QueryGym: Step-by-Step Interaction with Relational Databases

Haritha Ananthakrishnan[1], **Harsha Kokel**[1], **Kelsey Sikes**[2*],
Debarun Bhattacharjya[1], **Michael Katz**[1], **Shirin Sohrabi**[1], **Kavitha Srinivas**[1]

[1]IBM Research
[2]Colorado State University

Abstract

We introduce **QueryGym**, an interactive environment for building, testing, and evaluating LLM-based query planning agents. Existing frameworks often tie agents to specific query language dialects or obscure their reasoning; QueryGym instead requires agents to construct explicit sequences of relational algebra operations, ensuring engine-agnostic evaluation and transparent step-by-step planning. The environment is implemented as a Gymnasium interface that supplies observations—including schema details, intermediate results, and execution feedback—and receives actions that represent database exploration (e.g., previewing tables, sampling column values, retrieving unique values) as well as relational algebra operations (e.g., filter, project, join). We detail the motivation and the design of the environment. In the demo, we showcase the utility of the environment by contrasting it with contemporary LLMs that query databases. QueryGym serves as a practical testbed for research in error remediation, transparency, and reinforcement learning for query generation.

Demo — https://ibm.biz/QueryGym
Extended version — https://arxiv.org/abs/2509.21674

1 Introduction

The ability to translate natural language questions into executable database queries—across SQL and other query languages—has become a standard benchmark for evaluating an LLM's ability to reason over relational databases. Recent advances in LLMs' reasoning capabilities have dramatically improved performance of NL2Query benchmarks such as Spider and BIRDBench (Lei et al. 2025; Li et al. 2023; Granado, Lotufo, and Pereira 2025; Wang et al. 2025; Pourreza et al. 2025; Papicchio et al. 2025). However, a systematic understanding of how these models plan, explore, and manipulate relational data remains limited. Existing evaluation suites typically treat query generation as a static sequence-to-sequence problem, providing a single gold-standard SQL string and scoring the final output. This paradigm obscures intermediate reasoning, blocks error analysis, and rules out reinforcement learning methods that depend on an interactive agent-database loop.

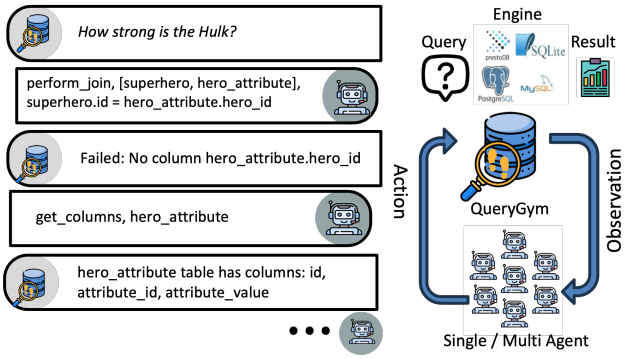

Figure 1: Sample trajectory in QueryGym, an interactive environment for query planning agents.

To address these gaps, we introduce **QueryGym**, an interactive, engine-agnostic environment that casts NL2Query tasks as a partially observable Markov Decision Process (POMDP). In QueryGym, an agent must interact with the database to answer a natural language question; see Fig. 1 for an example. The environment supplies rich textual observations—including schema metadata, intermediate table previews, and error feedback—while accepting actions that correspond to both exploratory probes (e.g., previewing a table, sampling column values) and relational algebra manipulations. By abstracting away any concrete query language (e.g., SQL, PostgreSQL), QueryGym enables truly engine-agnostic research, and its interactive database access naturally supports use-cases such as error remediation. Together, these features make QueryGym a versatile testbed for exploring and improving LLM-based query planning agents in a controlled, interactive setting.

2 QueryGym Environment

We cast NL2Query tasks (as instantiated in SQL or other query languages) as a POMDP tuple (S, A, Ω, T, R), where S is the set of environment states, A is the set of actions, Ω is the set of observations, $T : S \times A \to S$ is the deterministic transition function between states, and $R : S \times A \to \mathcal{R}$ is the reward function.

*Work done while at IBM Research

Environment States The state in QueryGym comprises of the full schema and database contents, the natural language question, and any intermediate tables, Common Table Expressions (CTEs) that the agent has materialized so far.

Actions The action space consists of 12 exploration operations (such as `preview_table` and `get_sample_values`) that probe the database, and 8 relational algebra operations (such as `perform_filter`, `perform_join`, and `perform_union`) that manipulate the tables. Each of these operations is associated with a fixed number of parameters. At each turn, the agent is expected to issue a command that defines the chosen operation to be performed and supplies concrete values for all its parameters.

Observation The textual observation supplied by the QueryGym at any given turn depends on the environment state and the previous command. They can be broadly categorized into 4 classes: *Overview*, the information about the current database schema and the natural language question; *Exploration Result*, the information requested by the explore operation; *Intermediate CTE Info*, the intermediate table that was created as a result of the last relational algebra operation; *Error Feedback*, error trace of any failure that occurred when attempting to apply a relational algebra operation, such as a missing column or unknown keyword.

Transition Function Currently, the transition function in QueryGym treats the underlying database as immutable. The exploration actions do not modify the state of the environment; they only furnish the requested information in the observation as text. For relational algebra operations, the transition function constructs a new CTE if the operation is successful, otherwise the error is passed to the observation and the state of the environment remains the same.

Reward Function An episode terminates as soon as the current CTE is *equivalent* to the target table, i.e., it contains exactly the same set of rows and columns, irrespective of ordering and column names. A large terminal reward is provided upon termination, whereas a small reward is provided when the CTE is a subset or superset of the target table.

Scope QueryGym currently supports translation of the relational algebra operations to SQLite and PostgreSQL but other SQL dialects could be easily supported by implementing a single class compatible with Gym API. Our system also automatically converts any NL2Query dataset into a POMDP, if provided in the BIRDBench format. Benchmarks supported by our system include: BIRDBench (Li et al. 2023), WikiSQL (Zhong, Xiong, and Socher 2017), Criteria2SQL (Yu et al. 2020), ACL SQL (Kaoshik et al. 2021), CoSQL (Yu et al. 2019a), FIBEN (Sen et al. 2020), SEOSS-Queries (Tomova, Hofmann, and Mäder 2022), SParC (Yu et al. 2019b), Spider-Syn (Gan et al. 2021), SQUALL (Shi et al. 2020) and BIRD-Critic (Li et al. 2025).

Agent We also provide a sample implementation of a LangChain-based Gymnasium agent that leverages a vLLM-hosted LLM to solve the task by iteratively generating actions from textual observations, handling error feedback, and progressively constructing the correct relational algebra plan until the final CTE matches the target table.

3 Use Cases

Engine Agnostic Research QueryGym enables researchers to develop, train, and evaluate query planning agents without being tied to any particular SQL dialect or database system. This design supports cross-engine comparisons, transfer-learning studies, and the creation of truly portable query-generation strategies that can be deployed on SQLite, PostgreSQL, MySQL, or any future relational engine with minimal adaptation.

Database Exploration Current NL2Query pipelines rely on schema linking, where relevant tables and columns are first identified given the question before generating a query. Instead of depending on this schema linking step, QueryGym provides inexpensive probing actions that allow an agent to engage in active data exploration, akin to human analysts. These exploratory operations serve two complementary purposes. First, an agent can use data exploration as an alternative to schema linking. Second, an agent can disambiguate ambiguous names. For example, a column named *date* in *employee* table could refer to either *birthdate* or *joining date*; by examining the column's data distribution, an agent can resolve such ambiguity. Exploratory actions foster transparent and interpretable agents in general, enabling them to justify why a particular query plan was chosen.

Error Remediation In many real-world scenarios, users may provide an initial SQL query that requires syntactic or semantic corrections. Recent benchmarks such as BIRD-Critic (Li et al. 2025) have emerged to evaluate LLMs' ability to perform such debugging tasks. In QueryGym, the initial SQL query is treated as the first action, and the environment generates CTEs and/or feedback observations. The agent then iteratively refines the query, progressively reducing the gap between the current CTE and the target answer table until the episode concludes successfully.

Reinforcement Learning The verifiable reward signal implemented in the QueryGym can be plugged directly into *RL with Verifiable Results* (RLVR) pipelines and an agent can be trained to reason and explore before answering the query. Moreover, the exploration trajectories can also be used for offline RL or imitation learning.

4 Demonstration Plan

The QueryGym demonstration consists of three components: an exploration interface, a blackbox SQL execution and analysis interface, and an agent interface. The exploration interface allows users to select example queries, inspect database schemas, and interactively execute actions such as retrieving tables or performing joins and filters. The blackbox SQL execution interface highlights the drawbacks of single-pass NL2Query generation by showing how a query produced by a large LLM can fail at a single step, rendering the entire generation useless, whereas breaking it down into actions enables targeted remediation. The agent interface then introduces an agent that generates stepwise plans using the environment actions and observations, presenting trajectories both as chat style replays and flow diagrams. In our demo, we use Llama 3.3-70b Instruct (Grattafiori et al. 2024) as the LLM of choice.

References

Gan, Y.; Chen, X.; Huang, Q.; Purver, M.; Woodward, J. R.; Xie, J.; and Huang, P. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. In *ACL/IJCNLP (1)*, 2505–2515. Association for Computational Linguistics.

Granado, F.; Lotufo, R.; and Pereira, J. 2025. RAISE: Reasoning Agent for Interactive SQL Exploration. In *Anais do XVI Simpósio Brasileiro de Tecnologia da Informação e da Linguagem Humana*, 170–181. Porto Alegre, RS, Brasil: SBC.

Grattafiori, A.; Dubey, A.; Jauhri, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Kaoshik, R.; Patil, R.; R, P.; Agarawal, S.; Jain, N.; and Singh, M. 2021. ACL-SQL: Generating SQL Queries from Natural Language. In *COMAD/CODS*, 423. ACM.

Lei, F.; Chen, J.; Ye, Y.; Cao, R.; Shin, D.; Su, H.; Suo, Z.; Gao, H.; Hu, W.; Yin, P.; Zhong, V.; Xiong, C.; Sun, R.; Liu, Q.; Wang, S.; and Yu, T. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *ICLR*. OpenReview.net.

Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; Zhou, X.; Ma, C.; Li, G.; Chang, K. C.; Huang, F.; Cheng, R.; and Li, Y. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *NeurIPS*.

Li, J.; Li, X.; Qu, G.; Jacobsson, P.; Qin, B.; Hui, B.; Si, S.; Huo, N.; Xu, X.; Zhang, Y.; Tang, Z.; Li, Y.; Widjaja, F.; Zhu, X.; Zhou, F.; Huang, Y.; Papakonstantinou, Y.; Ozcan, F.; Ma, C.; and Cheng, R. 2025. SWE-SQL: Illuminating LLM Pathways to Solve User SQL Issues in Real-World Applications. *CoRR*, abs/2506.18951.

Papicchio, S.; Rossi, S.; Cagliero, L.; and Papotti, P. 2025. Think2SQL: Reinforce LLM Reasoning Capabilities for Text2SQL. arXiv:2504.15077.

Pourreza, M.; Li, H.; Sun, R.; Chung, Y.; Talaei, S.; Kakkar, G. T.; Gan, Y.; Saberi, A.; Ozcan, F.; and Arik, S. Ö. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *ICLR*. OpenReview.net.

Sen, J.; Lei, C.; Quamar, A.; Özcan, F.; Efthymiou, V.; Dalmia, A.; Stager, G.; Mittal, A. R.; Saha, D.; and Sankaranarayanan, K. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.*, 13(11): 2747–2759.

Shi, T.; Zhao, C.; Boyd-Graber, J. L.; III, H. D.; and Lee, L. 2020. On the Potential of Lexico-logical Alignments for Semantic Parsing to SQL Queries. In *EMNLP (Findings)*, volume EMNLP 2020 of *Findings of ACL*, 1849–1864. Association for Computational Linguistics.

Tomova, M. T.; Hofmann, M.; and Mäder, P. 2022. SEOSS-Queries - a software engineering dataset for text-to-SQL and question answering tasks. *Data in Brief*, 42: 108211.

Wang, B.; Ren, C.; Yang, J.; Liang, X.; Bai, J.; Chai, L.; Yan, Z.; Zhang, Q.-W.; Yin, D.; Sun, X.; and Li, Z. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In Rambow, O.; Wanner, L.; Apidianaki, M.; Al-Khalifa, H.; Eugenio, B. D.; and Schockaert, S., eds., *Proceedings of the 31st International Conference on Computational Linguistics*, 540–557. Abu Dhabi, UAE: Association for Computational Linguistics.

Yu, T.; Zhang, R.; Er, H.; Li, S.; Xue, E.; Pang, B.; Lin, X. V.; Tan, Y. C.; Shi, T.; Li, Z.; Jiang, Y.; Yasunaga, M.; Shim, S.; Chen, T.; Fabbri, A. R.; Li, Z.; Chen, L.; Zhang, Y.; Dixit, S.; Zhang, V.; Xiong, C.; Socher, R.; Lasecki, W. S.; and Radev, D. R. 2019a. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. In *EMNLP/IJCNLP (1)*, 1962–1979. Association for Computational Linguistics.

Yu, T.; Zhang, R.; Yasunaga, M.; Tan, Y. C.; Lin, X. V.; Li, S.; Er, H.; Li, I.; Pang, B.; Chen, T.; Ji, E.; Dixit, S.; Proctor, D.; Shim, S.; Kraft, J.; Zhang, V.; Xiong, C.; Socher, R.; and Radev, D. R. 2019b. SParC: Cross-Domain Semantic Parsing in Context. In *ACL (1)*, 4511–4523. Association for Computational Linguistics.

Yu, X.; Chen, T.; Yu, Z.; Li, H.; Yang, Y.; Jiang, X.; and Jiang, A. 2020. Dataset and Enhanced Model for Eligibility Criteria-to-SQL Semantic Parsing. In *LREC*, 5829–5837. European Language Resources Association.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.