Volcano

Iterators, State, and Batches

Yihao Sun Utah State University Sept. 2026

Three Relations, Five Answers

$$\text{ans}(c, s, a) \leftarrow M(t, d, c),\ P(h, t, s),\ L(h, a, p)$$

$$E = \pi_{\text{Actor,Schedule,Address}}\left(\pi_{\text{Actor,Theater,Schedule}}(M \bowtie P) \bowtie L\right)$$

	Title	Director	Actor
M_0	filmA	Bergman	actor1
M_1	filmA	Bergman	actor2
M_2	filmB	Other	actor3

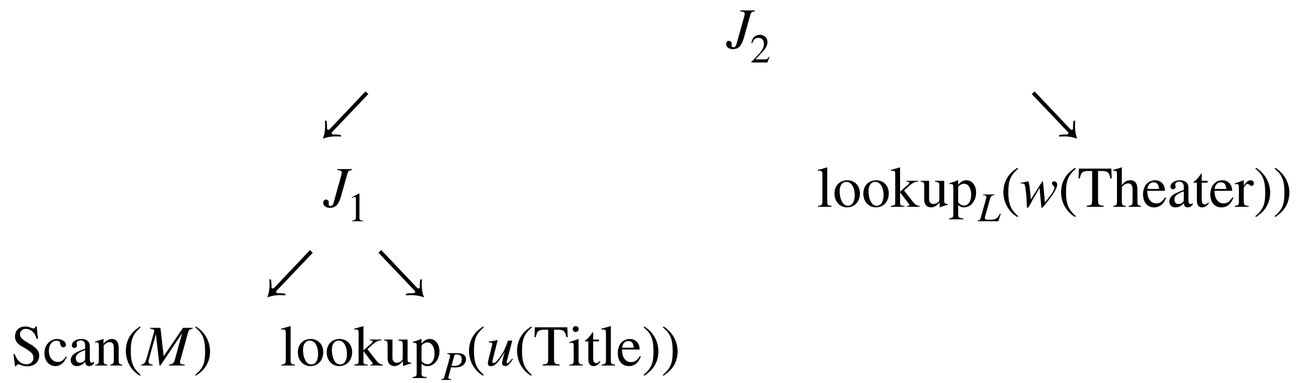
	Theater	Title	Schedule
P_0	Champo	filmA	18:00
P_1	Champo	filmA	21:00
P_2	Rialto	filmB	20:00

	Theater	Address	Phone
L_0	Champo	addr1	phone1
L_1	Rialto	addr2	phone2

For each actor, find the showing times and theater addresses.

An Indexed Nested-Loop Plan

$$E = \pi_{\text{Actor,Schedule,Address}}((M \bowtie P) \bowtie L)$$



u : current Movies tuple w : current result from J_1

Each outer tuple initializes an inner iterator through a keyed lookup.

The Outer Scan Has a State

$R = [r_0, \ldots, r_{n-1}], \qquad n = |R| \qquad$ (fixed input array)

$$S_{\text{scan}} = \{0, \ldots, n\}, \qquad i_0 = 0$$

$$\text{next}_R(i) = \begin{cases} (\text{Some}(r_i), i+1), & i < n, \\ (\text{None}, n), & i = n\,. \end{cases}$$

$$0 \xrightarrow{\text{Some}(M_0)} 1 \xrightarrow{\text{Some}(M_1)} 2 \xrightarrow{\text{Some}(M_2)} 3 \xrightarrow{\text{None}} 3$$

The Movies cursor advances only when the first join needs another outer tuple.

The Iterator Contract

$$\text{Option}(A) = \{\text{None}\} \cup \{\text{Some}(a) \mid a \in A\}$$

$$\text{Iterator}(A) = \exists S \,.\, \Big(s_0 : S,\ \text{next} : S \longrightarrow \text{Option}(A) \times S\Big)$$

Array scan: $S = \{0, \ldots, n\}, \quad s_0 = 0, \quad \text{next} = \text{next}_R$

The caller requests the next result. Each implementation defines its own state.

Open, Next, Close

$$\exists S.\quad \big(\text{open} : \mathbf{1} \to S,\quad \text{next} : S \to \text{Option}(A) \times S,\quad \text{close} : S \to \mathbf{1}\big)$$

Unopened $\xrightarrow{\text{open}}$ Active $\xrightarrow{\text{next}^*}$ Exhausted $\xrightarrow{\text{close}}$ Closed

Active $\xrightarrow{\text{close}}$ Closed (early stop)

$$\begin{aligned}\text{Scan}(M) :\quad & S = \{0,1,2,3\},\quad \text{open}() = 0\\ & \text{next} = \text{next}_M,\quad \text{close}(i) = ()\end{aligned}$$

Exhaustion ends the results. Closing releases the resources.

Join Algorithms, One Interface

Indexed nested-loop join

- Requires: outer stream and keyed lookup
- Retains: outer tuple and match cursor

Hash join

- Requires: two input streams, builds its own table
- Retains: hash table and probe progress

Merge join

- Requires: inputs ordered by the join key
- Retains: input cursors and equal-key group progress

The parent requests results through the same iterator interface.

A Key Initializes an Iterator

$\text{lookup}_P : \text{TitleValue} \longrightarrow \text{Iterator}(\text{Tuple}(P))$

$J_1 = \text{IndexJoin}(\text{Scan}(M), \text{lookup}_P)$

For current u : $c \leftarrow \text{lookup}_P(u(\text{Title}))$
request matches from c until exhausted
For the next u : initialize a new lookup

The keyed lookup is an extra capability required by this join algorithm.

An Initialized Index Lookup

$u = M_0 \qquad \sigma_{\text{Title}=u(\text{Title})}(P) = \{P_0, P_1\}$

$$K = H_P(k), \quad S_k = \{0,\ldots,|K|\}, \quad q_0 = 0$$

$H_P(\text{filmA}) = [0,1]$ matching source row IDs

$$\text{next}_k(q) = \begin{cases} (\text{Some}(P_{K[q]}), q+1), & q < |K| \\ (\text{None}, |K|), & q = |K| \end{cases}$$

The index yields row IDs. This iterator fetches their tuples.

State of the Indexed Joins

$K_i = H_P(M_i(\text{Title}))$ $\quad$ $T_w = H_L(w(\text{Theater}))$

Need : next outer $\quad$ Match : resume lookup $\quad$ Done : end

$$S_1 = \{\text{Need}(p) \mid 0 \le p \le 3\}$$
$$\cup \{\text{Match}(p, M_{p-1}, q) \mid 1 \le p \le 3,\ 0 \le q \le |K_{p-1}|\} \cup \{\text{Done}\}$$
$$s_1^0 = \text{Need}(0)$$

$$S_2 = \{\text{Need}(s) \mid s \in S_1\}$$
$$\cup \{\text{Match}(s, w, r) \mid s \in S_1,\ w \in M \bowtie P,\ 0 \le r \le |T_w|\} \cup \{\text{Done}\}$$
$$s_2^0 = \text{Need}(s_1^0)$$

These records belong to the indexed nested-loop algorithm.

Two Calls Through the Join Tree

$J_2\,.\,\text{next}()$: $J_1\,.\,\text{next}() \rightsquigarrow M_0 \cup P_0$

$c_L \leftarrow \text{lookup}_L(\text{Champo}), \quad c_L\,.\,\text{next}() \rightsquigarrow L_0$

First answer: (actor1,18:00,addr1)

$s_1 = \text{Match}(1, M_0, 1)$ one Title match remains

$s_2 = \text{Match}(s_1, M_0 \cup P_0, 1)$ Theater matches exhausted

Second call: $J_1\,.\,\text{next}() \rightsquigarrow M_0 \cup P_1$; new Champo lookup $\rightsquigarrow L_0$

$s_1' = \text{Match}(1, M_0, 2), \quad s_2' = \text{Match}(s_1', M_0 \cup P_1, 1)$

Second answer: (actor1,21:00,addr1)

J1 resumes its Title matches. J2 opens a Theater lookup for the new tuple.

Cursor in a Sorted Array

$V = \big[(\text{filmA},0),\ (\text{filmA},1),\ (\text{filmB},2)\big]$ (Title, row ID)

$k = \text{filmA}, \qquad [a, b) = [0,2)$

$S_k = \{a, \ldots, b\}, \qquad j_0 = a$

$$\text{next}_k(j) = \begin{cases} (\text{Some}(V_j\,.\,\text{rowID}), j+1), & j < b, \\ (\text{None}, b), & j = b\,. \end{cases}$$

Seek locates the key range. Each next call advances one matching entry.

Cursor in a Hash Bucket

$$h(\text{filmA}) = h(\text{filmB}) = 0$$

$$B = \big[(\text{filmA},0),\ (\text{filmB},2),\ (\text{filmA},1)\big]$$

$$k = \text{filmA}, \qquad S_k = \{0,\ldots,|B|\}, \qquad q_0 = 0$$

$$\text{next}_k(q) = \begin{cases} (\text{None}, |B|), & q = |B|, \\ (\text{Some}(B_q.\text{rowID}), q+1), & q < |B| \wedge B_q.\text{key} = k, \\ \text{next}_k(q+1), & q < |B| \wedge B_q.\text{key} \neq k\,. \end{cases}$$

A bucket supplies candidates. Equality decides which candidates are matches.

A Compact State for the First Join

$$K_i = H_P(M_i(\text{Title}))$$

$$K_0 = K_1 = [0,1], \qquad K_2 = [2]$$

i: current Movies position

j: next offset inside K_i

$$S_{\bowtie} = \{(i,j) \mid 0 \le i < 3,\ 0 \le j \le |K_i|\} \cup \{(3,0)\}$$

$$s_0 = (0,0), \qquad s_{\text{end}} = (3,0)$$

When $i < 3,\ j < |K_i|$: next pair $(M_i, P_{K_i[j]})$

The match offset selects an entry that contains a Pariscope row ID.

Semijoin Tests for a Match

$$P_C = \sigma_{\text{Theater=Champo}}(P), \qquad M \ltimes P_C = \{M_0, M_1\}$$

$$M \ltimes P_C = \pi_{\text{Title,Director,Actor}}(M \bowtie P_C)$$

$$e_i \iff \exists v \in P_C : \; v(\text{Title}) = M_i(\text{Title})$$

$$S_{\ltimes} = \{0,1,2,3\}, \qquad i_0 = 0$$

$$\text{next}_{\ltimes}(i) = \begin{cases} (\text{None},3), & i = 3, \\ (\text{Some}(M_i), i+1), & i < 3 \wedge e_i, \\ \text{next}_{\ltimes}(i+1), & i < 3 \wedge \neg e_i \,. \end{cases}$$

Four join results. Two semijoin results. All three Movies tuples remain stored.

CQ Zoo: A Path with a Condition

$$\text{ans}(z) \leftarrow \text{edge}(\text{Logan}, y),\ \ \text{Lakefront}(y, \text{BearLake}),\ \ \text{edge}(y, z)$$

$$\begin{aligned} \text{Logan} &\longrightarrow \text{GardenCity} \longrightarrow \{\text{Paris}, \text{Montpelier}\} \\ &\longrightarrow \text{Ogden} \longrightarrow \{\text{SLC}\} \\ \text{Lakefront} &= \{(\text{GardenCity}, \text{BearLake})\} \end{aligned}$$

First answer: ? Next neighbor offset: ? Second answer: ?

Which atom acts as a semijoin filter? What state remains after the first answer?

Hash Join Builds Its Lookup

$$H_k(t) = [\, r \mid 0 \le r < k,\ P_r(\text{Title}) = t \,], \qquad 0 \le k \le 3$$

$$H_0(t) = [\ ]$$

$$\text{open}: \quad \text{Build}(k, H_k) \longrightarrow \text{Build}(k+1, H_{k+1}) \quad (k < 3)$$

After open: $s_0 = \text{Probe}(0, 0, H_3)$

$$\underbrace{P \longrightarrow \text{Build}(H)}_{\text{complete build input}} \quad \Big| \quad \underbrace{M \longrightarrow \text{Probe}(H) \longrightarrow J_2}_{\text{produce results on demand}}$$

The build side finishes before this hash join can return its first result.

Why Work on a Batch?

$$\mathrm{Batch}_b(A) = \bigcup_{\ell=1}^{b} A^{\ell}, \qquad b \geq 1$$

$$\mathrm{nextBatch} : S_{\bowtie} \longrightarrow \mathrm{Option}(\mathrm{Batch}_b(A)) \times S_{\bowtie}$$

For N rows, batches filled to b :

Calls across one boundary: $N \longrightarrow \left\lceil \frac{N}{b} \right\rceil$

Key vector $\xrightarrow{\text{tight probe loop}}$ matching position vectors

bounded buffers; retain unfinished matches

Work on arrays inside each call, while keeping intermediate buffers bounded.

A Batch Can End Inside a Join

$b = 3$ Output shown as (Movies row ID, Pariscope row ID)

$B_1 = [(0,0), (0,1), (1,0)]$

$s_1 = (1,1)$ current outer row and next match offset

$$B_2 = [(1,1), (2,2)], \qquad s_2 = (3,0)$$

$$\text{nextBatch}(3,0) = (\text{None}, (3,0))$$

The first batch ends while Movies row 1 still has a match to return.

Filtering a Batch by Positions

Batch position	0	1	2
Theater	Champo	Champo	Rialto
Schedule	18:00	21:00	20:00

Theater = Champo : [true, true, false]

sel = [0,1] positions in this batch

Schedule[sel] = [18:00,21:00]

Pass the surviving positions. The input columns keep their values.

The First Join Produces Positions

$$M.\text{Title} = [\text{filmA}, \text{filmA}, \text{filmB}]$$
$$P.\text{Title} = [\text{filmA}, \text{filmA}, \text{filmB}]$$

Build on P : $H(\text{filmA}) = [0,1]$, $H(\text{filmB}) = [2]$

Movies IDs:	$\underbrace{[0,0,1]}$	$\underbrace{[1,2]}$
	batch 1	batch 2
Pariscope IDs:	$[0,1,0]$	$[1,2]$

Keep both source identities. Pariscope's Theater is needed by the next join.

Positions Through the Second Join

First-join pairs = [(0,0), (0,1), (1,0)]

Theater for these pairs = [Champo, Champo, Champo]

Matching Location IDs = [0,0,0]

Movies IDs = [0,0,1]

Pariscope IDs = [0,1,0]

Location IDs = [0,0,0]

Actor output = [actor1,actor1,actor2]

Schedule output = [18:00,21:00,18:00]

Address output = [addr1,addr1,addr1]

Theater is needed for the second join. Director and Phone can remain unread.

Who Drives the Next Step?

Pull: $\text{Project}\,.\,\text{next}() \longrightarrow J_2\,.\,\text{next}() \longrightarrow J_1\,.\,\text{next}()$

Push: a producer has an output tuple

↓

the consumer processes that tuple

Join algorithm Control direction Batch size Data layout

Which computation pauses, and what must remain in its state?

Cursor and Operator APIs

Operation	LevelDB / RocksDB	PostgreSQL	Velox	DuckDB
Interface	Ordered KV cursor	Tuple plan node	Batch operator	Source / pipeline
Next / advance	Next() (move only)	ExecProcNode()	getOutput()	GetData() (source)
Seek / new key	Seek(key)	IndexScan keys + ExecReScan()	No generic seek	No generic seek
Read output	key(), value()	slot_getattr(slot, …)	batch->childAt(c)	chunk.data[c]
Batch API / unit	No cursor nextBatch	One tuple per call	getOutput / addInput (RowVector)	GetData / Execute / Sink (DataChunk)
End / waiting	Valid(), status()	TupIsNull(slot)	isFinished / isBlocked	FINISHED / BLOCKED (source status)

Batching passes rows together. Vectorization computes over their column vectors.