# Lineage Querying at Interactive speeds using DuckDB

Sughosh Kaushik
Department of Computer Science, Columbia University

ACM SIGMOD PODS 2022
Philadelphia, PA, USA

## Motivation

**Query lineage for a random tuple in the ouput relation**

```
conn.execute("PRAGMA trace_lineage='OFF'")
conn.execute("PRAGMA lineage_query('SELECT * from Personal_Info WHERE age < 30 ORDER BY age DESC;','VALUE',0)")
print("Lineage of 0th tuple in the ouput is ")
print(conn.fetchall())
```

```
PragmaTraceLineage 0
Lineage of 0th tuple in the ouput is
[(2,)]
```

DuckDB users can query lineage at interactive speeds

## Lineage

**Query Q** : Select * from Personal_Info where ***age < 30***

Personal_Info

| | Name | Age |
|---|---|---|
| 0 | Alice | 25 |
| 1 | Jack | 31 |
| 2 | Bob | 26 |

Result of Q

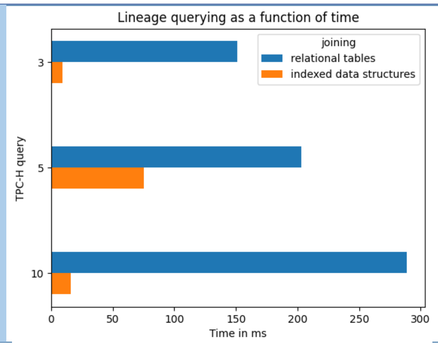
| | Name | Age |
|---|---|---|
| 0 | Alice | 25 |
| 1 | Bob | 26 |

Lineage of Q

| oid | iid |
|---|---|
| 0 | 0 |
| 1 | 2 |

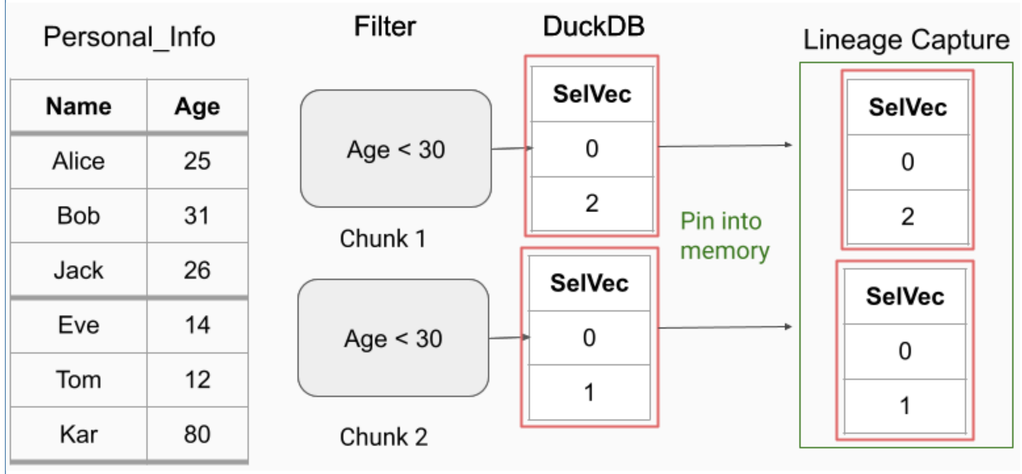
Applications: Data Debugging, Data explanations etc.

## Results



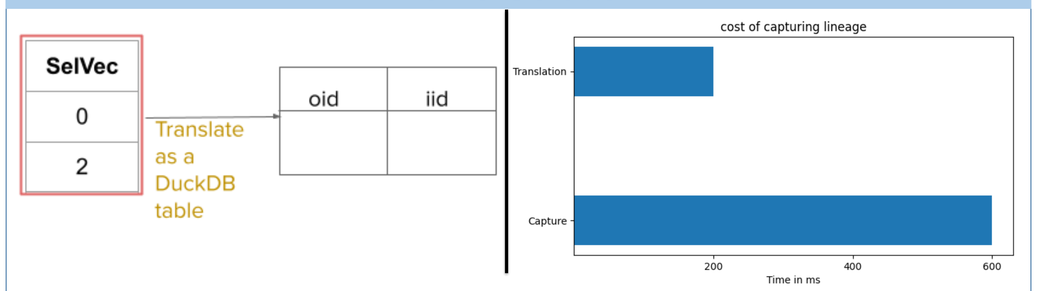

Time taken to query a random output tuple from the given TPC-H queries scale factor 1. The time is compared between joining relational tables and indexed data structures

## Background




Data structures in DuckDB's operator execution encode lineage.

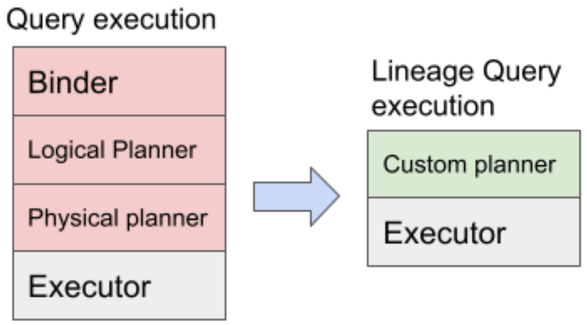
## Naive Solution




Translate the data structure to a relational table and query it using DuckDB's engine. The average cost of lineage capture with translation is published for executing TPC-H queries with a scale factor of 1. The translation is expensive! So we stop it.

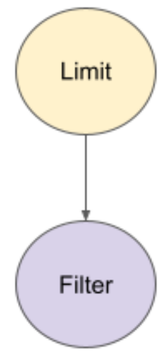
## Main Challenges

### Lineage Query Execution

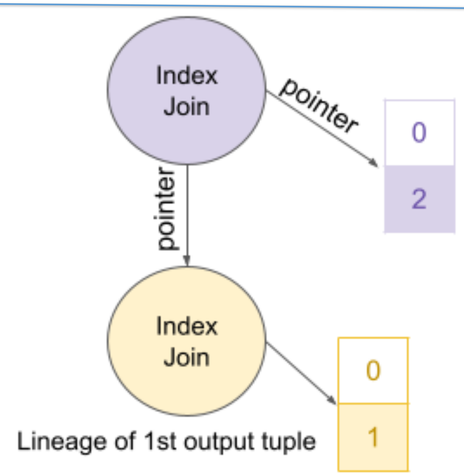



Control flow in DuckDB **2**

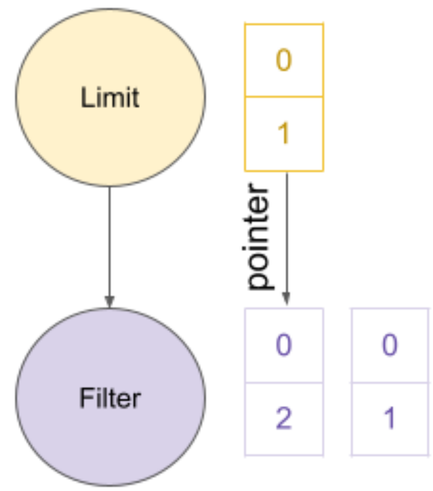
### Base Query Execution




Base query plan




Lineage query plan **3**




Lineage Capture **1**

1. During lineage capture, we build indexes over pinned data structures. The pointer represents an index to access the lineage of the previous operator.
2. Answering lineage queries involves joining operator lineages. We skip DuckDB's planning phase using a PRAGMA function since its unable to generate the optimal join plan. In the definition of PRAGMA, we create a custom plan.
3. The custom plan is built using DuckDB's physical index join operators. The pointer is passed between the join operators, and we see lineage querying in action.