Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

BoundGuard: Certifying Data-Skipping Index Upgrades under a Fixed Byte Budget

Tianhao Li

litianh@kean.edu

Wenzhou-Kean University

Research Article

Keywords:

Posted Date: November 27th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11891517/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

The VLDB Journal manuscript No.
(will be inserted by the editor)

BoundGuard: Certifying Data-Skipping Index Upgrades under a Fixed Byte Budget

Tianhao Li

Abstract BoundGuard upgrades an installed data-skipping index within its existing byte budget. On the same immutable file, it certifies at installation that the replacement preserves every incumbent block exclusion for any future positive single-column `IN` predicate. Pooling literals across query boundaries can forfeit an exponential factor in attainable pruning; we prove tight advice-bit bounds for balanced logs, characterize the guarantee by singleton containment, and show that deciding whether an equal-size range replacement can add at least d candidate blocks is NP-complete.

BoundGuard losslessly recodes the incumbent and allocates released bytes to certified range splits or exact key exceptions, checking both serialized images with the same full-domain sweep in 1.8–18.6 ms. Whereas unrestricted joint training loses 67,237 skips on subject-ordered Wikidata, both certified policies add zero candidates; at matched ceilings, splits skip more blocks overall on Owner, Retail, and Wikidata, while each policy uniquely excludes over 1,100 Owner query–block pairs.

On Owner's wide projection, trace replay yields a 1.049× modeled service speedup for exact key exceptions over strong scalar (64 KiB coalescing, four streams). Across 512 held-out Retail baskets, measured wall-time ratios are 1.002× serially and 1.000× with coalescing and four streams (95% CI includes one). Removing an interior candidate block can split a coalesced span into two requests, so candidate containment alone does not determine service time.

T. Li
Department of Computer Science, Wenzhou-Kean University, Wenzhou, Zhejiang, China
Department of Computer Science and Technology, Kean University, Union, NJ, USA
ORCID: 0009-0004-1891-5791
E-mail: litianh@kean.edu

Keywords Data skipping · Physical design · Query workloads · Columnar storage · Cardinality inference

1 Introduction

Consider a column store that has decided to spend its next four index bytes on an exact posting for a popular customer ID. On a request for that ID alone, each false candidate removed by the posting saves a read. On a request for twenty IDs, the same posting saves a read only on a block already ruled out by the other nineteen. The inputs to these two decisions can have exactly the same literal counts. What the counts omit is the boundary around the keys that must be excluded together. We call the set of distinct constants in one positive membership predicate a *query boundary*. A statement may contribute several boundaries.

The difficulty extends beyond estimating the return on four bytes. A designer can assign every observed list its exact loss under a proposed range merge, choose the most profitable merges, and still open substantially more blocks on the next requests. On the subject-ordered Wikidata P31 object column, complete-list training improves the training score over scalar training. On subsequent requests, the resulting index skips 67,237 fewer blocks than the original scalar. A storage operator replacing a sound, deployed index therefore faces two questions. Which associations in the log justify buying precision, and which properties of the replacement can be checked before the workload changes?

BoundGuard upgrades an installed data-skipping index within its existing byte budget (Fig. 1). It losslessly recodes the incumbent and allocates the released space to one of two policies, *certified range splits* or *exact key exceptions*. Both policies produce serialized replacements that the same full-domain verifier checks against the

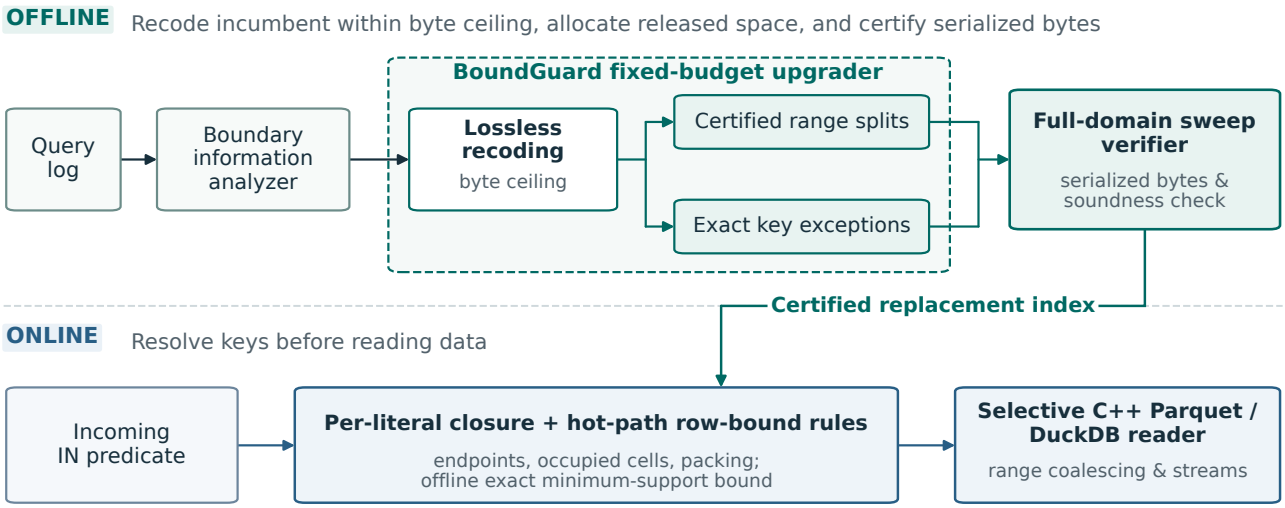


Fig. 1 BoundGuard upgrades an installed index for one immutable file generation. The offline path losslessly recodes the incumbent within its byte ceiling, allocates released space via either certified range splits or exact key exceptions, and checks the serialized replacement over the entire key domain. The online path combines per-literal closure with hot-path row-bound rules before issuing selective reads.

incumbent, including keys that have never appeared in a query. For every future positive membership query on the same immutable file generation, an accepted replacement preserves every incumbent exclusion. Workload probabilities enter only the purchase decision; a new workload can change how often the additional precision pays, while the certificate continues to cover the same file.

This contract separates BoundGuard from prior workload-aware designs. A global range table (GRT) allocates range precision from workload evidence [16], where a fresh allocation can move an exclusion away from a future key. CliffGuard searches for robust physical designs under an explicit workload uncertainty model [34], and budgeted Bloom collections optimize filter accuracy by access probability [32]. Our operator instead names an installed index, keeps its byte ceiling, and certifies predicate-by-predicate containment over all future lists at installation. BoundGuard first obtains a strong scalar incumbent by refining selected intervals, then changes its encoding via variable-length endpoints and shared mask references without altering a single candidate bit. Either allocation policy spends those freed bytes where whole-list scoring finishes block exclusions. Both the spatial ceiling and the read guarantee are relative to the named incumbent; a point dictionary purchased from scratch and a predicate/range hybrid are informative open competitors.

The execution path uses the same unit of reasoning. Given a list, the reader intersects the evidence bearing on each key, then takes the union across keys. A negative answer to a past predicate can remove several singleton possibilities at once. A positive answer requires at least one witnessing key. If the minimum number of witnesses already fills a block's row-count bound, the possible keys are exactly those appearing in a minimum support. Cheap hot-path rules capture nearly all of this gain on short posting lists using a count already paid for by the directory, while an exact solver supplies an offline tightness bound.

Three theoretical results connect the design to the reader. First, on a family of balanced logs, pooling literals across query boundaries can forfeit an exponential factor in worst-case attainable pruning: if each of r roles retains a fraction ρ of its keys, complete pairings buy a skip fraction ρ where pooled frequencies guarantee only ρ^r. We derive exact advice-bit bounds to close that gap. Second, for singleton-union readers, pointwise containment is necessary and sufficient for every future list to keep every incumbent exclusion; checking zero exposure takes one sweep, whereas deciding whether unrestricted exposure reaches a positive threshold is NP-complete. Third, exact exception records interact through shared blocks, so greedy per-byte selection has no constant approximation guarantee even on two lists. The installation certificate therefore guarantees zero regression independently of optimizer approximation quality.

Our evaluation follows the life of an exclusion across four real workloads. A grouped randomization test on saved Stack Overflow SQL holds literal counts fixed while changing training and test lists; fixed-marginal resampling yields $p = 0.0047$ (Holm 0.0235), weakening

to an exploratory tail of 0.0885 when duplicate-list structure is also held fixed. A chronological, customer-disjoint retail trace supplies 512 distinct invoice baskets, a later Wikidata trace tests temporal drift on two physical layouts, and IMDb posting lists isolate row-count inference. Under matched recoded ceilings, certified range splits skip more blocks in aggregate than exact key exceptions on Owner, Retail and Wikidata, while both policies certify zero added candidates and capture distinct query–block exclusions. Finally, a selective C++ Parquet/DuckDB reader measures serial, coalesced and concurrent reads against strong scalar, demonstrating both when logical exclusions speed up service time and when request coalescing splits spans.

Sections 2–4 establish the information bound, installation contract and execution rules. Section 5 defines the units of evidence. Sections 6 and 7 report logical and physical results against their named incumbents.

2 Query boundaries and the price of advice

2.1 The payoff of a complete exclusion

Fix an immutable column partitioned into B disjoint blocks. Let D_b be the set of distinct non-null keys in block b, and Q the finite set supplied to a positive `WHERE x IN` filter. A row passes exactly when its value belongs to Q. Removing repeated constants and null list entries preserves the rows that pass this filter. An empty resulting list selects no row. Write $H(Q) = \{b : D_b \cap Q \neq \varnothing\}$ for the true matching blocks. An index can return any candidate set $C(Q)$ containing $H(Q)$. The unit of benefit is one omitted pair (b, Q), counted with the frequency of that query occurrence.

In a block with exact extrema the exterior of its envelope is free to exclude. Let G_b contain the nonempty maximal gaps between successive distinct keys of D_b. Widening a certified empty interval to its maximal gap costs the same two endpoints and preserves every prior exclusion. A synopsis selects at most k gaps $S \subseteq G_b$. For every query that misses D_b, discard exterior keys and map the remaining literals to the gaps they occupy. This yields the set e of certificates that the query needs. Aggregate equal nonempty requirements into weights w_e. The additional reward above exterior-only pruning is

$$F_b(S) = \sum_{e \neq \varnothing} w_e \, \mathbf{1}\{e \subseteq S\}. \tag{1}$$

This expression credits a completed read once even when several selected gaps contribute. For two-key lists it contains densest k-subgraph: place a singleton gap at each vertex and issue one pair query per edge [21]. With general arities it has the set-union knapsack form [23, 17].

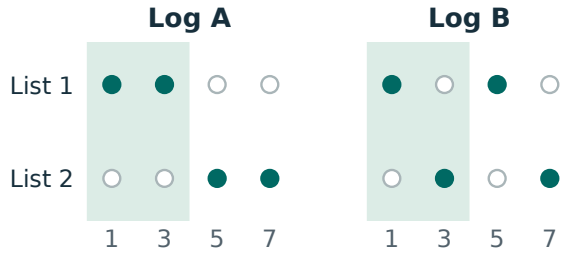


Fig. 2 Equal frequencies, different exclusions. Solid dots mark requested keys. Shading marks the two gaps installed on a block storing the even keys from 0 to 8. The installation skips the first list of log A. Both lists in log B still require a read.

Figure 2 realizes the distinction with one block, four keys and room for two gaps. In log A the pairs are $\{1, 3\}$ and $\{5, 7\}$; in log B they are $\{1, 5\}$ and $\{3, 7\}$. Both logs give frequency one to each key and frequency two to length two. The two installed singleton gaps at 1 and 3 complete one requirement in A. In B each list has one unresolved literal. Replicating every query any number of times leaves the scalar descriptions identical and scales the difference in reads by the replication factor.

2.2 What marginals alone guarantee

Suppose a list has r labelled slots with individual distributions $p_1, \ldots, p_r$. Let A be the keys certified absent by a fixed block synopsis and $a_j = p_j(A)$. The designer knows the marginal distributions but not their coupling. For this model a single design must work against every admissible joint distribution, including ones that repeat a literal in different slots.

Theorem 1 *The minimum skip probability for a fixed synopsis is* $\left[\sum_j a_j - r + 1\right]_+$. *With equal-price disjoint gaps, a deterministic maximin design buys the k largest gap weights* $\sum_j p_j(g)$.

Proof Missing an exclusion in at least one slot has probability at most $\sum_j (1 - a_j)$ by a union bound. To attain that bound up to one, place arcs of lengths $1 - a_j$ consecutively on a circle of circumference one. Their union covers exactly $\min\{1, \sum_j (1 - a_j)\}$. Draw one uniform point. For slot j, condition its draw from p_j on failure if the point is on arc j, and on success otherwise. This preserves each marginal and realizes the stated probability. The certified mass is additive across disjoint gaps, and taking a positive part is monotone. The top weights therefore give an optimal design. □

The probability bound is the Fréchet intersection bound [22]. Its physical interpretation is that a designer using only slot marginals can maximize the worst-case probability of certifying all slots by paying for individual gap mass. A zero lower bound in this expression specifies a compatible adverse grouping. It says nothing about how often a particular recorded grouping completes the same certificates.

There is a corresponding construction when every list has r distinct unordered keys. Let π_x be the inclusion probability of x. Lay intervals of length $\pi_x \le 1$ along $[0, r)$ with all certified keys first. For uniform $U \in [0, 1)$ take the keys hit by $U, U+1, \dots, U+r-1$. The resulting list has distinct keys and the prescribed marginals [30]. Its number of certified keys rounds their total inclusion mass to an adjacent integer. In particular the probability of hitting only certified keys attains the same positive-part expression. Descriptions that preserve more structure, such as a known ordering of slots, restrict the permissible couplings.

If a log supplies separate marginals for each list length, summing a single pooled score can discard even this guarantee. For length class t, let r_t be its arity, n_t its query count, o_t its total exterior literals and u_{tg} its count in gap g. The sharp guaranteed number of reads saved is

$$L(S) = \sum_{t} \left[o_t - (r_t - 1) n_t + \sum_{g \in S} u_{tg} \right]_{+} . \tag{2}$$

A discrete form of the circle construction attains each term using the n_t equally weighted query positions. To optimize (2), maximize over the set of length classes contributing their positive parts and take the largest combined gap weights for that set. The four odd keys in Fig. 2 address a different loss: conditioning on length alone cannot recover the missing pairings when every query has the same length.

2.3 How many bits recover full-information pruning?

Give each of r distinguishable roles m singleton gap values. A hidden matching partitions those keys into m lists, each using one key per role. The lists occur equally often. Such gaps can be placed between consecutive even data keys, so the example describes a realizable block index. Retain selected pairwise projections in the log. Each such projection is a bijection between two roles. Joining along a connected component of the projection graph recovers its m tuples. Isolated components still have an unknown relative matching [8].

Let the c components have sizes $d_1, \dots, d_c$. Covering a_j complete component tuples costs at least $d_j a_j$ gap slots. Under independent uniform relative permutations the expected covered fraction of complete lists is $\prod_j (a_j/m)$. Choosing uniform subsets of those sizes independently achieves that fraction on every fixed matching. Thus the exact randomized maximin fraction without additional advice is

$$\max_{\substack{0 \le a_j \le m,\ a_j \in \mathbb{Z} \\ \sum_j d_j a_j \le k}} \prod_{j=1}^{c} a_j/m. \tag{3}$$

With all roles isolated and $k = rh$, balancing the counts gives $(h/m)^r$. With the matching known the designer can buy the partners of h reference keys and obtain h/m. For fixed $h/m < 1$ the ratio grows exponentially in the arity. Storage capacity is identical in the two cases.

An advice encoder may additionally send one of t messages. Encoder and designer can use public randomness independent of the input. The message may depend on both that randomness and the matching. The criterion is the probability of reaching the optimum on the least favorable fixed matching.

Theorem 2 *For $k = rh$, $1 \le h \le m$, put $A = \binom{m}{h}^{c-1}$. The optimal guaranteed probability of reaching the full-information placement value is $\min(1, t/A)$. Zero error requires and suffices to have A message states. For $h = 1$ the optimal guaranteed expected skip fraction is $\min(1, t/m^{c-1})/m$.*

Proof A placement that saves h complete lists spends all rh available slots. It must pick exactly h tuples from every component. Under independent, uniform relative permutations, the partners of an h-set in the first component form independent uniform h-sets in the others. A fixed placement is optimal with probability at most $1/A$. After fixing the random tape the messages can name at most t placements. A union bound and averaging give a matching on which their success probability is at most $\min(1, t/A)$.

For achievability, fix h reference tuples. Sample without replacement $\min(t, A)$ addresses, each consisting of an h-set in every other component. If the true address appears, the encoder names it. For each fixed matching this succeeds with the claimed probability. Enumerating all addresses makes the protocol deterministic. With $h = 1$, a placement saves either one list or none, so the optimality probability divided by m gives the expected fraction. □

The fixed-width price is $\lceil \log_2 A \rceil$ bits beyond the retained projections. For four isolated roles with eight keys each and four slots, no advice yields $1/4096$. Three bits yield $1/512$, six bits yield $1/64$, and nine bits attain $1/8$. Each state names partners for the keys whose gaps will

actually be installed. By contrast, a synopsis of the log that must score *every* possible four-slot placement must answer whether each cross-component tuple occurs. At $k = r$ these answers identify the entire hidden matching, requiring $(m!)^{c-1}$ states. Advice bits and index bytes price two different objects.

2.4 The expected return from a partial address

When $h > 1$, a placement can miss the optimum while saving many complete lists. Make two partner addresses adjacent when they differ by a single swap within one component's h-set. Suppose t addresses can be chosen so every possible address is within distance one of some chosen address. Then the best guaranteed expected skip fraction is

$$\frac{h - 1 + \min(1, t/A)}{m}. \tag{4}$$

To see the upper bound, let X count complete lists certified by the installation. Since $X \le h$, one has $X \le h - 1 + \mathbf{1}\{X = h\}$. For the lower bound take a covering menu, extend it to $\min(t, A)$ entries and uniformly relabel the tuples within each component. The true address is now uniform for each fixed matching. It is in the menu with probability $\min(1, t/A)$. In the other cases the nearest entry preserves at least $h - 1$ of the reference lists. This proves (4) and specifies what information the encoder must actually transmit.

The cover condition can be checked separately from physical placement. Each closed neighborhood has size $V = 1 + (c - 1)h(m - h)$. Select each address with probability $\ln(V)/V$ and add every uncovered address. The expected number selected after repair is at most $A(1 + \ln V)/V$, so a cover of at most this size exists [4]. A designer with a smaller menu can still quantify its value in particular families. For two equal components with $m = 2h$, offer complementary h-sets of partners for a fixed reference set under shared random relabeling. If Z is its overlap with the first offer, the other saves $h - Z$ lists. The encoder chooses the larger. For $m = 8$ and $h = 4$, a single bit raises the expected fraction from $1/4$ to $11/35$, even though it reaches the full value $1/2$ with probability just $1/35$. A useful log can retain enough grouping to pay for part of the next index without reconstructing every observed list.

3 Certified physical design

3.1 The replacement contract

For each key x, let $T(x)$ be the set of blocks that really contain it. A singleton map $C_R(x)$ for the incumbent R contains $T(x)$. The reader for a finite list takes the bitwise union of its singleton maps. An upgrade U is accepted only if it satisfies the installation contract

$$\forall x,\ T(x) \subseteq C_U(x) \subseteq C_R(x), \quad \text{bytes}(U) \le \text{bytes}(R), \tag{5}$$

over the same immutable file generation, row-group numbering, key encoding and evaluation rule. The left inclusion $T(x) \subseteq C_U(x)$ is *soundness*: an erroneous index that clears a true hit bit could pass containment alone while corrupting SQL answers. The right inclusion $C_U(x) \subseteq C_R(x)$ is *dominance certification*: it preserves every incumbent exclusion on every future list, including keys absent from training or from the file and signed-integer boundary atoms. The byte inequality compares actual serialized wire lengths per file installation (e.g., Retail's 6,377-byte occupied strong-scalar payload inside its 6,464-byte physical slot). Sound indexes may be incomparable, so every claim names the incumbent, generation and evaluation rule.

Theorem 3 *For two sound singleton-union readers on the same immutable blocks the following are equivalent.*

1. *For every key x, $C_U(x) \subseteq C_R(x)$.*
2. *For every finite list Q, $C_U(Q) \subseteq C_R(Q)$.*
3. *For every finite list and every nonnegative vector of block prices, the total candidate cost under U is at most that under R.*

Proof Taking the union of contained sets proves the second statement from the first. Adding nonnegative prices for a subset proves the third. If the first fails, choose a witnessing singleton x and a block in $C_U(x) \setminus C_R(x)$. Assign price one to this block and zero to all others. This violates the third statement on a one-key list. □

The third statement permits a reader to value a large compressed chunk differently from a small one. It also distinguishes true protection from an equal or better total on a benchmark. Losing one large block and gaining several small blocks may improve the unweighted count while increasing the byte bill. Conversely, a zero price on a particular block can conceal a lost exclusion in one observed cost model. In our evaluation we therefore verify not only regressed-query counts, but also total newly admitted query–block pairs $\sum_Q |C_U(Q) \setminus C_R(Q)|$ and per-query maxima.

Checking the bytes to be installed. An integer interval map is piecewise constant. So are a file footer's envelope test and an exact posting that overrides one key. Decode the incumbent and challenger from their serialized byte

strings. Collect every interval start and successor of an interval end in either map; include the beginning and the past-the-end sentinel of the key domain. Add the starts and successors of the shared file extrema and of all point exceptions. Sort and deduplicate the events. On each half-open atom between consecutive events one representative suffices. If a bit of the challenger is set and its incumbent bit is clear, that key and block are a counterexample. Otherwise Theorem 3 certifies every list. A wider integer represents the successor of the largest signed key. A gap between stored keys, including a hole absent throughout the file, takes part in exactly the same sweep.

Soundness and containment have separate checks. For an exact range cell we rebuild its occupancy from the immutable keys; for a point posting we test equality to the file's actual incidence vector. Given a sound construction, the sweep needs only the decoded maps and shared metadata. If u distinct boundary events remain and a block bitmap occupies W machine words, a sorted sweep compares the masks in $O(uW)$ word operations after sorting, with mask construction charged separately. It emits the first counterexample for a rejected installation. The online reader does not pay for the sweep; a changed generation requires fresh supports and a new certificate.

Verifier cost and falsifiable rejections. Table 1 reports measured offline installation and verification timings across all 14 file installations (Owner 5-fold median plus Retail and all eight Wikidata cohort/layout files; common Wikidata archive ingest took 11.121 s and peaked at 464.2 MiB once before per-file builds). All columns exclude offline policy training and purchase selection: Serialize measures encoding the already-selected replacement (1.6–91.4 ms), Decode covers both byte images (0.4–12.6 ms), Soundness checks physical supports (3.5–215.9 ms against reusable per-file supports built in 21.1–1,234.4 ms on first deployment), and the full-domain containment Sweep itself takes only 1.8–18.6 ms across 218–1,720 atoms (bitmap width 3–20 64-bit words). To test rejection paths directly, we take the accepted 6,376-byte Retail range-split image (PASS on 301 atoms against the 6,377-byte ceiling) and construct three controlled failures: (1) setting bit 0 at byte 75 produces an equal-length image rejected by the containment sweep with witness (key 12, block 0); (2) clearing bit 0 at byte 23 produces an equal-length image rejected by the soundness check with witness (key 1, block 0); and (3) supplying the 8,192-row Retail generation identifier with the valid 512-row image triggers the catalog generation mismatch check before decoding.

3.2 Why zero exposure is the tractable threshold

A storage operator may instead ask for the maximum number of added reads on a list of width at most k. Define

$$D_k(U, R) = \max_{|Q| \le k} |C_U(Q) \setminus C_R(Q)|. \tag{6}$$

This quantity charges a newly admitted block even if the replacement saves another block. For $k \ge 1$, it lies between D_1 and $\min(B, kD_1)$. Indeed, each newly admitted block has at least one singleton witness in the replacement and no singleton witness in the incumbent for any member of Q. Checking $D_k = 0$ thus costs one singleton sweep for every width. Computing a nonzero allowance poses a combinatorial choice.

Theorem 4 *Given two equal-byte GRTs with exact occupancy and shared extrema, deciding whether a list can admit at least d new blocks is NP-complete. The integer domain and the file are part of the input. Each singleton can be restricted to at most one added block, and the witnessing keys can all occur in the data.*

Proof Reduce from Independent Set [26]. Use a block for each vertex v of a graph and one common block. Give v a private triple of consecutive integer keys a_v, x_v, b_v. Put a_v in the blocks of its open neighbors $N(v)$ and b_v in the blocks of $N(v) \cup \{v\}$. The common block holds all keys of all triples. Two exterior anchors in every block give shared extrema. On each triple the incumbent stores $[a_v, x_v]$ and $[b_v, b_v]$. Its challenger uses $[a_v, a_v]$ and $[x_v, b_v]$. Their endpoint and bitmap counts agree and their occupancies are exact.

The only differing masks are at the middle keys. Apart from the common block, the old mask for x_v is $N(v)$ and the new one is $N(v) \cup \{v\}$. For a set S of selected middle keys the added blocks correspond to the vertices in $S \setminus \bigcup_{v \in S} N(v)$. These vertices form an independent set; choosing the middle keys of an independent set adds exactly its cardinality. Other keys have equal masks and can only remove elements from the difference. Consequently the maximum is the independence number, or its minimum with k under a width cap. For membership in NP a certificate needs at most one key per elementary domain atom. Verifying its unions and comparing their population counts takes polynomial time. □

For two keys a reader can still construct a precise diagnostic. Let $U_x = C_U(x) \setminus C_R(x)$. The added blocks for $\{x, y\}$ are

$$(U_x \cup U_y) \setminus (C_R(x) \cup C_R(y)).$$

Table 1 Offline installation and verifier cost (median of five pinned single-CPU repetitions after warm-up). Times are in ms and exclude policy training and purchase selection; paired cells report certified range splits / exact key exceptions. Support extraction is reusable across upgrades on the same generation; Serialize measures encoding the selected replacement; Decode covers both serialized images; Sweep measures the full-domain containment scan itself (1.8–18.6 ms); peak RSS is the worker high-water mark across both policies (MiB).

Installation	Groups	Support	Serialize	Decode	Sound.	Sweep	Atoms	RSS (MiB)
Owner (5-fold med.)	1,222	1197.7	65.8/71.6	8.4/8.8	160.6/167.4	18.0/16.9	1,207/1,181	134.4
Retail (202 groups)	202	28.8	6.7/7.7	0.8/0.7	8.3/10.2	2.9/3.0	301/339	96.2
Wikidata P31-obj (Subj.)	469	951.0	19.8/22.1	3.2/3.4	27.6/30.3	8.6/8.6	833/843	140.8
Wikidata P31-obj (Arch.)	469	95.3	14.4/18.9	3.5/3.3	32.5/35.1	10.6/9.1	982/935	96.7
Wikidata P31-subj (Subj.)	469	1228.6	2.0/1.9	1.3/1.3	75.7/73.6	13.0/13.3	1,296/1,292	151.7
Wikidata P31-subj (Arch.)	469	433.2	15.4/33.0	3.7/3.4	81.1/76.9	14.8/15.8	1,513/1,720	108.2
Wikidata P27-obj (Subj.)	140	255.3	3.4/3.9	1.0/1.1	4.9/6.2	3.6/3.8	445/446	91.0
Wikidata P27-obj (Arch.)	140	24.3	1.6/2.1	0.4/0.4	3.5/4.7	2.0/1.8	224/218	76.3
Wikidata P106-obj (Subj.)	135	248.0	2.5/2.7	1.0/1.0	4.6/5.8	3.7/4.0	464/462	88.4
Wikidata P106-obj (Arch.)	135	21.1	2.4/2.6	1.1/1.0	4.7/6.0	4.4/3.9	460/458	75.9

Two singleton exposures in disjoint blocks need not add up, since the other literal may already admit one of those blocks in the incumbent. This identity explains why the positive-threshold problem depends on combinations even though its zero-threshold version factors completely. A verifier certifying the binary install decision can use the tractable threshold.

3.3 Range refinement under the installed byte ceiling

A GRT stores m disjoint, sorted intervals, their endpoints and occupancy over B blocks [16]. With signed 32-bit endpoints and block-major dense rows the baseline payload is $8m + B\lceil m/8 \rceil$ bytes. Per-block extrema are already available. A singleton lookup locates its interval and intersects its occupancy with the extrema test. In the common range construction the training distribution of individual literals selects initial local gaps. Align their endpoints into global cells and merge adjacent cells until the on-wire table fits. The original scalar score charges increases in per-literal occupancy according to frequency.

The boundary analyzer can instead price the immediate loss of a merge on the complete lists observed in training. Let K_q be the blocks currently skipped for q. For an adjacent merge u let N_{uq} be the union of the new singleton candidates within the merged hull for the literals in q, with the block envelope applied. If f_q is the occurrence count, the exact immediate loss is

$$\Delta(u) = \sum_q f_q \, |K_q \cap N_{uq}|. \tag{7}$$

The merged occupancy is the union of the two former masks. Outside the hull the maps agree. Inside the hull only blocks previously skipped can produce a new read, and the intersection in (7) counts each once. Candidate merges can be updated on the training lists with literals in affected cells. Equal primary scores use the scalar cost and deterministic geometric ties. In the controlled comparison the two builders share their training-selected cut limit, geometry and range count.

An exact charge on the observed log is still allowed to coarsen precision. Suppose a block contains 2 and a proposed merge begins to admit 1. No training read is lost on $\{1, 2\}$ because 2 already requires the block. A later $\{1, 3\}$ can be forced to read it. BoundGuard instead uses a trained range table as a *proposal* for subdivision. For each installed interval I and overlapping proposal interval J, find actual keys in $I \cap J$. Drop an empty intersection. For a nonempty one, store the extrema of its actual keys and its exact block occupancy. Every resulting mask is contained in the incumbent mask for I. The construction covers every stored key in I and can never admit a previously excluded key/block pair. Any subset of the local replacements therefore preserves the incumbent.

Looking up the intersection in actual supports matters. A block with keys 1 and 5 occupies both $[1, 3]$ and $[3, 5]$, yet need not contain 3. Intersecting two occupancy bitmaps would retain a false candidate that a fresh occupancy test removes. For a fixed subdivision with one hull and one mask per nonempty cell, the extrema and exact mask just described are the smallest sound choices: every stored key must be covered, and every contributing block must retain its bit. These are local improvements, so a byte-limited installer can choose among them independently while their benefits on complete lists can interact.

The builder serializes each proposal using a dense bitmap, delta-coded occupied or absent block identifiers, or occupied runs, and chooses the shortest exact form. Mode tags, counts, endpoints and the header enter the price. It takes nonpositive byte changes and treats the remaining space as a budget for positive-price splits.

A scalar refinement scores these splits using singleton gain and a finer scalar target built at twice the initial range capacity. A joint target at the same construction capacity supplies a controlled alternative. Construction capacity is transient. The installed index contains only the accepted cells and must fit the incumbent's occupied length. The stronger scalar produced this way is the reference incumbent R for BoundGuard's two allocation policies.

3.4 Lossless recoding and two allocation policies

Let R now denote that strong scalar map. BoundGuard keeps its ranges and re-encodes endpoints as an anchor followed by gaps and spans. Identical occupancy masks can either be inlined or written once and addressed by number. The encoder compares the actual lengths of the two streams. Geometry-only decoding has to recover the same intervals and masks. This creates usable space even if there is no profitable geometric split. A nominal allocation of 32 bytes per block is a ceiling; a strong incumbent occupying fewer bytes sets a correspondingly stricter ceiling for the replacement.

From this identical recoded geometry, BoundGuard supports two allocation policies that both submit their serialized replacements to the same verifier (5):

Certified range splits. This policy spends the released bytes on further interval subdivisions of the recoded strong scalar. Per parent range, it proposes a physical median and cuts around up to eight physically present training keys, storing exact child extrema and group occupancies. It prices proposals by newly completed training exclusions per additional serialized byte, updates gains after each purchase, and reopens the serialized result for the full-domain sweep.

Exact key exceptions. For a purchased set of keys S, this policy defines the installed singleton map by

$$C_{R,S}(x) = \begin{cases} T(x), & x \in S, \\ C_R(x), & x \notin S. \end{cases} \tag{8}$$

Each record pays for the key offset, mode and encoded posting. When shorter it can encode the false-positive mask to remove from the incumbent rather than the blocks to retain. In the primary catalog, considered keys must occur in training, occur physically and have a false candidate under R. An extended catalog also permits a training key absent throughout the file to buy an empty posting. Feasibility follows from $T(x) \subseteq C_{R,S}(x) \subseteq C_R(x)$.

For a false incumbent candidate (b, Q) let the *blocking keys* be

$$E_{bQ} = \{x \in Q : b \in C_R(x)\}. \tag{9}$$

They are exactly the members of the list that have to buy their true posting to eliminate this read, so the exception policy saves the read if and only if $E_{bQ} \subseteq S$. Equal nonempty blocker sets form weighted terms of (1). The frequency control scores a key by its training frequency times eliminated false candidates per record byte; the whole-list rule buys the most newly completed weight per byte after each purchase, breaking ties by singleton benefit per byte and key order. Both policies are finite training heuristics; neither searches all certified replacements, and both obey the exact same installation contract.

3.5 The price of optimizing a protected purchase

Proposition 1 *Under a byte limit, greedy whole-list gain per byte has no positive constant approximation ratio for the additional training reads saved by exact records, even for two distinct lists with physically present keys.*

Proof One concrete range construction uses four blocks with supports

$$\{0, 1, 2, 3, 4, 8, 16, 24, 32, 40\},$$
$$\{0, 4, 8, 16, 24, 32\}, \quad \{50\}, \quad \{60\}.$$

Give $\{1, 2\}$ multiplicity $M \geq 2$ and $\{3, 40\}$ multiplicity one. With one local gap the first block picks the earliest widest zero-frequency gap $[9, 15]$; the second picks $[1, 3]$. Under a six-cell budget, merging $[1, 3]$ and $[4, 8]$ yields the six occupied ranges $[0, 0]$, $[1, 8]$, $[16, 32]$, $[33, 40]$, $[50, 50]$, and $[60, 60]$, occupying $8 \cdot 6 + 4\lceil 6/8 \rceil = 52$ bytes. Its $[1, 8]$ cell admits block two for keys 1, 2 and 3, while the extrema there exclude 40. A lossless 44-byte recoding leaves 8 bytes, enough for two of the three four-byte exact records. Only buying key 3 completes an exclusion initially. Having bought it, greedy can purchase at most one of 1 and 2. Its gain is one. Buying the two complementary records instead saves M reads. Both purchases fit and preserve every incumbent exclusion. Letting M increase proves the claim. □

The instance isolates the economic interaction. The prices of keys 1 and 2 do not change when the other is bought; their return changes. For an exact training optimization, connect records that appear in the same requirement and enumerate the cost–reward frontier

within each connected component. Requirements in different components depend on disjoint records. One byte-budget knapsack over these frontiers therefore gives the global optimum, including the choice of buying nothing. If m_j records lie in a component, a subset transform or direct enumeration computes its attainable rewards [10]. This procedure can be expensive at training time. The installer verifies its output with precisely the same domain sweep used for a greedy result. The NP-completeness result concerns estimating the exposure of an unprotected rival; the greedy counterexample concerns training reward inside the set of protected alternatives. Neither property enters the online correctness argument.

4 Executing from certified block facts

4.1 Closing the evidence per literal

The installed directory supplies a sound starting map for every key. A reader may also retain the outcome of earlier complete predicates on the same file generation. For each stored record, let P be its literal set and $H(P)$ the *exact* set of blocks with a match. A recorded miss on block b means that none of the literals of P is present there. A recorded hit means that at least one is present. The identity of the witness need not have been recorded. A cache of exact query strings would use this information only for a new occurrence of P.

Let $G(x)$ denote the block candidates from exact extrema and, where available, occupancy in disjoint range cells. Without a cardinality bound, the strongest candidate map implied by these facts is

$$M(Q) = \bigcup_{x \in Q} \left(G(x) \cap \bigcap_{\{P:\, x \in P\}} H(P) \right). \tag{10}$$

An empty inner intersection contributes every block. Soundness follows by taking a block with a true witness x. All predicates containing x matched the block and its geometric cell was occupied. For completeness, choose a block on the right with witness x and start from any support consistent with its observations. Insert x into that support. It falls within the known extrema and in an already occupied range. Every recorded predicate containing it was positive already. It alters no negative response. Hence the observations admit a support where the incoming query matches on this block. The completeness statement pertains to these exact observations. Further trusted information can tighten G.

The order of the operators has operational consequences. Suppose a block previously missed $\{11, 17\}$ and $\{23, 31\}$. A later $\{11, 31\}$ can be skipped although neither stored list contains the entire new list. The first miss explains the absence of 11 and the second the absence of 31. Taking an intersection per literal preserves both reasons. Conversely, a past hit on $\{11, 17\}$ by itself cannot attest that 11 occurs. Interpreting that hit as a positive point posting could change the SQL answer. The block-level formulation allows one literal to have many negative witnesses while keeping positive obligations as disjunctions [19,2].

A paid exact key exception can be used before the closure step. For its key a block outside the posting is certainly absent; a block inside it is a true candidate. For every other key the installed map applies. A predicate/range hybrid can spend part of a fixed budget on exact past outcomes and the rest on a GRT. This hybrid exercises (10) directly. The experimental hybrid is an independent access path with its own fitted GRT. Only candidates that pass the sweep against the selected incumbent are termed certified upgrades.

4.2 One spare support position

The number of rows already recorded for a Parquet group or a posting list is an upper bound on the number of distinct non-null keys in it. This seems a weak bound until one asks how many different keys are already required to explain the available positive metadata. On a given nonempty block, let D be the singleton domain surviving the geometric and negative tests. Its exact minimum and maximum are mandatory elements. Each occupied cell and each exact positive predicate imposes an obligation to hit a subset of D. A feasible support contains the mandatory keys, lies in D, and hits each obligation. A conservative zone-map bound can restrict D without creating a mandatory endpoint. A positive Bloom response also creates no obligation.

For $T \subseteq D$, write $s(T)$ for the minimum cardinality of a feasible support containing T. Let $s = s(\varnothing)$ and let d be an available upper bound on support size. The actual distinct keys of the block give a feasible support of size at most d.

Theorem 5 *An admitted key x remains possible exactly when $s(\{x\}) \le d$. If $d > s$, every admitted key remains possible. If $d = s$, the possible keys are the union of all minimum supports.*

Proof A support counted by $s(\{x\})$ realizes all recorded tests and contains x. If it has at most d members it witnesses the possibility; conversely any possible data support establishes the inequality. Inserting an admitted x in a minimum support preserves every observation. Thus $s \le s(\{x\}) \le s + 1$. With slack at least one, every such x qualifies. At equality a support incorporating x must itself have minimum size. □

For an exact row count, a nonempty feasible support can be extended to the required length by repeating an existing value. With an exact non-null distinct count, it can be enlarged by adding other admitted values until the count is reached, assuming the observations are consistent. If $d < s$ the metadata cannot all describe the stated block and generation. Where first and last keys coincide, one mandatory witness suffices. Empty or all-null blocks can be handled by an explicit validity fact before the nonempty analysis.

The theorem distinguishes three physically different situations. A block with spare positions admits every singleton passing (10). With no spare position, an admitted key can still occur if it substitutes for some required witness. Finally, a key may consume a position while satisfying no outstanding obligation that can be dropped; that key forces a support of size $s+1$. A single fixed minimizing support would wrongly reject possible substitutes. The correct set takes the union over *all* minima, or tests $s(\{x\})$ separately for a candidate key.

4.3 Computing a support certificate

The reader can settle many blocks without solving a covering instance. First reserve the distinct extrema. Then count occupied cells with neither endpoint as a witness. Reserving one more witness for each outstanding positive predicate gives a feasible upper bound on s when the obligations are consistent. If even that upper bound leaves a spare position, the cached closure suffices. For individual keys that survive, reserve x and inspect which obligations it and the endpoints already cover. A disjoint packing of the remaining obligations gives a lower bound on the further keys required. If the resulting bound exceeds d, x is impossible. A lower bound below d leaves the possibility open.

An exact residual test is finite even when the integer domain is large. Split the admitted keys into classes according to their occupied cell and the positive predicates they satisfy. Keys in the same class are interchangeable for covering; one representative suffices for a minimum. Include the mandatory elements first and remove obligations they already hit. For each candidate signature A of obligations define $h(U)$ as the minimum additional witnesses needed to cover the outstanding set U. Fix one obligation $i \in U$. Then

$$h(\varnothing) = 0, \qquad h(U) = 1 + \min_{A:\, i \in A} h(U \setminus A). \tag{11}$$

Every feasible support must choose some witness for i, and every term gives a valid first choice. A state with no such signature is infeasible. Due to the cell structure one may instead accumulate attainable masks of positive predicates separately per occupied cell. The larger dependence is exponential in the number of unresolved obligations. The cheap slack and packing rules can precede the exact computation without changing any exclusion. Mandatory reservation of a test literal implements $s(\{x\})$ with the same machinery.

For a directory of movie posting lists, successive directory offsets already give d. A compiled geometric check can sometimes move a decision all the way to construction. Consider disjoint occupancy buckets and a non-endpoint key in an occupied bucket. A realization containing this key needs the two distinct endpoints, the key itself, and a witness for every occupied bucket not yet represented. If that count exceeds the row limit, clear the non-endpoint candidate bit for that bucket, retaining explicit endpoint handling. This is an implication about every member of the bucket. Exact positive predicate obligations, by contrast, can distinguish keys within it. The hierarchy from endpoint count through geometry and packing to exact support gives each additional exclusion a specific justification.

4.4 From a candidate mask to a selective scan

A candidate mask identifies the row groups whose key column the reader must open. BoundGuard enumerates their identifiers in physical order. Its C++17 source reads the selected compressed key chunks through the Parquet interface, decodes the keys and constructs a row-position selection. When the query projects a payload column, the source fetches that column only for groups having at least one selected row. Arrow batches carry the surviving values into DuckDB, which evaluates the remaining SQL operations, including grouping, aggregation and ordering [5]. For a predicate evaluated inside a larger query, this route preserves all passing rows and their multiplicities. Excluding the whole group requires every literal to be settled.

Let K_g be the compressed key bytes in group g, let W_{gc} be its compressed bytes for column c, and let J be the required payload projection. When each selected chunk is requested once the byte count is

$$B(P, Q, J) = \sum_{g \in C_P(Q)} K_g + \sum_{g \in H(Q)} \sum_{c \in J} W_{gc}. \tag{12}$$

There are $|C_P(Q)|$ key requests and $|H(Q)|\,|J|$ payload requests on this path. The second term in (12) depends on real matches and the projection. It is common to sound policies on the specified scan. Thus at a fixed projection a protected subset removes actual key-chunk requests, and a wider projection can dilute its proportional saving. An application can separately observe the

cost of locating the candidates, decompressing the keys, materializing payloads and finishing the SQL computation.

The distinction also specifies the system boundary of our implementation. The timed driver probes a decoded index, passes group IDs to the C++ source, registers the Arrow stream and executes the query through DuckDB. File-open and index-deserialization costs belong to installation or session setup. The posting-list executor implements the support reasoning on a different physical unit, one movie's list. The shared interface between the two is the certified set of candidates; the disk layout determines what opening a candidate entails.

5 Workloads, budgets and measurement

5.1 Sources and holdout units

Each source provides a different connection between a written predicate and the physical object it probes. For Owner the grouping is present in saved SQL. For Online Retail the members of a recorded invoice define the constants of an analytic query. For Wikidata the service log supplies occurrence weights and a chronological split. For IMDb a join produces the key set used to route to candidate movies. We retain these distinctions in both the numerators and the denominators of the results.

Owner. Saved Stack Exchange queries from SEDE and SENLIDB provide positive membership predicates required by a `WHERE` conjunction [24,11]. The primary OwnerUserId comparison uses 32 predicate occurrences from 27 SENLIDB query identifiers, 26 normalized SQL bodies and 24 distinct literal sets. One statement may contribute multiple occurrences. Five folds keep a normalized body and all its occurrences together. Each fold uses the union of the other four for training. A physical projection stores seven columns for 9,904,936 posts from the public 2008, 2014 and 2020 files [41]. Published row order is preserved within the 1,222 experimental groups of up to 8,192 rows. Every full five-fold replay therefore contains $32 \cdot 1,222 = 39,104$ block probes. The same data file serves the narrow and wide projections.

Online Retail. UCI Online Retail records actual invoice identifiers, customers, times, stock codes and quantities [15]. From the 541,909 released lines, remove cancellations and nonpositive quantity rows. Freeze the 103,217 qualifying lines preceding 1 March 2011, preserving their file order. The 3,102 historical stock codes receive ordered integer IDs. A future code outside that dictionary receives a deterministic distinct negative ID, with collisions checked over the observed codes. All policies receive the same encoded relation and literals. For the query workload, an eligible invoice has one known customer, one date and at least two distinct stock codes. Its complete set of codes instantiates a count and quantity-sum query on the historical relation. This retains co-occurrence, including codes absent from the snapshot, under a single SQL template.

Customers are partitioned by a fixed SHA-256 hash modulo four. The March–August training set uses the three nonzero buckets. The September–December test set uses bucket zero. Within each month invoices are ordered by a separate fixed hash of the invoice ID, taking up to 192 training and 128 test invoices. Process training before testing and deduplicate the full canonical list across both. Each selection rule operates on source identifiers, time or list identity, independently of any index result. The resulting 1,152 training lists come from 800 customers; the 512 distinct test lists come from 348 other customers. Test width ranges from two to 529 with median 19. Both the original chronological prefix and the exact invoice-to-query mapping are available for checking the split.

Wikidata. Seven four-week intervals from the public organic query log contain 3,530,948 successful requests [31]. We extract an entity `VALUES` list bound to a direct property in the same outer graph pattern. A request can yield more than one scan input. The first four intervals in 2017 train the design; the three 2018 intervals test it. The physical snapshot contains 20,624,575 entity-to-entity Wikidata5M edges [48,47]. A property and argument side qualifies with at least 20 distinct training sets and 32 full groups. Four cohorts meet the criterion. Together they use 6,090,044 edges and yield 2,690 test occurrences from 1,231 distinct normalized bodies. Each cohort is replayed in archive edge order and in subject/object order. Counting the same occurrence once in each layout gives 5,380 occurrence/layout evaluations.

IMDb. The Join Order Benchmark supplies complete join queries over the IMDb data [29]. Thirty-three occurrences from twelve SQL templates resolve a positive keyword list through the keyword dictionary and the fact relation. They map to ten distinct sets of keyword IDs. A split by connected components keeps a shared complete set and a related template together. The physical relation contains all 4,523,930 movie–keyword facts, routed into 476,794 movie posting lists. A block in this experiment is one entire list; its integral endpoints and row-count bound belong to that directory. A separate SQL join over the fact relation establishes which movies match each query.

5.2 Matched space and independent truth

The original scalar GRT and the unrestricted joint GRT use the same fixed-width range format. The strong scalar uses a finer scalar target and exact, compressed range occupancies. Certified range splits, frequency exceptions and whole-list exact key exceptions all start from this *same* recoded strong map. A standalone point index buys exact key postings and falls back to the common footer on other keys. The Owner hybrid stores complete-predicate evidence and uses the residual bytes for a range table; its online closure can use the same predicate evidence on a new list. A gain over the original GRT measures a deployment decision that starts with that GRT. A gain over the strong scalar measures the remaining value of a certified upgrade after refinement. Each table specifies which comparison it makes.

On Retail all policies are serialized into physically equal-size slots, including a two-byte length field and padding. For 202 groups at 512 rows each, every method occupies 6,464 bytes. For 13 groups at 8,192 rows each, it occupies 416 bytes. The useful payload length is also recorded: in the first layout the raw dense GRT uses 6,384 bytes, the strong scalar 6,377, and both certified range splits and exact key exceptions 6,376. Padding enforces the identical installation price while the strong index supplies the tighter occupied ceiling. We count header lengths, tags, key identifiers and mask references; the shared base column, footer and historic code dictionary are fixed across methods.

Each incoming query is checked separately against source-row truth. On Owner, key and projected payload digests are verified for all three input years. A reference full scan establishes result values and true group masks. Retail tests additionally reconstruct the base rows and invoice lists from the original workbook; the count and quantity sum are checked for both layouts and all methods. The file footer maps each compressed request offset and length back to its group and physical column. A sweep over signed 32-bit atoms checks the containment chain independently of the held-out keys. On Wikidata the stored supports rebuild group incidence; IMDb uses a direct fact-table join and block aggregates by shard. These checks separate soundness, preservation of the incumbent and aggregate speed. For the earlier regrouping, factorial and native-file controls we use the archived summaries; the protection and storage-profile figures use the verified masks and request traces.

5.3 Wall time, service time and uncertainty

The reader measures an application request when a nonempty read of a physical range completes. For a simulated single-stream link, each such read pays a configured time to first byte (TTFB) τ and a transfer term proportional to the returned compressed byte count. The wall-clock mode waits to a monotonic-clock deadline for *each* read. It preserves any excess scheduling delay. Independent harness sessions may advance together, each with its own link allowance. The original three storage profiles service one outstanding range per query; Sect. 7.3 evaluates multi-range coalescing and concurrent streams on the same compressed-byte accounting. The three wide-projection profiles use $(1\text{ ms}, 50\text{ MB/s})$, $(3, 400)$ and $(20, 150)$; the last also runs the key-only projection. These are controlled service parameters. We count data GETs and bytes directly and use decimal MB throughout.

The timed region starts before the index probe and ends after DuckDB has returned all rows and the Arrow stream is unregistered. The footer and decoded index are resident. A separate trace mode retains every physical offset in order. The mean for each regime weights every test occurrence once. In the original three profiles, speedup is the ratio of total original-scalar time to the total time of the compared method on those same occurrences; Sect. 7.3 uses strong scalar as the direct reference. Percentile intervals use 10,000 paired cluster bootstrap samples, with the SQL body as cluster for Owner and the customer for Retail. An entire cluster carries all its method outcomes into a draw. The storage experiment obtains one wall-clock measurement per query, method and limited profile. The intervals describe uncertainty over query composition with fitted policies held fixed.

For the continuous cost curve we combine the median host path for each occurrence with the measured stream of requests. This allows a change in bandwidth or request latency while keeping the bytes and the SQL fixed. After discarding one warm-up repetition, the host calibration runs five serial repetitions; each repetition records both elapsed time and process CPU time, and medians are taken independently over repetitions 1–5 for each projection, occurrence, SQL body and method. Figure 3 distinguishes the calibrated crossover from wall-clock observations on the object-store profile. The relation between a crossing, a measured speedup and a zero-regression guarantee is examined in Sect. 7.

6 Workload transfer and certified pruning

6.1 Owner: isolating the value of a boundary

On held-out Owner bodies the original scalar and unrestricted joint GRTs skip 3,799 and 5,721 of 39,104 probes. Both dense tables occupy 38,580 bytes per installation. The extra 1,922 block exclusions represent a

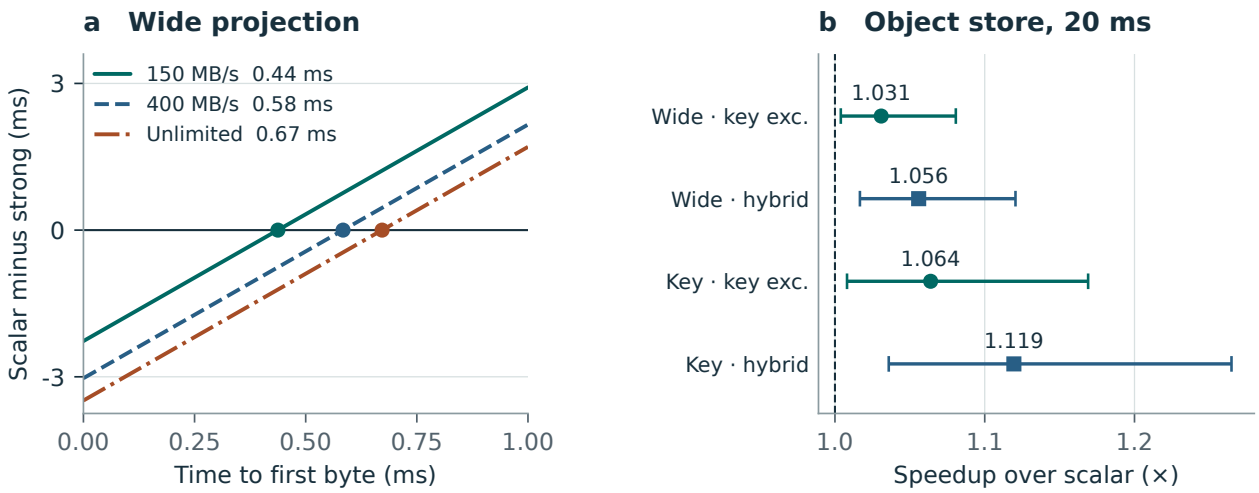


Fig. 3 CPU–I/O crossover and object-store execution on Owner. In (a), lines use measured host time and actual request traces. The zero crossings compare the *strong scalar* to the original scalar at three bandwidths. In (b), points and paired 95% intervals are *wall-clock* speedups at 150 MB/s. Key exc. designates the certified exact-key exception policy. The hybrid uses its own range and predicate budget.

Table 2 Owner grouping controls. The first two statistics are extra held-out joint skips over scalar. The final row fixes the observed test folds and counts absolute joint skips.

Control	Draws	Median	Observed
Disjoint fold margins	9,999	390	1,922
Margins and equality	9,999	1,665	1,922
Training trades, fixed tests	16	3,852.5	5,721

difference of 4.92 percentage points. A whole-list merge has exact training loss given by (7). Attributing the observed gain to the grouping in this corpus requires a reference distribution that preserves the evidence available to a scalar designer. Two fitted index totals alone do not supply that distribution.

Represent each fold by a binary matrix, with one row per occurrence and one column per literal. A Curveball trade changes two rows by uniformly redistributing their exclusive keys while retaining their intersection and lengths [43,12]. Every traded table has the same arity per row and frequency per key. The initial geometry, 64 local cuts, stored blocks and byte ceiling are fixed. A reversible hub-and-spoke randomization test places the observed table and simulated spokes on the same footing under the fixed-margin null [9]. Its exchangeability follows from reversibility and the common hub, without assuming that a particular walk length produces an independent uniform table.

In the coherent test, randomize the five disjoint fold tables first, then reconstruct each training complement and test on its corresponding randomized fold. The statistic is the sum of joint-minus-scalar skips over all five held-out folds. The fixed-marginal Curveball test yields $p = 0.0047$ (Holm 0.0235), compared with an exploratory tail proportion of 0.0885 when duplicate-list structure is also fixed. This is evidence that grouping changes the relative payoff of the fitted designs on these folds, while the signal weakens under the stricter conditioning. The archived whole-log control reports a median of 390 and a central 95% interval of $[-647, 1, 537]$. Among 9,999 null draws, 46 attain or exceed the observed 1,922, giving $p = (46 + 1)/10,000 = 0.0047$. A second family fixes the observed test inputs and randomizes each training table. Its one-sided fold probabilities are 0.2320, 0.1824, 0.0001, 0.0086 and 0.0083. Holm adjustment across that family and the coherent test gives 0.0235 for the coherent test and 0.3648, 0.3648, 0.0006, 0.0332 and 0.0332 for the folds. Overlapping training complements are not independent observations of a pooled statistic.

A stricter conditioning choice preserves the full 32 by 32 equality relation between occurrence lists. Distinct lists may be traded only within equal fold-multiplicity signatures, and proposals creating a new equality are rejected. This preserves arities, margins and repeat structure while allowing 18 of 24 distinct lists to move. In 9,999 coherent draws the central interval becomes $[1, 367, 2, 237]$, with 884 draws reaching 1,922 (exploratory rank 0.0885). This measures the contribution

Table 3 Owner 2×2 refinement ablation. Both axes are construction choices. All four alternatives protect the original scalar GRT and use the same packed format. Entries count held-out skips.

Source of finer cuts	Singleton score	Whole-list score
Scalar target	3,965	5,165
Joint target	3,988	5,844

Table 4 Owner body holdout at a nominal 32 B/group. The contract column names the protected map. Key exceptions designate BoundGuard's exact-key policy; open designs have no containment relation to strong scalar.

Physical design	Contract	Skips
Original scalar GRT	Incumbent	3,799
Unrestricted joint GRT	Open	5,721
Strong scalar	Original scalar	3,965
Frequency exceptions	Strong scalar	5,547
Key exceptions, present	Strong scalar	5,939
Key exceptions, extended	Strong scalar	6,064
Predicate/range hybrid	Open	7,533

of precise grouping after the occurrence-equivalence pattern has already been disclosed. In the independently tabulated 16-seed fixed-test control, training trades yield 2,629 to 4,698 absolute skips. Table 2 distinguishes that measure from the two coherent differences.

Boundaries can change the cuts proposed to the installer or which cuts it buys. The factorial control crosses the source of the finer target with the score used to purchase a subdivision (Table 3). With the scalar target frozen, whole-list scoring raises the held-out count by 1,200. Changing the target while retaining singleton scoring raises it by 23. When both decisions use boundaries the improvement over the scalar/scalar arm is 1,879. All four plans preserve the original scalar. A higher aggregate than the scalar/scalar arm does not establish containment of that stronger arm. Table 4 explicitly installs the latter requirement.

The strong scalar skips 3,965 probes. Buying present-key exceptions by frequency raises the value to 5,547. Pricing complete exclusions raises it to 5,939, a gain of 1,974 over the protected map and 392 over the frequency control on the same catalog. Allowing present and absent training keys to buy priced records raises the whole-list total to 6,064. An absent key is certified only by the ordinary geometry or a paid record. The open hybrid obtains 7,533 skips by allocating space between predicate implications and its own ranges. It is an attractive latency comparator for a store free to replace its access path; some of its queries admit extra blocks relative to strong scalar.

The five strong scalar installations use 192,604 bytes in total. Lossless recoding reduces that to 182,907. The present-key whole-list installations use 192,520 bytes in total and buy 280 key records across the five files. These are alternative trained installations, not five copies needed by one reader. The domain sweeps cover 6,933 signed-key atoms altogether. Each passes the zero-addition test. Thus the 1,974 observed gain is compatible with a stronger statement: no other list issued against the corresponding generation can turn an incumbent exclusion into a required block.

6.2 Online Retail: 512 customer-disjoint baskets

The retail trace compares six equal-byte designs on two physical groupings of the same time-ordered relation. A planner buying metadata for the historical snapshot sees the training invoices only. Evaluation uses distinct baskets from different customers and later months. All methods receive each basket in full, including keys outside the historical catalog. With 512-row groups there are 103,424 query/group opportunities. With 8,192-row groups there are 6,656. A regressed query in Table 5 is one that requires more candidate groups than strong scalar.

At the finer granularity exact key exceptions skip 3,227 groups, the largest count among the six original policies in this table, against 2,892 for strong scalar, 2,614 for unrestricted joint and 2,515 for standalone point postings. Both open alternatives regress on 40 queries. Equal regression counts conceal different candidate changes: joint admits 680 formerly excluded blocks across the held-out queries (up to 110 on one invoice), and points admit 441. The two policies also save other blocks, so those numbers differ from their net deficits. Key exceptions add zero candidate blocks, save 335 in total and remove 335 key GETs. A workload summary therefore benefits from reporting both net reads and the direction of every changed relation (Fig. 4).

The saved compressed input is 540,920 bytes relative to strong scalar, or 0.266% of key plus payload traffic. A paired customer-cluster interval for the percentage is $[0.035, 0.592]\%$. Counts and quantities match the original workbook on every test. On the coarser layout key exceptions skip 138 groups against 130 for strong scalar. They omit eight key requests and 118,367 bytes, or 0.092% of the strong scalar total, with customer-cluster interval $[0, 0.212]\%$. Larger groups reduce the number of individually excludable units while giving each changed decision a larger compressed byte value. The certificate itself does not depend on the scale: all 3,104 elementary signed-key atoms pass the independent containment test for each layout.

Table 5 UCI Online Retail over 512 held-out baskets. Both layouts charge exactly 32 B/group for every policy. MB are compressed key and payload bytes in decimal millions. Worse counts queries opening more groups than strong scalar. Key exceptions denote BoundGuard's present-key policy.

	512 rows/group, 6,464 index bytes				8,192 rows/group, 416 index bytes			
Physical design	Skips	MB	GETs	Worse	Skips	MB	GETs	Worse
Original scalar	2,886	203.557	192,835	2	130	129.158	13,015	0
Unrestricted joint	2,614	204.007	193,107	40	130	129.158	13,015	0
Strong scalar	2,892	203.547	192,829	0	130	129.158	13,015	0
Frequency exceptions	2,963	203.432	192,758	0	136	129.065	13,009	0
Key exceptions	3,227	203.006	192,494	0	138	129.039	13,007	0
Point index	2,515	204.161	193,206	40	136	129.065	13,009	0

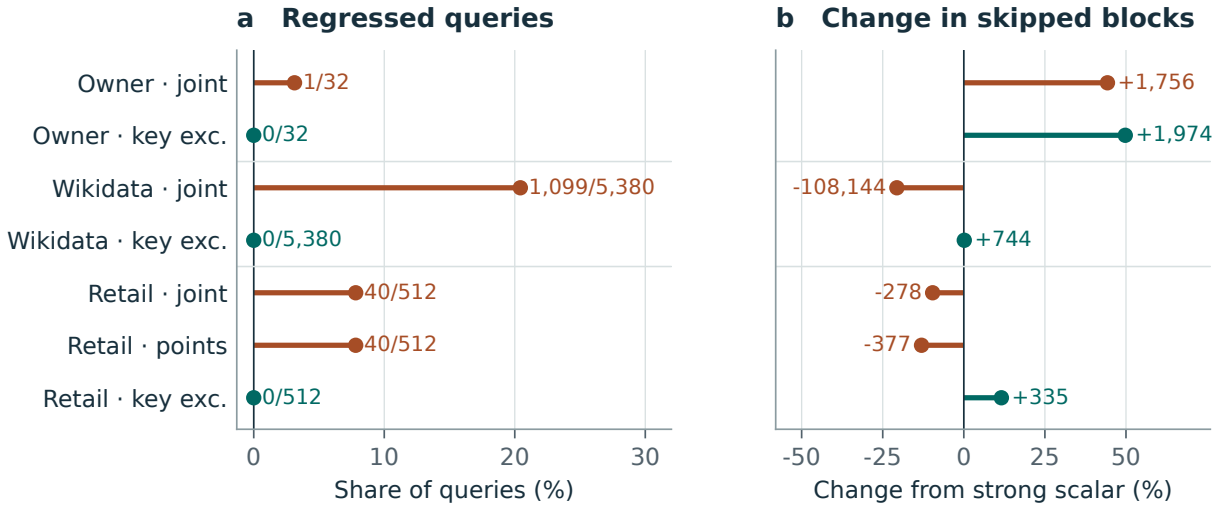


Fig. 4 Both panels compare with the *strong scalar* for each workload. Left, a regressed query has more candidate blocks; labels give the numerator and denominator. Right, signed net extra skips are printed beside the points, whose position is relative to incumbent skips. Wikidata counts each request once per layout; Retail uses 512-row groups. Key exc. designates BoundGuard's present-key exception policy.

6.3 Wikidata: protecting the later workload

P31-object supplies a chronological test of learned range geometry. With subject ordering, the original scalar skips 150,593 training probes and joint skips 156,983. The 1,203 later occurrences yield 564,207 group probes. Scalar skips 209,018 groups, versus 141,781 for joint. The deficit of 67,237 is relative to the *original* scalar. Both maps contain every true hit. The difference is the number of false candidates that must be examined. The same requests in archive edge order give 205,452 and 205,480. Holding the requests fixed and changing the row order separates exposure to drift from physical locality.

Strong scalar skips 239,861 groups in subject order. Its 327 cells occupy 14,905 bytes versus 14,924 for the dense original map. Lossless recoding occupies 13,571 bytes. Exact key exceptions buy 69 present-key records, use 14,899 bytes and skip 240,061 groups here. In archive order they add 440 exclusions above that layout's strong scalar (Table 6), and P31-subject in archive order supplies another 104, summing to 744. Across all four cohorts and both orders strong scalar skips 523,933 and key exceptions 524,677 (frequency control 524,558) over 1,585,260 query/group opportunities.

The named baselines describe different deployment decisions. A store choosing between fresh fixed-width range tables on P31-object would have lost 67,237 skips by selecting the better training score in subject order. A store already running the finer scalar is deciding whether to upgrade *that* map: key exceptions gain 744 skips across both layouts with zero added blocks. Figure 4 uses strong scalar consistently, so the joint deficit shown there for the complete Wikidata workload is 108,144. No cancellation between costlier and cheaper occurrence/layout pairs is needed for either certified policy.

Table 6 Wikidata temporal transfer at 32 B/group. Each row uses its own incumbent and identical later occurrences across policies. Archive denotes published edge order. Frequency and Key exc. denote present-key BoundGuard exception policies.

Filter	Layout	Occurrences	Original scalar	Unrestricted joint	Strong scalar	Frequency	Key exc.
P31 object	Subject	1,203	209,018	141,781	239,861	240,046	240,061
P31 object	Archive	1,203	205,452	205,480	216,430	216,870	216,870
P31 subject	Subject	62	28,694	28,695	28,695	28,695	28,695
P31 subject	Archive	62	2,742	4,580	3,694	3,694	3,798
P27 object	Subject	1,394	34,441	34,441	34,441	34,441	34,441
P27 object	Archive	1,394	0	0	0	0	0
P106 object	Subject	31	407	407	407	407	407
P106 object	Archive	31	405	405	405	405	405
All	Both	5,380	481,159	415,789	523,933	524,558	524,677

A `VALUES` list may include an entity outside the fixed property column while passing a range test. A list composed of absent values can therefore be sensitive to merges even when the logical result is empty. An explicit paid empty posting would settle its key in all blocks; the primary temporal comparison credits only physically present keys, so its 744 saved reads require no external membership service.

6.4 What the recovered bytes can buy

Upgrading from a strong scalar changes both the wire encoding and how the freed bytes are spent. Comparing BoundGuard's two allocation policies isolates the second decision: for each training fold or layout, both policies start from the identical recoded strong scalar and face the same occupied wire ceiling (on Retail, the 6,377-byte occupied strong-scalar length inside the 6,464-byte slot).

Table 7 shows a consistent aggregate ordering alongside strong per-query complementarity. Certified range splits skip more blocks in aggregate on all three workloads: $+1,998$ vs. $+1,974$ on Owner, $+356$ vs. $+335$ on Retail, and $+2,203$ vs. $+744$ on Wikidata. Yet the two policies win on different queries: in split-wins / ties / exception-wins order, the counts are 4 / 26 / 2 on Owner, 12 / 497 / 3 on Retail, and 171 / 5,201 / 8 on Wikidata. On Owner, their net difference of 24 hides 1,170 query–block exclusions unique to range splits and 1,146 unique to exact key exceptions. Of the split-only exclusions, 707 involve a key absent from training; tightening an interval can settle an unseen test key together with its trained neighbors, whereas an exception records exact block support for a key whose neighbors occupy different groups. On Retail all 174 split-only exclusions involve previously observed blocking keys (with 153 exception-only exclusions). On subject-ordered Wikidata P31-object, range splits skip 241,548 blocks vs. 240,061 for key exceptions, and all 1,493 split-only exclusions involve an unseen test key; in archive order both reach 216,870.

Evaluating exact exposure relative to strong scalar confirms the installation contract: total newly admitted query–block pairs $\sum_Q |C_U(Q) \setminus C_R(Q)|$ (and single-query maxima) under unrestricted joint training are 32 (max 15) on Owner, 680 (max 110) on Retail, and 109,173 (max 317) on Wikidata, whereas both certified range splits and exact key exceptions record 0 / 0 across all three workloads.

6.5 IMDb: pruning a movie posting list

A posting-list directory changes both the unit of access and the metadata already loaded. The movie–keyword relation groups all entries for a movie together. Its offsets are needed to find the next movie even for an uncapped reader. The difference of adjacent offsets supplies the current movie's row count at zero incremental count bytes. We compare routers on the same 476,794 lists and 33 occurrence queries. A query/list probe represents a possible list open. The number of facts within an admitted list can vary.

Across 15,734,202 probes, uncapped closure skips 8,502,994. Reserving mandatory endpoints lifts the count to 9,749,816. Incorporating occupied-cell and negative-test geometry raises it to 9,926,924. Packing disjoint outstanding obligations yields 9,977,188, and exact support yields 9,977,208. The cheap endpoint, occupied-cell and packing rules together deliver 1,474,194 additional exclusions, while the exact minimum-support solver contributes the final 20. For this layout we deploy the cheap rules directly on the hot execution path and use the exact solver offline to certify their tightness and the attainable upper bound under these metadata. Together they supply 1,474,214 additional exclusions, reducing list opens from 7,231,208 to 5,756,994 (against 1,049,264 true hit probes). A separate exact distinct-count field would cost 953,588 additional bytes and buy 6,272 further exclusions. The zero-additional-byte result uses only the row count.

Table 7 BoundGuard's two certified allocation policies from an identical recoded scalar. Both policies yield zero regressed queries and zero newly admitted query–block pairs ($\sum_Q |C_U(Q) \setminus C_R(Q)| = 0$) in every row. Bytes sum alternative installations across folds or layouts; the ceiling is enforced separately per file.

		Held-out blocks skipped			Occupied wire bytes		
Workload	Queries	Strong scalar	Range splits	Key exceptions	Scalar ceiling	Range splits	Key exceptions
Owner	32	3,965	5,963	5,939	192,604	192,597	192,520
Retail	512	2,892	3,248	3,227	6,377	6,376	6,376
Wikidata	5,380	523,933	526,136	524,677	64,658	60,238	60,331

Table 8 IMDb routing on four occupied ranges plus cached predicates. All rows share the same directory. Cumulative gain is relative to uncapped per-literal closure. The exact-match oracle certifies 14,684,938 empty probes.

Inference available	List skips	Lists opened	Cumulative gain
Per-literal closure	8,502,994	7,231,208	0
Mandatory endpoints	9,749,816	5,984,386	1,246,822
Occupied cells	9,926,924	5,807,278	1,423,930
Disjoint packing	9,977,188	5,757,014	1,474,194
Minimum support	9,977,208	5,756,994	1,474,214

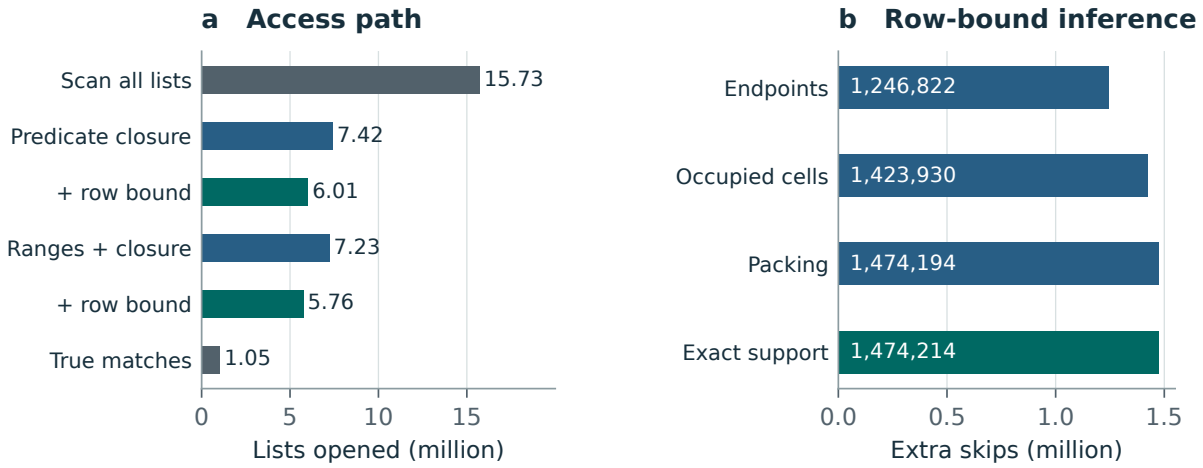


Fig. 5 Movie posting-list access paths from the row-count and incidence tables. In (a), the first row opens every directory entry; adding a row bound applies exact minimum-support reasoning to the preceding router. True matches give the lower bound on list opens. In (b), improvements accumulate relative to four-range uncapped closure. The last 20 exclusions require an offline exact support test.

Two access paths illustrate why both the unit and its metadata should be reported (Fig. 5). With cached predicates alone, closure skips 8,309,968 and exact row-bounded inference skips 9,720,036. Adding four occupied ranges improves both starting and ending points. The fact array and row counts are identical. The range bits create positive occupancy obligations and exclude values in empty ranges, while exact predicate records contribute separate obligations. The minimum-support test exploits their conjunction. The gain in Table 8 measures saved list opens on this path; physical payload requests on the Owner file are a different quantity.

The shard result sums independent contributions over 64 disjoint movie-ID shards, preserving the full relation. Summing the length bins gives the archived all-length row for each inference rule. The six template/set folds test all 33 occurrences once. A direct SQL join gives the exact set of matching movies, while the routing check asks whether every matching movie remains a candidate. These equalities check distinct points in the access path.

6.6 Catalog and service workloads near exact support

An incumbent that is already precise defines a useful control for installation. If $C_R(x) = T(x)$, an exact record for x has no false candidates to remove. At zero residual reward the incumbent is feasible at its own byte length and can be retained. Remaining opportunity can be measured against actual support or bounded from the physical order.

In SDSS SkyServer, 2,186 completed DR14 requests used a single `Galaxy` template with 300 object-ID slots [39, 40, 1]. The 1,093 later requests have no object ID in common with the first half. For contiguous groups sorted by unique object ID, a present literal can be admitted only to its own block. An absent literal can fall inside at most one block envelope. The number of false candidate blocks is therefore at most the number of distinct absent literals in the request, independently of group size. The later queries have 327,793 distinct-per-request literals and 327,471 returned rows. These counts leave an upper bound of 322 excess reads across the batch. The bound is conditional on the specified order and does not require the service's private partition map.

SQLShare connects saved SQL and transparent views to consented base tables [25]. Twenty-three eligible predicates resolve to four physical columns. The conference-venue column contains 1,221,660 rows in 150 groups. Ten predicates from seven bodies give 1,500 opportunities in the body holdout, with exact membership certifying 1,459 exclusions. The gap between exact support and an installed map bounds the remaining reward before the installer buys a single key. These services complement the drift experiment by quantifying the precision a replacement inherits.

7 End-to-end system performance

7.1 What enters the I/O bill

The physical Owner file occupies 149,609,388 bytes. A query issues one key-column range request per candidate group and one request per projected payload column in each true-hit group. Every nonempty `Read` or `ReadAt` of a compressed data chunk records its offset and returned length, verified against the Parquet footer, candidate mask and exact matching rows. Dictionary and data bytes belong to the same requested column chunk. A GET here denotes one uncoalesced data-range request on the stated single-stream path.

Table 9 accounts for the traffic of all five policies. The 32 wide aggregates issue 70,390 GETs (1,574,825,693 bytes) with original scalar, 70,224 GETs (1,568,988,953 bytes) with strong scalar, and 68,250 GETs (1,499,376,109 bytes) with exact key exceptions—1,974 fewer GETs and 69,612,844 fewer bytes (−4.44%) than strong scalar, matching the logical gain in Table 4 event by event. The 345,419,086 payload bytes across 35,085 requests are identical across all sound policies, while the open hybrid reaches 1,443,370,019 total bytes. A key-only count removes the common payload term from (12), whereas a wide projection adds shared latency and transfer time that dilutes proportional speedup without changing key-chunk pruning.

7.2 Three serial storage profiles

For each actual request i of size b_i the limiter charges $\tau + b_i/\beta$, where β is the configured stream bandwidth, enforced via a monotonic-clock wait after the backing read returns. Independent harness sessions have private allowances (40 in the network and object profiles, 12 in the gateway profile). The timer spans index probe, selective source execution and DuckDB result retrieval.

Table 10 reports measured SQL wall times on the limited reader. Relative to original scalar, exact key exceptions speed up wide queries by 1.038× in the 1 ms, 50 MB/s gateway profile (3.710 to 3.576 s) and 1.033× in the 3 ms, 400 MB/s network profile (7.429 to 7.190 s), with paired 95% intervals strictly above one. Under the 20 ms, 150 MB/s object-store profile, wide queries improve from 45.191 to 43.837 s (1.031× [1.004, 1.081] vs. original scalar; 1.028× [1.003, 1.078] vs. strong scalar), and key-only counts improve from 22.701 to 21.337 s (1.064× [1.008, 1.169]). The open hybrid reaches 1.056× wide and 1.119× narrow (Fig. 3b). In the archived local evicted-page control, original scalar and unrestricted joint read 76.94 and 75.46 MB per query (ratio 1.020 [1.002, 1.054]), where OS readahead separates device bytes from application chunk bytes.

7.3 Coalesced reads and a shared link

Candidate-block containment guarantees that a replacement never increases logical block reads, yet physical request coalescing can alter request counts. Consider three 4 KiB chunks at offsets 0, 60 and 120 KiB under a 64 KiB merge gap: reading all three bridges two 56 KiB gaps into one GET, whereas excluding the middle chunk exposes a 116 KiB gap and splits that read into two GETs. In our captured Owner wide trace (query 7, 256 KiB gap), key chunks for groups 1,064, 1,065 and 1,067 sit across adjacent gaps of 92,421 and 225,581 bytes; excluding group 1,065 widens the gap to 353,929

Table 9 Measured traffic for all 32 wide Owner aggregates on the same seven-column Parquet file. The five projected payload columns are fetched only from groups containing a true hit. Key exceptions denote BoundGuard's present-key policy.

Policy	Key GETs	Payload GETs	Key bytes	Payload bytes	Total bytes
Original scalar	35,305	35,085	1,229,406,607	345,419,086	1,574,825,693
Unrestricted joint	33,383	35,085	1,162,712,739	345,419,086	1,508,131,825
Strong scalar	35,139	35,085	1,223,569,867	345,419,086	1,568,988,953
Key exceptions	33,165	35,085	1,153,957,023	345,419,086	1,499,376,109
Hybrid	31,571	35,085	1,097,950,933	345,419,086	1,443,370,019

Table 10 Owner end-to-end wall time under serial reads. Mean seconds per held-out query, speedup over original scalar, and paired SQL-body cluster 95% intervals. Key exceptions denote BoundGuard's present-key policy.

	Gateway, wide: 1 ms, 50 MB/s			Network, wide: 3 ms, 400 MB/s		
Policy	s	Ratio	95% interval	s	Ratio	95% interval
Original scalar	3.710	1.000	Reference	7.429	1.000	Reference
Unrestricted joint	3.589	1.034	[1.0090, 1.0733]	7.186	1.034	[1.0112, 1.0692]
Strong scalar	3.702	1.002	[0.9958, 1.0098]	7.416	1.002	[0.9936, 1.0109]
Key exceptions	3.576	1.038	[1.0051, 1.0949]	7.190	1.033	[1.0055, 1.0834]
Hybrid	3.479	1.066	[1.0206, 1.1404]	7.018	1.058	[1.0182, 1.1246]
	Object, wide: 20 ms, 150 MB/s			Object, key only: 20 ms, 150 MB/s		
Policy	s	Ratio	95% interval	s	Ratio	95% interval
Original scalar	45.191	1.000	Reference	22.701	1.000	Reference
Unrestricted joint	43.989	1.027	[1.0059, 1.0624]	21.477	1.057	[1.0132, 1.1253]
Strong scalar	45.085	1.002	[0.9997, 1.0063]	22.584	1.005	[0.9989, 1.0150]
Key exceptions	43.837	1.031	[1.0039, 1.0807]	21.337	1.064	[1.0081, 1.1690]
Hybrid	42.799	1.056	[1.0167, 1.1205]	20.278	1.119	[1.0359, 1.2648]

bytes and splits one local span, even as total key-stage GETs on that query drop from 123 to 81.

To measure coalescing and concurrency directly against *strong scalar*, we evaluate a two-stage reader sharing a 20 ms TTFB, 150 MB/s link per query. Stage 0 fetches selected key chunks in physical order; stage 1 fetches projected payload chunks only for groups with a decoded match, reusing any payload chunk already contained in a stage-0 span. Within each stage, adjacent ranges whose gap fits the threshold are merged up to an 8 MiB span, charging all intervening bytes. Waves of up to C GETs share the link bandwidth while overlapping first-byte latency. Speedups divide strong-scalar time by challenger time, with 10,000 paired cluster resamples (SQL bodies on Owner, customers on Retail); intervals spanning one are unresolved.

Owner modeled service time. Table 11 replays recorded C++ offsets across 32 predicates. At a 64 KiB gap and four streams, wide-projection GETs fall from 41,383 (strong scalar) to 39,435 (key exceptions), transferred bytes fall from 1,599,178,528 to 1,528,739,034, and mean modeled service time improves from 6,808.787 to 6,490.362 ms (1.04906× [1.00580, 1.13104])—a larger proportional advantage than uncoalesced reads (1.02941×) because coalescing compresses the common payload stage. Varying C from 1 to 8 cuts absolute latency sharply while keeping relative speedups steady at 64 KiB. At a 256 KiB gap, span splitting causes wide key exceptions to issue 1,434 GETs vs. 1,280 for strong scalar despite transferring 239,154,102 fewer bytes: the single-stream point estimate dips to 0.97330× (95% CI [0.83103, 1.10767], unresolved), while eight streams recover 1.03640× (95% CI [1.00002, 1.09849]). On the six complete source SQL statements, incumbent and exception requests are identical, so scheduling alone cannot separate them.

Retail measured wall time. On UCI Online Retail, the C++17 reader executes physical range reads, decodes candidate keys, discovers hit groups and fetches missing quantity chunks before feeding Arrow batches into DuckDB (up to 64 interleaved clients, one SQL thread and emulated link per client; all 4,096 runs match full-column SQL truth). Under raw serial reads ($C = 1$), key exceptions save 335 GETs and 540,920 bytes, yielding a measured wall speedup of 1.00219× [1.00050, 1.00447] (certified range splits: 1.00206× [1.00052, 1.00409], Table 12). Under 64 KiB coalescing and four streams, a single merged span covers many candidate groups on this sub-megabyte relation and supplies later payload chunks; key exceptions save 127,487 transferred bytes while issuing one extra GET (939 vs. 938), yielding an unresolved

Table 11 Owner modeled service-time replay of recorded Parquet offsets (20 ms TTFB, 150 MB/s shared link). GETs are key-exception totals over 32 queries per projection; times are mean milliseconds; ratios and paired 95% intervals use strong scalar as reference.

		Key-only projection				Wide projection			
Gap (KiB)	C	GETs	ms	Ratio	95% interval	GETs	ms	Ratio	95% interval
Raw	1	33,165	20968.5	1.05953	[1.00682, 1.16265]	68,250	42968.6	1.02905	[1.00352, 1.07717]
	4	33,165	5431.0	1.05952	[1.00681, 1.16253]	68,250	10993.0	1.02941	[1.00358, 1.07808]
	8	33,165	2839.2	1.05948	[1.00688, 1.16234]	68,250	5662.4	1.02982	[1.00363, 1.07904]
64	1	32,722	20694.8	1.05954	[1.00687, 1.16242]	39,435	24965.4	1.04936	[1.00579, 1.13184]
	4	32,722	5359.2	1.05942	[1.00690, 1.16200]	39,435	6490.4	1.04906	[1.00580, 1.13104]
	8	32,722	2809.2	1.05929	[1.00696, 1.16170]	39,435	3414.7	1.04877	[1.00581, 1.13035]
256	1	1,031	1482.7	1.04324	[0.92749, 1.19397]	1,434	1738.8	0.97330	[0.83103, 1.10767]
	4	1,031	1008.9	1.05363	[1.00704, 1.13132]	1,434	1080.0	1.02241	[0.97135, 1.08298]
	8	1,031	932.1	1.05671	[1.01028, 1.13888]	1,434	973.8	1.03640	[1.00002, 1.09849]

Table 12 End-to-end Retail SQL on 512 held-out invoices (native C++ reader + DuckDB on the 202-group Parquet file). Service is scheduled link time; wall time includes native execution and DuckDB. Ratios compare against strong scalar within each profile; intervals spanning one are unresolved.

Profile	Method	GETs	MB	Service (ms)	Ratio	Wall (ms)	Ratio	95% wall interval
Raw, $C = 1$	Strong scalar	192,829	203.547	7535.033	1.00000	7568.673	1.00000	[1.00000, 1.00000]
	Joint	193,107	204.007	7545.899	0.99856	7577.543	0.99883	[0.99590, 1.00184]
	Range splits	192,473	202.974	7521.120	1.00185	7553.131	1.00206	[1.00052, 1.00409]
	Key exceptions	192,494	203.006	7521.940	1.00174	7552.116	1.00219	[1.00050, 1.00447]
64 KiB, $C = 4$	Strong scalar	938	349.457	41.191	1.00000	74.929	1.00000	[1.00000, 1.00000]
	Joint	936	349.654	41.037	1.00375	76.191	0.98344	[0.91282, 1.05836]
	Range splits	942	349.202	41.266	0.99819	76.557	0.97874	[0.90535, 1.05900]
	Key exceptions	939	349.330	41.228	0.99909	74.933	0.99995	[0.93168, 1.07229]

wall ratio of 0.99995× [0.93168, 1.07229] (range splits: 0.97874× [0.90535, 1.05900], also unresolved).

Table 13 separates the logical contract from physical scheduling and single-run host jitter. Under both profiles, certified range splits and exact key exceptions incur *zero* per-invoice regressions in candidate blocks and transferred bytes (whereas unrestricted joint adds up to 110 blocks and 188,629 bytes on one invoice). Yet under 64 KiB coalescing, span splitting causes one key-exception invoice (and two range-split invoices) to pay up to 19.247–19.985 ms of extra link service time. Because wall time records one host-scheduled execution per invoice and policy, even byte-identical runs fluctuate in elapsed time.

7.4 When does a saved read pay for its CPU cost?

For policy P on query q denote host processing time by h_{Pq}, total GET count by n_{Pq} and returned compressed bytes by b_{Pq}. A trace-based time for a serial service is

$$\widehat{t}_{Pq}(\tau, \beta) = h_{Pq} + n_{Pq}\tau + b_{Pq}/\beta. \tag{13}$$

Each h_{Pq} is measured through completion of the query on the host; counts and bytes come from ordered C++ range events. Process CPU time was recorded alongside host elapsed time in the same calibration repetition (taking independent medians over repetitions 1–5, which come from different repetitions in 25 of 350 groups); the host term in (13) is elapsed time, since replacing it by process CPU would omit elapsed work belonging to the query.

For a reference policy R let Δn and Δb be its average GETs and bytes minus the challenger's, and Δh be challenger minus reference host time. When $\Delta n > 0$, the break-even TTFB is

$$\tau_* = \max\{0, (\Delta h - \Delta b/\beta)/\Delta n\}. \tag{14}$$

On the wide projection strong scalar saves an average of 5.1875 requests and 182,398 bytes against original scalar. At 50 MB/s saved transfer time alone covers the extra host work; required TTFB is 0.437 ms at 150 MB/s, 0.584 ms at 400 MB/s, and 0.672 ms at unlimited bandwidth (Fig. 3a). For exact key exceptions the same model gives a zero threshold against original scalar on the wide workload. Theorem 3 governs candidate sets, not host lookup costs.

7.5 Parquet page indexes and split-block Bloom filters

Parquet also offers metadata at page and column-chunk granularity: a column index records page bounds, an offset index locates surviving pages, and split-block Bloom

Table 13 Retail per-invoice tails relative to strong scalar. Triples count improved / tied / regressed invoices; Max is the largest single-invoice increase. Strong scalar has 512 ties and zero loss. Wall time reflects one host-scheduled run per invoice and policy.

		Candidate blocks		Transferred bytes		Service (ms)		Wall (ms)	
Gap	Method	Counts	Max	Counts	Max	Counts	Max	Counts	Max
Raw	Joint	14/458/40	110	14/458/40	179,322	14/458/40	2201.196	242/0/270	2211.814
	Range splits	18/494/0	0	18/494/0	0	18/494/0	0.000	259/0/253	400.922
	Key exceptions	8/504/0	0	8/504/0	0	8/504/0	0.000	254/0/258	97.374
64 KiB	Joint	14/458/40	110	3/497/12	188,629	7/497/8	19.247	257/0/255	499.083
	Range splits	18/494/0	0	5/507/0	0	3/507/2	19.985	254/0/258	482.625
	Key exceptions	8/504/0	0	4/508/0	0	3/508/1	19.247	252/0/260	452.801

Table 14 Archived native Parquet controls. Mean compressed key MB per Owner list on the common key-only layout. Sparse allocations price physical file growth including metadata pointers.

Key-file access path	Key MB
Row-group extrema	41.705
Matched-budget sparse page index	41.531
Full fine page index	41.473
Matched-budget sparse Bloom	41.553
Joint-first group filter + full page index	35.480
Hybrid group filter + full page index	33.602
Bloom at target 1% false-positive rate	8.627

filters exclude column chunks on negative answers [7, 6]. The archived native controls use a common key-only layout and price actual on-file growth including metadata pointers (Table 14).

With 4 KiB pages, 4,848 key pages add 132,060 bytes of page-index structures (149,168 bytes total file growth). A matched-budget sparse page index (314 groups, 38,518–38,526 extra bytes per fold) reads 41.531 MB vs. 41.473 MB for the full page index; prepending the protected joint-first group filter drops the latter to 35.480 MB. A 32-byte split-block Bloom filter for every group adds 65,988 bytes, a matched-budget sparse Bloom (38,054–38,462 bytes/fold) reads 41.553 MB, and a 1% false-positive-rate Bloom adds 14.16 MB to read 8.627 MB. Bloom negatives are sound exclusions, while Bloom positives cannot act as required witnesses in minimum-support inference.

8 Related work

Block-level exclusion. Small materialized aggregates store useful facts about predefined segments [33]. Block range indexes and column imprints use extrema, interval occupancy or closely related information to avoid scanning blocks [35, 38]. A global range table builds a common set of boundaries and spends its occupancy bits according to individual-value popularity [16]. Our GRT comparison fixes the data, occupancy semantics and capacity while varying how a newly admitted read is charged. GRT supplies a workload-driven range allocation; BoundGuard takes an installed allocation as its reference and certifies singleton containment on every elementary domain interval, including intervals absent from the log.

Equality-encoded bitmap indexes provide the natural full-precision alternative for a key with a reusable posting [13]. Partial indexes restrict an access path to a selected subset [42]. PatchIndex records exceptions to approximate constraints in distributed data processing [27]. BoundGuard's exact-key policy is narrower: it replaces one installed key-to-block map by exact block support, paying for the key address and positive or corrective posting, with return determined by the remaining blockers in each list.

Workload-aware design and robustness. GSOP, Qd-trees and Pando use access patterns to improve physical organization or block skipping [45, 49, 44]. In particular a workload-aware partitioner can already evaluate complete predicates. BoundGuard changes metadata on fixed blocks; moving tuples, as in the Wikidata layout comparison, requires a separate certificate.

Adaptive Range Filters redistribute precision following observed false positives, Proteus chooses approximate range filters using a sampled workload, and space-budgeted Bloom collections allocate filter bits according to access frequency [3, 28, 32]. CliffGuard searches physical designs for robust workload performance over an explicit uncertainty neighborhood [34]. BoundGuard addresses a distinct contract: given an installed reference index and a fixed byte ceiling on an immutable file generation, every candidate block admitted by an upgrade for any future membership list must already be admitted by the incumbent. Lossless recoding frees space within the installed ceiling for either certified range splits or exact key exceptions, and a full-domain sweep verifies the serialized replacement. Workload compression likewise asks which properties of a query log to retain [20]; our balanced-family bound identifies the exact grouping

information needed for read exclusions independently of physical index bytes.

Prior predicate outcomes and support constraints. Semantic caching reuses known predicate regions and their results; predicate caches store and reuse the results of filter evaluation [19,36,18]. For a disjunctive membership predicate, a recorded block miss entails one negative fact about each literal, even for later lists absent from the cache. Noiseless group testing supplies the related notions of elimination from negative pools and minimum satisfying sets [2]. The file contributes two more kinds of information: a stored extremum can mandate a specific witness, and an occupied range can mandate at least one witness in a category. Theorem 5 characterizes exactly when an existing row count certifies further exclusions.

Interactions in index selection. Automatic physical design and index-interaction analyses already capture dependencies among candidate access paths [14,37]. In this setting the interaction derives directly from physical truth: a term of (1) is the set of records completing one unpaid exclusion. Within each connected component training gain is local, while a knapsack allocates bytes across components. Regardless of allocation heuristic, installation checks two frozen candidate maps.

9 Conclusion

A query boundary records which negative facts must hold together to eliminate a read. Recovering that grouping can change pruning by an exponential factor even with balanced marginals, yet training evidence alone cannot certify behavior under workload shift. BoundGuard separates purchase from certification: it losslessly recodes an installed index within its byte ceiling, allocates the released space to either certified range splits or exact key exceptions, and verifies soundness and full-domain singleton containment on the serialized replacement. Across Owner, Retail and Wikidata, certified range splits yield more aggregate skips while both policies certify zero added candidates and capture distinct query–block exclusions. At execution, per-literal closure and cheap hot-path row-bound rules settle further candidates, with exact minimum-support solving providing an offline tightness bound. Finally, our coalesced and concurrent I/O studies show that candidate-block containment and request scheduling have distinct guarantee scopes: removing an interior candidate block never adds logical reads, yet can occasionally split a coalesced GET span and alter physical service time.

Scope and Limitations. The guarantee covers positive single-column membership on one immutable generation. Saved Owner SQL supplies observed bodies with unknown production weights; the 512 retail predicates instantiate one template from genuine invoices. Gains can be concentrated: six Owner predicates improve under exact key exceptions, eight of 512 retail baskets improve at 512 rows/group (18 under range splits), and November contributes 286 of 335 exception retail skips. On resident local file pages the wide key-exception $1.021\times$ interval $[0.984, 1.079]$ is unresolved. Strong scalar costs an extra 3.166 ms of process CPU and 3.485 ms of host time per wide query against original scalar. On the six complete Owner SQL statements, strong scalar and key exceptions issue identical requests and bytes. SkyServer range objectives tie in tested layouts, SQLShare's protected scalar leaves no further held-out exclusion for eligible key records, and in 256 held-out TPC-H Q16 queries on SF1 every group contains all 50 sizes, leaving zero opportunities for the eight-value filter [46]. Wikidata uses a later partial graph snapshot. Object-store profiles use one host, private per-query links, a single wall observation per Retail invoice/policy, and a specified wave scheduler for within-query coalescing; production services may pipeline completions, share bandwidth across queries or impose range limits of their own.

Statements and Declarations

Funding. No funding was received for conducting this study.

Competing Interests. The author has no competing interests to declare that are relevant to the content of this article.

Author Contributions. T.L. conceptualized the study, developed the theoretical analysis and software implementation, conducted all experiments, prepared Figs. 1–5 and Tables 1–14, and wrote and reviewed the manuscript.

Data Availability. All datasets evaluated in this study are publicly available from their cited repositories (Stack Overflow SEDE/SENLIDB [24,11,41], UCI Online Retail [15], Wikidata organic query logs and Wikidata5M [31,48,47], IMDb Join Order Benchmark [29], SDSS SkyServer DR14 [39,40], SQLShare [25], and TPC-H [46]). The source code, verification scripts, serialized index artifacts, and summary tables supporting the findings of this study are available from the author upon reasonable request.

References

1. Abolfathi, B., et al.: The fourteenth data release of the sloan digital sky survey: First spectroscopic data from the extended baryon oscillation spectroscopic survey and from the second phase of the apache point observatory galactic evolution experiment. Astrophysical Journal Supplement Series **235**(2), 42 (2018). https://doi.org/10.3847/1538-4365/aa9e8a
2. Aldridge, M., Baldassini, L., Johnson, O.: Group testing algorithms: Bounds and simulations. IEEE Transactions on Information Theory **60**(6), 3671–3687 (2014). https://doi.org/10.1109/TIT.2014.2314472
3. Alexiou, K., Kossmann, D., Larson, P.Å.: Adaptive range filters for cold data: Avoiding trips to Siberia. Proceedings of the VLDB Endowment **6**(14), 1714–1725 (2013). https://doi.org/10.14778/2556549.2556556
4. Alon, N., Spencer, J.H.: The Probabilistic Method, 4th edn. Wiley (2016)
5. Apache Arrow contributors: PyArrow 18.1 api reference: ParquetFile (2024). URL https://arrow.apache.org/docs/18.1/python/generated/pyarrow.parquet.ParquetFile.html
6. Apache Parquet contributors: Parquet format: Bloom filter. URL https://parquet.apache.org/docs/file-format/bloomfilter/
7. Apache Parquet contributors: Parquet format: Page index. URL https://parquet.apache.org/docs/file-format/pageindex/
8. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the desirability of acyclic database schemes. Journal of the ACM **30**(3), 479–513 (1983). https://doi.org/10.1145/2402.322389
9. Besag, J., Clifford, P.: Generalized monte carlo significance tests. Biometrika **76**(4), 633–642 (1989). https://doi.org/10.1093/biomet/76.4.633
10. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM Journal on Computing **39**(2), 546–563 (2009). https://doi.org/10.1137/070683933
11. Brad, F., Iacob, R.C.A., Hosu, I.A., Rebedea, T.: Dataset for a neural natural language interface for databases (NNLIDB). In: Proc. 8th Int. Joint Conf. on Natural Language Processing (IJCNLP), pp. 906–914 (2017). URL https://aclanthology.org/I17-1091/
12. Carstens, C.J.: Proof of uniform sampling of binary matrices with fixed row sums and column sums for the fast curveball algorithm. Physical Review E **91**, 042,812 (2015). https://doi.org/10.1103/PhysRevE.91.042812
13. Chan, C.Y., Ioannidis, Y.E.: Bitmap index design and evaluation. In: Proceedings of ACM SIGMOD, pp. 355–366 (1998). https://doi.org/10.1145/276304.276336
14. Chaudhuri, S., Narasayya, V.R.: An efficient cost-driven index selection tool for Microsoft SQL Server. In: Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 146–155 (1997)
15. Chen, D.: Online retail (2015). https://doi.org/10.24432/C5BW33. Dataset, UCI Machine Learning Repository
16. Chen, J., Sherkat, R., Andrei, M., Gerwens, H.: Global range encoding for efficient partition elimination. In: Proceedings of the 21st International Conference on Extending Database Technology, pp. 453–456 (2018). https://doi.org/10.5441/002/edbt.2018.45
17. Chlamtáč, E., Dinitz, M., Konrad, C., Kortsarz, G., Rabanca, G.: The densest k-subhypergraph problem. SIAM Journal on Discrete Mathematics **32**(2), 1458–1477 (2018). https://doi.org/10.1137/16M1096402
18. ClickHouse: Introducing the query condition cache (2025). URL https://clickhouse.com/blog/introducing-the-clickhouse-query-condition-cache
19. Dar, S., Franklin, M.J., Jónsson, B.T., Srivastava, D., Tan, M.: Semantic data caching and replacement. In: Proceedings of the 22nd International Conference on Very Large Data Bases, pp. 330–341 (1996). URL https://www.vldb.org/conf/1996/P330.PDF
20. Deep, S., Gruenheid, A., Koutris, P., Naughton, J., Viglas, S.: Comprehensive and efficient workload compression. Proceedings of the VLDB Endowment **14**(3), 418–430 (2020). https://doi.org/10.14778/3430915.3430931
21. Feige, U., Kortsarz, G., Peleg, D.: The dense k-subgraph problem. Algorithmica **29**(3), 410–421 (2001). https://doi.org/10.1007/s004530010050
22. Fréchet, M.: Généralisation du théorème des probabilités totales. Fundamenta Mathematicae **25**, 379–387 (1935). https://doi.org/10.4064/fm-25-1-379-387
23. Goldschmidt, O., Nehme, D., Yu, G.: Note: On the set-union knapsack problem. Naval Research Logistics **41**(6), 833–842 (1994). https://doi.org/10.1002/1520-6750(199410)41:6<833::AID-NAV3220410611>3.0.CO;2-Q
24. Hazoom, M., Malik, V., Bogin, B.: Text-to-SQL in the wild: a naturally-occurring dataset based on stack exchange data. In: Proceedings of the 1st Workshop on Natural Language Processing for Programming, pp. 77–87 (2021). https://doi.org/10.18653/v1/2021.nlp4prog-1.9
25. Jain, S., Moritz, D., Halperin, D., Howe, B., Lazowska, E.: SQLShare: Results from a multi-year SQL-as-a-service experiment. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 281–293 (2016). https://doi.org/10.1145/2882903.2882957
26. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press (1972). https://doi.org/10.1007/978-1-4684-2001-2_9
27. Kläbe, S., Sattler, K.U., Baumann, S.: PatchIndex: exploiting approximate constraints in distributed databases. Distributed and Parallel Databases **39**, 833–853 (2021). https://doi.org/10.1007/s10619-021-07326-1
28. Knorr, E.R., Lemaire, B., Lim, A., Luo, S., Zhang, H., Idreos, S., Mitzenmacher, M.: Proteus: A self-designing range filter. In: Proceedings of the 2022 International Conference on Management of Data, pp. 1670–1684 (2022). https://doi.org/10.1145/3514221.3526167
29. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? Proceedings of the VLDB Endowment **9**(3), 204–215 (2015). https://doi.org/10.14778/2850583.2850594
30. Madow, W.G.: On the theory of systematic sampling, II. The Annals of Mathematical Statistics **20**(3), 333–354 (1949). https://doi.org/10.1214/aoms/1177729988
31. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of Wikidata: Semantic technology usage in Wikipedia's knowledge graph. In: Proceedings of the 17th International Semantic Web Conference, *LNCS*, vol. 11137, pp. 376–394 (2018). https://doi.org/10.1007/978-3-030-00668-6_23
32. Mersy, G., Wang, Z., Sintos, S., Krishnan, S.: Optimizing collections of Bloom filters within a space budget. Proceedings of the VLDB Endowment **17**(11), 3551–3564 (2024). https://doi.org/10.14778/3681954.3682020

33. Moerkotte, G.: Small materialized aggregates: A light weight index structure for data warehousing. In: Proceedings of the 24th International Conference on Very Large Data Bases, pp. 476–487 (1998). URL https://www.vldb.org/conf/1998/p476.pdf
34. Mozafari, B., Goh, E.Z.Y., Yoon, D.Y.: CliffGuard: A principled framework for finding robust database designs. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1167–1182 (2015). https://doi.org/10.1145/2723372.2749454
35. PostgreSQL Global Development Group: PostgreSQL 18 documentation: BRIN indexes (2025). URL https://www.postgresql.org/docs/18/brin.html
36. Schmidt, T., Kipf, A., Horn, D., Saxena, G., Kraska, T.: Predicate caching: Query-driven secondary indexing for cloud data warehouses. In: ACM SIGMOD Companion, pp. 347–359 (2024). https://doi.org/10.1145/3626246.3653395
37. Schnaitter, K., Polyzotis, N., Getoor, L.: Index interactions in physical design tuning: Modeling, analysis, and applications. Proceedings of the VLDB Endowment **2**(1), 1234–1245 (2009). https://doi.org/10.14778/1687627.1687766
38. Sidirourgos, L., Kersten, M.: Column imprints: A secondary index structure. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 893–904 (2013). https://doi.org/10.1145/2463676.2465306
39. Sloan Digital Sky Survey: Skyserver data release 14: Catalog archive, schema and sql access. URL https://skyserver.sdss.org/dr14/
40. Sloan Digital Sky Survey: Skyserver traffic: Sql query log. URL https://skyserver.sdss.org/log/en/traffic/sql.asp
41. Stack Overflow contributors and ClickHouse: Stack overflow public dataset in Parquet. URL https://clickhouse.com/docs/get-started/sample-datasets/stackoverflow. Annual projections for 2008, 2014 and 2020
42. Stonebraker, M.: The case for partial indexes. Tech. Rep. UCB/ERL M89/17, University of California, Berkeley (1989)
43. Strona, G., Nappo, D., Boccacci, F., Fattorini, S., San-Miguel-Ayanz, J.: A fast and unbiased procedure to randomize ecological binary matrices with fixed row and column totals. Nature Communications **5**, 4114 (2014). https://doi.org/10.1038/ncomms5114
44. Sudhir, S., Tao, W., Laptev, N., Habis, C., Cafarella, M., Madden, S.: Pando: Enhanced data skipping with logical data partitioning. Proceedings of the VLDB Endowment **16**(9), 2316–2329 (2023). https://doi.org/10.14778/3598581.3598601
45. Sun, L., Franklin, M.J., Wang, J., Wu, E.: Skipping-oriented partitioning for columnar layouts. Proceedings of the VLDB Endowment **10**(4), 421–432 (2016). https://doi.org/10.14778/3025111.3025123
46. Transaction Processing Performance Council: TPC Benchmark H: Standard specification, revision 2.17.3 (2018). URL https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.3.pdf
47. Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., Tang, J.: Wikidata5M: Knowledge graph (transductive). Zenodo, dataset (2019). https://doi.org/10.5281/zenodo.5546383
48. Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., Tang, J.: KEPLER: A unified model for knowledge embedding and pre-trained language representation. Transactions of the Association for Computational Linguistics **9**, 176–194 (2021). https://doi.org/10.1162/tacl_a_00360
49. Yang, Z., Chandramouli, B., Wang, C., Gehrke, J., Li, Y., Minhas, U.F., Larson, P.Å., Kossmann, D., Acharya, R.: Qd-tree: Learning data layouts for big data analytics. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 193–208 (2020). https://doi.org/10.1145/3318464.3389770