SQL Execution of Smoked Duck

Sughosh Kaushik
Department of Computer Science, Columbia University

ACM SIGMOD PODS 2022
Philadelphia, PA, USA

Introduction

Problem statement:

Computing fine grained provenance at interactive speeds

Q : SELECT * from Personal_Info WHERE age < 30 ORDER BY age DESC; LQ : Lineage(Q, 0);

Personal_Info

Name	Age	
Alice	25	0
Jack	31	1
Bob	26	2

Result of Filter

Name	Age	
Alice	25	0
Bob	26	1

Result of Order By

Name	Age	
Bob	26	0
Alice	25	1

Lineage of Filter

iid	oid
0	0
2	1

⋈

Lineage of Order By

iid	oid
1	0
0	1

DuckDB

Personal_Info

Name	Age
Alice	25
Jack	31
Bob	26

Age

Age
25
31
26

Age < 30

SelVec

SelVec
0
2

Background:

DuckDB is a vectorized execution engine that supports late materialization. It generates data structures during operator execution that encode lineage. Smoked Duck captures lineage by pinning these data structures into memory.

Smoked Duck

Personal_Info

Name	Age
Alice	25
Jack	31
Bob	26

Filter: Age < 30

DuckDB

SelVec
0
2

move

Smoked Duck: Operator Lineage — pinned operator lineage in memory

Expensive costs:

1) Formatting data structure pinned into memory to a relational table.
2) Joining lineage data structures to compute fine grained provenance which entails full scans over lineage data

Operator Lineage — pinned operator lineage in memory

Expensive

oid	iid

lineage_query(base query, oid)

iid	oid

Operator Lineage

⋈ Joins Expensive

iid	oid

Operator Lineage

SQL Execution

Contribution:

The two expensive costs mentioned are resolved using changes proposed by Smoked Duck. The changes are integrated into DuckDB's execution engine as mentioned below. Thus by leveraging DuckDB's vectorized execution engine we compute fine grained provenance at interactive speeds.

lineage_query(base query, oid)

Order By

filter

Base query plan

Order By operator lineage — pointer — filter operator lineage

Maintaining pointer from one operator to its immediate child operator

Lineage of Order By: oid | iid ⋈ Lineage of Filter: oid | iid

To avoid costs of relational table representation join the data structures pinned into memory using a pointer. Embed the pointer into operator state of DuckDB.

Operator Lineage: Artifacts of DuckDB — pinned operator lineage in memory — lineage_query(oid)

PROBLEM

Operator Lineage: Artifacts of DuckDB — Custom Index — pinned operator lineage in memory — lineage_query(oid)

SOLUTION

To avoid full scans use custom index and add these indexes to the generic index interface of DuckDB.

lineage_query(base query, oid)

limit

filter

Base query plan

Index Join — Index Join — oid chunk scan

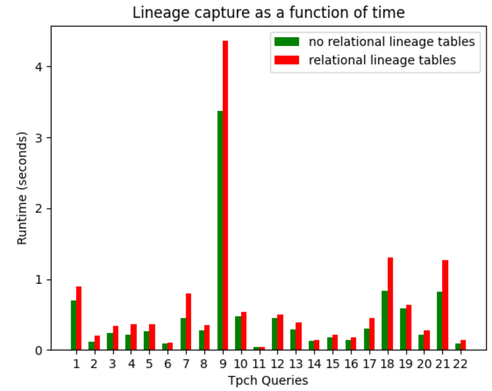
Lineage query plan I

Hash Join — oid chunk scan — Hash Join — scan — scan

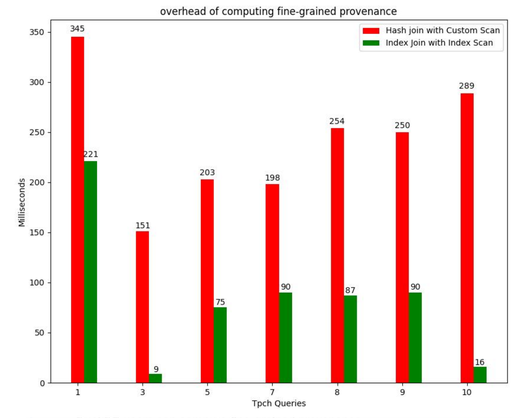
Lineage query plan II

Bypass DuckDB's internal planner and build the optimized plan with the pushed down filter.

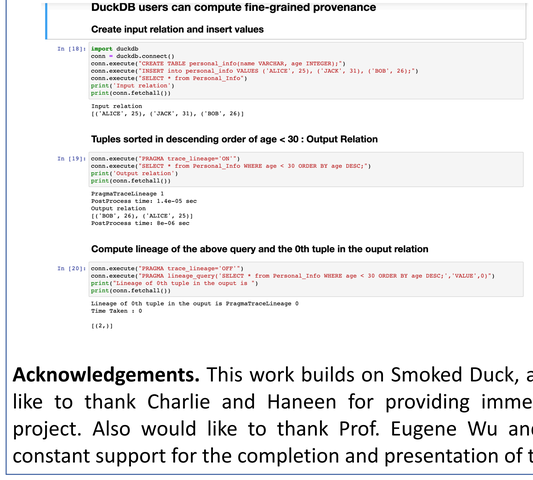
Results



Overhead of capturing fine-grained provenance



Overhead of querying fine-grained provenance



DuckDB users can now query fine-grained provenance

Acknowledgements. This work builds on Smoked Duck, a project in the WuLab. I would like to thank Charlie and Haneen for providing immense guidance throughout this project. Also would like to thank Prof. Eugene Wu and the members of WuLab for constant support for the completion and presentation of the work.