1.1

Created table, filled the table and then running select

```
1  INSERT INTO labs.test_index_plan(num, load_date)
2  SELECT random(), x
3  FROM generate_series('2017-01-01 0:00'::timestamptz,
4   '2021-12-31 23:59:59'::timestamptz, '10 seconds'::interval) x;
5
```

Data Output | Messages | Notifications

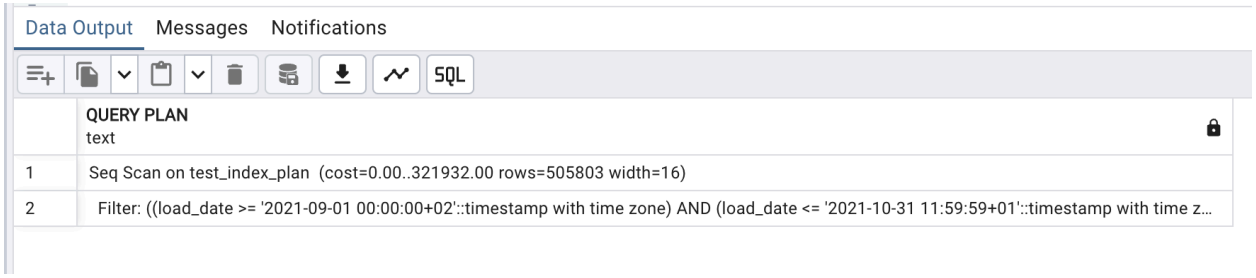
```
INSERT 0 15776640

Query returned successfully in 14 secs 573 msec.
```

```
3  FROM labs.test_index_plan
4  WHERE load_date BETWEEN '2021-09-01 0:00' AND '2021-10-31  11:59:59'
5   ORDER BY 1;
```

Data Output | Messages | Notifications

	num double precision	load_date timestamp with time zone
1	3.7785302797743725e-07	2021-09-18 04:18:50+02
2	1.7337147915075235e-06	2021-09-07 07:42:10+02
3	4.347209249688788e-06	2021-09-02 00:30:40+02
4	9.30310046332039e-06	2021-09-16 17:55:40+02
5	9.484747716115294e-06	2021-10-14 06:25:30+02
6	1.4716955323290648e-05	2021-09-25 02:34:00+02
7	1.6589733947691698e-05	2021-09-14 07:24:10+02
8	1.757896585585783e-05	2021-10-29 15:59:40+02
9	1.854417591284907e-05	2021-09-13 02:04:10+02
10	1.996650084068463e-05	2021-10-07 21:45:20+02
11	2.2806653216189687e-05	2021-10-08 07:12:10+02
12	2.3523807902714466e-05	2021-09-17 03:52:10+02
13	2.4936612992076945e-05	2021-10-03 16:35:20+02
14	2.6178197982229534e-05	2021-09-06 18:24:40+02

With Explain it provides a basic plan without execution



Data Output | Messages | Notifications

SQL

	QUERY PLAN text
1	Seq Scan on test_index_plan (cost=0.00..321932.00 rows=505803 width=16)
2	Filter: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…

With Explain analyze, provides execution details with actual timings.

	QUERY PLAN text
1	Seq Scan on test_index_plan (cost=0.00..321932.00 rows=505803 width=16) (actual time=646.924..690.514 rows=523080 loops=1)
2	Filter: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…
3	Rows Removed by Filter: 15253560
4	Planning Time: 0.066 ms
5	Execution Time: 701.314 ms

With EXPLAIN (ANALYZE, BUFFERS), includes buffer usage details, offering insights into I/O performance.

	QUERY PLAN text
1	Seq Scan on test_index_plan (cost=0.00..321932.00 rows=505803 width=16) (actual time=647.314..694.439 rows=523080 loops=1)
2	Filter: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…
3	Rows Removed by Filter: 15253560
4	Buffers: shared hit=15611 read=69669
5	Planning Time: 0.093 ms
6	Execution Time: 705.936 ms

1.2

Btree index

With explain:

The planner predicts that using the index will significantly reduce the number of rows scanned compared to a sequential scan, The overall cost is reduced, indicating a more efficient plan due to the index.

	QUERY PLAN text
1	Sort (cost=74967.21..76231.70 rows=505798 width=16)
2	Sort Key: num
3	-> Index Scan using idx_load_date_btree on test_index_plan (cost=0.43..18402.40 rows=505798 width=16)
4	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…

With explain analyze, the index scan is much faster than a sequential scan, with actual times showing significant improvement, total execution time confirms the efficiency of using the index.

	QUERY PLAN text
1	Sort (cost=74967.21..76231.70 rows=505798 width=16) (actual time=181.290..215.522 rows=523080 loops=1)
2	Sort Key: num
3	Sort Method: external merge Disk: 13336kB
4	-> Index Scan using idx_load_date_btree on test_index_plan (cost=0.43..18402.40 rows=505798 width=16) (actual time=1.841..66.593 rows=523080 loo…
5	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…
6	Planning Time: 0.133 ms
7	Execution Time: 227.419 ms

Explain analyze, buffers. the majority of the pages were found in shared buffers, indicating efficient caching and use of temporary space for sorting is highlighted by the temp read and written values.

	QUERY PLAN text
1	Index Scan using idx_load_date_btree on test_index_plan (cost=0.43..18402.40 rows=505798 width=16) (actual time=0.028..73.014 rows=523080 loop…
2	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…
3	Buffers: shared hit=4260
4	Planning Time: 0.097 ms
5	Execution Time: 96.384 ms

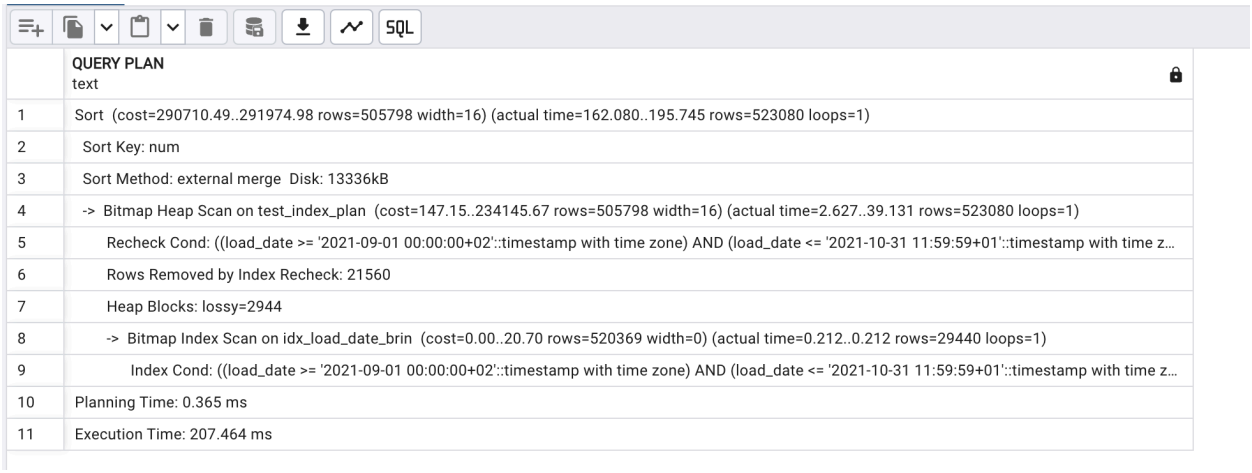
If shared hit is high, PostgreSQL found most data in RAM (faster).

Brin index

With explain

	QUERY PLAN text
1	Sort (cost=290710.49..291974.98 rows=505798 width=16)
2	Sort Key: num
3	-> Bitmap Heap Scan on test_index_plan (cost=147.15..234145.67 rows=505798 width=16)
4	Recheck Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…
5	-> Bitmap Index Scan on idx_load_date_brin (cost=0.00..20.70 rows=520369 width=0)
6	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z…

With explain analyze



	QUERY PLAN text
1	Sort (cost=290710.49..291974.98 rows=505798 width=16) (actual time=162.080..195.745 rows=523080 loops=1)
2	Sort Key: num
3	Sort Method: external merge Disk: 13336kB
4	-> Bitmap Heap Scan on test_index_plan (cost=147.15..234145.67 rows=505798 width=16) (actual time=2.627..39.131 rows=523080 loops=1)
5	Recheck Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z...
6	Rows Removed by Index Recheck: 21560
7	Heap Blocks: lossy=2944
8	-> Bitmap Index Scan on idx_load_date_brin (cost=0.00..20.70 rows=520369 width=0) (actual time=0.212..0.212 rows=29440 loops=1)
9	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z...
10	Planning Time: 0.365 ms
11	Execution Time: 207.464 ms

With explain analyze, buffer

	QUERY PLAN text
1	Sort (cost=290710.49..291974.98 rows=505798 width=16) (actual time=160.108..207.219 rows=523080 loops=1)
2	Sort Key: num
3	Sort Method: external merge Disk: 13336kB
4	Buffers: shared hit=2948, temp read=1667 written=1675
5	-> Bitmap Heap Scan on test_index_plan (cost=147.15..234145.67 rows=505798 width=16) (actual time=2.254..43.169 rows=523080 loops=1)
6	Recheck Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z...
7	Rows Removed by Index Recheck: 21560
8	Heap Blocks: lossy=2944
9	Buffers: shared hit=2948
10	-> Bitmap Index Scan on idx_load_date_brin (cost=0.00..20.70 rows=520369 width=0) (actual time=0.200..0.200 rows=29440 loops=1)
11	Index Cond: ((load_date >= '2021-09-01 00:00:00+02'::timestamp with time zone) AND (load_date <= '2021-10-31 11:59:59+01'::timestamp with time z...
12	Buffers: shared hit=4
13	Planning:
14	Buffers: shared hit=1
15	Planning Time: 0.105 ms
16	Execution Time: 224.225 ms

BRIN is more space it also allows for an Index Scan, which still requires accessing the table (heap) to retrieve the actual rows. Non-zero heap fetches indicate additional I/O operations, which can slow down performance compared to B-Tree. More buffer reads and writes compared to B-Tree, indicating more disk I/O operations. Slightly lower performance compared to B-Tree, but generally it is very effective for large datasets with wide ranges.

2.1

Optimized query:

```
INSERT INTO emp (empno, ename, job, mgr, hiredate) VALUES
   (1, 'SMITH', 'CLERK', 13, '17-DEC-80'),
   (2, 'ALLEN', 'SALESMAN', 6, '20-FEB-81'),
   (3, 'WARD', 'SALESMAN', 6, '22-FEB-81'),
   (4, 'JONES', 'MANAGER', 9, '02-APR-81'),
   (5, 'MARTIN', 'SALESMAN', 6, '28-SEP-81'),
   (6, 'BLAKE', 'MANAGER', 9, '01-MAY-81'),
   (7, 'CLARK', 'MANAGER', 9, '09-JUN-81'),
   (8, 'SCOTT', 'ANALYST', 4, '19-APR-87'),
   (9, 'KING', 'PRESIDENT', NULL, '17-NOV-81'),
   (10, 'TURNER', 'SALESMAN', 6, '08-SEP-81'),
   (11, 'ADAMS', 'CLERK', 8, '23-MAY-87'),
   (12, 'JAMES', 'CLERK', 6, '03-DEC-81'),
   (13, 'FORD', 'ANALYST', 4, '03-DEC-81'),
   (14, 'MILLER', 'CLERK', 7, '23-JAN-82');
```

2.2

```
23  COPY (SELECT num, load_date
24        FROM labs.test_index_plan
25        WHERE load_date BETWEEN '2021-09-01 00:00:00' AND '2021-09-01 11:59:59')
26  TO '/Users/pixel/Desktop/EPAM_STAGE2/test_index_plan_short.csv'
27  WITH DELIMITER ',' CSV HEADER;
28
29
```

Data Output | Messages | Notifications

```
COPY 4320

Query returned successfully in 58 msec.
```

2.3 With this code it is making sure that the emp table is updated correctly without duplicating rows, which helps to maintain data integrity and consistency.

```
INSERT INTO labs.emp (empno, ename, job, mgr, hiredate)
VALUES
    (1,  'SMITH',  'MANAGER',   13, '2021-12-01'),
    (14, 'KELLY',  'CLERK',      1, '2021-12-01'),
    (15, 'HANNAH', 'CLERK',      1, '2021-12-01'),
    (11, 'ADAMS',  'SALESMAN',   8, '2021-12-01'),
    (4,  'JONES',  'ANALYST',    9, '2021-12-01')
ON CONFLICT (empno)
DO UPDATE SET
    ename = EXCLUDED.ename,
    job = EXCLUDED.job,
    mgr = EXCLUDED.mgr,
    hiredate = EXCLUDED.hiredate;
```

Data Output | Messages | Notifications

```
INSERT 0 5
```

	empno [PK] numeric (4)	ename character varying (10)	job character varying (9)	mgr numeric (4)	hiredate date
1	2	ALLEN	SALESMAN	6	1981-02-20
2	3	WARD	SALESMAN	6	1981-02-22
3	5	MARTIN	SALESMAN	6	1981-09-28
4	6	BLAKE	MANAGER	9	1981-05-01
5	7	CLARK	MANAGER	9	1981-06-09
6	8	SCOTT	ANALYST	4	1987-04-19
7	9	KING	PRESIDENT	[null]	1981-11-17
8	10	TURNER	SALESMAN	6	1981-09-08
9	12	JAMES	CLERK	6	1981-12-03
10	13	FORD	ANALYST	4	1981-12-03
11	1	SMITH	MANAGER	13	2021-12-01
12	14	KELLY	CLERK	1	2021-12-01
13	15	HANNAH	CLERK	1	2021-12-01
14	11	ADAMS	SALESMAN	8	2021-12-01
15	4	JONES	ANALYST	9	2021-12-01