Zeco_catastrophy_list_default_id_company_idx
A-Zeco_catastrophy_list_id_company_idx
```

### Результат скопированных данных из eco\_catastrophy:

```
SELECT tableoid::regclass AS partition, count(*) FROM eco_catastrophy_list
GROUP BY tableoid;
```

partition	count
eco_catastrophy_list_0	50115
eco_catastrophy_list_1_3	150255
eco_catastrophy_list_7_9	99576
eco_catastrophy_list_2_4_6_8_2020	73562
eco_catastrophy_list_2_4_6_8_2021	126492

### Создание партии по хешу:

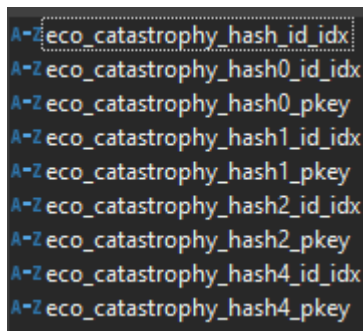
```
CREATE TABLE eco_catastrophy_hash (
  "id" SERIAL PRIMARY KEY NOT NULL,
  id_company int4 NOT NULL,
  id_type_catastrophe int4 NOT NULL,
  data_catastrophe TIMESTAMP NOT NULL,
  number_victims int4 NOT NULL,
  effects TEXT,
  detriment DECIMAL,
  id_department int4[]
) PARTITION BY HASH ("id");
```

### Создание секций:

```
CREATE TABLE eco_catastrophy_hash0 PARTITION OF eco_catastrophy_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE eco_catastrophy_hash1 PARTITION OF eco_catastrophy_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 1);
CREATE TABLE eco_catastrophy_hash2 PARTITION OF eco_catastrophy_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 2);
CREATE TABLE eco_catastrophy_hash4 PARTITION OF eco_catastrophy_hash
FOR VALUES WITH (MODULUS 4, REMAINDER 3);
```

### Создание индекса:

```
CREATE INDEX ON eco_catastrophy_hash ("id");
```



A screenshot of a database catalog window showing a list of indexes. The first index is 'eco\_catastrophy\_hash\_id\_idx' which is a primary key index on the 'id' column of the 'eco\_catastrophy\_hash' table. Below it, there are secondary indexes for each partition: 'eco\_catastrophy\_hash0\_id\_idx', 'eco\_catastrophy\_hash1\_id\_idx', 'eco\_catastrophy\_hash2\_id\_idx', and 'eco\_catastrophy\_hash4\_id\_idx'. Each of these partition indexes is also a primary key index on the 'id' column of its respective partition table.

### Заполнение с автоматической раскладкой по секциям:

```
INSERT INTO eco_catastrophy_hash
SELECT * FROM eco_catastrophy ON CONFLICT ("id") DO NOTHING;
```

### Распределение строк по секциям происходит равномерно:

```
SELECT tableoid::regclass AS partition, count(*) FROM eco_catastrophy_hash
GROUP BY tableoid;
```

partition	count
eco_catastrophy_hash0	124817
eco_catastrophy_hash1	125337
eco_catastrophy_hash2	124916
eco_catastrophy_hash4	124930

### Сравнение и анализ партиций и обычной таблицы:

Создадим обычную таблицу (*eco\_catastrophy\_test*) с такими же полями как у партиций и добавим индекс по *id* (*CREATE INDEX ON eco\_catastrophy\_test ("id")*) и заполним данными скопировав из партиции *eco\_catastrophy*).

```
EXPLAIN (ANALYZE) SELECT * FROM eco_catastrophy;
```

QUERY PLAN
Append (cost=0.00..17583.85 rows=500590 width=109) (actual time=0.172..109.407 rows=500000 loops=1)
-> Seq Scan on eco_catastrophy_2020_3kv (cost=0.00..2018.26 rows=50326 width=110) (actual time=0.171..15.838 rows=50326 loops=1)
-> Seq Scan on eco_catastrophy_2020_4kv (cost=0.00..4351.46 rows=134046 width=110) (actual time=0.324..29.372 rows=134046 loops=1)
-> Seq Scan on eco_catastrophy_2021_1kv (cost=0.00..7180.50 rows=260650 width=109) (actual time=0.058..34.598 rows=260650 loops=1)
-> Seq Scan on eco_catastrophy_2021_2kv (cost=0.00..1514.78 rows=54978 width=110) (actual time=0.043..11.417 rows=54978 loops=1)
-> Seq Scan on eco_catastrophy_default (cost=0.00..15.90 rows=590 width=109) (actual time=0.012..0.012 rows=0 loops=1)
Planning Time: 8.036 ms
Execution Time: 119.112 ms

**EXPLAIN (ANALYZE) SELECT \* FROM eco\_catastrophy\_list;**

QUERY PLAN
Append (cost=0.00..16295.25 rows=500550 width=110) (actual time=0.029..83.043 rows=500000 loops=1)
-> Seq Scan on eco_catastrophy_list_0 (cost=0.00..1380.15 rows=50115 width=109) (actual time=0.028..9.338 rows=50115 loops=1)
-> Seq Scan on eco_catastrophy_list_1_3 (cost=0.00..4141.55 rows=150255 width=110) (actual time=0.020..20.373 rows=150255 loops=1)
-> Seq Scan on eco_catastrophy_list_2_4_6_8_2020 (cost=0.00..2027.62 rows=73562 width=110) (actual time=0.088..11.328 rows=73562 loops=1)
-> Seq Scan on eco_catastrophy_list_2_4_6_8_2021 (cost=0.00..3484.92 rows=126492 width=110) (actual time=0.016..13.976 rows=126492 loops=1)
-> Seq Scan on eco_catastrophy_list_7_9 (cost=0.00..2742.76 rows=99576 width=109) (actual time=0.021..10.563 rows=99576 loops=1)
-> Seq Scan on eco_catastrophy_list_default (cost=0.00..15.50 rows=550 width=120) (actual time=0.020..0.020 rows=0 loops=1)
Planning Time: 12.254 ms
Execution Time: 92.455 ms

**EXPLAIN (ANALYZE) SELECT \* FROM eco\_catastrophy\_hash;**

QUERY PLAN
Append (cost=0.00..16277.00 rows=500000 width=109) (actual time=0.039..111.655 rows=500000 loops=1)
-> Seq Scan on eco_catastrophy_hash0 (cost=0.00..3439.17 rows=124817 width=109) (actual time=0.039..24.985 rows=124817 loops=1)
-> Seq Scan on eco_catastrophy_hash1 (cost=0.00..3453.37 rows=125337 width=109) (actual time=0.036..23.587 rows=125337 loops=1)
-> Seq Scan on eco_catastrophy_hash2 (cost=0.00..3442.16 rows=124916 width=109) (actual time=0.030..21.559 rows=124916 loops=1)
-> Seq Scan on eco_catastrophy_hash4 (cost=0.00..3442.30 rows=124930 width=110) (actual time=0.034..19.950 rows=124930 loops=1)
Planning Time: 7.492 ms
Execution Time: 123.029 ms

**EXPLAIN (ANALYZE) SELECT \* FROM eco\_catastrophy\_test;**

QUERY PLAN
Seq Scan on eco_catastrophy_test (cost=0.00..13776.00 rows=500000 width=110) (actual time=0.291..237.331 rows=500000 loops=1)
Planning Time: 5.364 ms
Execution Time: 249.222 ms

Из результатов видим что по затраченному времени на запрос (Execution Time) показатель почти у всех на одном уровне (не считая *eco\_catastrophy\_test*), быстрее справилась партиция разбитая по списку (*eco\_catastrophy\_list*) – 92.455 ms, а медленнее обычная таблица (*eco\_catastrophy\_test*) – 249.222 ms, что почти в 2 раза, хотя количество записей одинаково.

Общее количество строк (rows), полученных при последовательном сканировании (Seq Scan) у всех одинаково – 500000. У партиций данный параметр можно посмотреть по каждой секции.

Оценка затратности операции (cost) у партиций почти одинаковы (*eco\_catastrophy* – 17583.85, *eco\_catastrophy\_list* – 16295.25, *eco\_catastrophy\_hash* – 16277.00) а у обычной таблицы (*eco\_catastrophy\_test*) – 13766.00.

Добавим к запросу условие на дату и сортировку. Сортировка, для партиции – *eco\_catastrophy* будет по полю *data\_catastrophe* (поле, по которому разбита партиция на секции), для *eco\_catastrophy\_list* будет по полю *id\_company* (поле, по которому разбита партиция по секциям), для *eco\_catastrophy\_hash* будет по полю *id* (поле, по которому разбита партиция по секциям) и для *eco\_catastrophy\_test* будет по полю *id*.



```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy
WHERE data_catastrophe>='2021-04-01' ORDER BY data_catastrophe;
```

QUERY PLAN
Merge Append (cost=0.45..7756.52 rows=55568 width=110) (actual time=0.020..34.861 rows=54978 loops=1)
Sort Key: eco_catastrophy_2021_2kv.data_catastrophe
-> Index Scan using eco_catastrophy_2021_2kv_data_catastrophe_idx on eco_catastrophy_2021_2kv (cost=0.29..6270.35 rows=54978 width=110) (actual time=0.019..34.859 rows=54978 loops=1)
Index Cond: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
-> Index Scan using eco_catastrophy_default_data_catastrophe_idx on eco_catastrophy_default (cost=0.15..930.47 rows=590 width=109) (actual time=0.001..0.001 rows=0 loops=1)
Index Cond: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Planning Time: 7.547 ms
Execution Time: 36.036 ms

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy_list
WHERE data_catastrophe>='2021-04-01' ORDER BY id_company;
```

QUERY PLAN
Gather Merge (cost=13436.41..18859.93 rows=46484 width=110) (actual time=127.618..162.163 rows=54978 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=12436.39..12494.49 rows=23242 width=110) (actual time=80.298..82.489 rows=18326 loops=3)
Sort Key: eco_catastrophy_list_1_3.id_company
Sort Method: external merge Disk: 2968kB
Worker 0: Sort Method: quicksort Memory: 1400kB
Worker 1: Sort Method: external merge Disk: 2832kB
-> Parallel Append (cost=0.00..10750.83 rows=23242 width=110) (actual time=30.496..64.008 rows=18326 loops=3)
-> Parallel Seq Scan on eco_catastrophy_list_1_3 (cost=0.00..3743.82 rows=9959 width=110) (actual time=18.188..21.900 rows=5504 loops=3)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 44581
-> Parallel Seq Scan on eco_catastrophy_list_2_4_6_8_2021 (cost=0.00..3150.09 rows=12881 width=110) (actual time=22.815..29.020 rows=11050 loops=2)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 52196
-> Parallel Seq Scan on eco_catastrophy_list_7_9 (cost=0.00..2479.18 rows=6519 width=109) (actual time=36.511..42.518 rows=10801 loops=1)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 88775
-> Parallel Seq Scan on eco_catastrophy_list_0 (cost=0.00..1247.49 rows=3346 width=109) (actual time=18.587..21.820 rows=5564 loops=1)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 44551
-> Parallel Seq Scan on eco_catastrophy_list_default (cost=0.00..14.04 rows=108 width=120) (actual time=0.001..0.001 rows=0 loops=1)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Planning Time: 5.126 ms
Execution Time: 164.815 ms

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy_hash
WHERE data_catastrophe>='2021-04-01' ORDER BY "id";
```

QUERY PLAN
Gather Merge (cost=15211.58..20514.69 rows=45452 width=109) (actual time=166.775..182.536 rows=54978 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=14211.56..14268.38 rows=22726 width=109) (actual time=94.725..96.156 rows=18326 loops=3)
Sort Key: eco_catastrophy_hash1.id
Sort Method: external merge Disk: 4040kB
Worker 0: Sort Method: quicksort Memory: 3369kB
Worker 1: Sort Method: quicksort Memory: 1431kB
-> Parallel Append (cost=0.00..12567.10 rows=22726 width=109) (actual time=57.129..82.609 rows=18326 loops=3)
-> Parallel Seq Scan on eco_catastrophy_hash1 (cost=0.00..3121.60 rows=8274 width=109) (actual time=47.016..55.011 rows=13933 loops=1)

```

Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 111404
-> Parallel Seq Scan on eco_catastrophy_hash4 (cost=0.00..3111.60 rows=8114 width=110) (actual time= 16.578..18.486 rows=4568 loops=3)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 37075
-> Parallel Seq Scan on eco_catastrophy_hash2 (cost=0.00..3111.50 rows=7920 width=109) (actual time=41.328..49.729 rows=13610 loops=1)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 111306
-> Parallel Seq Scan on eco_catastrophy_hash0 (cost=0.00..3108.77 rows=7776 width=109) (actual time=76.030..83.853 rows=13730 loops=1)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 111087
Planning Time: 7.186 ms
Execution Time: 184.856 ms

```

## EXPLAIN (ANALYZE)

```

SELECT * FROM eco_catastrophy_test
WHERE data_catastrophe>='2021-04-01' ORDER BY "id";

```

```

QUERY PLAN
Gather Merge (cost=14053.40..19440.74 rows=46174 width=110) (actual time= 122.495..161.342 rows=54978 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=13053.37..13111.09 rows=23087 width=110) (actual time= 72.899..76.275 rows= 18326 loops=3)
Sort Key: id
Sort Method: external merge Disk: 2144kB
Worker 0: Sort Method: external merge Disk: 2144kB
Worker 1: Sort Method: external merge Disk: 2224kB
-> Parallel Seq Scan on eco_catastrophy_test (cost=0.00..11380.17 rows=23087 width=110) (actual time=42.982..55.853 rows=18326 loops=3)
Filter: (data_catastrophe >= '2021-04-01 00:00:00'::timestamp without time zone)
Rows Removed by Filter: 148341
Planning Time: 2.898 ms
Execution Time: 164.194 ms

```

Из результатов видно, что первая партиция (*eco\_catastrophy*) разбитая по диапазонам дат справилась лучше всех – по затраченному времени (Execution Time) на запрос – 36.036 ms, оценка затратности операции (cost) – 7756.52. Это очевидно потому что условие поиска и сортировка происходила по ключу разбиения партиции на секции, что привело к тому что поиск нужных записей проводился в 2-х секциях – *eco\_catastrophy\_20201\_2kv* (это секция с данными за период от 01.04.2021 до 01.07.2021, как раз попадающая под условие поиска *data\_catastrophe>='2021-04-01'*) и *eco\_catastrophy\_default* (это секция где хранятся данные которые не попали в остальные секции по разным причинам, но на данный момент она пуста). Также стоит отметить, что, последовательное сканирование (Seq Scan) сменилось на Index Scan — используется индекс *eco\_catastrophy\_20201\_2kv\_data\_catastrophe\_idx* (для секции *eco\_catastrophy\_20201\_2kv* и *eco\_catastrophy\_default\_data\_catastrophe\_idx* для секции *eco\_catastrophy\_default*) для условий *WHERE* и читает таблицу при отборе строк.

Вторая (*eco\_catastrophy\_list*) и третья (*eco\_catastrophy\_hash*) партиции справилась хуже, им пришлось сканировать все секции в поисках нужных записей, т.к. ключом разбиение у второй является поле *id\_company*, а у третьей поле *id*, а не поле дат (*data\_catastrophe*) как у первой партиции. Но стоит отметить, что, у второй партиции (*eco\_catastrophy\_list*) подсекция *eco\_catastrophy\_list\_2\_4\_6\_8\_2020* не была задействована в сканировании. Напомню, что мы секцию *eco\_catastrophy\_list\_2\_4\_6\_8* разбили на 2 подсекции – *eco\_catastrophy\_list\_2\_4\_6\_8\_2020* (подсекция с данными у которых поле *data\_catastrophe* лежит в пределах 2020 года) и *eco\_catastrophy\_list\_2\_4\_6\_8\_2021* (подсекция с данными у которых поле *data\_catastrophe* лежит в пределах 2021 года). Поэтому т.к. у нас условие найти данные, у которых *data\_catastrophe>='2021-04-01'*, секция *eco\_catastrophy\_list\_2\_4\_6\_8\_2020* с данными за 2020 год отбрасывается. Потом партиции сортируются: вторая (*eco\_catastrophy\_list*) по *id\_company*, третья (*eco\_catastrophy\_hash*) по *id*. Используется последовательное сканирование секций, но в параллельном режиме (Parallel Seq Scan).



Обычная таблица справилась чуть лучше. По затраченному времени (Execution Time) на запрос показатель равен 164.194 ms, что наравне со второй партицией (*eco\_catastrophy\_list*) – 164.815 ms. Оценка затратности операции (cost) – 19440.74, что хуже, чем у первой (7756.52) и второй (18859.93) партиций, но лучше, чем у третьей (20514.69) партиции.

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy WHERE id_company=5 ORDER BY data_catastrophe;
```

QUERY PLAN
Gather Merge (cost=15838.53..20819.61 rows=42692 width=109) (actual time=141.728..172.134 rows=50327 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=14838.50..14891.87 rows=21346 width=109) (actual time=94.757..97.017 rows=16776 loops=3)
Sort Key: eco_catastrophy_2021_1kv.data_catastrophe
Sort Method: external merge Disk: 2512kB
Worker 0: Sort Method: external merge Disk: 2136kB
Worker 1: Sort Method: quicksort Memory: 2651kB
-> Parallel Append (cost=0.00..13303.55 rows=21346 width=109) (actual time=0.255..78.437 rows=16776 loops=3)
-> Parallel Seq Scan on eco_catastrophy_2021_1kv (cost=0.00..5931.55 rows=11081 width=109) (actual time=0.337..28.370 rows=8748 loops=3)
Filter: (id_company = 5)
Rows Removed by Filter: 78135
-> Parallel Seq Scan on eco_catastrophy_2020_4kv (cost=0.00..3996.63 rows=8211 width=110) (actual time=8.025..52.330 rows=6739 loops=2)
Filter: (id_company = 5)
Rows Removed by Filter: 60285
-> Parallel Seq Scan on eco_catastrophy_2020_3kv (cost=0.00..1885.04 rows=2920 width=110) (actual time=6.885..27.945 rows=4992 loops=1)
Filter: (id_company = 5)
Rows Removed by Filter: 45334
-> Parallel Seq Scan on eco_catastrophy_2021_2kv (cost=0.00..1369.25 rows=3326 width=110) (actual time=0.036..13.862 rows=5614 loops=1)
Filter: (id_company = 5)
Rows Removed by Filter: 49364
-> Parallel Seq Scan on eco_catastrophy_default (cost=0.00..14.34 rows=35 width=109) (actual time=0.001..0.001 rows=0 loops=1)
Filter: (id_company = 5)
Planning Time: 3.480 ms
Execution Time: 175.555 ms

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy_list WHERE id_company=5 ORDER BY id_company;
```

QUERY PLAN
Bitmap Heap Scan on eco_catastrophy_list_1_3 (cost=915.18..4184.45 rows=50421 width=110) (actual time=23.025..68.221 rows=50327 loops=1)
Recheck Cond: (id_company = 5)
Heap Blocks: exact=2639
-> Bitmap Index Scan on eco_catastrophy_list_1_3_id_company_idx (cost=0.00..902.58 rows=50421 width=0) (actual time=22.586..22.586 rows=50327 loops=1)
Index Cond: (id_company = 5)
Planning Time: 0.236 ms
Execution Time: 69.485 ms

```
EXPLAIN (ANALYZE)
SELECT * FROM eco_catastrophy_hash WHERE id_company=5 ORDER BY "id";
```

QUERY PLAN
Gather Merge (cost=15065.59..19964.53 rows=41988 width=109) (actual time=151.616..169.143 rows=50327 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=14065.57..14118.05 rows=20994 width=109) (actual time=106.395..108.315 rows=16776 loops=3)
Sort Key: eco_catastrophy_hash1.id
Sort Method: external merge Disk: 3184kB
Worker 0: Sort Method: quicksort Memory: 2727kB
Worker 1: Sort Method: quicksort Memory: 2813kB

```

-> Parallel Append (cost=0.00..12558.44 rows=20994 width=109) (actual time=0.308..96.590 rows= 16776 loops=3)
-> Parallel Seq Scan on eco_catastrophy_hash1 (cost=0.00..3121.60 rows=7395 width=109) (actual time=0.427..27.563 rows=4211 loops=3)
    Filter: (id_company = 5)
    Rows Removed by Filter: 37568
-> Parallel Seq Scan on eco_catastrophy_hash4 (cost=0.00..3111.60 rows=7309 width=110) (actual time=0.227..39.776 rows=6271 loops=2)
    Filter: (id_company = 5)
    Rows Removed by Filter: 56195
-> Parallel Seq Scan on eco_catastrophy_hash2 (cost=0.00..3111.50 rows=7595 width=109) (actual time=0.023..71.875 rows= 12711 loops=1)
    Filter: (id_company = 5)
    Rows Removed by Filter: 112205
-> Parallel Seq Scan on eco_catastrophy_hash0 (cost=0.00..3108.77 rows=7340 width=109) (actual time=0.037..52.043 rows=12442 loops=1)
    Filter: (id_company = 5)
    Rows Removed by Filter: 112375
Planning Time: 0.202 ms
Execution Time: 171.902 ms

```

## EXPLAIN (ANALYZE)

```
SELECT * FROM eco_catastrophy_test WHERE id_company=5 ORDER BY "id";
```

```

QUERY PLAN
Gather Merge (cost=13887.79..18788.13 rows=42000 width=110) (actual time=105.399..131.946 rows=50327 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Sort (cost=12887.77..12940.27 rows=21000 width=110) (actual time=59.105..60.348 rows= 16776 loops=3)
    Sort Key: id
    Sort Method: external merge Disk: 3208kB
    Worker 0: Sort Method: quicksort Memory: 2488kB
    Worker 1: Sort Method: quicksort Memory: 3022kB
  -> Parallel Seq Scan on eco_catastrophy_test (cost=0.00..11380.17 rows=21000 width=110) (actual time=0.294..49.014 rows=16776 loops=3)
    Filter: (id_company = 5)
    Rows Removed by Filter: 149891
Planning Time: 0.080 ms
Execution Time: 135.120 ms

```

Поменяв условие поиска записей в *WHERE* (теперь ищем все записи у которых поле *id\_company* равно пяти), видно что вторая партиция (*eco\_catastrophy\_list*) разбитая по списку справилось лучше всех, по затраченному времени (Execution Time) на запрос – 69.485 ms, оценка затратности операции (cost) – 4184.45. Это потому, что условие поиска и сортировка происходила по ключу разбиения её секций. Поэтому поиск происходит только в одной секции – *eco\_catastrophy\_list\_1\_3*, где хранятся записи с *id\_company* равным 1,3 и 5 что удовлетворяет нашему условию *WHERE id\_company=5*. Также стоит отметить, что, последовательное сканирование (Seq Scan) сменилось на Bitmap Index Scan — используется индекс *eco\_catastrophy\_list\_1\_3\_id\_company\_idx* для определения нужных нам записей, а затем PostgreSQL лезет в саму таблицу (Bitmap Heap Scan), чтобы убедиться, что эти записи на самом деле существуют.

Первая партиция (*eco\_catastrophy*) которую разбивали по диапазонам дат и третья партиция (*eco\_catastrophy\_hash*) разбитая по хешу, справилась хуже, т.к. поиск проводился по всем секциям партиций. Результаты первой партиции (*eco\_catastrophy*) - по затраченному времени (Execution Time) на запрос – 175.555 ms, оценка затратности операции (cost) – 20819.61. Результаты третьей партиции (*eco\_catastrophy\_hash*) - по затраченному времени (Execution Time) на запрос – 171.902 ms, оценка затратности операции (cost) – 19964.53. У обеих партиций используется последовательное сканирование секций, но в параллельном режиме (Parallel Seq Scan).

Показатели обычной таблицы по затраченному времени (Execution Time) на запрос – 135.120 ms, оценка затратности операции (cost) – 18788.13.

Добавим еще индексов (по ключевым полям) к партициям и к обычной таблице, и проверим на других запросах, как будет происходить их построение и какие из индексов будут участвовать в построении запроса, а какие нет.

```

CREATE INDEX ON eco_catastrophy ("id_company");
CREATE INDEX ON eco_catastrophy ("id");
CREATE INDEX ON eco_catastrophy_list ("data_catastrophe");
CREATE INDEX ON eco_catastrophy_list ("id");
CREATE INDEX ON eco_catastrophy_hash ("id_company");
CREATE INDEX ON eco_catastrophy_hash ("data_catastrophe");
CREATE INDEX ON eco_catastrophy_test ("id_company");
CREATE INDEX ON eco_catastrophy_test ("data_catastrophe");

```

```

EXPLAIN (ANALYZE) SELECT
    "id", id_company, data_catastrophe::date, number_victims, effects, detriment
FROM eco_catastrophy WHERE data_catastrophe>='2021-02-25' AND
    id_company IN(7,5) ORDER BY "id", id_company;

```

QUERY PLAN
Sort (cost=9730.88..9795.66 rows=25913 width=61) (actual time=79.862..82.285 rows=25674 loops=1)
Sort Key: eco_catastrophy_2021_1kv.id, eco_catastrophy_2021_1kv.id_company
Sort Method: external merge Disk: 1808kB
-> Append (cost=928.21..7831.28 rows=25913 width=61) (actual time=3.977..64.346 rows=25674 loops=1)
-> Bitmap Heap Scan on eco_catastrophy_2021_1kv (cost=928.21..6322.31 rows=14685 width=61) (actual time=3.976..52.169 rows=14615 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 37692
Heap Blocks: exact=4574
-> Bitmap Index Scan on eco_catastrophy_2021_1kv_id_company_idx (cost=0.00..924.53 rows=52226 width=0) (actual time=3.490..3.490 rows=52226 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
-> Bitmap Heap Scan on eco_catastrophy_2021_2kv (cost=202.70..1362.14 rows=11111 width=61) (actual time=0.738..11.075 rows=11059 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Heap Blocks: exact=965
-> Bitmap Index Scan on eco_catastrophy_2021_2kv_id_company_idx (cost=0.00..199.92 rows=11111 width=0) (actual time=0.643..0.643 rows=11059 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
-> Bitmap Heap Scan on eco_catastrophy_default (cost=5.21..17.26 rows=117 width=61) (actual time=0.006..0.006 rows=0 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
-> Bitmap Index Scan on eco_catastrophy_default_id_company_idx (cost=0.00..5.18 rows=117 width=0) (actual time=0.005..0.005 rows=0 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
Planning Time: 8.845 ms
Execution Time: 84.292 ms

```

EXPLAIN (ANALYZE)SELECT
    "id", id_company, data_catastrophe::date, number_victims, effects, detriment
FROM eco_catastrophy_list WHERE data_catastrophe>='2021-02-25' AND
    id_company IN(7,5) ORDER BY "id", id_company;

```

QUERY PLAN
Sort (cost=8358.54..8423.16 rows=25849 width=61) (actual time=80.615..82.480 rows=25674 loops=1)
Sort Key: eco_catastrophy_list_1_3.id, eco_catastrophy_list_1_3.id_company
Sort Method: external merge Disk: 1808kB
-> Append (cost=0.42..6464.09 rows=25849 width=61) (actual time=0.070..62.989 rows=25674 loops=1)
-> Index Scan using eco_catastrophy_list_1_3_data_catastrophe_idx on eco_catastrophy_list_1_3 (cost=0.42..3799.74 rows=13075 width=61) (actual time=0.070..62.989 rows=13075 loops=1)
Index Cond: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Filter: (id_company = ANY ('{7,5}':integer[]))
Rows Removed by Filter: 25377
-> Index Scan using eco_catastrophy_list_7_9_data_catastrophe_idx on eco_catastrophy_list_7_9 (cost=0.29..2535.11 rows=12774 width=61) (actual time=0.006..62.989 rows=12774 loops=1)
Index Cond: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Filter: (id_company = ANY ('{7,5}':integer[]))
Rows Removed by Filter: 12684
Planning Time: 4.215 ms
Execution Time: 84.213 ms

EXPLAIN (ANALYZE) SELECT

```
"id", id_company, data_catastrophe::date, number_victims, effects, detriment
FROM eco_catastrophy_hash WHERE data_catastrophe>='2021-02-25' AND
id_company IN(7,5) ORDER BY "id", id_company;
```

QUERY PLAN
Sort (cost=14192.63..14257.34 rows=25884 width=61) (actual time=109.247..110.992 rows=25674 loops=1)
Sort Key: eco_catastrophy_hash0.id, eco_catastrophy_hash0.id_company
Sort Method: external merge Disk: 1808kB
-> Append (cost=456.98..12295.36 rows=25884 width=61) (actual time=10.362..95.598 rows=25674 loops=1)
-> Bitmap Heap Scan on eco_catastrophy_hash0 (cost=456.98..3045.18 rows=6458 width=61) (actual time=10.361..24.559 rows=6273 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 18716
Heap Blocks: exact=2191
-> Bitmap Index Scan on eco_catastrophy_hash0_id_company_idx (cost=0.00..455.36 rows=25404 width=0) (actual time=1.626..1.626 rows=24989 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
-> Bitmap Heap Scan on eco_catastrophy_hash1 (cost=446.70..3039.33 rows=6447 width=61) (actual time=9.694..25.013 rows=6421 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 18622
Heap Blocks: exact=2200
-> Bitmap Index Scan on eco_catastrophy_hash1_id_company_idx (cost=0.00..445.08 rows=25101 width=0) (actual time=1.648..1.648 rows=25043 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
Index Cond: (id_company = ANY ('{7,5}':integer[]))
-> Bitmap Heap Scan on eco_catastrophy_hash2 (cost=457.99..3050.38 rows=6536 width=61) (actual time=11.589..25.965 rows=6447 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 18676
Heap Blocks: exact=2193
-> Bitmap Index Scan on eco_catastrophy_hash2_id_company_idx (cost=0.00..456.35 rows=25537 width=0) (actual time=1.683..1.683 rows=25123 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
-> Bitmap Heap Scan on eco_catastrophy_hash4 (cost=446.28..3031.04 rows=6443 width=61) (actual time=9.289..19.005 rows=6533 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 18492
Heap Blocks: exact=2193
-> Bitmap Index Scan on eco_catastrophy_hash4_id_company_idx (cost=0.00..444.66 rows=25044 width=0) (actual time=1.659..1.659 rows=25025 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
Planning Time: 8.479 ms
Execution Time: 112.640 ms

EXPLAIN (ANALYZE) SELECT

```
"id", id_company, data_catastrophe::date, number_victims, effects, detriment
FROM eco_catastrophy_test WHERE data_catastrophe>='2021-02-25' AND
id_company IN(7,5) ORDER BY "id", id_company;
```

QUERY PLAN
Sort (cost=14030.83..14095.61 rows=25910 width=61) (actual time=87.876..90.080 rows=25674 loops=1)
Sort Key: id, id_company
Sort Method: external merge Disk: 1808kB
-> Bitmap Heap Scan on eco_catastrophy_test (cost=1776.45..12131.47 rows=25910 width=61) (actual time=13.850..74.350 rows=25674 loops=1)
Recheck Cond: (id_company = ANY ('{7,5}':integer[]))
Filter: (data_catastrophe >= '2021-02-25 00:00:00':timestamp without time zone)
Rows Removed by Filter: 74506
Heap Blocks: exact=8776
-> Bitmap Index Scan on eco_catastrophy_test_id_company_idx (cost=0.00..1769.97 rows=100950 width=0) (actual time=11.943..11.943 rows=100180 loops=1)
Index Cond: (id_company = ANY ('{7,5}':integer[]))
Planning Time: 2.067 ms
Execution Time: 91.764 ms



Из полученных результатов видим, что первая партиция (разбитая по диапазону) и вторая партиция (разбитая по спискам) справилась лучше всех.

Результаты первой партиции (*eco\_catastrophy*) - по затраченному времени (Execution Time) на запрос – 84.292 ms, оценка затратности операции (cost) – 9795.66. Результаты второй партиции (*eco\_catastrophy\_list*) - по затраченному времени (Execution Time) на запрос – 84.213 ms, оценка затратности операции (cost) – 8423.16. В первой партиции поиск записей происходит с помощью Bitmap Index Scan — который использует индексы: *eco\_catastrophy\_2021\_1kv\_id\_company\_idx* для секции *eco\_catastrophy\_2021\_1kv*, *eco\_catastrophy\_2021\_2kv\_id\_company\_idx* для секции *eco\_catastrophy\_2021\_2kv* и *eco\_catastrophy\_default\_id\_company\_idx* для секции *eco\_catastrophy\_default*, чтобы определить нужные нам записи, а затем PostgreSQL лезет в саму таблицу: (Bitmap Heap Scan), чтобы убедиться, что эти записи на самом деле существуют. Следует отметить, что это новый индекс, который мы создали для этой партиции по полю *id\_company*. Во второй партиции поиск записей происходит с помощью Index Scan — который использует индексы: *eco\_catastrophy\_list\_1\_3\_data\_catastrophe\_idx* для секции *eco\_catastrophy\_list\_1\_3*, *eco\_catastrophy\_list\_7\_9\_data\_catastrophe\_idx* для секции *eco\_catastrophy\_list\_7\_9*, чтобы определить нужные нам записи при условии *WHERE*, читает таблицу при отборе строк. Следует отметить, что это новый индекс, который мы создали для этой партиции по полю *data\_catastrophe*.

Третья партиция (*eco\_catastrophy\_hash*) справилась хуже всех. Результаты: по затраченному времени (Execution Time) на запрос – 112.640 ms, оценка затратности операции (cost) – 14257.34. Поиск записей происходит по всем секциям с помощью Bitmap Index Scan — который использует новый индекс, который мы создали для этой партиции по полю *id\_company*.

Результаты таблицы *eco\_catastrophy\_test*: по затраченному времени (Execution Time) на запрос – 91.764 ms, оценка затратности операции (cost) – 14095.61. Обычная таблица по затраченному времени и по оценке затратности операций уступает первой и второй партициям, но лучше чем партиция разбитая по хешу но не на много. Также стоит отметить, что по сравнению с предыдущими запросами, последовательное сканирование (Seq Scan) или последовательное сканирование в параллельном режиме (Parallel Seq Scan) сменилось на Bitmap Index Scan — используется новый индекс *eco\_catastrophy\_test\_id\_company\_idx* для определения нужных нам записей, а затем PostgreSQL лезет в саму таблицу (Bitmap Heap Scan), чтобы убедиться, что эти записи на самом деле существуют.

#### EXPLAIN (ANALYZE)

```
SELECT id_company, max(detriment), min(detriment)
FROM eco_catastrophy WHERE data_catastrophe > '2021-01-01'
AND number_victims < 1000
GROUP BY id_company ORDER BY id_company;
```

QUERY PLAN
Finalize GroupAggregate (cost=8842.81..8894.48 rows=200 width=68) (actual time= 101.189..101.221 rows= 10 loops= 1)
Group Key: eco_catastrophy_2021_1kv.id_company
-> Gather Merge (cost=8842.81..8889.48 rows=400 width=68) (actual time= 101.184..105.963 rows= 30 loops= 1)
Workers Planned: 2
Workers Launched: 2
-> Sort (cost=7842.79..7843.29 rows=200 width=68) (actual time=48.472..48.473 rows= 10 loops= 3)
Sort Key: eco_catastrophy_2021_1kv.id_company
Sort Method: quicksort Memory: 25kB
Worker 0: Sort Method: quicksort Memory: 25kB
Worker 1: Sort Method: quicksort Memory: 25kB
-> Partial HashAggregate (cost=7833.14..7835.14 rows=200 width=68) (actual time=48.377..48.380 rows= 10 loops= 3)
Group Key: eco_catastrophy_2021_1kv.id_company
-> Parallel Append (cost=0.00..7734.28 rows=13182 width=16) (actual time=0.452..41.651 rows= 10541 loops= 3)
-> Parallel Seq Scan on eco_catastrophy_2021_1kv (cost=0.00..6203.06 rows=10867 width=16) (actual time=0.456..31.938 rows= 8676 loops= 3)
Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
Rows Removed by Filter: 78208
-> Parallel Seq Scan on eco_catastrophy_2021_2kv (cost=0.00..1450.10 rows=3267 width=16) (actual time=0.036..26.976 rows= 5596 loops= 3)
Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))



```

Rows Removed by Filter: 49382
-> Parallel Seq Scan on eco_catastrophe_default (cost=0.00..15.21 rows=2 width=16) (actual time=0.005..0.005 rows=0 loops=1)
    Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
Planning Time: 2.064 ms
Execution Time: 106.090 ms

```

EXPLAIN (ANALYZE)

```

SELECT id_company, max(detriment), min(detriment)
FROM eco_catastrophe_list WHERE data_catastrophe>'2021-01-01'
AND number_victims<1000
GROUP BY id_company ORDER BY id_company;

```

```

QUERY PLAN
Finalize GroupAggregate (cost=12436.33..12488.00 rows=200 width=68) (actual time=104.255..104.270 rows=10 loops=1)
  Group Key: eco_catastrophe_list_1_3.id_company
  -> Gather Merge (cost=12436.33..12483.00 rows=400 width=68) (actual time=104.251..125.693 rows=20 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Sort (cost=11436.31..11436.81 rows=200 width=68) (actual time=57.463..57.464 rows=7 loops=3)
      Sort Key: eco_catastrophe_list_1_3.id_company
      Sort Method: quicksort Memory: 25kB
      Worker 0: Sort Method: quicksort Memory: 25kB
      Worker 1: Sort Method: quicksort Memory: 25kB
      -> Partial HashAggregate (cost=11426.67..11428.67 rows=200 width=68) (actual time=57.430..57.432 rows=7 loops=3)
        Group Key: eco_catastrophe_list_1_3.id_company
        -> Parallel Append (cost=0.00..11328.19 rows=13130 width=16) (actual time=10.046..51.919 rows=10541 loops=3)
          -> Parallel Seq Scan on eco_catastrophe_list_1_3 (cost=0.00..3964.78 rows=5675 width=16) (actual time=7.561..16.405 rows=3177 loops=3)
            Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
            Rows Removed by Filter: 46908
          -> Parallel Seq Scan on eco_catastrophe_list_2_4_6_8_2021 (cost=0.00..3336.11 rows=7271 width=16) (actual time=0.314..26.593 rows=6347 loops=1)
            Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
            Rows Removed by Filter: 56899
          -> Parallel Seq Scan on eco_catastrophe_list_7_9 (cost=0.00..2625.61 rows=3706 width=16) (actual time=15.586..30.639 rows=6257 loops=1)
            Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
            Rows Removed by Filter: 93319
          -> Parallel Seq Scan on eco_catastrophe_list_0 (cost=0.00..1321.19 rows=1849 width=16) (actual time=6.875..20.003 rows=3142 loops=1)
            Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
            Rows Removed by Filter: 46973
          -> Parallel Seq Scan on eco_catastrophe_list_default (cost=0.00..14.85 rows=36 width=36) (actual time=0.001..0.001 rows=0 loops=1)
            Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
Planning Time: 6.165 ms
Execution Time: 125.872 ms

```

EXPLAIN (ANALYZE)

```

SELECT id_company, max(detriment), min(detriment)
FROM eco_catastrophe_hash WHERE data_catastrophe>'2021-01-01'
AND number_victims<1000
GROUP BY id_company ORDER BY id_company;

```

```

QUERY PLAN
Finalize GroupAggregate (cost=14363.94..14415.61 rows=200 width=68) (actual time=126.026..126.047 rows=10 loops=1)
  Group Key: eco_catastrophe_hash1.id_company
  -> Gather Merge (cost=14363.94..14410.61 rows=400 width=68) (actual time=126.017..131.798 rows=30 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Sort (cost=13363.92..13364.42 rows=200 width=68) (actual time=71.309..71.310 rows=10 loops=3)
      Sort Key: eco_catastrophe_hash1.id_company
      Sort Method: quicksort Memory: 25kB
      Worker 0: Sort Method: quicksort Memory: 25kB
      Worker 1: Sort Method: quicksort Memory: 25kB
      -> Partial HashAggregate (cost=13354.28..13356.28 rows=200 width=68) (actual time=71.167..71.171 rows=10 loops=3)

```

```

Group Key: eco_catastrophy_hash1.id_company
-> Parallel Append (cost=0.00..13254.97 rows=13241 width=16) (actual time=19.216..65.534 rows=10541 loops=3)
    -> Parallel Seq Scan on eco_catastrophy_hash1 (cost=0.00..3305.91 rows=4665 width=16) (actual time=5.537..17.785 rows=2636 loops=3)
        Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
        Rows Removed by Filter: 39143
    -> Parallel Seq Scan on eco_catastrophy_hash4 (cost=0.00..3295.32 rows=4562 width=16) (actual time=7.529..20.051 rows=3925 loops=2)
        Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
        Rows Removed by Filter: 58541
    -> Parallel Seq Scan on eco_catastrophy_hash2 (cost=0.00..3295.20 rows=4755 width=16) (actual time=12.209..40.143 rows=7967 loops=1)
        Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
        Rows Removed by Filter: 116949
    -> Parallel Seq Scan on eco_catastrophy_hash0 (cost=0.00..3292.33 rows=4712 width=16) (actual time=26.601..60.873 rows=7898 loops=1)
        Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
        Rows Removed by Filter: 116919
Planning Time: 0.698 ms
Execution Time: 131.945 ms

```

#### EXPLAIN (ANALYZE)

```

SELECT id_company, max(detriment), min(detriment)
FROM eco_catastrophy_test WHERE data_catastrophe>'2021-01-01'
AND number_victims<1000
GROUP BY id_company ORDER BY id_company;

```

```

QUERY PLAN
Finalize GroupAggregate (cost=13000.93..13003.52 rows=10 width=68) (actual time=105.145..105.164 rows=10 loops=1)
  Group Key: id_company
  -> Gather Merge (cost=13000.93..13003.27 rows=20 width=68) (actual time=105.137..118.582 rows=30 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Sort (cost=12000.91..12000.94 rows=10 width=68) (actual time=56.376..56.377 rows=10 loops=3)
      Sort Key: id_company
      Sort Method: quicksort Memory: 25kB
      Worker 0: Sort Method: quicksort Memory: 25kB
      Worker 1: Sort Method: quicksort Memory: 25kB
    -> Partial HashAggregate (cost=12000.65..12000.75 rows=10 width=68) (actual time=56.340..56.342 rows=10 loops=3)
      Group Key: id_company
      -> Parallel Seq Scan on eco_catastrophy_test (cost=0.00..11901.00 rows=13286 width=16) (actual time=0.394..51.561 rows=10541 loops=3)
          Filter: ((data_catastrophe > '2021-01-01 00:00:00'::timestamp without time zone) AND (number_victims < 1000))
          Rows Removed by Filter: 156126
Planning Time: 0.117 ms
Execution Time: 118.643 ms

```

Из полученных результатов видим что первая партиция (разбитая по диапазонам) производит поиск только в секциях где хранятся данные за 2021 год – это секция *eco\_catastrophy\_2021\_1kv* и секция *eco\_catastrophy\_2021\_2kv*, а также секция отвечающая за данные которые не попали ни в одну из секций – это секция *eco\_catastrophy\_default*. Результаты первой партиции (*eco\_catastrophy*) - по затраченному времени (Execution Time) на запрос – 106.090 ms, оценка затратности операции (cost) – 8894.48. Следует отметить, что используется последовательное сканирование секций в параллельном режиме (Parallel Seq Scan).

Вторая партиция (разбитая по спискам) производит последовательное сканирование секций в параллельном режиме (Parallel Seq Scan) по всем секциям кроме одной – *eco\_catastrophy\_list\_2\_4\_6\_8\_2020*. Напомню, что мы секцию *eco\_catastrophy\_list\_2\_4\_6\_8* разбили на 2 подсекции – *eco\_catastrophy\_list\_2\_4\_6\_8\_2020* (подсекция с данными у которых поле *data\_catastrophe* лежит в пределах 2020 года) и *eco\_catastrophy\_list\_2\_4\_6\_8\_2021* (подсекция с данными у которых поле *data\_catastrophe* лежит в пределах 2021 года). Поэтому т.к. у нас условие найти данные, у которых *data\_catastrophe>'2021-01-01'*, секция *eco\_catastrophy\_list\_2\_4\_6\_8\_2020* с данными за 2020 год отбрасывается, что нам на пользу. Результаты: по затраченному времени (Execution Time) на запрос – 125.872 ms, оценка затратности операции (cost) – 12488.00.

Третья партиция также производит последовательное сканирование секций в параллельном режиме (Parallel Seq Scan) по всем секциям. Результаты: по затраченному времени (Execution Time) на запрос – 131.945 ms, оценка затратности операции (cost) – 14415.61.

Результаты *eco\_catastrophy\_test*: по затраченному времени (Execution Time) на запрос – 118.643 ms, оценка затратности операции (cost) – 13003.52. Обычная таблица по затраченному времени и оценке затратности операций опять уступает партициям (первой (*eco\_catastrophy*) – разбитая по диапазонам дат и второй (*eco\_catastrophy\_list*) – разбитая по спискам), но чуть лучше показывает результат по сравнению с третьей партицией. Из этого следует что таблицы, которые хранят большое количество данных лучше преобразовывать в партиции.