Persona-Conditioned Query Generation for Selectivity-Controlled Workloads

Angjela Davitkova
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
angjela.davitkova@cs.rptu.de

Sebastian Michel
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
sebastian.michel@cs.rptu.de

ABSTRACT

The generation of realistic queries that encompass diverse workloads and selectivity patterns is essential for a rigorous evaluation of database systems. Meanwhile, persona-driven modeling has become prominent in fields such as gaming and recommendation systems for simulating heterogeneous behaviors. Leveraging the capabilities of large language models, we present PCOQ, a persona-conditioned query generation system that enables the creation of realistic query workloads. Unlike conventional workload generators, PCOQ harnesses large language models to embody diverse user personas, enabling the generation of queries that simulate the behavioral diversity of real users. Additionally, PCOQ addresses inherent challenges in LLM-based range query generation, where exact data counting is infeasible, and mitigates the computational overhead of executing queries to allow selectivity-constrained generation. In this demo, users can define and edit personas, choose query selectivity, and generate workloads in real time, with PCOQ providing an interactive platform for testing workload generation.

PVLDB Reference Format:
Angjela Davitkova and Sebastian Michel. Persona-Conditioned Query Generation for Selectivity-Controlled Workloads. PVLDB, 19(12): 4714 - 4717, 2026.
doi:10.14778/3827998.3828104

1 INTRODUCTION

Workload generation is a cornerstone of database system evaluation, directly impacting benchmarking, query optimization, and the assessment of components such as cardinality estimators and indexes. Modern database systems face increasing challenges due to growing data volumes and complex, evolving workloads. Obtaining representative real-world data is difficult due to privacy constraints, limited access to query logs, and the cold-start problem, while reliance on static datasets often leads to bias, distribution shifts, and poor generalization. To address these limitations, generative models [2, 11] offer a way to automate the creation of diverse and representative workloads.

Concurrently, LLM agent-driven modeling is widely adopted across multiple domains to simulate realistic user behavior. In e-commerce [9], agents inspired by personas are used to capture browsing and purchasing patterns for recommendation and personalization systems, while in gaming [6], they inform the design of non-player characters and adaptive gameplay scenarios. Despite the success of existing LLM-based SQL and workload generation approaches, they *lack explicit user modeling*. In the absence of structured guidance about user intent, query style, and selectivity preferences, generated workloads often become directionless and homogeneous, failing to capture the diversity of real-world database access patterns and reducing their utility for realistic benchmarking and evaluation. Extending persona-driven approaches to query generation for typical database users could allow database workloads to reflect heterogeneous user behaviors, enabling more representative, informative, and robust evaluation.

Furthermore, query workloads are shaped by heterogeneous users with distinct intents. Some users issue broad exploratory queries over large regions, while others focus on highly selective queries targeting dense areas. Capturing this diversity is critical for realistic evaluation and for exposing weaknesses in system components, such as bias in estimators or index structures. However, as shown in related work [2], while query generation techniques can advance significantly with the emergence of large language models, these models provide benefits when generating queries based on contextual similarity rather than targeting specific selectivity levels. If we consider a query that aims to find *specific time periods where the revenue of cosmetics sales reached a target percentage*, language models are more capable of dealing with the semantic connotation rather than capturing the actual cardinality. As a result, to create the required benchmarks, users are forced into an inefficient trial-and-error process. They write a query with predicates assuming uniform distribution, execute it, observe the result size, and iteratively adjust the query until they approximate the desired cardinality. This process is time-consuming, error-prone, and inefficient at scale. Additionally, generating large volumes of queries with large language models is computationally costly, and even if such models develop improved inference speed and stronger counting capabilities, their effectiveness remains inherently constrained by context window limitations and token budgets.

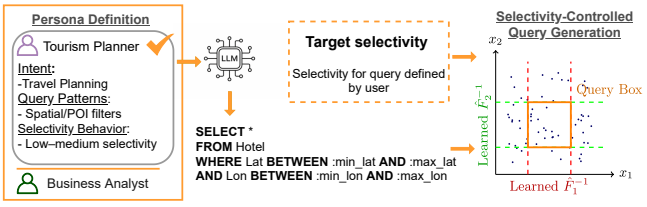

Figure 1: Persona & selectivity-controlled query generation

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828104

In this demo, we present **Persona-Conditioned Query Generation (PCOQ)**, a framework that improves workload generation along two complementary dimensions: realistic user behavior modeling and selectivity-controlled range query synthesis. PCOQ introduces *personas* constructed as parameterized abstractions of user intent, capturing query patterns, selectivity behavior, and data focus, enabling the generation of workloads that better mimic typical user behavior. Additionally, PCOQ supports *selectivity-controlled query generation* through efficient inverse CDF transformations, enabling fast, memory-efficient generation of multiple selectivity-constrained queries in parallel without scanning the dataset.

Overall, PCOQ demonstrates to users the generation of workloads that are both behaviorally realistic and selectivity-accurate, providing a practical and efficient foundation for systematic benchmarking, stress testing, and training modern database components.

2 APPROACH OVERVIEW

PCOQ generates query workloads by conditioning on **parameterized personas** and **selectivities** that capture specific query behaviors and comprises two components, shown in Figure 1.

The first component, *Persona Module*, encodes user archetypes. Each persona specifies the desired behavior for queries, including the expected selectivity range, preferred intentions, and the columns involved. By defining personas at a parameterized level, PCOQ can generate a wide variety of queries from a single persona, systematically covering both typical and edge-case query patterns.

The second component, the *Selectivity-Controlled Query Generation*, combines knowledge of the underlying data distribution with the persona specifications to instantiate concrete range queries. The system models data distributions using empirical CDFs and translates persona parameters into regions in data space. These parameterized queries are then filled with actual data values, ensuring that the resulting queries respect the intended selectivity, spatial coverage, and column constraints defined by the persona.

Personas and selectivities interact at two levels. A persona can incorporate default selectivity tendencies. Independently, users may override this prior by specifying an explicit target selectivity.

2.1 Persona Module

A *database persona* represents an abstract class of users defined by their interaction patterns with a dataset. Rather than modeling individual queries, a persona captures the underlying *intent* and *behavior* that drive query generation. This abstraction allows us to systematically model and reproduce realistic workloads. Since personas are only an approximate model of user behavior, their realism depends on metadata quality and the model's prior knowledge. A database persona is defined by several dimensions:

Intent: The high-level goal of the user that determines the purpose of the query. Examples include analyzing business metrics, debugging system performance, or detecting anomalies. Intent influences both the structure and semantics of generated queries.

Query Patterns: The user query types that capture structural characteristics such as aggregations, filtering, and grouped analytical queries, using operators like GROUP BY, JOIN, and COUNT(*). More complex queries are straightforward extensions and are beyond the scope of the initial use case.

Selectivity Behavior: The typical fraction of data accessed by the user's queries. Some personas issue highly selective queries targeting small subsets of data, while others perform broad scans over large portions of the dataset. Modeling selectivity is critical for evaluating components such as cardinality estimators and indexes.

Examples. Consider two representative personas. A **Business Analyst** is primarily interested in understanding high-level trends and business metrics, such as revenue over time. This persona typically issues aggregation queries with `GROUP BY` clauses over temporal attributes, accessing moderate portions of the data while focusing on meaningful subsets such as specific time ranges or product categories. In contrast, a **System Tester (Cardinality Estimator)** is concerned with evaluating the accuracy of database components. This persona generates `COUNT(*)` queries on carefully controlled data regions, systematically varying selectivity to stress-test estimation behavior across edge cases and typical scenarios.

2.2 Selectivity-Controlled Query Generation

Range queries are among the most fundamental operations in modern data management systems, spanning analytical workloads, scientific databases, and spatial information systems. For developing, benchmarking, and optimizing database components, such as query processors, cardinality estimators, and indexes, there is a critical need for generating synthetic range queries with well-characterized and precise properties, particularly target selectivity. For scenarios where LLMs struggle to produce range queries with precise selectivity, due to their inherent difficulty in accurately reasoning about counts or distributions, we propose the following approach.

2.2.1 Overview of the Approach. The core idea of our methodology is to translate selectivity targets into filter predicates based on the statistical properties of the data. The process of selectivity-controlled query generation can be formally outlined as follows.

Distribution Estimation: For an attribute A_i, we estimate the empirical cumulative distribution function (CDF) $F_{A_i}(x)$, which represents the probability that the attribute value is less than or equal to x: $F_{A_i}(x) = P(A_i \leq x) = \frac{|\{x_j \in D | x_j \leq x\}|}{|D|}$, where D is the dataset, and $|\cdot|$ is the cardinality of the set.

Inverse CDF Calculation: Once the CDF is estimated, the inverse CDF (also known as the quantile function) $F^{-1}_{A_i}$ is calculated with the goal of determining the range values $[a, b]$ of A_i that corresponds to a specific selectivity target $S \in [0, 1]$, where S is the desired proportion of data. This step allows for a fast generation of the range boundaries for the attribute A_i.

Learned Quantile Regressor: As our aim is to have minimal storage and parallel computation, we avoid explicitly storing the quantile function and train a machine learning model, such as a decision tree or a neural network, that directly learns to predict the attribute value x corresponding to a given range satisfying the selectivity requirement S. Formally, we train a parametric function $f_S(\theta)$, where θ are the learned model parameters, to approximate: $f_S(p; \theta) \approx F^{-1}_{A_i}(p)$. Once trained, the model can be used to efficiently predict the attribute value boundary $x \approx f_S(p; \theta^*)$. This offers key advantages: minimal storage compared to heavy estimators or distribution tables, and query generation *without directly accessing the underlying dataset.*

Algorithm 1 Query Range Generation Using Inverse CDF

Require: Regressors $\{\hat{F}_1^{-1}, \ldots, \hat{F}_d^{-1}\}$, target selectivity $S \in [0, 1]$

1: $d \leftarrow$ number of dimensions
2: $\boldsymbol{\alpha} \sim \text{Dirichlet}(\mathbf{1}_d)$ ▷ Sample random proportions
3: $log_s \leftarrow \log S$
4: **for** $i = 1$ to d **do**
5: $\quad s_i \leftarrow \exp(\alpha_i \times log_s)$ ▷ Per-dimension selectivity
6: $\quad r_i \leftarrow s_i/2$ ▷ Half range per dimension
7: $\quad p_i \sim \mathcal{U}(r_i, 1 - r_i)$ ▷ Random center quantile
8: $\quad p^i_{\text{start}} \leftarrow p_i - r_i$
9: $\quad p^i_{\text{end}} \leftarrow p_i + r_i$
10: $\quad a_i \leftarrow \hat{F}_i^{-1}(p^i_{\text{start}})$
11: $\quad b_i \leftarrow \hat{F}_i^{-1}(p^i_{\text{end}})$
12: **return** lower bound vector $a = [a_1, \ldots, a_d]$ and upper bound vector $b = [b_1, \ldots, b_d]$

2.2.2 Query Generation. To generate queries that target a specific fraction of the data, we leverage the trained regressors. Each regressor $\hat{F}_i^{-1}$ approximates the quantile function for a single attribute i, allowing us to map target selectivities to actual data values. We first illustrate the procedure for a single attribute before generalizing to multiple dimensions. When considering a single dimension for numeric or ordinal attributes (e.g., age, salary, temperature), given a target selectivity S (i.e., target fraction of the data to retrieve), we first randomly sample a set of starting values. For each value corresponding to the start of the range $a_i = \hat{F}_i^{-1}(p_{start})$, we determine the end of the range as: $b_i = \hat{F}_i^{-1}(p_{start} + S)$, where $\hat{F}_i^{-1}$ is the estimated inverse CDF (quantile function). This ensures that the interval $[a_i, b_i]$ approximately covers a fraction S of the data. We repeat the procedure to generate the required number of queries, introducing variability and coverage across the data space.

The Algorithm 1 for higher dimensions operates in several steps. First, the approach has to randomly determine the *per-dimension selectivity* (Algorithm 1, Lines 2–5). Since selectivity compounds multiplicatively across dimensions, one way to compute the per-dimension selectivity is $s_i = S^{1/d}$, where d is the number of dimensions. However, queries typically do not have equal selectivity across all columns. Thus, we enforce that the selectivity varies per dimension. To distribute a total selectivity budget $S \in (0, 1]$ across d dimensions such that the product of per-dimension selectivities equals S, we apply the following method. Let $\log S$ denote the total log-selectivity. We sample a vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ from a Dirichlet distribution $\boldsymbol{\alpha} \sim \text{Dirichlet}(\mathbf{1})$, ensuring that the proportions satisfy $\sum_{i=1}^{d} \alpha_i = 1$. We allocate the log-selectivity proportionally to each dimension, i.e., $\log s_i = \alpha_i \times \log S$, and we recover the per-dimension selectivity $s_i = S^{\alpha_i}$. This method ensures that the product of the per-dimension selectivities exactly matches the target selectivity: $\prod_{i=1}^{d} s_i = \prod_{i=1}^{d} S^{\alpha_i} = S^{\sum_{i=1}^{d} \alpha_i} = S^1 = S$. The multiplicative decomposition of selectivity assumes approximate independence across dimensions. Correlations may affect accuracy, but can be refined through future post-processing. Once the per-dimension selectivity is determined, we need to perform the quantile range selection (Algorithm 1, Lines 6–9). A symmetric range of width s is centered around a randomly chosen quantile $p \sim \mathcal{U}(s/2, 1 - s/2)$. The start and end quantiles are then computed as $p - s/2$ and $p + s/2$. Finally, for each dimension i, the learned inverse CDF (Algorithm 1, Lines 10–11) is used to map quantiles to actual values: $a_i = \hat{F}_i^{-1}(p - s/2), \quad b_i = \hat{F}_i^{-1}(p + s/2)$. The final query box is constructed by the lower bound vector $a = [a_1, \ldots, a_d]$ and upper bound vector $b = [b_1, \ldots, b_d]$. Thus PCOQ enables selectivity-controlled generation, even in high dimensions, provided that the models reasonably approximate the distributions.

3 DEMONSTRATION OVERVIEW

PCOQ is an interactive system that provides an intuitive user interface while enabling precise, data-aware workload generation for benchmarking database components.

The general workflow begins with the user either uploading a dataset or selecting from a set of preloaded datasets provided by the system. PCOQ automatically analyzes the data and constructs a representation of its distribution. In particular, the system computes empirical cumulative distribution functions (CDFs) for each dimension, which serve as the foundation for inverse query generation. Once the distribution model is initialized, the system enables persona-driven query generation, where each persona encodes a user archetype by specifying desired query characteristics such as selectivity ranges, geometric shapes, and spatial biases. The parameters guide the sampling process in probability space, ensuring that generated queries reflect both the underlying data distribution and the behavioral traits of the selected persona. PCOQ also supports AI-generated personas inferred automatically from the dataset.

The PCOQ demo highlights how persona-driven and selectivity-driven query generation can be used to explore and evaluate query workloads. Each scenario emphasizes a different aspect of the system, providing insights into query and system behavior.

3.1 Demonstration Scenarios

In our demonstration (Figure 2), participants are invited to generate queries that are both *data-aware* and *selectivity-controlled*, enabling them to efficiently create workloads targeting specific system components, such as cardinality estimators, indexes, or join operators. By automating the transformation from raw data to distribution-aware query generation, PCOQ provides a practical and scalable solution for workload design in modern data systems.

3.1.1 Scenario 1: Persona Exploration. In the first scenario, users explore how different personas influence query generation. The workflow begins with selecting a dataset, either by uploading a custom dataset or choosing from a set of preloaded examples. Once the dataset is loaded, users can choose from predefined personas that encode common user archetypes or create a custom persona tailored to their application-specific testing needs. In addition, PCOQ supports LLM-generated personas, which are automatically inferred by analyzing the dataset's columns and metadata. These LLM personas capture likely user intents and query patterns without manual specification, enabling rapid exploration of realistic and diverse workloads even on unfamiliar datasets. For example, in a hotel dataset, a persona could be a *Travel Agent*, generating queries focused on package deals and preferred customer ratings. Queries are then generated using an LLM conditioned on the selected persona, producing semantically meaningful and diverse query templates that

Figure 2: PCOQ UI showcasing different demo scenarios

reflect the persona's intended behavior. Users can visually inspect the workload queries to see how different personas produce distinct query patterns, including variations in selected columns, predicate structures, and overall query design. This scenario highlights the expressiveness of personas and demonstrates their ability to model a wide spectrum of query intents, from analytical workloads to system-level testing scenarios, providing a high-level abstraction for understanding and exploring workload diversity.

3.1.2 Scenario 2: Selectivity-Controlled Query Generation. The second scenario highlights controllable query selectivity and the rapid creation of test workloads. Here, users can select a persona and specify a target selectivity. Rather than relying solely on LLM-generated queries, PCOQ combines LLM-based query templates with a lightweight, parameterized query generation mechanism. Using an LLM, PCOQ generates semantic query structures, such as relevant columns and predicates, and applies selectivity-aware adjustments and inverse-CDF mapping to enforce target selectivity constraints. This hybrid approach allows the system to generate queries that are both semantically meaningful and quantitatively precise, while supporting fast batch generation of large workloads for testing and benchmarking. Compared to purely LLM-based generation, this method is significantly faster, more reliable, and ensures that queries meet the specified selectivity targets. Users can interactively adjust selectivity parameters and instantly observe how query ranges and expected result sizes change, making it ideal for creating large-scale, controlled datasets for system evaluation.

3.1.3 Scenario 3: Query Analysis and Evaluation. In the third scenario, users evaluate queries directly on the dataset, with the system executing them and reporting metrics such as selectivity, result size, and coverage over the data distribution. Users can assess how well the generated queries match their intended selectivity and compare different query generation approaches, including LLM-based persona queries and selectivity-controlled parameterized queries. This scenario highlights PCOQ 's support for *end-to-end workload analysis*, enabling users not only to generate queries but also to validate and compare their impact on the underlying data.

4 RELATED WORK

Query Generation and LLMs: Text-to-SQL work increasingly uses LLMs [3, 5, 7, 8], with GPT models showing strong performance [4]. Other approaches [2, 11] for query workload generation were proposed but lack selectivity and persona control.

Cardinality-Controlled Query Generation: Bruno et al. [1] propose a hill-climbing heuristic that adjusts range predicates using execution feedback to synthesize SQL queries with target cardinalities. Mishra et al. [10] introduce a space-pruning method that samples predicate values and focuses on top-k regions closest to the target cardinality. LearnedSQLGen [12] applies reinforcement learning with execution feedback to generate constraint-satisfying queries, but requires repeated data access. *In contrast to related work, our method is persona-driven, lightweight, and fast, making it ideal for benchmarking and stress-testing database components.*

5 CONCLUSION

We introduced a persona-conditioned query generation system that combines LLM-based persona intent modeling with a selectivity-aware, parameterized workload generation mechanism to produce realistic yet controllable query workloads for database evaluation. The demonstration illustrates how persona-driven behaviors shape query intent, how selectivity constraints are efficiently enforced, and how the resulting workloads can be inspected and validated interactively. Overall, PCOQ provides an integrated framework for generating and analyzing tailored query workloads, offering a key direction for future database testing and benchmarking.

REFERENCES

[1] Nicolas Bruno, Surajit Chaudhuri, and Dilys Thomas. 2006. Generating Queries with Cardinality Constraints for DBMS Testing. *IEEE Trans. Knowl. Data Eng.* 18, 12 (2006), 1721–1725.

[2] Angjela Davitkova and Sebastian Michel. 2025. Automated Training of Learned Database Components with Generative AI. In *Novel Optimizations for Visionary AI Systems Workshop at SIGMOD 2025.*

[3] Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect. In *COLING*. International Committee on Computational Linguistics, 2166–2187.

[4] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (2024), 1132–1145.

[5] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proc. ACM Manag. Data* 1, 2 (2023), 147:1–147:28.

[6] Sihao Hu, Tiansheng Huang, Fatih Ilhan, Selim F. Tekin, Gaowen Liu, Ramana Kompella, and Ling Liu. 2024. A Survey on Large Language Model-Based Game Agents. *CoRR* abs/2404.02039 (2024).

[7] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.* 32, 4 (2023), 905–936.

[8] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *CoRR* abs/2408.05109 (2024).

[9] Saab Mansour, Leonardo Perelli, Lorenzo Mainetti, George Davidson, and Stefano D'Amato. 2025. PAARS: Persona Aligned Agentic Retail Shoppers. *CoRR* abs/2503.24228 (2025).

[10] Chaitanya Mishra, Nick Koudas, and Calisto Zuzarte. 2008. Generating targeted queries for database testing. In *SIGMOD Conference*. ACM, 499–510.

[11] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. *Proc. VLDB Endow.* 18, 11 (2025), 4144–4157.

[12] Lixi Zhang, Chengliang Chai, Xuanhe Zhou, and Guoliang Li. 2022. LearnedSQLGen: Constraint-aware SQL Generation using Reinforcement Learning. In *SIGMOD Conference*. ACM, 945–958.