Elf
— Multi-Dimensional Tree Structure —

David Broneske, Martin Schäler, Veit Köppen
Multi-Column Selection Predicates

- Single scans for each predicate reasonable
  - No B-Tree would beat it!
- Combined selectivity pretty high
- Case for a multi-dimensional index structure?

TPC-H Query Q6

Q6.1 \( l\text{._shipdate} \geq [\text{DATE}] \) and \( l\text{._shipdate} < [\text{DATE}] + '1' \) year
Q6.2 \( l\text{._discount between [DISCOUNT]} - 0.01 \) and \( [\text{DISCOUNT]} + 0.01 \)
Q6.3 \( l\text{._quantity} < [\text{QUANTITY}] \)
Conceptual Design of Elf [BKSS17]

- Tree-based index structure
- Level = data of one column
- Nodes (DimensionLists) consist of
  - Value
  - Pointer
- Partitioning creates an N-1 dimensional subspace
Elf — Construction

**Procedure:**

1. Sort data on current column $C_i$

2. Create node entry in DimensionList for each distinct value.

3. Repeat from step 1 for each logical partition (subset) with $C_{i+1}$
Elf — Properties

Prefix-Redundancy Elimination

➡ Reduced storage consumption

Ordered Node Entries

➡ Early pruning possible (e.g. C2 = 0)

Fixed Search Depth

➡ Limits hops, no recursion

Optimizations: MonoLists, Linearization, HashMap
Elf Optimizations: MonoList

Column-wise Nodes
Represent data of one column for several tuples

Row-wise Nodes
Represent data of a single tuple across several columns

Solution:
- 1-Element nodes share common suffix
- Create MonoList for all paths w/o branching in underlying dimension
Elf Optimizations: MonoList cont.

- MonoLists half the storage
- 70% storage space of original data

Elf with MonoLists

Result:

- MonoLists half the storage comparison of Elf
- 70% storage space of original data
**Elf Optimizations: Linearization**

**Idea:**

- Transform pointer addresses to array offsets
- Reduces cache misses and storage consumption

**Linearized Elf**

- Pairs of value, pointer
- MSB set at value → List ends
- MSB set at pointer → Child is MonoList

**Figure:** Linearized Elf
Elf Optimizations: Linearization cont.

Elf with MonoLists

Solution - Pre-Order Traversal

Recursively:

1. Create current DimensionList, remember pointer address

2. For each child
   
   1. If child is DimensionList: Go to Step 1
   
   2. If child is MonoList: Write MonoList, propagate pointer, visit next child

3. Propagate pointer of DimensionList to parent
Elf Optimizations: HashMap

Observation — Values in first dimension are

1. **Unique**: Due to prefix-redundancy elimination
2. **Ordered**: Due to ordered DimensionList entries
3. **(Maybe) Dense**: E.g., by dictionary compression

→ Store pointers only!
   (Position of pointer encodes value belonging to it)

Fig. 4. Hash-map property of the first DimensionList

. 7. Optimized memory layout
Elf — Micro Benchmark

Number of Columns vs. Selectivity:

- Dominant impact factor: selectivity
- Threshold SIMDScan vs Elf: 11-18%
Elf - Summary

• Efficient for highly selective workloads
• The more columns are queried, the better the selectivity threshold
• Elf scales linear with a factor smaller than 1

Challenges:
• Complex Predicates
• Insertions
• Column order
References


