Decisionhouse: Prescriptive Analytics in the Data Stack

Matteo Brucato Fjodor Kholodkov Soren Little Jakob Mayer Duc Nguyen
OSM Data
{matteo, fjodor, soren, jakob, duc}@osm-data.com

ABSTRACT

Data platforms have evolved by making data-intensive workloads native: SQL and query optimizers eliminated bespoke data-retrieval programs; Lakehouses added first-class support for ML training and serving over the same data. Prescriptive analytics (computing optimal actions subject to constraints over data) is equally data-intensive, yet remains outside the platform: every optimization problem requires a hand-built pipeline from data extraction to solver invocation, rebuilt from scratch whenever the data or the requirements change. We propose *Decisionhouses*, a new class of data infrastructure that makes prescriptive analytics *native*. A Decisionhouse provides (i) DeQL (Decision Query Language), a declarative SQL extension where users express decision problems over relational data; (ii) automatic formulation selection that exploits query and data semantics to pick the right problem class and solver—a choice that can change a query's complexity class from NP-hard to polynomial; and (iii) end-to-end integration of optimization into the data platform, from query parsing through solver execution. Decisionhouses can help address several challenges that have kept optimization outside data platforms, including pipeline brittleness, formulation expertise, structural blindness, and scalability cliffs, and make decision-making as accessible as querying data.

PVLDB Reference Format:
Matteo Brucato, Fjodor Kholodkov, Soren Little, Jakob Mayer, and Duc Nguyen. Decisionhouse: Prescriptive Analytics in the Data Stack. PVLDB, 19(10): 2755 - 2762, 2026.
doi:10.14778/3828612.3828629

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/osm-data/decisionhouse-analysis.

1 INTRODUCTION

A SQL database can easily answer *"what does it cost to ship from each warehouse to each store?"* with a single join. It cannot answer *"which warehouse-to-store routes should we activate to minimize total cost while meeting every store's demand without exceeding any warehouse's supply?"* This *shipping problem* is a *prescriptive* question [25, 47, 57]: given options from relational data, constraints from business policy, and a measurable objective, compute the best course of action. Such questions arise across many domains, including supply chain routing, GPU allocation, shift scheduling, batch sequencing, and portfolio selection. Answering each today requires leaving the database for a bespoke pipeline that extracts data, encodes a formulation, runs an optimization solver, and maps results back. Each new problem requires its own pipeline. We call this accumulated cost *decision debt* (Figure 1a).

In these pipelines, data extraction, solver invocation, and result mapping have established tooling. Producing the *formulation*, however, is still hard. The formulation specifies the problem's variables, objective, and constraints for a particular *solver* (an algorithm that finds an assignment optimizing the objective subject to the constraints). *Mixed-Integer Linear Programming* (MILP) is one of the most common: continuous or integer variables with linear objective and constraints. MILPs model problems from scheduling and routing to resource allocation and portfolio selection [69].

The shipping problem can be written as a MILP, but not every problem that fits MILP is best solved as one. Shipping itself can equivalently be written as a *minimum-cost flow* problem, provided certain conditions hold: warehouses and stores form a bipartite supply-demand graph, with supply at each warehouse and demand at each store [5]. Conditions like these are called the problem's *structure*. Both formulations produce the same optimum, but the structure determines the complexity: MILP is NP-hard in general, while minimum-cost flow runs in polynomial time. Many other decision problems exhibit this pattern: resource allocation, worker assignment, and route planning all admit specialized formulations of much lower complexity than generic MILP.

Such structure is present in the data and the schema, but generic formulations obscure it: once committed to MILP, the problem becomes a flat coefficient matrix in which supply caps and demand requirements are indistinguishable rows. SQL preserves the schema's structure (entities, keys, relationships) but cannot express prescriptive questions. Modeling languages such as AMPL [24], Pyomo [30], and JuMP [22] encode a chosen formulation but offer no support for selecting one, and they lack the relational view of the data that makes structure visible.

What is missing is a layer between data and modeling languages: the *Decisionhouse* (Figure 1b), a new class of data infrastructure for prescriptive analytics that internalizes per-problem pipelines behind a declarative interface that preserves the problem structure. That interface is *DeQL* (Decision Query Language, pronounced "dee-kwl"), a SQL extension where decision options are declared as relations, each row carries a decision variable, and constraints and objectives are aggregate expressions. Because these constructs stay anchored to the schema—column names, joins, and group-bys—structural patterns are visible to the system. The user writes one DeQL query per decision problem; the Decisionhouse handles data binding, formulation selection, solving, and result mapping. Decision queries produce standard relations, so results compose with SQL and other decision queries without the separate result-mapping step that bespoke pipelines require.

Proceedings of the VLDB Endowment, Vol. 19, No. 10 ISSN 2150-8097.
doi:10.14778/3828612.3828629

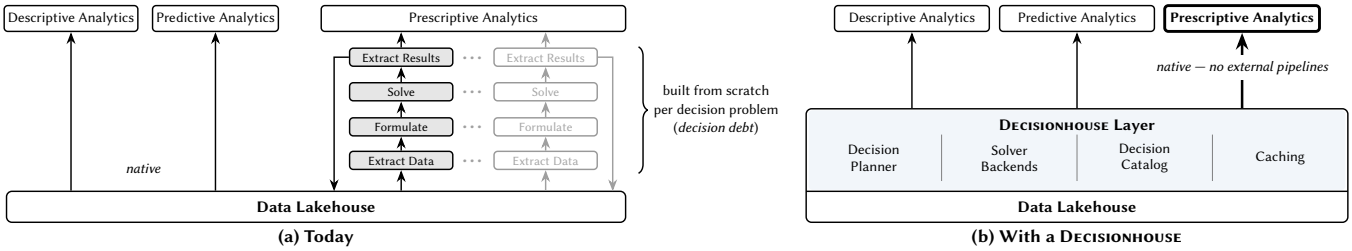

Figure 1: Prescriptive analytics in data platforms: per-problem external pipelines today (a) vs. native in a Decisionhouse (b).

Given the shipping problem as a DeQL query, a Decisionhouse can recognize its bipartite supply-demand structure and dispatch the query to a minimum-cost flow solver rather than MILP (Section 4). Shipping is one of many decision problems where bipartite supply-demand structure makes minimum-cost flow applicable. GPU allocation is another such case: in our experiments, the DeQL query runs up to 355× faster than MILP at 1.8M candidate assignments; at 7.4M, MILP times out after 10 minutes while min-cost flow solves in 2.2 seconds. Both formulations produce identical optimal solutions, so the speedup comes entirely from structure recognition (Section 4.3).

Decisionhouses also give LLMs a target language and runtime for prescriptive analytics: a smaller, verifiable alternative to the free-form solver code generated by today's end-to-end LLM optimization tools [4, 31, 43]. For descriptive analytics, text-to-SQL is mature at the front [44] and LLM-synthesized engines are emerging at the back [41, 68]; Decisionhouses open this decomposition to prescriptive analytics: text-to-DeQL, DeQL-to-code synthesis, and explicit formulation selection in between. End-to-end tools skip this step: the formulation is whatever the LLM emits, often MILP [32, 70]; each deployment is a standalone script with no specification to verify against. Whether the formulation comes from a rule, a statistic, or an LLM, DeQL is the contract it must satisfy.

This paper defines **eight requirements** for prescriptive data infrastructure (Section 2), presents **DeQL** (Section 3) and a **Decisionhouse architecture** with a three-IR pipeline and automatic formulation selection (Section 4), and outlines **open problems** at the intersection of database systems, combinatorial optimization, and scalable solving—including data-centric directions on structure detection, cost models, physical design, responsive execution, and decision governance (Section 5). This work extends prior research on in-database prescriptive analytics [16, 19, 29, 46] from single-class package queries to a general-purpose system. The design is informed by our ongoing industry collaborations in logistics, process mining, and cloud platforms.

2 THE DECISIONHOUSE

Building a Decisionhouse is now feasible and timely. Lakehouse architectures [71] and open table formats [9] decouple data from its host engine, so any system can read it. PaQL [16] absorbed the optimization pipeline for package queries; abstraction-based decomposition [16, 29, 46] scales it to billions of candidates. Recent LLM capabilities at natural-language-to-optimization translation [4, 31, 43, 72] demonstrate that this kind of front-end is feasible; in a Decisionhouse, the LLM target shifts from solver scripts to DeQL, making text-to-DeQL a natural LLM front-end on top of the Decisionhouse.

Table 1: Existing systems vs. Decisionhouse requirements. ✓ addressed, ○ partial, ✗ not in scope. Partial notes: [1]OR expertise required; [2]data connectors, not integrated; [3]generic, not extensible; [4]binary selection only; [5]logging, not decision-specific; [6]LLM formulation may be incorrect; [7]solver-level time limits; [8]scalable, no progress feedback.

Requirement	Solvers	Modeling langs	OR Platf.	PaQL	SolveDB	LLM	Decisionhouse
(R1) Accessible	✗	○[1]	○[1]	✓	○[1]	✓	✓
(R2) Data-native	✗	✗	○[2]	✓	✓	✗	✓
(R3) Extensible	✓	✓	✓	✗	○[3]	✓	✓
(R4) Composable	✗	✗	✗	○[4]	✓	✗	✓
(R5) Auto-plan	✗	✗	✗	✗	✗	✗	✓
(R6) Governance	✗	✗	○[5]	○[5]	○[5]	✗	✓
(R7) Guarantees	✓	✓	✓	✓	✓	○[6]	✓
(R8) Responsive	○[7]	○[7]	○[7]	○[8]	✗	○[7]	✓

Requirements. Eight requirements together define the Decisionhouse. **(R1) Accessibility**. Analysts need an interface above integer programming [4, 27, 43, 72]. **(R2) Data-native**. Optimization consumes data in place—relational tables, Parquet, Delta Lake—without manual extraction [16, 64]. **(R3) Extensibility**. The system handles diverse patterns (subset selection, resource allocation, assignment, scheduling) and admits new constraint types. **(R4) Composability**. The output of one decision query is consumable as input to another, echoing relational algebra's closure property; languages without it exhibit expressibility gaps [26]. **(R5) Automatic planning**. Algorithm choice follows from problem structure: a supply-demand query is exponential as MILP but polynomial as network flow. **(R6) Governance**. Decisions must be explainable and auditable for regulatory compliance [1, 23]. **(R7) Guarantees**. Decisions must be feasible and bounded; solvers already provide optimality gaps and feasibility certificates. **(R8) Responsiveness**. Users expect interactive response: progress feedback, time-bounded execution, and best-found solutions within user budgets [2, 73].

Automatic planning (R5)—the Decisionhouse's defining capability, absent from every other system (Table 1)—requires detecting structure from the declarative formulation: solvers do limited matrix-level recognition but miss the query-level structure that drives formulation choice, and LLMs generate solver code end-to-end with no planning stage or correctness guarantees.

3 DeQL: THE DECISION QUERY LANGUAGE

`SELECT` answers "what data exists?" DeQL introduces `DECIDE` to answer "what actions to take?" `PREDICT` invokes a pre-trained model; `DECIDE` triggers a translation and optimization pipeline. DeQL preserves the relational model end-to-end: queries consume relations

gpu_pools

pool_id	capacity
H100-E	500
A100-W	300

workloads

workload_id	demand
llm-serve	200
embed-batch	250
fine-tune	150

assignments

pool_id	workload_id	cost	latency_ms
H100-E	llm-serve	10	45
H100-E	embed-batch	15	120
A100-W	embed-batch	8	85
A100-W	fine-tune	14	160

```
CREATE CANDIDATES gpu_assignments
DECISION KEY (pool_id, workload_id) AS
  SELECT a.pool_id, a.workload_id,
         p.capacity, w.demand, a.cost, a.latency_ms
  FROM assignments a
  JOIN gpu_pools p ON a.pool_id = p.pool_id
  JOIN workloads w ON a.workload_id = w.workload_id;

DECIDE allocation_plan
FROM gpu_assignments
DECISION COLUMNS (
    active   SELECTION BINARY,
    quantity CONTINUOUS BETWEEN 0 AND capacity)
WHERE latency_ms <= 200
SUBJECT TO
  CONSTRAINT pool_capacity: SUM(quantity) <= capacity BY pool_id,
  CONSTRAINT meet_demand:   SUM(quantity) = demand BY workload_id
MINIMIZE SUM(cost * quantity);
```

Figure 2: GPU allocation example: input tables (top) and DeQL workflow (bottom).

and produce relations. We outline the major features below; full details are in the language specification [17].

Running example: GPU allocation. An AI inference platform allocates GPU-hours from compute pools to workloads at minimum cost, subject to capacity and demand constraints. Figure 2 shows the input tables and query: pools with available capacity, workloads with compute demand, and assignments with costs and latencies.

`CREATE CANDIDATES` factors the candidate definition into a first-class database construct: a named query like a view, annotated with a `DECISION KEY` that declares which columns identify each *decision entity* (metadata that neither tables nor views carry). In the GPU example, the `SELECT` joins three tables to produce one row per pool–workload pair; each such row is a *candidate*, and `DECISION KEY (pool_id, workload_id)` declares that each pair is one decision entity.

The `DECIDE` statement optimizes over the candidates and materializes the result as a new table (e.g., `allocation_plan`). `DECISION COLUMNS` declares the *decision variables* (one per candidate): unknowns computed at query time, unlike data columns which are read from the database; they otherwise behave like regular columns in expressions and output. Each variable declares a domain: `BINARY` for select/not-select, or `CONTINUOUS` with data-driven bounds (e.g., `BETWEEN 0 AND capacity`). A `BY` clause instantiates one constraint per group: with two pools (H100-E, A100-W), `SUM(quantity) <= capacity BY pool_id` generates two constraints, one capping each pool's allocation. The optional `SELECTION` modifier makes `active` a binary on/off switch: when `active = 0`, `quantity` is forced to zero and the row is omitted from the result, so this linking need not be written by hand (generalizing PaQL's [16] package semantics). The `MINIMIZE/MAXIMIZE` clause specifies the objective; appending `WITHIN X%` bounds the optimality gap, and appending `TIMEOUT X s` caps solve time, letting users trade optimality for speed (Section 4.2).

Three further constructs complete the sketch. `WHERE` in `CREATE CANDIDATES` removes physically infeasible candidates; `WHERE` in

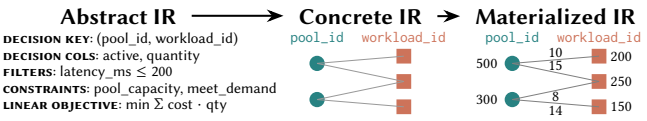

Figure 3: GPU query at each IR: Abstract (symbolic), Concrete (min-cost flow, symbolic), Materialized (data-populated).

`DECIDE` applies per-query policy filters (e.g., latency SLAs). Multi-level problems (assigning items to batches while choosing each batch's timing) join candidate tables in `DECIDE`: each keeps its own `DECISION KEY`, so its decision variables are not duplicated across the join. Stochastic extensions attach distributional annotations to data columns and probabilistic guarantees to constraints. The language specification [17] covers each in detail.

`CREATE ABSTRACTION` defines a coarser view of the candidate space—e.g., clustering millions of GPU assignments into thousands of representative groups—enabling decomposition methods [59] like SketchRefine [16] and partial-state abstraction [50] to meet `WITHIN/TIMEOUT` budgets at scale:

```
CREATE ABSTRACTION assignment_clusters
ON gpu_assignments (cost, demand, capacity) USING KMEANS(k=1000);
```

SketchRefine solves a reduced problem on representatives, then refines only promising clusters, shrinking effective problem size by orders of magnitude within bounded error. We discuss extensions to other problem classes and anytime variants in Section 5.

DeQL rests on relational algebra: `CREATE CANDIDATES` takes a SQL query as body, grouped constraints use `GROUP BY` semantics, and column references are relational attributes. Extending SQL reuses these primitives directly. Beyond syntax, the relational grounding yields several benefits: `DECIDE` produces a relation, so decision queries compose with SQL and with other `DECIDE` queries without an export/import step; the decision planner reads candidate-set statistics for formulation selection (Section 4.2); and candidate sets and decision outputs reuse the host store's physical design and governance (provenance, audit, replay). This mirrors in-database ML extensions like BigQuery ML and Snowflake ML, which add new declarative constructs (e.g., `CREATE MODEL`) to SQL while pure-SQL operations work unchanged. The same primitives already express different optimization problems—subset selection, bipartite matching, multi-level batching, and job-shop scheduling (worked through in the language specification [17])—and like SQL, DeQL is designed to grow with new constructs and keywords over time (Section 5).

4 DECISION QUERY PROCESSING

A Decisionhouse ingests DeQL queries and returns decisions as new relational tables. The query-processing pipeline comprises four components connected by three intermediate representations, each marking a distinct commitment (Figure 3 traces the GPU query through them). The **Parser** produces **Abstract IR**: the decision problem stated symbolically over schema columns—variables, grouped constraints, objective. The **Decision Planner** produces **Concrete IR**, the chosen formulation (still symbolic), by detecting structural patterns in the Abstract IR (Section 4.2): for the GPU query, bipartite supply-demand structure selects min-cost flow over generic MILP. Committing the formulation against database statistics before materialization lets one Concrete IR be reused across

many Materialized IRs, one per data subset (SketchRefine-style decomposition [16]) or per parameter setting (what-if analysis).

The **Materializer** produces **Materialized IR** by binding Concrete IR's symbolic expressions to data fetched from federated sources (e.g., lakehouse, operational database, ML pipeline). The **Executor** dispatches Materialized IR to the solver (e.g., HiGHS/Gurobi/SCIP for LP/MILP, OR-Tools' cost-scaling solver for min-cost flow), writes the solution to the `DECIDE` output table, addressable by later SQL or DeQL queries, and logs the decision context (query, formulation, solver parameters, solution, timestamp) for audit and replay. This decoupling of formulation choice from solver dispatch lets future backends (e.g., GPU-accelerated LP solvers, network-flow hardware) plug in without changing the planner.

4.1 The Three-IR Design

The Abstract-to-Concrete boundary plays the same role as SQL planning: it separates *what* to optimize from *how* to formulate it. Because Abstract IR is symbolic, type checking, name resolution, and linearity checks all happen against it before data is fetched, rejecting invalid queries (e.g., non-linear objectives) early. The Concrete-to-Materialized boundary is what optimization adds beyond SQL planning: solving requires a global view of the data—a single numeric instance assembled from candidate rows—which must be materialized as a distinct stage. The materialization strategy depends on the formulation: a network flow needs supply and demand vectors indexed by node, while a generic MILP needs a flat coefficient matrix. Deferring data binding until planning is finalized lets the planner choose both the formulation and its materialization strategy [48, 55].

Abstract IR is the parser's structured representation of the problem. For the GPU query (Figure 2), it contains two decision variables keyed on `(pool_id, workload_id)`—a binary `active` flag and a continuous `quantity`—supply and demand constraints grouped via `BY`, and a linear cost-minimization objective, all stated symbolically over column references. Shortcuts like `SELECTION` desugar to explicit constraints (details in the language specification [17]).

Concrete IR encodes the chosen formulation. Growing the family of targets across new problem classes (and the structure-detection logic that recognizes them) is part of the research agenda (Section 5). Current targets include a generic MILP with a general constraint matrix, an LP relaxation that drops integrality, and a min-cost network flow as a bipartite supply-demand graph. For the GPU query, the planner selects min-cost flow because the bipartite `DECISION KEY` and supply-demand `BY` constraints match flow's structure (Section 4.2 details the criteria); the middle column of Figure 3 shows the resulting graph template.

Materialized IR is the data-populated form: the Concrete IR's symbolic expressions evaluated against actual rows. For the GPU query, the right column of Figure 3 populates the bipartite graph directly from Figure 2's three tables: pool nodes carry capacities from `gpu_pools`, workload nodes carry demands from `workloads`, and arcs carry costs from `assignments`—vectors that a min-cost-flow solver consumes directly. For LP/MILP targets the Materialized IR is the flat coefficient form (objective vector $\mathbf{c}$, constraint matrix $\mathbf{A}$, bound vectors $\mathbf{l}$, $\mathbf{u}$) instead of a graph. Decomposition methods like SketchRefine [16, 29, 46] reuse one Concrete IR across many Materialized IRs—once over cluster representatives, then once per refined cluster.

Correctness. Each stage of the three-IR pipeline (Abstract IR, Concrete IR, Materialized IR) must preserve the original DeQL query's constraints and objective. To establish this, it suffices to verify each transformation independently—desugaring, structure detection, formulation choice, materialization. Decomposition methods carry bounded-error theorems (e.g., PaQL/sPaQL [16, 19] for subset `SELECTION`). `WITHIN`/`TIMEOUT` budgets bound the solver's output. Extending the algebraic foundation [58] across the full language (including stochastic and relaxation [15] constructs) is an important research direction (Section 5).

4.2 The Decision Planner

MILP is the planner's default target, and structural patterns in the query enable more efficient polynomial-time alternatives such as min-cost flow [5], LP relaxation, or the Hungarian algorithm [38] when they apply. This choice can change the effective complexity class from NP-hard to polynomial (Section 4.3). We call the problem of selecting the fastest applicable formulation from a declarative query the *Decision Optimization Problem* (DeOP). Solvers optimize *within* a fixed coefficient matrix; the decision planner optimizes *which instance* the solver receives, using query structure and database statistics that no solver has access to.

Planner objective. The planner minimizes solve time subject to the user's quality budget (`WITHIN` X% or `TIMEOUT`; optimal by default). A simple rule yields large gains: prefer any applicable polynomial-time formulation (min-cost flow, LP relaxation, Hungarian) over MILP when structure detection succeeds, since the complexity-class ordering is known in advance and the gap is often orders of magnitude (Section 4.3). Refining this with a proper cost model (particularly for ranking multiple polynomial alternatives when several apply) is central to our research agenda (Section 5).

Structure detection from Abstract IR. The Abstract IR preserves structural patterns that flat coefficient matrices obscure. In the GPU allocation query (Figure 2), the `DECISION KEY (pool_id, workload_id)` reveals bipartite structure; the `BY` constraints match supply-demand flow conservation; the objective is linear in the flow variable. Together, these identify a min-cost network flow, enabling min-cost flow instead of branch-and-bound (Section 4.3). Single-component constraint groups with bare-variable sums reveal total unimodularity, so LP relaxation is exact (Hoffman–Kruskal [5]); bipartite queries whose constraint bounds are the literal one (e.g., `COUNT(*) = 1`) reduce to assignment, solvable by the Hungarian algorithm [38] in $O(n^3)$. Each is a deterministic pattern match on the IR, analogous to recognizing star joins from query graphs. The same patterns recur across decision workloads: bipartite supply-demand structures transportation, ad placement, and resource-allocation problems; total unimodularity holds for bipartite matching and assignment [5]. Extending the planner to detect further patterns is part of the research agenda (Section 5).

Statistics-driven planning. Not all structure is visible in the IR; some shows up only in the candidate data. The planner reads per-column statistics on the candidate set (`MIN`, `MAX`, `COUNT DISTINCT`, all-integer flag) computed at `CREATE CANDIDATES` time as a byproduct of the data scan, analogous to SQL `ANALYZE`. `COUNT(DISTINCT`

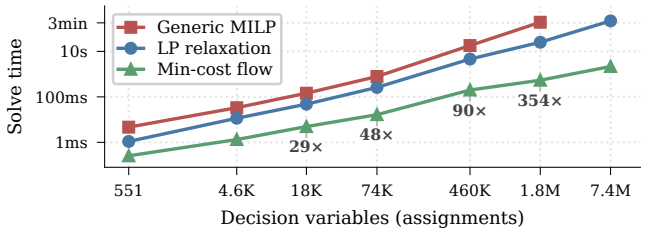

Figure 4: Solve time for the GPU allocation query (Figure 2). All three formulations produce identical optima up to 1.8M assignments; min-cost flow is 29–355× faster than MILP. At 7.4M, MILP times out (10 min); min-cost flow solves in 2.2 s.

`cost)` = 1 on a min-cost flow query with equality demands reduces it to max-flow, since a constant cost objective then makes any feasible flow optimal. Coefficients confined to $\{-1, 0, 1\}$ on candidate columns can complete a data-dependent total-unimodularity check [5] that the IR alone cannot finish, reducing MILP to LP relaxation (Section 5).

Planning under uncertainty. Stochastic queries [19, 29] introduce uncertainty in input attributes, adding another dimension to planning (Section 5).

4.3 Analysis

We are building DH-0, an initial Decisionhouse implementation in Rust, using Apache DataFusion [40] for query federation, Apache Arrow [6] as columnar format, and HiGHS [33]/Gurobi [28] for solving. The current release includes a DeQL parser, all three IRs, a working SketchRefine implementation, and direct solving. Data sources include DuckDB, ODBC databases, Microsoft Fabric, and all DataFusion-supported backends. DeQL Studio, an interactive interface to this prototype, is demonstrated in a companion paper [18].

Formulation selection. To evaluate automatic structure recognition, we solve the GPU allocation query from Figure 2 at seven scales up to 7.35M candidate assignments (consistent with hyperscale GPU clusters that reach $O(10^5)$ GPUs [39, 66]) using three formulations that the decision planner would select between: generic MILP (no recognition), LP relaxation (TU detection drops binary variables), and min-cost network flow (full network structure detection). Generic MILP and LP relaxation run on HiGHS [33] (branch-and-bound and dual simplex, respectively); min-cost flow runs on OR-Tools' [56] cost-scaling solver. Each formulation has a 10-minute time limit on random instances (median of 3 runs; Figure 4).

Speedup grows superlinearly (29× at 18K, 90× at 460K, 355× at 1.8M—from over 3 minutes to under 1 second). All three formulations return provably optimal solutions with identical objective values where they complete; the speedup comes entirely from structure recognition. The transportation matrix is totally unimodular [5], so LP relaxation produces integer solutions automatically (binary variables redundant); recognizing bipartite supply-demand additionally enables min-cost flow, changing the complexity class. At scale, structure recognition determines whether the query succeeds at all; a declarative system can detect and exploit structure the user did not specify. At 7.4M, MILP fails to find any feasible solution in 10 minutes while min-cost flow returns the optimum in 2.2 seconds. Which formulations apply depends on the query: without bipartite structure, min-cost flow is unavailable, and LP relaxation or MILP applies depending on total unimodularity. This experiment isolates one dimension of the DeOP: structure detection from the Abstract IR. Statistics-driven detection from candidate data is a second dimension (Section 4.2); execution-strategy choices like decomposition and pruning are a third dimension, an open research direction (Section 5).

Incremental materialization. When successive `DECIDE` queries share the same Abstract and Concrete IR—the common case in what-if analysis, where an analyst varies constraint parameters while the formulation stays fixed—only the Materialized IR changes, and only partially: bound vectors update but the constraint matrix sparsity pattern is unchanged. The system retains both the Materialized IR and the solver's internal state (preprocessed model and factored simplex basis), propagating only the changed bounds through the solver's incremental update API. The solver re-solves from its previous basis without reconstructing the model. On the GPU allocation benchmark, varying pool capacities across 50 what-if queries yields a per-query speedup that grows with problem size once the model is built and reused: 8–11× at 460K assignments and 12–15× at 1.8M.

5 RESEARCH AGENDA

We outline eight research directions tightly coupled to the Decisionhouse architecture.

Formal foundations for DeQL. DeQL needs an algebraic foundation analogous to relational algebra for SQL, covering typed decision variables, multi-table candidate spaces, grouped constraints, and non-linear or stochastic constructs (quadratic objectives, constraint programming, inline ML-model scoring via `PREDICT`, probabilistic predicates). Prior work covers subset selection [16, 19] and a relational algebra for decision queries via exponentiation [58]; extending to stochastic, probabilistic, and ML-augmented constructs remains open. Each new construct is admissible only via a translation rule and an equivalence theorem, making DeQL extensible.

Automatic reformulation and structure detection. Current systems have limited support for structure detection: CPLEX's network extractor [36] recovers pure node-arc substructure with specialized heuristics [14], but broader semantic patterns are lost once the problem is flattened to a coefficient matrix. The Abstract IR preserves these patterns. Detecting them draws on database query plan pattern matching [61] and OR reformulation theory [20, 60, 62]. New pattern detectors should plug into the planner without modifying the rest of the pipeline. Where syntactic rules and statistical methods fall short, the planner can draw on LLM judgment as a third input, with the DeQL query serving as the contract any chosen formulation must satisfy.

Cost models for decision query optimization. DeOP requires predicting execution cost and solution quality before committing to a plan. MILP solve time resists closed-form characterization; prediction errors of 10–100× are common even with ML models trained on millions of instances [35], partly because predictions operate on flat coefficient matrices with no semantic information. The three-IR design exposes a richer feature space: problem type (assignment vs. scheduling), constraint topology (bipartite keys, grouped supply-demand), candidate set size per dimension, and abstraction depth—analogous to the cardinality estimates and index statistics that make database cost models work.

Physical design for decision queries. Candidate sets introduce physical design questions absent from traditional query processing. Whether a candidate set should be virtual, in-memory, or persisted depends on join complexity, reuse across queries, solver access patterns, and data freshness. Physical layout must serve solver access patterns, which differ from query scans: the join structure in `CREATE CANDIDATES` groups candidates by dimension key, defining constraint matrix sparsity patterns to precompute once and reuse. When candidate sets derive from large joins, full materialization may be infeasible; factorized representations [54] avoid materializing joins for aggregation, but computing solver coefficients directly from factorized data remains open. The same materialization question applies to `DECIDE` output, which appends new decision columns to existing data columns.

Responsive execution. Decisions are coupled through shared constraints, so the row- and batch-independent scaling SQL and ML rely on does not apply directly. Decomposition methods like SketchRefine [16] partition the candidate space into subproblems and coordinate at the boundaries. The three-IR pipeline can then compute bounds at coarse abstraction levels before full materialization, and fall back to a different Concrete IR when the solver stalls. This mirrors adaptive query processing [10], except that adaptation crosses formulations rather than access paths. Several questions remain open. First, abstraction-based decomposition must be extended beyond package queries to scheduling, network flow, and assignment. Second, abstractions need to be designed so that they yield valid solutions at any interruption point. Third, scheduling across heterogeneous solver backends (where the formulation/dispatch split admits parallel formulation racing and per-block routing) requires new cost models and routing policies.

Optimization-workload benchmarks. The community needs a TPC-H equivalent for prescriptive workloads—spanning data access and optimization: diverse problem types (knapsack, assignment, scheduling), realistic scales, multiple constraint patterns, and ground-truth optima to compare systems and measure progress.

Decision governance. Decisions that affect people—hiring, allocation, scheduling—require an audit trail, infeasibility explanations, and traceability of outputs through formulation choices. The GDPR's "right to explanation" applies to automated decisions, yet no infrastructure directly supports it.

LLM-driven prescriptive pipelines. The decomposition databases are adopting (text-to-SQL [42, 44] + SQL-to-code synthesis [41, 68]) maps onto prescriptive analytics as text-to-DeQL + DeQL-to-code synthesis, with the declarative query as the stable intermediate. Text-to-DeQL benchmarks, validation methods for generated queries (a missing constraint silently yields an optimal solution to the wrong problem), and LLM-synthesized specialized execution are open problems.

6 RELATED WORK

Optimization tools and platforms. Commercial (Gurobi [28], CPLEX [36]) and open-source (HiGHS [33], SCIP [11]) solvers provide the algorithmic foundation for optimization. Modeling languages (AMPL [24], Pyomo [30], JuMP [22], MiniZinc [51], OR-Tools) add expressivity but require optimization expertise and lack database integration. OR platforms (FICO Xpress [23], AIMMS [12], IBM CPLEX Studio [36]) bundle these with data connectors and enterprise logging, covering everything from formulation onward but leaving formulation manual: models come in platform-specific languages, data is mapped imperatively, and formulations are selected by hand. Lakehouse integration runs through notebooks [21]. Algorithm selectors such as AutoFolio [45] choose among solvers from instance features but operate on already-serialized coefficient matrices; they cannot detect that a MILP encodes network-flow or transportation structure. Supply chain platforms [37, 53] embed domain-specific heuristics for planning but are not general-purpose. A Decisionhouse sits above all of these, using solvers as backends while providing data-native interfaces and automatic planning.

LLMs and optimization. LLMs that translate natural-language descriptions into solver code [3, 4, 27, 31, 43, 72] provide what we call *operational intelligence* (understanding which situation to model). They do not provide *mathematical intelligence*: each generated formulation is a one-off solver script with no specification to verify against, accuracy degrades with combinatorial structure and size [32, 49, 70], and no shared planning layer transfers across queries. When frontier models attempt to produce decisions directly without a solver, feasibility is unreliable across both models and domains [67]. We envision LLMs as complementary front-ends, translating user intent into DeQL, while a Decisionhouse provides verified translation, formulation selection, and guaranteed solving.

Decision support systems. Classical DSS [65] provide dashboards, what-if analysis, and scenario comparison to aid human judgment. A Decisionhouse computes optimal decisions from declarative specifications; classical DSS presents alternatives for manual evaluation.

In-database optimization. An early SQL-based decision query language was proposed by Brodsky et al. [13]. SolveDB [64] embedded solvers in PostgreSQL via UDFs; User-Defined Operators [63] compile custom algorithms into the query pipeline with zero-copy data access, but in both cases the solver executes after formulation selection is fixed; the database planner cannot see the mathematical structure. PaQL [16] pioneered declarative package queries, scaled by SketchRefine [59] to millions of candidates and Progressive Shading [46] to billions; Pratten et al. [58] provide an algebraic foundation for subset selection and optimization in relational algebra; DeciDB [34] extends DuckDB with a `DECIDE` clause inside `SELECT`. DeQL makes `DECIDE` a top-level statement and adds a structure-aware planner that detects properties of the constraint matrix (total unimodularity, network-flow shape, block-angular decomposability) and selects among LP, MILP, and decomposition formulations, which none of these systems, including commercial ones like LogicBlox [8] and RelationalAI [7], detect. DeQL extends SQL rather than replacing it, unlike SaneQL [52].

7 CONCLUSION

SQL and query optimizers eliminated bespoke data-retrieval programs; Lakehouses added first-class support for ML. Deciding (computing optimal actions under constraints) is next. As autonomous systems take on consequential decisions over data, the decision debt of ad-hoc pipelines—hard to audit, reproduce, or govern—is not sustainable. Decisionhouses provide the missing layer: DeQL decisions are declarative, automatically planned across formulations and solvers, and auditable end-to-end.

REFERENCES

[1] Aera Technology. 2024. Aera Decision Cloud: Cognitive Automation for the Enterprise. https://www.aeratechnology.com/

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*. ACM, 29–42. https://doi.org/10.1145/2465351.2465355

[3] Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. 2024. OptiMUS-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale. arXiv:2407.19633

[4] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024 (Proceedings of Machine Learning Research)*, Vol. 235. PMLR / OpenReview.net, 577–596. https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html

[5] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

[6] Apache Software Foundation. 2024. Apache Arrow: A cross-language development platform for in-memory analytics. https://arrow.apache.org

[7] Molham Aref, Paolo Guagliardo, George Kastrinis, Leonid Libkin, Victor Marsault, Wim Martens, Mary McGrath, Filip Murlak, Nathaniel Nystrom, Liat Peterfreund, Allison Rogers, Cristina Sirangelo, Domagoj Vrgoc, David Zhao, and Abdul Zreika. 2025. Rel: A Programming Language for Relational Data. In *Companion of the 2025 International Conference on Management of Data (SIGMOD/PODS)*. ACM, 283–296. https://doi.org/10.1145/3722212.3724450

[8] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1371–1382. https://doi.org/10.1145/2723372.2742796

[9] Michael Armbrust, Tathagata Das, Sameer Paranjpye, Reynold Xin, Shixiong Zhu, Ali Ghodsi, Burak Yavuz, Mukul Murthy, Joseph Torres, Liwen Sun, Peter Boncz, Mostafa Mokhtar, Herman Van Hovell, Adrian Ionescu, Alicja Luszczak, Michal Switakowski, Takuya Ueshin, Xiao Li, Michal Szafranski, Pieter Senster, and Matei Zaharia. 2020. Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3411–3424.

[10] Ron Avnur and Joseph M. Hellerstein. 2000. Eddies: Continuously Adaptive Query Processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*. ACM, 261–272. https://doi.org/10.1145/342009.335420

[11] Ksenia Bestuzheva, Mathieu Besançon, Weikun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros M. Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. 2023. Enabling Research through the SCIP Optimization Suite 8.0. *ACM Trans. Math. Software* 49, 2 (2023), 22:1–22:21. https://doi.org/10.1145/3585516

[12] Johannes Bisschop. 2006. *AIMMS – Optimization Modeling*. Lulu.com.

[13] Alexander Brodsky, Mayur M. Bhot, Manasa Chandrashekar, Nathan E. Egge, and Xiaoyang Sean Wang. 2009. A decisions query language (DQL): high-level abstraction for mathematical programming over databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*. ACM, 1059–1062. https://doi.org/10.1145/1559845.1559981

[14] Gerald G. Brown and William G. Wright. 1984. Automatic identification of embedded network rows in large-scale optimization models. *Mathematical Programming* 29, 1 (1984), 41–56. https://doi.org/10.1007/BF02591728

[15] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2014. Improving package recommendations through query relaxation. In *Proceedings of the First International Workshop on Bringing the Value of "Big Data" to Users, Data4U@VLDB 2014, Hangzhou, China, September 1, 2014*. ACM, 13. https://doi.org/10.1145/2658840.2658843

[16] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2018. Package queries: efficient and scalable computation of high-order constraints. *The VLDB Journal* 27, 5 (2018), 693–718. https://doi.org/10.1007/S00778-017-0483-4

[17] Matteo Brucato, Fjodor Kholodkov, Soren Little, Jakob Mayer, and Duc Nguyen. 2026. DeQL: A Decision Query Language for Prescriptive Analytics over Relational Data. arXiv:2606.19751

[18] Matteo Brucato, Fjodor Kholodkov, Soren Little, Jakob Mayer, and Duc Nguyen. 2026. DeQL Studio: Declarative Decision-Making over Relational Data. *Proceedings of the VLDB Endowment* 19, 12 (2026).

[19] Matteo Brucato, Nishant Yadav, Azza Abouzied, Peter J. Haas, and Alexandra Meliou. 2020. Stochastic Package Queries in Probabilistic Databases. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020*. ACM, 269–283. https://doi.org/10.1145/3318464.3389765

[20] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. 2010. Extended formulations in combinatorial optimization. *4OR* 8, 1 (2010), 1–48. https://doi.org/10.1007/S10288-010-0122-Z

[21] Databricks. 2024. Assembling Toy Brick Sets with Gurobi & Databricks: A Gentle Introduction to Optimization. https://www.databricks.com/blog/assembling-toy-brick-sets-gurobi-databricks-gentle-introduction-optimization

[22] Iain Dunning, Joey Huchette, and Miles Lubin. 2017. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Rev.* 59, 2 (2017), 295–320. https://doi.org/10.1137/15M1020575

[23] FICO. 2024. FICO Platform: Enterprise Decision Management. https://www.fico.com/en/fico-platform

[24] Robert Fourer, David M. Gay, and Brian W. Kernighan. 2003. *AMPL: A Modeling Language for Mathematical Programming* (2nd ed.). Thomson Brooks/Cole, Pacific Grove, CA.

[25] Davide Frazzetto, Thomas Dyhre Nielsen, Torben Bach Pedersen, and Laurynas Siksnys. 2019. Prescriptive analytics: a survey of emerging trends and technologies. *The VLDB Journal* 28, 4 (2019), 575–595. https://doi.org/10.1007/S00778-019-00539-Y

[26] Amélie Gheerbrant, Leonid Libkin, Liat Peterfreund, and Alexandra Rogova. 2025. GQL and SQL/PGQ: Theoretical Models and Expressive Power. *Proceedings of the VLDB Endowment* 18, 6 (2025), 1798–1810. https://doi.org/10.14778/3725688.3725707

[27] Gurobi Optimization. 2024. Gurobot. https://www.gurobi.com/solutions/gurobot/

[28] Gurobi Optimization, LLC. 2026. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[29] Riddho R. Haque, Anh L. Mai, Matteo Brucato, Azza Abouzied, Peter J. Haas, and Alexandra Meliou. 2025. Stochastic SketchRefine: Scaling In-Database Decision-Making under Uncertainty to Millions of Tuples. *Proceedings of the VLDB Endowment* 18, 9 (2025), 3106–3118.

[30] William E. Hart, Jean-Paul Watson, and David L. Woodruff. 2011. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation* 3, 3 (2011), 219–260. https://doi.org/10.1007/S12532-011-0026-8

[31] Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025. ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling. *Operations Research* 73, 6 (2025), 2986–3009. https://doi.org/10.1287/OPRE.2024.1233

[32] Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. 2024. When Large Language Model Meets Optimization. arXiv:2405.10098

[33] Qi Huangfu and J. A. J. Hall. 2018. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (2018), 119–142. https://doi.org/10.1007/S12532-017-0130-5

[34] HuDa Lab. 2026. DeciDB: Optimization, Native in SQL. https://decidb.com. Software (v0.1.0), MIT License.

[35] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm Runtime Prediction: Methods & Evaluation. *Artificial Intelligence* 206 (2014), 79–111.

[36] IBM. 2024. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio

[37] Kinaxis. 2024. Introducing Maestro: The First AI-Infused Supply Chain Orchestration Platform. https://www.kinaxis.com/en/news/press-releases/2024/introducing-maestro-first-ai-infused-supply-chain-orchestration-platform

[38] Harold W. Kuhn. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* 2, 1–2 (1955), 83–97. https://doi.org/10.1002/nav.3800020109

[39] Neeraj Kumar, Pol Mauri Ruiz, Vijay Menon, Igor Kabiljo, Mayank Pundir, Andrew Newell, Daniel Lee, Liyuan Wang, and Chunqiang Tang. 2024. Optimizing Resource Allocation in Hyperscale Datacenters: Scalability, Usability, and Experiences. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024*. USENIX Association, 507–528.

[40] Andrew Lamb, Yijie Shen, Daniël Heres, Jayjeet Chakraborty, Mehmet Ozan Kabak, Liang-Chi Hsieh, and Chao Sun. 2024. Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago, Chile, June 9-15, 2024*. ACM, 5–17. https://doi.org/10.1145/3626246.3653368

[41] Jiale Lao and Immanuel Trummer. 2026. GenDB: The Next Generation of Query Processing—Synthesized, Not Engineered. arXiv:2603.02081

[42] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24–28, 2025*. OpenReview.net. https://openreview.net/forum?id=XmProj9cPs

[43] Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. 2023. Large Language Models for Supply Chain Optimization. arXiv:2307.03875

[44] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[45] Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. 2015. AutoFolio: An Automatically Configured Algorithm Selector. *Journal of Artificial Intelligence Research* 53 (2015), 745–778.

[46] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2024. Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1146–1158. https://doi.org/10.14778/3641204.3641222

[47] Alexandra Meliou, Azza Abouzied, Peter J. Haas, Riddho R. Haque, Anh L. Mai, and Vasileios Vittis. 2025. Data Management Perspectives on Prescriptive Analytics. In *28th International Conference on Database Theory, ICDT 2025, Barcelona, Spain, March 25-28, 2025 (LIPIcs)*, Vol. 328. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:12. https://doi.org/10.4230/LIPICS.ICDT.2025.2

[48] Hubert Mohr-Daurat, Xuan Sun, and Holger Pirk. 2023. BOSS - An Architecture for Database Kernel Composition. *Proceedings of the VLDB Endowment* 17, 4 (2023), 877–890. https://doi.org/10.14778/3636218.3636239

[49] Mahdi Mostajabdaveh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. 2025. Evaluating LLM Reasoning in the Operations Research Domain with ORQA. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*. AAAI Press, 24902–24910. https://doi.org/10.1609/AAAI.V39I23.34673

[50] Samer B. Nashed, Justin Svegliato, Matteo Brucato, Connor Basich, Rod Grupen, and Shlomo Zilberstein. 2021. Solving Markov Decision Processes with Partial State Abstractions. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, 813–819. https://doi.org/10.1109/ICRA48506.2021.9561435

[51] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. 2007. MiniZinc: Towards a Standard CP Modelling Language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings (Lecture Notes in Computer Science)*, Vol. 4741. Springer, 529–543. https://doi.org/10.1007/978-3-540-74970-7_38

[52] Thomas Neumann and Viktor Leis. 2024. A Critique of Modern SQL and a Proposal Towards a Simple and Expressive Query Language. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. https://vldb.org/cidrdb/2024/a-critique-of-modern-sql-and-a-proposal-towards-a-simple-and-expressive-query-language.html

[53] o9 Solutions. 2024. AI-Powered Integrated Business Planning Platform. https://o9solutions.com/

[54] Dan Olteanu and Maximilian Schleich. 2016. Factorized Databases. *ACM SIGMOD Record* 45, 2 (2016), 5–16. https://doi.org/10.1145/3003665.3003667

[55] Pedro Pedreira, Orri Erling, Konstantinos Karanasos, Scott Schneider, Wes McKinney, Satyanarayana R. Valluri, Mohamed Zaït, and Jacques Nadeau. 2023. The Composable Data Management System Manifesto. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2679–2685. https://doi.org/10.14778/3603581.3603604

[56] Laurent Perron and Vincent Furnon. 2026. OR-Tools v9.15. Google. https://developers.google.com/optimization/

[57] Warren B. Powell. 2022. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. John Wiley & Sons, Hoboken, NJ.

[58] David Robert Pratten, Luke Mathieson, and Fahimeh Ramezani. 2025. Relational Algebras for Subset Selection and Optimisation. arXiv:2509.06439

[59] David F. Rogers, Robert D. Plante, Richard T. Wong, and James R. Evans. 1991. Aggregation and Disaggregation Techniques and Methodology in Optimization. *Operations Research* 39, 4 (1991), 553–582. https://doi.org/10.1287/OPRE.39.4.553

[60] Martin W. P. Savelsbergh. 1994. Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *INFORMS Journal on Computing* 6, 4 (1994), 445–454. https://doi.org/10.1287/IJOC.6.4.445

[61] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD*. 23–34. https://doi.org/10.1145/582095.582099

[62] Hanif D. Sherali and Warren P. Adams. 1999. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Nonconvex Optimization and Its Applications, Vol. 31. Springer US. https://doi.org/10.1007/978-1-4757-4388-3

[63] Moritz Sichert and Thomas Neumann. 2022. User-Defined Operators: Efficiently Integrating Custom Algorithms into Modern Databases. *Proceedings of the VLDB Endowment* 15, 5 (2022), 1119–1131. https://doi.org/10.14778/3510397.3510408

[64] Laurynas Siksnys and Torben Bach Pedersen. 2016. SolveDB: Integrating Optimization Problem Solvers Into SQL Databases. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM 2016, Budapest, Hungary, July 18-20, 2016*. ACM, 14:1–14:12. https://doi.org/10.1145/2949689.2949693

[65] Ralph H. Sprague, Jr. 1980. A Framework for the Development of Decision Support Systems. *MIS Quarterly* 4, 4 (1980), 1–26. https://doi.org/10.2307/248957

[66] Chunqiang Tang. 2025. Meta's Hyperscale Infrastructure: Overview and Insights. *Commun. ACM* 68, 2 (2025), 52–63. https://doi.org/10.1145/3701296

[67] Joseph Tso, Preston Schmittou, Quan Huynh, and Jibran Hutchins. 2026. ConstraintBench: Benchmarking LLM Constraint Reasoning on Direct Optimization. arXiv:2602.22465

[68] Johannes Wehrstein, Timo Eckmann, Matthias Jasny, and Carsten Binnig. 2026. Bespoke OLAP: Synthesizing Workload-Specific One-size-fits-one Database Engines. arXiv:2603.02001

[69] Laurence A. Wolsey. 2020. *Integer Programming* (2nd ed.). John Wiley & Sons.

[70] Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=fsDZwS49uY

[71] Matei Zaharia, Ali Ghodsi, Reynold Xin, and Michael Armbrust. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org. https://vldb.org/cidrdb/2021/lakehouse-a-new-generation-of-open-platforms-that-unify-data-warehousing-and-advanced-analytics.html

[72] Xinzhi Zhang, Zeyi Chen, Humishka Zope, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Janardhan Kulkarni, Ishai Menache, and Sirui Li. 2025. OptiMind: Teaching LLMs to Think Like Optimization Experts. arXiv:2509.22979

[73] Shlomo Zilberstein. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17, 3 (1996), 73–83. https://doi.org/10.1609/AIMAG.V17I3.1232