PGConf.dev 2026, Vancouver, BC

Batching in Executor

Amit Langote

Microsoft

Agenda

- Why batching
- Initial design
- Second Thoughts (about initial design)
- Towards a batch-aware TupleTableSlot
- Beyond batched scans

Per-tuple cost and prior work

Per-tuple costs

- Indirect calls through plan-node and table-AM dispatch
- Data-dependent scan-loop branch
- I-cache pressure from bouncing between nodes

Prior work on per-tuple costs

- Opcode-based expression evaluation (v10)
- JIT compilation (v11)
- Faster tuple deforming (v18)
- Scan inlining (v18)

Storage produces batches

- Heap: 8 KB row-oriented pages
- Column-store TAM: column-oriented page or page groups
- Index access: B-tree leaf pages contain many TIDs
- The executor reaches into storage one tuple at a time
- Consuming those batches would amortize the per-tuple costs

Two approaches to batched execution

- Write a new batch-native, push-based, pipelined column engine alongside the existing executor
  - Years of work, two engines to maintain
- Make the existing executor batch-aware
  - Infra exists: ExecProcNode dispatch, TupleTableSlot, AM callbacks
- This talk: the second

Initial design: TupleBatch between nodes

- TupleBatch (later renamed RowBatch) is a new object type
- Holds N slots of tuples
- Crosses node boundaries via ExecProcNodeBatch()
- Four TAM callbacks
  - scan_begin_batch, scan_getnextbatch, scan_reset_batch, scan_end_batch
  - scan_getnextbatch() replaces scan_getnextslot in the batch path
- Init-time selection among compile-time-specialized ExecSeqScan variants (same shape as v18 scan inlining)
  - To avoid a per-tuple branch on whether batching is active

Prototypes of batched qual and aggregation

- Batched qual
  - ExecQualBatch() over the batch via new EEOPs in ExecInterpExpr; AND-trees of leakproof ops only, per-tuple fallback otherwise
  - One call per batch instead of per tuple; expression eval call overhead amortized
- Batched aggregate transitions
  - Agg consumed batches from SeqScan; e.g. count(*) became count += batch->ntuples per batch
  - Transition function called per batch instead of per tuple
- Prototype microbenchmarks
  - batched-scan ~25%, + batched-qual ~40%, + batched-agg ~50%

First revision

- N slots was the wrong shape: each slot is one tuple's worth of state; N of them is N times that, not a batch
- Replaced by a single re-pointable slot
- New repoint_slot() RowBatchOps entry advances it to each tuple in turn
- Slot contract preserved (heap uses TTSOpsBufferHeapTuple)
- Heap microbenchmarks, fully cached: +11 to +22%

Discomfort with RowBatch

```
struct RowBatchOps
{
    void (*repoint_slot)(RowBatch *b, int pos);
};

struct RowBatch
{
    void           *am_payload;   /* HeapPageBatch, ... */
    RowBatchOps    *ops;          /* heap_repoint_slot(), ... */
    int             nrows;
    int             pos;
    TupleTableSlot *slot;         /* BufferHeapTupleTableSlot for heap */
};
```

- RowBatch is mostly a wrapper: a slot, an ops table, and am_payload
- RowBatchOps is starting to look like TupleTableSlotOps
- Drop RowBatch; the slot wraps the AM's batch directly

The new design

- TableAmRoutine.scan_getnextbatch
  - AM fetches a batch into a TupleTableSlot (previously into a RowBatch)
- To allow batched writes into a TupleTableSlot, the TTS infrastructure adds:
  - TableAmRoutine.batch_slot_callbacks
    - AM returns the TupleTableSlotOps for its batch slot type
  - TupleTableSlotOps.batch_next
    - slot-side cursor advance
- An AM extends TupleTableSlot to hold its batch payload; batch_next lets the existing executor consume tuples one at a time
- Example (heap): BatchBufferHeapTupleTableSlot extends BufferHeapTupleTableSlot, holds visible tuples on a buffer page
- Non-batch AMs leave the new callbacks NULL; per-tuple paths remain

Putting it together

- SeqScan
  - Four batch variants (ExecSeqScanBatch ± WithQual ± WithProject); init-time selection in ExecInitSeqScan (AM supports it, forward scan, not in EPQ, not a system catalog)
  - SeqNextBatch calls slot_batch_next() within a batch; calls table_scan_getnextbatch() when exhausted
  - ExecScanExtended untouched
- Heap
  - BatchBufferHeapTupleTableSlot extends BufferHeapTupleTableSlot; pointer into HeapScanDesc.rs_vistuples[], count, cursor
  - heap_getnextbatch() advances to the next page, calls heap_prepare_pagescan(), hands the array to the slot
  - tts_batch_buffer_heap_get_next(): advances the cursor and returns the next tuple in rs_vistuples
  - Batch unit: visible tuples on one page

Benchmarks

- 15-40% reduction in query time for fully cached heap tables
- Varies with: tuples per page, qual presence, selectivity, all-visible vs not-all-visible

AM fetch costs, before and after

- Both paths: one vtable call per tuple. The work inside the call differs.
- Non-batch (getnextslot)
  - Enter heapgettup_pagemode: loop bookkeeping, lineindex/linesleft, scan key check, struct copy, rs_cindex update
  - Return, then ExecStoreBufferHeapTuple: buffer comparison, flag clearing, tid setting
- Batch (batch_next)
  - Pointer assignment, cursor bump, flag set, return
- Plus one scan_getnextbatch per page (amortized across all tuples on the page)
- Still per tuple either way:
  - Slot deform, qual evaluation, projection
  - ExecProcNode from parent — batches don't cross node boundaries yet

Batching the scan's per-tuple work

- Batched tuple deform
  - AM callback populates per-column arrays in the slot
  - Heap deforms HeapTupleData[] into col_values[][] / col_isnull[][]
  - A columnar AM that already has columnar arrays just exposes them
- Batched qual evaluation
  - Likely shape: an opcode that advances the batch slot's cursor from inside expression eval
  - Qual results accumulated in a bitmask
  - ExecQual advances through tuples without returning to the scan node
- Batched projection
  - Writes per-column arrays into the output slot

Further out

- SIMD vectorization on column arrays once they exist in the slot
- Batches crossing node boundaries: Agg, Sort, Join consuming batches, batched aggregate transitions
- Planner involvement, eventually
  - If batches propagate up the tree, plan choice may need to account for it

Thanks!

Threads on pgsql-hackers:

- "Batching in executor"
- "scan_getnextbatch() and BatchBufferHeapTupleTableSlot” (not yet posted)