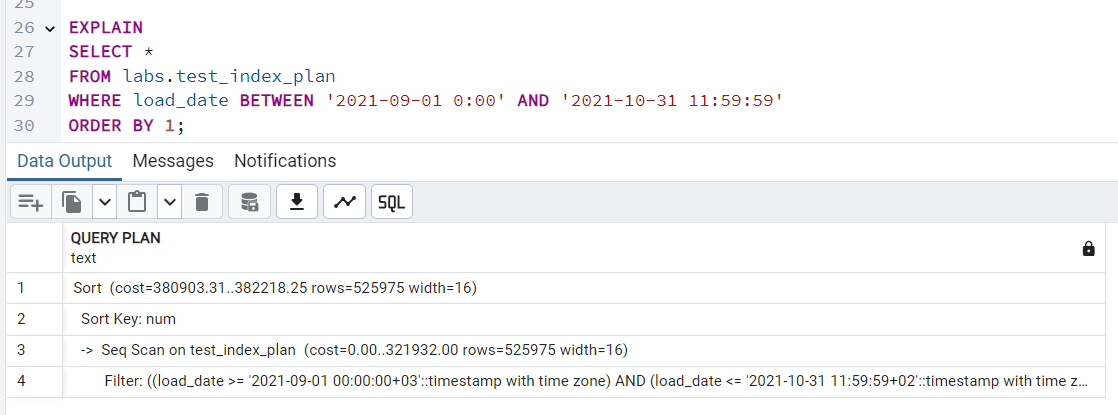
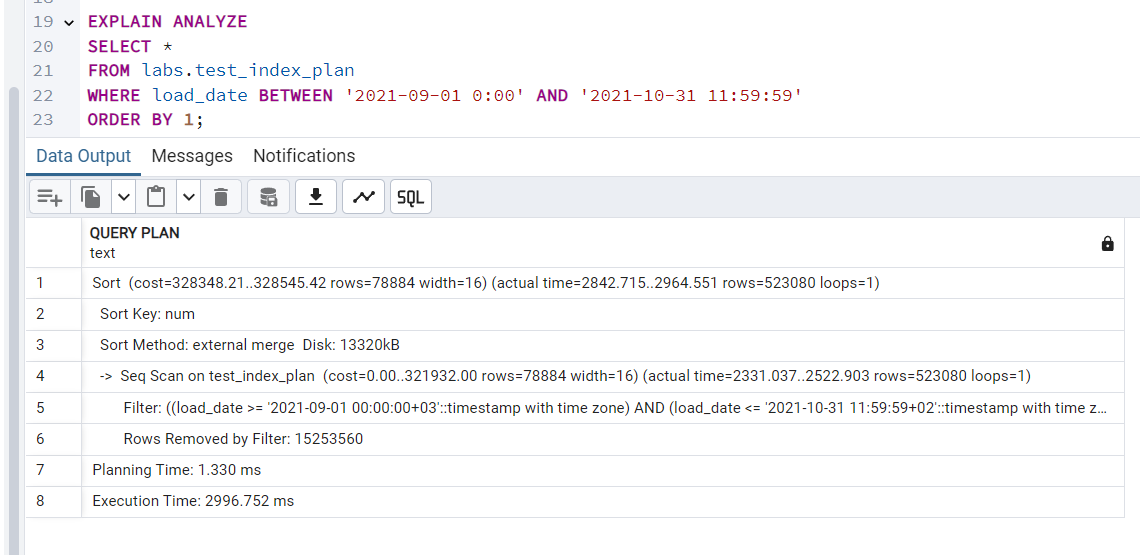
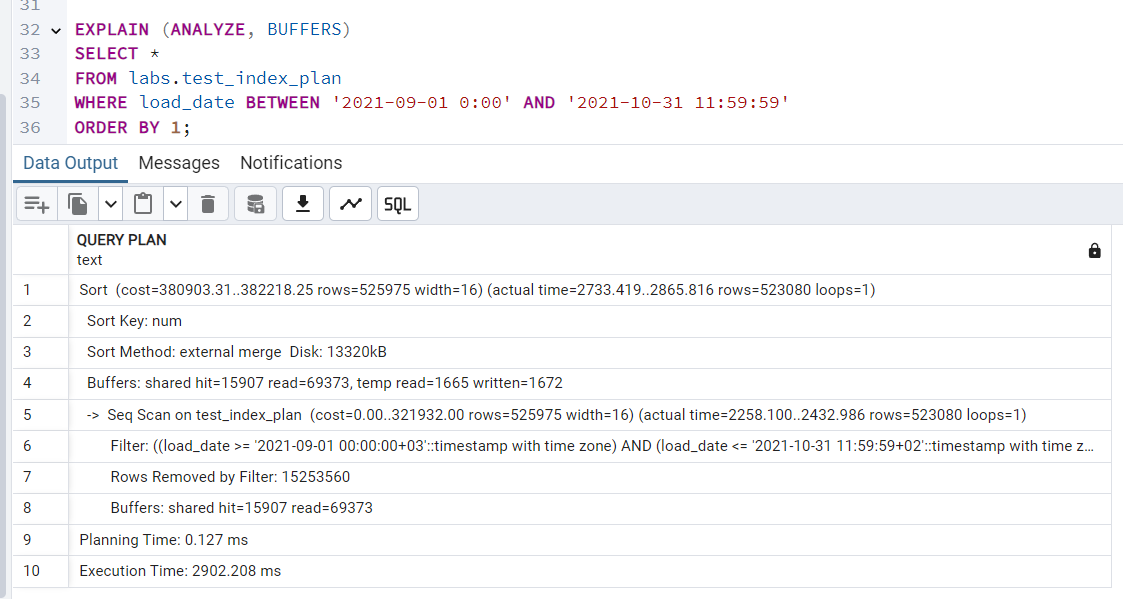
**Task 1.**

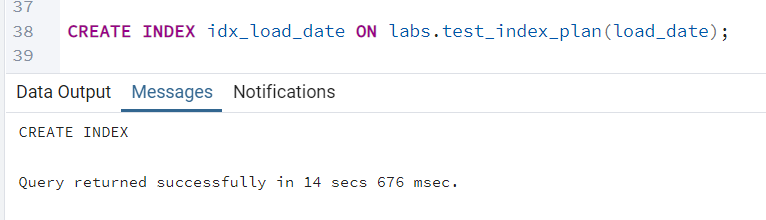
****





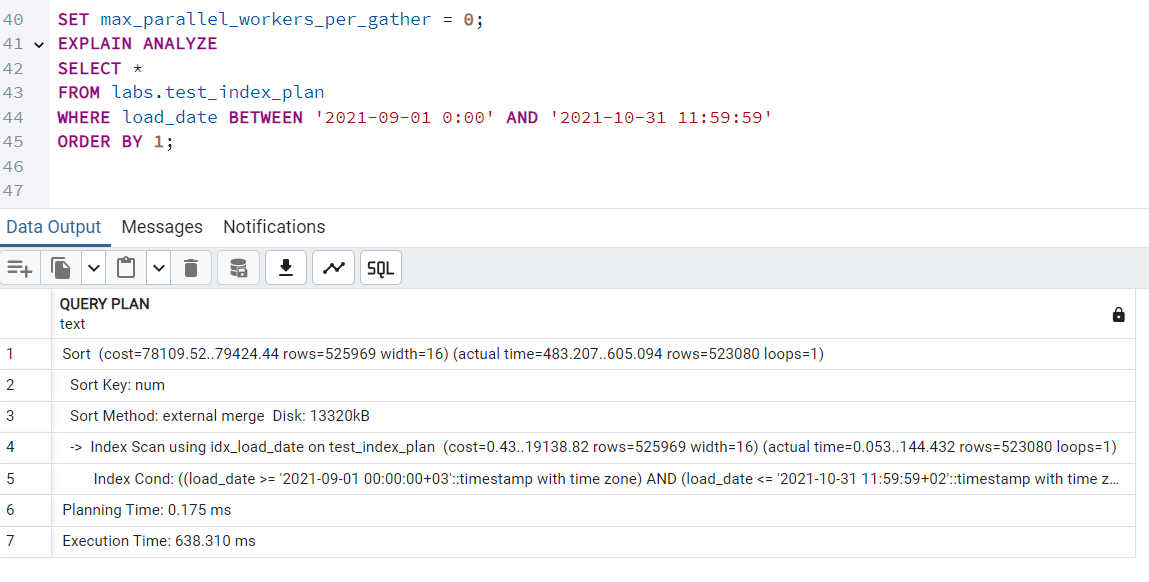
* PostgreSQL performs a sequential scan because there is no index.
* The cost indicates the estimated effort needed for scanning the table.
* First execution is slow because data must be read from disk.
* Second execution is faster because PostgreSQL caches the relevant pages in shared buffers.
* Execution plan structure remains the same (Seq Scan), but buffer usage changes significantly.

**Task 2.**

****

****

* Index Scan replaces Seq Scan → PostgreSQL now uses idx\_load\_date instead of scanning all rows.
* Execution Time Reduced
* Rows Filtered Efficiently: The index allows PostgreSQL to jump directly to relevant rows.



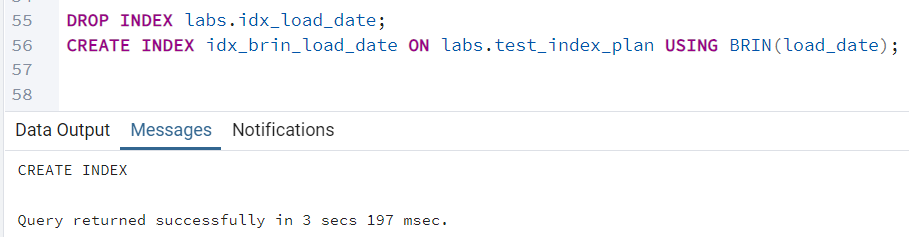
* **Second execution is faster** since PostgreSQL retrieves index pages from memory.

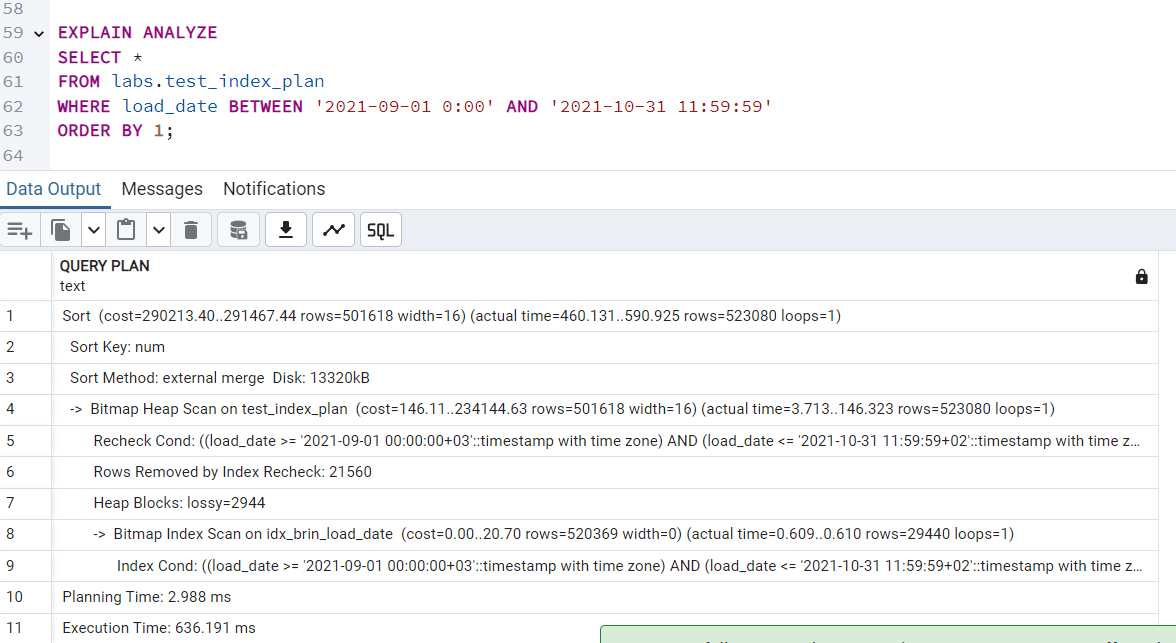
To switch to **Index-Only Scan**, the query must avoid fetching extra columns from the table. **VACUUM ANALYZE** the table so PostgreSQL knows which pages don’t need extra visibility checks.



Why is this faster?

* Heap Fetches = 0: PostgreSQL reads only the index (avoiding full table access).
* Execution Time Drops Further.







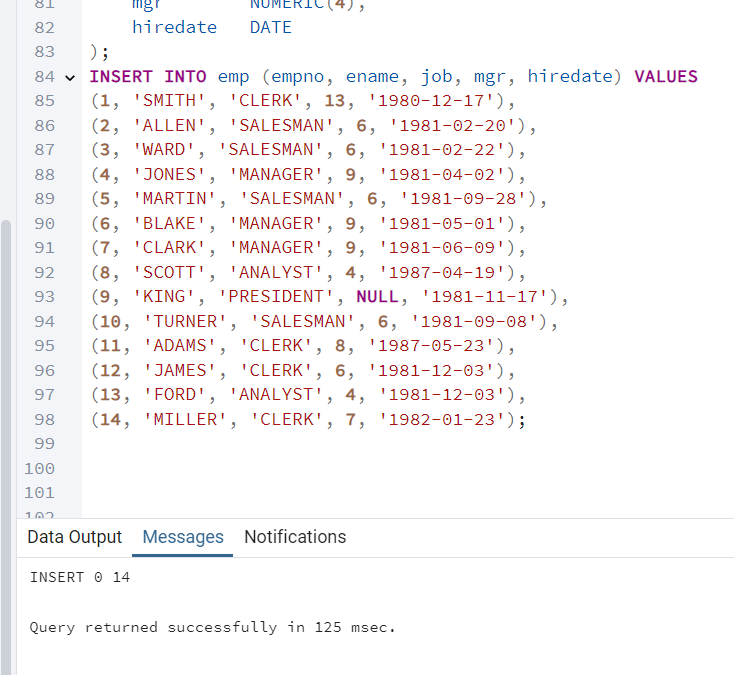
* **BRIN uses a Bitmap Heap Scan** instead of Index Scan because it **retrieves block ranges** instead of individual rows.
* **Ideal for very large datasets** when **full indexing isn't feasible**.

**Task 3.**



What Happens?

* PostgreSQL copies all rows from test\_index\_plan into test\_inserts.
* Since an index exists on load\_date, PostgreSQL will update the index after each insert, which could slow down performance for large datasets.



* **Use Bulk Inserts (**INSERT ... VALUES (...)**)** instead of multiple INSERT statements.

**Task 4.**

The COPY command is the most efficient way to move data between PostgreSQL and files. This task involves exporting data from the test\_index\_plan table into CSV files and importing it back into a new table.

* DELIMITER ',' ensures columns are separated by commas.
* CSV HEADER includes column names as the first row.

****

**Task 5.**

UPSERT is a combination of INSERT and UPDATE. It ensures:

* If a record exists, it gets updated.
* If a record does not exist, it gets inserted.

WITH data\_to\_insert(empno, ename, job, mgr, hiredate) AS (

VALUES

(1, 'SMITH', 'MANAGER', 13, '2021-12-01'::DATE),

(14, 'KELLY', 'CLERK', 1, '2021-12-01'::DATE),

(11, 'ADAMS', 'SALESMAN', 8, '2021-12-01'::DATE),

(4, 'JONES', 'ANALIST', 9, '2021-12-01'::DATE), -- Changed name

(15, 'HANNAH', 'CLERK', 1, '2021-12-01'::DATE) -- Ensure unique ename

)

-- Perform the UPSERT (insert or update on conflict)

INSERT INTO emp (empno, ename, job, mgr, hiredate)

SELECT empno, ename, job, mgr, hiredate

FROM data\_to\_insert

ON CONFLICT (empno)

DO UPDATE

SET ename = EXCLUDED.ename,

job = EXCLUDED.job,

mgr = EXCLUDED.mgr,

hiredate = EXCLUDED.hiredate;

