How Database Interfaces Shape Agent Performance: An Empirical Study of SQL and Executable Semantic Layers

Anonymous authors
Paper under double-blind review

Abstract

Database agents are typically built to query relational databases through SQL, but emerging semantic layers offer a fundamentally different agent-facing abstraction. We ask how the choice of agent-facing database interface shapes agent performance and failure modes. We systematically compare direct SQL access with an executable semantic-graph interface on BIRD Mini-Dev and LiveSQLBench-Lite. The results reveal that no interface dominates across workloads. On BIRD, graph settings are stronger, with the advantage increasing from 0.5 percentage points on Simple questions to 4.9 percentage points on Challenging questions. On LiveSQLBench-Lite, SQL leads by about four percentage points under both metadata conditions. Workload and agent-behavior analyses suggest that interface choice redistributes rather than eliminates reasoning difficulty: semantic structure can simplify identifying the relevant data and relationships, while analytically intensive workloads place greater pressure on constructing and finalizing queries. These findings establish the database interface as a first-class design choice for database agents and show that effective interface design depends on how reasoning complexity is allocated between reusable structure and query construction.

1 Introduction

Database agents are commonly formulated as text-to-SQL systems: given a natural-language request and a relational schema, the agent identifies relevant tables and columns, constructs SQL, executes it, and returns an answer. This formulation has driven substantial progress in schema linking, query generation, execution-guided refinement, and interactive database agents. It also embeds a largely implicit design choice: because relational database management systems expose SQL, the relational schema and SQL are typically treated as the natural interface for the agent.

That choice is no longer inevitable. Modern semantic layers expose a conceptual view of data—entities, attributes, relationships, metrics, and other governed abstractions—while mappings connect this view to relational storage. If the semantic layer is executable, an agent can query that abstraction directly rather than merely use it as auxiliary context for generating SQL. The agent-visible database interface then changes from tables, keys, and joins to semantic objects and relationships, even though the underlying data remain the same.

This shift raises a basic question for database-agent design: *what should the interface between an agent and a database be?* More specifically, how does replacing direct relational access with an executable semantic abstraction change agent performance, which tasks become easier or harder, and where failures occur? This question is distinct from asking whether additional schema documentation improves text-to-SQL. It treats the executable interface itself as a first-class design variable.

We study this question through matched comparisons of two agent-facing interfaces over the same underlying relational data: a *direct relational interface*, in which the agent reasons over tables, columns, keys, and SQL execution, and an *executable semantic interface*, instantiated as a virtual property graph queried through Cypher and mapped to PostgreSQL. To avoid conflating executable abstraction with richer textual context, we independently vary *descriptive schema information*—

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

natural-language descriptions and examples—within each interface. We also separate *interface answerability* from *agent performance*. For both benchmarks, we restrict the primary evaluation to a common answerable population: 450 BIRD Mini-Dev questions and 173 LiveSQLBench-Lite questions for which the benchmark target is stable and realizable through both interfaces.

The result is not a universal win for either interface. On BIRD Mini-Dev, graph settings outperform their matched SQL settings. Under descriptive schema information, the graph advantage grows from 0.5 percentage points on Simple questions to 3.0 points on Moderate questions and 4.9 points on Challenging questions. LiveSQLBench-Lite shows the opposite pattern: SQL leads graph by 4.1 percentage points with descriptive schema information and by 4.2 points with structural information alone. This advantage is not confined to one difficulty level; SQL remains ahead of its matched graph setting on Simple, Moderate, and Challenging questions. Difficulty alone therefore does not determine which interface is preferable—the interaction between interface and workload matters.

To understand this reversal, we examine both workload structure and agent behavior. A database agent must identify the right data—the relevant entities, attributes, relationships, identifiers, and populations—and construct the right query—the filters, aggregations, rankings, temporal operations, and computational stages needed to produce the answer. Our analyses suggest that changing the interface shifts where difficulty appears between these two activities. An executable semantic interface can encode recurring relationships and mapping decisions directly into the agent-visible abstraction, making some data-identification decisions easier. At the same time, the final computation must still be expressed in the interface's query language. LiveSQLBench-Lite contains substantially more CTEs, window operations, ranking, JSON access, and multi-stage analytical computation, and graph agents are much more likely than SQL agents to exhaust the interaction budget without producing a final executable query.

This paper makes three contributions:

- We formulate the *agent–database interface* as a first-class design dimension for database agents, and systematically study direct relational access and executable semantic graph access over the same underlying relational data.
- We develop a controlled evaluation framework for comparing database interfaces and apply it across BIRD Mini-Dev and LiveSQLBench-Lite, allowing us to systematically characterize their strengths and weaknesses and to examine whether observed differences arise from descriptive metadata or from the executable interface itself.
- We uncover a workload-dependent interface reversal and connect it to structured task and agent-behavior evidence. Graph access is stronger on BIRD, whereas SQL is stronger on the more analytically intensive LiveSQLBench-Lite workload. Analyses show that changing the interface redistributes failures between identifying the relevant data and constructing the required query, motivating workload-aware interface designs that allocate complexity between reusable structure and agent-side query construction.

2 Database Interfaces for Agents

2.1 Direct Relational Interface

The dominant database-agent setup exposes the physical relational organization of the database directly to the model. The agent observes tables, columns, types, primary and foreign keys, and possibly textual descriptions. It must map the user request to those objects, choose join paths and record populations, and express the required computation in SQL. We call this a *direct relational interface*. The interface is powerful and general, but physical database organization is part of the model's reasoning surface.

Formally, the agent performs a sequence of decisions of the form

$$\text{question} \rightarrow \text{relational objects} \rightarrow \text{SQL program} \rightarrow \text{DBMS execution}.$$

The SQL program may combine both data-access decisions, for example, which tables and keys represent a relationship, and analytical decisions such as filtering, grouping, ranking, temporal logic, and staged aggregation.

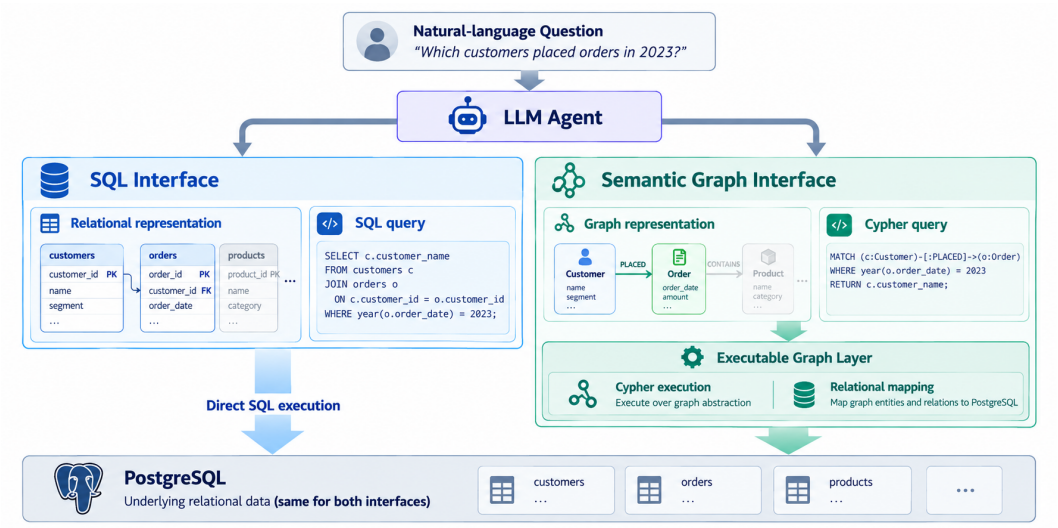


Figure 1: **Overview of our comparison between relational SQL and semantic graph interfaces for LLM-based database agents.** We evaluate two separate agent settings that differ in the database interface exposed to the LLM, while keeping the natural-language questions, agent configuration, and underlying PostgreSQL data fixed. In the *SQL setting*, the agent is presented with a relational representation of the database and generates SQL, which is executed directly on PostgreSQL. In the *semantic graph setting*, the agent is instead presented with a graph representation of the same database, where tables and their connections are exposed as entities, attributes, and relationships; it generates Cypher, which is executed through an executable graph layer that maps graph operations to the same underlying relational data. We compare agent performance across these two settings to study how the choice of database interface affects question answering.

2.2 Executable Semantic Interface

An executable semantic layer presents a different agent-visible abstraction. Domain entities and relationships become query primitives, while a mapping layer connects those semantic objects to the same underlying relational data. In our implementation, the agent observes graph nodes, properties, and edges, generates Cypher, and executes it through PuppyGraph, which maps the virtual property graph to PostgreSQL (PuppyGraph, 2026). The graph neither materializes benchmark answers nor replaces the source database; it changes the vocabulary through which the agent accesses it.

The resulting interaction is

$$\text{question} \rightarrow \text{semantic objects} \rightarrow \text{semantic query} \rightarrow \text{semantic graph engine} \rightarrow \text{DBMS execution}.$$

Named relationships can encode reusable connectivity decisions that a SQL agent would otherwise reconstruct from tables and keys. At the same time, the semantic query must still express the requested analytical operations, such as filters, ranking, temporal operations.

2.3 What Does It Mean to Compare Interfaces?

Two distinctions are essential. First, *interface expressibility* is not the same as *agent usability*. A failure to generate a correct query does not imply that the interface cannot express the target computation. We therefore define a shared answerable population before comparing agent accuracy and, where possible, verify graph expressibility with oracle reference-query translations.

Second, *semantic structure* is not the same as *descriptive metadata*. A graph already changes executable structure by exposing entities and relationships as queryable objects. Natural-language descriptions and examples are an additional source of information that can be supplied to either interface. Separating these factors is necessary to determine whether an observed effect comes from the executable abstraction or simply from richer textual context.

Table 1: Four experimental settings. Descriptive metadata is a control axis; graph topology is fixed between Graph–Structural and Graph–Descriptive.

Setting	Agent-visible representation	Query language	Descriptive metadata
SQL–Structural	Tables, columns, types, and keys	SQL	No
SQL–Descriptive	Same relational structure plus descriptions	SQL	Yes
Graph–Structural	Nodes, properties, types, and directed edges	Cypher	No
Graph–Descriptive	Same graph structure plus descriptions/examples	Cypher	Yes

These distinctions lead to three empirical questions. **RQ1:** Does an executable semantic interface reliably replace direct SQL access, or is the interface effect workload-dependent? **RQ2:** What observable workload and agent-behavior characteristics accompany the direction of that effect? **RQ3:** How does changing the executable abstraction redistribute agent failures?

3 Experimental Framework

3.1 Controlled Interface Comparison

We instantiate the relational interface with PostgreSQL SQL and the graph interface with Cypher over a PuppyGraph virtual graph mapped to the same PostgreSQL source. The graph schema for each database is generated using PuppyGraph's built-in schema-construction agent. Within each benchmark, the settings share the source data, read-only ReAct-style agent loop (Yao et al., 2023), model, iteration budget, benchmark-knowledge scope, and evaluation logic. The comparison is designed to vary the executable interface while keeping the underlying task and agent protocol fixed.

To separate the effect of the executable interface from that of richer natural-language schema information, we independently vary the amount of descriptive metadata exposed to the agent. This yields the four settings in Table 1. We use *Structural* to denote schema and topology information without descriptive meanings, and *Descriptive* to denote the same executable structure augmented with faithful descriptions and examples. This factorial design allows us to compare SQL and graph interfaces under matched metadata conditions while separately examining the effect of descriptive metadata within each interface.

3.2 Benchmarks and Experiment Protocol

BIRD Mini-Dev contains 500 questions over 11 databases (Li et al., 2023). LiveSQLBench-Lite contains 180 questions over 18 databases and places greater emphasis on multi-stage analytical queries (BIRD Team, 2025). We use GPT-5 (OpenAI, 2025) as the underlying language model for all agent configurations on both benchmarks. Our analysis focuses on matched *within-benchmark* comparisons between relational and graph interfaces, rather than absolute accuracy comparisons across benchmarks, whose workloads and evaluation protocols differ. To account for stochastic variation in agent execution, we independently run each experimental configuration three times under the same benchmark and interface conditions. Unless otherwise specified, we report performance aggregated across the three runs.

3.3 Common Answerable Population

A fair agent comparison requires separating targets that an interface can represent from programs that an agent can successfully discover. We therefore define a common primary population for which the benchmark target is stable and the deployed interfaces are comparable.

For BIRD, an independent audit identifies 39 benchmark-invalid cases and 11 PuppyGraph execution or representation non-comparable cases. We exclude the same 50 cases from all four settings, leaving 450 questions. For LiveSQLBench-Lite, seven malformed or nondeterministic samples are removed, leaving 173 valid deterministic questions per round. An oracle audit translates and probes

the retained LiveSQLBench-Lite samples through the compatible graph representation: all 173 have graph-equivalent answers, including 152 automatic/set matches and 21 conservative semantic equivalents. Thus, a graph-agent failure on the retained LiveSQLBench-Lite set is not, by itself, evidence that the target is inexpressible through the graph interface.

3.4 Evaluation Protocol

We use a two-stage evaluation procedure that combines rule-based execution-result comparison with LLM-based adjudication. We first apply the benchmark's rule-based execution evaluator to compare the predicted and reference query results. Predictions marked incorrect by this evaluator are then reviewed by Claude Opus 5 (Anthropic, 2026) to identify false negatives caused by semantically equivalent but representationally different outputs. The same judge and adjudication criteria are applied to all SQL and graph settings.

Our adjudication is intentionally invariant to presentation differences that do not change the answer. Redundant columns, auxiliary information, aliases, ordering when not required by the question, and equivalent output representations do not make an otherwise correct result incorrect. In contrast, predictions remain incorrect when they materially change the requested entities, filters, aggregation grain, ranking or selection logic, formulas, or answer values. This answer-equivalence criterion allows false negatives from rule-based comparison to be corrected while preserving a common semantic standard across the two interfaces despite differences in query language and result representation.

4 Does the Database Interface Matter?

4.1 The Preferred Interface Reverses Across Workloads

Table 2 summarizes the overall performance of the two interfaces under matched schema-information conditions. On BIRD Mini-Dev, the executable semantic interface performs better than direct SQL access under both conditions. Graph–Structural reaches 78.1% compared with 77.1% for SQL–Structural, a gain of 1.0 percentage point, while Graph–Descriptive reaches 80.1% compared with 77.5% for SQL–Descriptive, a gain of 2.6 points.

The direction reverses on LiveSQLBench-Lite. Graph–Structural reaches 53.0% compared with 57.2% for SQL–Structural, a difference of -4.2 percentage points, while Graph–Descriptive reaches 52.4% compared with 56.5% for SQL–Descriptive, a difference of -4.1 points. Thus, neither interface provides a consistent advantage across the two workloads: the semantic interface is stronger on BIRD, whereas direct SQL access is stronger on LiveSQLBench-Lite.

Table 2: Overall accuracy (%) and matched interface effects. Δ denotes Graph minus SQL accuracy in percentage points (pp); positive values indicate higher Graph accuracy and negative values indicate higher SQL accuracy.

	Structural			Descriptive		
Benchmark	SQL	Graph	Δ	SQL	Graph	Δ
BIRD Mini-Dev	77.1	78.1	$+1.0$	77.5	**80.1**	$+2.6$
LiveSQLBench-Lite	**57.2**	53.0	-4.2	56.5	52.4	-4.1

4.2 The Interface Effect Varies with Workload Difficulty

The aggregate reversal remains visible when performance is stratified by benchmark difficulty (Table 3), but the interaction with difficulty differs substantially across workloads. On BIRD, the advantage of the semantic interface becomes larger on more difficult questions. Under descriptive schema information, Graph exceeds SQL by 0.5 percentage points on Simple questions, 3.0 points on Moderate questions, and 4.9 points on Challenging questions. The same direction appears under structural schema information, where the Graph–SQL difference increases from 0.5 points on Simple questions to 0.9 points on Moderate questions and 1.9 points on Challenging questions.

LiveSQLBench-Lite exhibits a different pattern. SQL remains ahead of its matched graph setting at every difficulty level under both schema-information conditions. With descriptive schema information, the Graph–SQL differences are -0.8, -5.4, and -4.5 percentage points for Simple, Moderate, and Challenging questions, respectively. Under structural schema information, the corresponding differences are -4.8, -3.9, and -4.5 points.

These results rule out a simple interpretation in which semantic interfaces become preferable merely as questions become more difficult. Higher benchmark difficulty is associated with a growing graph advantage on BIRD, but not on LiveSQLBench-Lite, where SQL retains an advantage across all three levels. The effect of the database interface therefore depends on the characteristics of the workload, motivating a closer examination of what computations the two benchmarks require and how the corresponding agents fail.

Table 3: Accuracy (%) by benchmark difficulty. Δ denotes Graph minus SQL accuracy in percentage points (pp).

		Structural			Descriptive		
Benchmark	Difficulty	SQL	Graph	Δ	SQL	Graph	Δ
BIRD Mini-Dev	Simple	85.3	85.8	$+0.5$	86.5	87.0	$+0.5$
	Moderate	75.1	76.0	$+0.9$	74.8	77.8	$+3.0$
	Challenging	68.9	70.8	$+1.9$	69.7	74.6	$+4.9$
LiveSQLBench-Lite	Simple	76.2	71.4	-4.8	70.6	69.8	-0.8
	Moderate	50.4	46.5	-3.9	52.7	47.3	-5.4
	Challenging	52.6	48.1	-4.5	50.4	45.9	-4.5

5 What Determines Which Interface Works Better?

The benchmark-level reversal suggests that the value of an interface depends on the structure of the task it is used to solve. We distinguish two broad sources of query complexity. *Relationship navigation* concerns identifying and connecting the relevant entities, relationships, and populations. *Analytical composition* concerns organizing those data into multi-stage computations involving derived metrics, aggregation, ranking, temporal logic, or structured outputs. We use reference SQL as an observable proxy for these task characteristics and then examine how each regime contributes to the Graph–SQL performance difference.

5.1 The Benchmarks Emphasize Different Task Regimes

Table 4 summarizes syntactic characteristics of the reference SQL over the benchmarks. Both benchmarks require substantial relational navigation: approximately 80% of queries in each contain at least one explicit JOIN. The main difference lies in the computational structure built around those relationships. LiveSQLBench-Lite contains substantially more multi-stage queries: 68.2% use a CTE, 71.7% contain multiple SELECT blocks, 20.8% use window operations, and 32.4% use JSON operations. These constructs are uncommon or absent in BIRD Mini-Dev.

To make this distinction more explicit, we derive a simple task taxonomy from the reference programs. We label a task *analytical-heavy* when its reference program contains analytical structures such as window or JSON operations, percentile or set operations, LATERAL computation, multiple CTE stages, or at least three SELECT stages. Among the remaining tasks, those containing an explicit relational join are labeled *relationship-centric*; the rest are labeled *local/simple*. These categories are workload descriptors rather than intrinsic measures of question difficulty.

Table 5 reveals a sharp difference in workload composition. 78.4% of BIRD Mini-Dev questions are relationship-centric, whereas only 3.3% fall into the analytical-heavy category. LiveSQLBench-Lite shows the opposite profile: 64.2% of its questions are analytical-heavy, while only 26.0% are relationship-centric. Thus, the distinction between the two benchmarks is not whether relationships must be traversed but whether relationship navigation constitutes most of the query structure or is embedded within a substantially richer analytical program.

Table 4: Benchmark characteristics. JOIN counts are computed from explicit JOIN operators; remaining entries report the share of queries containing the corresponding construct.

Feature	BIRD Mini-Dev ($n = 450$)	LiveSQLBench-Lite ($n = 173$)
Mean explicit JOINs	1.04	2.15
At least one JOIN	80.9%	78.6%
At least two JOINs	19.3%	53.2%
CTE	1.1%	68.2%
Multiple SELECT blocks	11.6%	71.7%
Window operation	1.1%	20.8%
Explicit ranking function	1.1%	16.8%
JSON operation	0.0%	32.4%
Percentile aggregate	0.0%	2.3%

Table 5: Distribution of reference queries across task regimes in the evaluation sets.

Task regime	BIRD Mini-Dev	LiveSQLBench-Lite
Local/simple	18.2%	9.8%
Relationship-centric	**78.4%**	26.0%
Analytical-heavy	3.3%	**64.2%**

5.2 Different Task Regimes Drive the Interface Effect

We next connect workload structure to agent performance. Because the three regimes differ substantially in size, directly comparing within-regime percentage-point differences can overemphasize small buckets. We therefore report each regime's *weighted interface effect*, defined as

$$C_r = \frac{n_r}{N} \left(\mathrm{Acc}_{\mathrm{Graph},r} - \mathrm{Acc}_{\mathrm{SQL},r} \right),$$

where n_r is the number of questions in regime r and N is the benchmark size. C_r measures how many percentage points that regime contributes to the overall Graph–SQL accuracy gap. Positive values favor Graph and negative values favor SQL; the regime contributions sum to the overall interface difference for the analyzed round.

Table 6: Contribution of each task regime to the overall interface effect. Values are weighted Graph-minus-SQL effects in percentage points (%).

Benchmark	Task regime	Structural	Descriptive
BIRD Mini-Dev	Local/simple	$+0.2$	0.0
	Relationship-centric	$\mathbf{+2.4}$	$\mathbf{+3.1}$
	Analytical-heavy	$+0.2$	$+0.4$
LiveSQLBench-Lite	Local/simple	-1.2	-0.6
	Relationship-centric	0.0	-1.2
	Analytical-heavy	$\mathbf{-1.7}$	$\mathbf{-4.6}$

The decomposition reveals complementary task regimes. On BIRD, the graph advantage is driven primarily by relationship-centric questions: they contribute $+2.4$ points under structural metadata and $+3.1$ points under descriptive metadata, while the other regimes contribute comparatively little. On LiveSQLBench-Lite, the largest contribution instead comes from analytical-heavy questions, which contribute -1.7 and -4.6 points under the structural and descriptive conditions, respectively. Thus, the benchmark-level reversal is not merely associated with different workload mixtures: the dominant task regime in each benchmark also accounts for the largest share of the corresponding interface effect.

5.3 Why Do the Regimes Favor Different Interfaces?

The BIRD results are consistent with the graph interface reducing part of the data-navigation burden. In SQL, the agent must map the question to physical tables, keys, and join paths; in the graph interface, recurring relationships are exposed directly as executable edges. This can simplify tasks whose main challenge is identifying the relevant entities, relationships, and population, especially when the downstream computation is relatively direct. BIRD disagreement cases support this interpretation, with graph-favored examples frequently involving relationship, population, and terminal-entity selection across the relevant database schema.

LiveSQLBench-Lite shifts the difficulty toward analytical composition. Relationship navigation is often followed by multiple computational stages, including derived metrics, aggregation, ranking, temporal operations, and structured outputs. These stages must still be constructed in Cypher. Consistent with this burden, graph agents fail to produce a final executable query in 75 run-cases across the two graph settings, compared with only one such failure across the SQL settings. Inspection of failed runs indicates that these failures often arise while agents are constructing ranking, list, temporal, or staged-aggregation constructions rather than from a lack of graph answerability.

Together, these results suggest that interface choice redistributes rather than removes complexity. Semantic graph access is most useful when relationship navigation dominates the task, whereas SQL is more advantageous when navigation is embedded within a richer analytical program. We interpret these patterns as converging evidence rather than a causal decomposition of query difficulty. The alignment between workload structure, regime-level performance, and agent behavior nevertheless provides consistent evidence for how different interfaces shift the agent's reasoning burden.

6 What Role Does Descriptive Metadata Play?

Descriptive metadata has only a modest and workload-dependent effect. On BIRD, adding descriptions improves graph accuracy by +2.0 percentage points and SQL accuracy by +0.4 points; within the graph setting, the gain grows with benchmark difficulty. On LiveSQLBench-Lite, however, descriptive metadata slightly reduces pooled accuracy for both graph (-0.6 points) and SQL (-0.7 points), while paired outcomes show substantial case turnover despite the small net effect.

Detailed audits suggest a two-sided role: descriptions can clarify subtype, relationship path, metric meaning, and intended grain, but can also increase ambiguity by making multiple identifier roles or plausible query plans salient. Overall, descriptive metadata should be treated as a separate control from the executable interface itself: richer descriptions can change which cases an agent solves, but they do not explain the workload-dependent Graph–SQL reversal.

7 Implications for Database-Agent Design

7.1 Interface Design Is Complexity Allocation

A database interface does more than expose data: it determines which decisions must be reconstructed by the agent and which are encoded in reusable system structure. A useful design question is therefore not simply how much semantic information an interface provides, but *which decisions it removes from the model and which computations it makes easier or harder to express*.

Our results suggest that good interface design should minimize the forms of complexity that dominate the target workload rather than minimize interface complexity in the abstract. An interface is useful when it moves recurring, error-prone decisions out of the generated program without making the remaining computation substantially harder to construct. From this perspective, interface design is a problem of allocating complexity between the system and the agent.

7.2 Semantic Layers Should Be Designed as Agent-Facing Abstractions

Semantic layers need not be justified as universal replacements for relational access or SQL. In database-agent systems, their value can instead come from serving as controlled agent-facing ab-

stractions: exposing stable domain entities and relationships, constraining valid access paths, and making recurring database organization explicit.

The important design question is what work remains for the agent after this abstraction has been introduced. More generally, database agents need not rely on a single interface for every stage of every task. System designers can treat semantic structure, executable access, and analytical operators as composable design choices, provided that the resulting interface reduces rather than merely relocates the agent's reasoning burden.

More broadly, database-agent interfaces should be evaluated not only by how intuitive or semantically rich they appear, but by the reasoning work they induce at query time. The most effective interface is one that encodes reusable database decisions while leaving the agent with a query-construction problem well matched to the workload.

8 Related Work

Text-to-SQL and database agents. Text-to-SQL research has traditionally treated the relational schema and SQL as the executable interface. Earlier systems explicitly model schema relations or introduce intermediate representations to simplify SQL generation (Wang et al., 2020; Guo et al., 2019; Gan et al., 2021). Recent LLM-based systems increasingly decompose schema selection, generation, execution, and refinement (Pourreza & Rafiei, 2023; Talaei et al., 2024), while newer benchmarks emphasize large schemas, metadata, and multi-step workflows (Lei et al., 2025). Interactive database agents further use execution feedback and iterative reasoning rather than one-shot generation (Li et al., 2025; Huo et al., 2026). These approaches improve reasoning over relational databases but generally retain direct relational access as the endpoint. Our work asks whether that endpoint itself should be treated as a design choice.

Semantic layers, ontologies, and knowledge graphs. Ontology-based data access and virtual knowledge-graph systems expose conceptual entities and relationships while mappings connect them to physical sources (Calvanese et al., 2017; Xiao et al., 2019). Recent LLM-based work uses knowledge graphs to bridge user terminology and relational schemas or compares question answering across relational and graph representations (Sequeda et al., 2024; Nascimento et al., 2026). Concurrent work also studies semantic layers as intermediate representations compiled into SQL (Kim et al., 2026) and disentangles structural from semantic schema information for LLM-based SQL generation (Su et al., 2026). Our study is complementary: rather than introducing a new semantic-layer mechanism, we evaluate the semantic abstraction as the *executable agent interface* and control descriptive metadata separately.

Executable graph interfaces. Property graphs provide an alternative executable surface in which entities and relationships are directly queried through languages such as Cypher (Francis et al., 2018). Benchmarks such as CypherBench show that graph-query generation introduces its own grounding and reasoning challenges (Feng et al., 2025). Our setting uses a graph interface not because graph query generation is itself the target task, but because it provides an executable semantic abstraction over the same relational source. This enables a controlled study of how moving from physical relational objects to semantic entities and relationships changes agent behavior.

9 Conclusion

Database agents must both identify the relevant data and construct the computation needed to answer a question. Our results show that the interface through which an agent accesses a database changes how these two difficulties are distributed. In matched comparisons, executable semantic graphs perform better on BIRD Mini-Dev, whereas SQL performs better on the more analytically structured LiveSQLBench-Lite workload. Task-regime analysis further shows that BIRD's graph advantage is driven primarily by relationship-centric questions, while LiveSQLBench-Lite's SQL advantage is driven primarily by analytical-heavy questions. Executable graph structure can reduce recurring relationship-navigation and schema-linking decisions, but the resulting analytical program must still be constructed and finalized. As analytical composition becomes more demanding, that program-construction burden can outweigh the benefit of the semantic abstraction.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, rewriting, and organization of portions of the manuscript, as well as with consistency checks during paper preparation. All AI-assisted text was reviewed and edited by the authors. Experimental results and analyses were checked against the underlying evaluation outputs. The authors take responsibility for the final content, claims, and artifacts presented in this work.

REFERENCES

Anthropic. Introducing claude opus 5. https://www.anthropic.com/news/claude-opus-5, July 2026. Accessed: 2026-09-25.

BIRD Team. LiveSQLBench: A dynamic and contamination-free benchmark for evaluating LLMs on real-world text-to-SQL tasks. https://livesqlbench.ai/, 2025. Benchmark artifact; accessed 2026-08-27.

Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017. doi: 10.3233/SW-160217.

Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. CypherBench: Towards precise retrieval over full-scale modern knowledge graphs in the LLM era. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8934–8958. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.438. URL https://aclanthology.org/2025.acl-long.438/.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andres Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 1433–1445. ACM, 2018. doi: 10.1145/3183713.3190657.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. Natural SQL: Making SQL easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 2030–2042. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.174. URL https://aclanthology.org/2021.findings-emnlp.174/.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4524–4535. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1444. URL https://aclanthology.org/P19-1444/.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. Bird-interact: Re-imagining text-to-sql evaluation via lens of dynamic interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nHrYBGujps.

Ha Jeong Kim, Saksonita Khoeurn, and Ye Ji Yoon. A semantic-layer-mediated agent for natural language to SQL over heterogeneous enterprise databases. *arXiv preprint arXiv:2606.31041*, 2026. URL https://arxiv.org/abs/2606.31041. Submitted to FITAT 2026 for peer review.

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 I. Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=XmProj9cPs.

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 Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 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*, volume 36, 2023. doi: 10.52202/075280-1835.

Jinyang Li, Xiaolong Li, Ge Qu, Per Jacobsson, Bowen Qin, Binyuan Hui, Shuzheng Si, Nan Huo, Xiaohan Xu, Yue Zhang, Ziwei Tang, Yuanshuai Li, Florensia Widjaja, Xintong Zhu, Feige Zhou, Yongfeng Huang, Yannis Papakonstantinou, Fatma Ozcan, Chenhao Ma, and Reynold Cheng. Swe-sql: Illuminating llm pathways to solve user sql issues in real-world applications. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3247.

Eduardo R. S. Nascimento, Caio Viktor S. Avila, Yenier Torres Izquierdo, Grettel M. García, Lucas Feijó L. Andrade, Matheus O. Silva, Michelle S. P. Facina, Melissa Lemos, and Marco A. Casanova. A text-to-sql strategy based on large language models and knowledge graphs for real-world databases. *Data & Knowledge Engineering*, 164:102580, 2026. doi: 10.1016/j.datak.2026.102580.

OpenAI. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`, August 2025. Accessed: 2026-09-25.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1577.

PuppyGraph. Querying PostgreSQL data as a graph. PuppyGraph Documentation, 2026. URL `https://docs.puppygraph.com/`. Accessed 2026-08-27.

Juan Sequeda, Dean Allemang, and Bryon Jacob. A benchmark to understand the role of knowledge graphs on large language model's accuracy for question answering on enterprise sql databases. In *Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems and Network Data Analytics*, GRADES-NDA '24, pp. 1–12. Association for Computing Machinery, 2024. doi: 10.1145/3661304.3661901.

Daniel Yitian Su, Sophie Yiran Su, Qiang Sun, Yihao Ding, and Wei Liu. Disentangling structure and semantics: How schema representation affects LLM-based SQL generation. *arXiv preprint arXiv:2608.20356*, 2026. URL `https://arxiv.org/abs/2608.20356`. Work in progress.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7567–7578. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.677. URL `https://aclanthology.org/2020.acl-main.677/`.

Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. Virtual knowledge graphs: An overview of systems and use cases. *Data Intelligence*, 1(3):201–223, 2019. doi: 10.1162/dint_a_00011.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=WE_vluYUL-X`.