

Problem statement

Fetch all associated contracts for the Ethereum wallet address:
0xb735bf53abc79525a4f585a004a620d08cc66b27.

Steps

- Before creating an index on the transaction_hash column of the traces table:

Output

The screenshot shows a PostgreSQL query editor interface. The query is a recursive query using the `WITH RECURSIVE` clause to fetch associated contracts for a specific Ethereum wallet address.

```
1 WITH RECURSIVE associated_contracts AS (  
2     -- Base case: fetch contracts deployed directly by the DCW.  
3     SELECT  
4         tr.to_address AS contract_address -- 40 char hexadecimal string for Ethereum as per paper  
5     FROM  
6         transactions t  
7     JOIN  
8         traces tr  
9     /*  
10        * We join transaction table with traces table using transaction hash (64 char hexadecimal  
11        * string for Ethereum as per paper Section 2). There can be multiple traces referencing  
12        * one transaction hash. So, we assume there is another field like trace_id in the trace  
13        * table making a composite primary key: (transaction_hash, trace_id).  
14        */  
15     ON t.hash = tr.transaction_hash  
16     WHERE  
17         t.from_address = '0xb735bf53abc79525a4f585a004a620d08cc66b27' -- 40 char hexadecimal str  
18         AND tr.trace_type IN ('create')
```

The query results are displayed in the Data Output tab, showing a table with the following columns and data:

	contract_address
1	0xc1fde2af03bd15caf2d4a0db8d7e532c381e016
2	0xc45da2f9e5aa77e71fd4bab5b3556ecedf393...
3	0x7b0753f8ec09623d46471fb2c3dcbe498931a8...
4	0xf9530b545bbc92f85b4ab0b76f272f070440eb...
5	0xf7bdc6ace9bdebdfe35f167f6c31c88cf8d9925c

Total rows: 34 Query complete 00:00:09.846

Query Plan using EXPLAIN

```
QUERY PLAN
CTE Scan on associated_contracts (cost=3752262.34..3752265.76 rows=171 width=32)
 CTE associated_contracts
  -> Recursive Union (cost=121885.22..3752262.34 rows=171 width=43)
    -> Gather (cost=121885.22..338811.18 rows=1 width=43)
      Workers Planned: 2
      -> Parallel Hash Join (cost=120885.22..337811.08 rows=1 width=43)
        Hash Cond: ((tr.transaction_hash)::text = (t.hash)::text)
        -> Parallel Seq Scan on traces tr (cost=0.00..216910.14 rows=4190 width=110)
          Filter: ((to_address IS NOT NULL) AND ((trace_type)::text = 'create'::text))
        -> Parallel Hash (cost=120884.47..120884.47 rows=60 width=67)
          -> Parallel Seq Scan on transactions t (cost=0.00..120884.47 rows=60 width=67)
            Filter: ((from_address)::text = '0xb735bf53abc79525a4f585a004a620d08cc66b27'::text)
      -> Hash Join (cost=217962.84..341344.94 rows=17 width=43)
        Hash Cond: ((t_1.from_address)::text = (ac.contract_address)::text)
        -> Gather (cost=217962.51..341317.81 rows=7104 width=86)
          Workers Planned: 2
          -> Parallel Hash Join (cost=216962.51..339607.41 rows=2960 width=86)
            Hash Cond: ((t_1.hash)::text = (tr_1.transaction_hash)::text)
            -> Parallel Seq Scan on transactions t_1 (cost=0.00..119136.38 rows=699238 width=110)
            -> Parallel Hash (cost=216910.14..216910.14 rows=4190 width=110)
              -> Parallel Seq Scan on traces tr_1 (cost=0.00..216910.14 rows=4190 width=110)
                Filter: ((to_address IS NOT NULL) AND ((trace_type)::text = 'create'::text))
          -> Hash (cost=0.20..0.20 rows=10 width=32)
            -> WorkTable Scan on associated_contracts ac (cost=0.00..0.20 rows=10 width=32)
```

• Index creation

The screenshot shows a PostgreSQL client window with three tabs. The active tab is 'postgres/postgres@localhost*'. The interface includes a toolbar with icons for file operations, query execution, and help. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query: `CREATE INDEX idx_traces_transaction_hash ON traces(transaction_hash);`. The query is numbered 1, 2, and 3. At the bottom, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the message 'CREATE INDEX' and 'Query returned successfully in 35 secs 348 msec.'.

postgres/postgres... X postgres/postgres... X postgres/postgres@localhost* X

postgres/postgres@localhost

Query Query History

```
1 CREATE INDEX idx_traces_transaction_hash ON traces(transaction_hash);
2
3
```

Data Output Messages Notifications

CREATE INDEX

Query returned successfully in 35 secs 348 msec.

Total rows: Query complete 00:00:35.348

- After creating an index on the transaction_hash column of the traces table:

Output

postgres/postgre... × postgres/postgres... × postgres/postgres@localhost* ×

postgres/postgres@localhost

Query Query History

```

1 WITH RECURSIVE associated_contracts AS (
2     -- Base case: fetch contracts deployed directly by the DCW.
3     SELECT
4         tr.to_address AS contract_address -- 40 char hexadecimal string for Ethereum as per paper
5     FROM
6         transactions t
7     JOIN
8         traces tr
9     /*
10    * We join transaction table with traces table using transaction hash (64 char hexadecimal
11    * string for Ethereum as per paper Section 2). There can be multiple traces referencing
12    * one transaction hash. So, we assume there is another field like trace_id in the trace
13    * table making a composite primary key: (transaction_hash, trace_id).
14    */
15    ON t.hash = tr.transaction_hash
16 WHERE
17     t.from_address = '0xb735bf53abc79525a4f585a004a620d08cc66b27' -- 40 char hexadecimal str
18     AND tr.trace_type IN ('create')

```

Data Output Messages Notifications

Showing rows: 1 to 34

	contract_address character varying
1	0x39686413825d51cecec10016d55eff7bb4d0ed...
2	0x1a04e4fced95f6feeb35ac3cdfa4406cc58613c9
3	0xf7bdc6ace9bdebdfe35f167f6c31c88cf8d9925c
4	0xc1fde2af03bd15fcdf2d4a0db8d7e532c381e016
5	0x23bf9a3f7db387946dfc9c6420cc08c1880c1f9c

Total rows: 34 Query complete 00:00:02.852

Query Plan using EXPLAIN

```
QUERY PLAN
CTE Scan on associated_contracts (cost=1521392.04..1521395.46 rows=171 width=32)
 CTE associated_contracts
  -> Recursive Union (cost=1000.55..1521392.04 rows=171 width=43)
    -> Gather (cost=1000.55..123123.26 rows=1 width=43)
      Workers Planned: 2
        -> Nested Loop (cost=0.56..122123.16 rows=1 width=43)
          -> Parallel Seq Scan on transactions t (cost=0.00..120884.47 rows=60 width=67)
            Filter: ((from_address)::text = '0xb735bf53abc79525a4f585a004a620d08cc66b27'::text)
          -> Index Scan using idx_traces_transaction_hash on traces tr (cost=0.56..20.63 rows=1 width=110)
            Index Cond: ((transaction_hash)::text = (t.hash)::text)
            Filter: ((to_address IS NOT NULL) AND ((trace_type)::text = 'create'::text))
        -> Nested Loop (cost=0.88..139826.71 rows=17 width=43)
          -> Hash Join (cost=0.33..135258.32 rows=3914 width=67)
            Hash Cond: ((t_1.from_address)::text = (ac.contract_address)::text)
            -> Seq Scan on transactions t_1 (cost=0.00..128925.71 rows=1678171 width=110)
            -> Hash (cost=0.20..0.20 rows=10 width=32)
              -> WorkTable Scan on associated_contracts ac (cost=0.00..0.20 rows=10 width=32)
          -> Index Scan using idx_traces_transaction_hash on traces tr_1 (cost=0.56..1.16 rows=1 width=110)
            Index Cond: ((transaction_hash)::text = (t_1.hash)::text)
            Filter: ((to_address IS NOT NULL) AND ((trace_type)::text = 'create'::text))
```

Observation

We can see that the query time **decreased** from **9.85s** to **2.85s** after creating an index on the transaction_hash column in the traces table. The query plans generated using the EXPLAIN clause confirm the fact that creating the index avoided full table lookups for traces data and led to a **~71% drop** in the query execution time.