

Algorithms for merged indexes

Götz Graefe

HP Labs

Palo Alto, CA

Agenda

- Definition
- Database design
- Value proposition
- Data structures
- Transaction support
- Index utilities
- Query processing
- Prior work
- Conclusions

Definition: merged indexes

- Ordinary B-tree indexes
 - No record-to-record pointers, merely sorted
 - Efficient query processing & updates
 - Key range locking, recovery, loading, etc.
- Master-detail clustering
 - Object roots and components
 - Clustered & non-clustered indexes
 - Hash clustering, multi-dimensional clustering
 - Bitmap indexes, columnar storage

Example sort order

...
Order 4711, Customer “Schmidt”, ...
Order 4711, Line 1, Quantity 3, ...
Order 4711, Line 2, Quantity 1, ...
Order 4711, Line 3, Quantity 9, ...
Order 4712, Customer “Meier”, ...
Order 4712, Line 1, Quantity 1, ...
...

Typical classes of examples

- Complex objects
 - “Customers, orders & details, invoices & details”
 - Multiple clustered indexes
 - An object’s entire information on one disk page
- Many-to-many relationships, join indexes
 - “Students and courses,” “parts and suppliers”
 - Mixed clustered and non-clustered indexes
- Domain indexes
 - One “customer index” for all types of accounts
 - Multiple non-clustered indexes

Database design algorithm

- Define entities & relationships, tables & views
 - Ensure appropriate normal forms
 - No de-normalization for performance
- Define traditional indexes
 - Enable index-to-index navigation as needed
- Merge indexes
 - Cluster complex objects
 - Create domain indexes
 - Reduce I/O in index-to-index navigation
 - Replace de-normalization by merged indexes

Value proposition: clustering

- Less I/O per transaction, less buffer churn
- De-normalization for access performance
 - Match object-oriented applications
- Short-stroking and storage costs
 - Lower I/O rate per data volume
 - More data volume per disk arm
 - Factor 2-4x cost savings for OLTP
 - Reduced space and infrastructure
- Less power, heat, cooling, costs

Value proposition: merged indexes

- De-normalization in physical database design
 - Indexes on base tables & materialized views
 - Clustered & non-clustered indexes
 - Hash clusters, multi-dimensional clusters
 - Bitmap indexes, columnar storage
- Varying count of shared clustering columns
 - E.g., customers, orders, order details
- Add & remove individual indexes anytime
 - Including introduction of new clustering keys
- B-tree code including all optimizations

Data structures

- B-tree supporting binary sort order only
 - Requires full key normalization
 - Simplifies compression, prefix & suffix truncation
- “Domain tags” for clustering key fields
- Index identifier is last column with domain tag
- Sort order = sequence of normalization
 - Clustering fields: domain tag, null bit, data value
 - Index identifier: domain tag, index identifier
 - Data fields: null bit, data value

Record layout

“Customer identifier”	Domain tag
123	Data value
“Order number”	Domain tag
4711	Data value
“Index identifier”	Domain tag
“Orders.orderkey_index”	Identifier value
“2006/12/20”	Data value
“Urgent”	Data value
...	Data values

Example sort order

Customer “Meier”, City “Aachen”, ...
Customer “Meier”, Order 0241, Urgency “Now”, ...
Customer “Meier”, Order 0241, Line 1, ...
Customer “Meier”, Order 5100, Urgency “Now”, ...
Customer “Meier”, Order 5100, Line 1, ...
Customer “Meier”, Order 5100, Line 2, ...
Customer “Meier”, Invoice 52056, RID 0x9fca3846
Customer “Meier”, Invoice 52056, Line 1, RID ...
Customer “Schmidt”, City “Köln”, ...
Customer “Schmidt”, Order 4711, ...

Hash & multi-dimensional indexes

- Computed column as leading key column
 - B-tree sorted on this column
 - Additional columns as appropriate
- Hash indexes
 - Very large interior nodes (MBs) and prefetch
 - Interpolation search on uniform distribution
 - Ghost slots
- Multi-dimensional indexes
 - Z-order Peano curve similar to UB-trees
 - Sufficient to focus query on a bounding box

Bitmap indexes

- Interleaved with other indexes
 - Customer with orders and bitmap of part numbers
- Bitmaps are just another form of compression
 - Record per row → non-unique values repeated
 - Record per unique value → lists of row identifiers
 - Segmentation, compression by differences
 - Bitmap as compression of dense list segments

Columnar storage

- Not non-clustered indexes
 - Primary data store, row order versus key order
- Not vertical partitioning
 - Strict single columns
- “Concatenated index” by index identifier first
 - Storage negligible due to prefix truncation
- Row identifier as second field
 - Negligible due to compression by slot arithmetic
 - Run-length encoding if appropriate
- Row-format delta in the same merged index

Transaction support

- Primarily ordinary B-tree mechanisms
 - Key range locking
 - Insertion and deletion via ghost records
- Object locking
 - Approximated by hierarchical locking
 - Assumes split optimization as for prefix truncation
 - Benefits of lock escalation and de-escalation
- Summary views
 - Increment locks

Locking in bitmap indexes

- Aren't bitmap indexes read-only?
 - Don't need to be – segmentation for fast updates
- Increment locks for bitmaps as summaries
- Ordinary key range locking for bitmaps as compressed lists
 - Data structures protected by latches
 - Logging as in ordinary B-trees

Locking in columnar storage

- Isn't columnar storage read-only?
 - Doesn't need to be – differential row store
- Locking within delta only
 - Only the delta changes ...
 - Large phantom locks
- Locking in both main and delta: overhead
 - Large range locks cover an entire column
 - Small range locks give field-level locking

Deferred maintenance

- Partitioned B-trees for traditional indexes
 - Correct query results by union view
 - Deferred verification of constraints
- Merged indexes permit alternative formats
 - Example: Row delta for columnar storage
- Index reorganization optimizations
 - No log flush on commit
 - Commit & release locks on demand
 - Avoid contents logging with careful buffer writes
 - Exploit many-core, transactional memory, flash

Bulk insertion and deletions

- Delta partitions in partitioned B-trees
 - Incremental representation changes
 - Fast large contents change
- Large insertions
 - Bulk append followed by reorganization
- Large deletions
 - Reorganization followed by bulk removal
- Merged indexes permit data transformation
 - Between temporary and permanent format

Index utilities

- Primarily ordinary B-tree mechanisms
- Index addition and removal
 - Bulk update algorithms
- Online index creation
 - Both “side file” and “no side file” methods apply
- Partial indexes, “instant indexes”
 - Based on partitioned B-trees

Query processing

- Primarily ordinary B-tree mechanisms
- Poor scans of individual indexes
- Fast joins – index nested loops join
 - Techniques to resume B-tree descent
 - Fast verification of foreign key constraints
- Query optimization
 - Modified cost functions
 - Appropriate heuristics for search guidance
 - Probably no new primitive transformations

Prior work

- Härder: Generalized access paths (1978)
 - B-tree record = key plus multiple RID lists
 - Fast verification of foreign key constraints (1996)
- Valduriez: Join indices (1987)
 - B-trees on primary key and foreign keys
- Oracle product
 - Clustered indexes only, fixed clustering key
- Graefe: Executing nested queries (2003)
 - Record layout of merged indexes, use for caching

Other related work

- Hierarchical and network data models
 - Physical data independence
- Object-oriented databases
 - Relying on application for pointers
- Bitmap indexes
- Columnar storage

Orthogonal techniques

- Cache optimizations
- Partitioned B-trees
 - Sort, index creation, bulk insertion & deletion
- Write-optimized B-trees
 - Defragmentation, RAID, flash memory
- Escrow (“increment”) locking
 - High concurrency in indexed summary views
- Hierarchical locking
 - High concurrency in leaves, large range locks

Summary and conclusions

- Master-detail clustering
 - Physical database design, not de-normalization
 - Substantial benefits in cost and performance
- Compatible with relational databases
 - Data definition, transactions, query processing
- Moderate modifications of B-tree algorithms
 - Key normalization using domain tags
 - Scan logic like Tandem's MDAM
 - Index addition & removal efficient by partitioning