Tursio for Credit Unions: Structured Data Search with Automated Context Graphs

Shivani Tripathi
shivani@tursio.ai
Tursio
Bellevue, USA

Ravi Shetye
ravi@tursio.ai
Tursio
Bellevue, USA

Shi Qiao
shi@tursio.ai
Tursio
Bellevue, USA

Alekh Jindal
alekh@tursio.ai
Tursio
Bellevue, USA

ABSTRACT

Extracting actionable insights from structured databases in regulated industries is often hindered by complex schemas, legacy systems, and stringent data governance requirements. We present Tursio, a secure, on-premises database search platform that enables business users to query enterprise databases using natural language. Tursio automatically infers a *context graph*—a schema-level metadata structure that captures join paths, column semantics, and domain annotations—and uses it to systematically generate accurate query plans through LLM-assisted compilation, grounding, and rewriting. Unlike existing AI/BI tools that require extensive manual context curation, Tursio automates this end-to-end and deploys entirely on-premises. We demonstrate Tursio through realistic scenarios in the credit union domain, and discuss its applicability to other regulated settings.

PVLDB Reference Format:
Shivani Tripathi, Ravi Shetye, Shi Qiao, and Alekh Jindal. Tursio for Credit Unions: Structured Data Search with Automated Context Graphs. PVLDB, 19(12): 4786 - 4789, 2026.
doi:10.14778/3827998.3828122

1 INTRODUCTION

Credit unions (CUs) are not-for-profit financial cooperatives owned and operated by their members. They prioritize member service over profit maximization, often resulting in better rates and lower fees compared to traditional banks. As of 2025, there are more than 4,500 CUs in the United States serving over 144 million members [1]. Globally, this number goes up to 85,000 CUs serving 274 million members worldwide [13]. Despite their reach, CUs have limited resources and face increasing pressure to keep pace with the larger banks. Therefore, understanding member lives better and delivering services personalized to their financial dreams is crucial for credit unions' survival and growth.

Unfortunately, getting member insights from operational data is challenging for credit unions. The most widely adopted core banking platform in the credit union industry, Symitar from Jack Henry [12], was designed in the late 1970s to cater to the operational needs. It has highly normalized (3NF) hierarchical schema, centered around Account with many related tables, which is tedious for analytics and reporting. Consequently, various stakeholders send data requests to report writers or the IT team, who manually write scripts to extract the required information. This process is time-consuming, taking hours to days, leading to delays in decision-making and missed opportunities for member engagement.

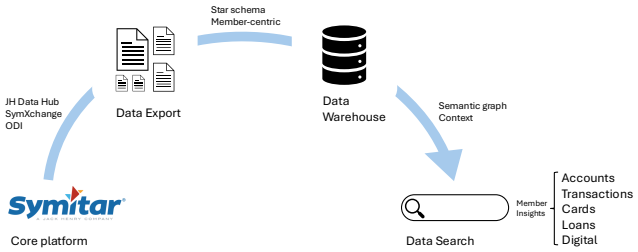


Figure 1: The journey of Symitar core banking data.

The current approach to address the above challenges is to modernize Symitar data by moving it into a relational or data warehousing platform and preparing it for analytics and reporting. Figure 1 shows the journey of Symitar data from core platform to a data warehouse, extracted via mechanisms like Operational Data Integration (ODI), the SymXchange API, or the Jack Henry Data Hub.

Despite the modernization efforts, stakeholders in a CU are still not self-served and they rely on experts for pulling data out. In fact, searching for information is a broader productivity problem; a recent Gartner study reveals that 47% of employees struggle to find the information they need to perform their jobs effectively, while a Microsoft report says 62% of digital workers spend too much time hunting for information [9]. Therefore, the journey of Symitar data is incomplete until it becomes easily searchable by a broader set of stakeholders, i.e., taking the operational data from the Symitar data goldmine all the way to refined insights.

In this paper, we present how Tursio supports CUs in that final step — enabling structured data search for faster member insights. Tursio's key contribution is automating the construction of a *context graph*—a schema-level metadata structure that integrates inferred

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828122

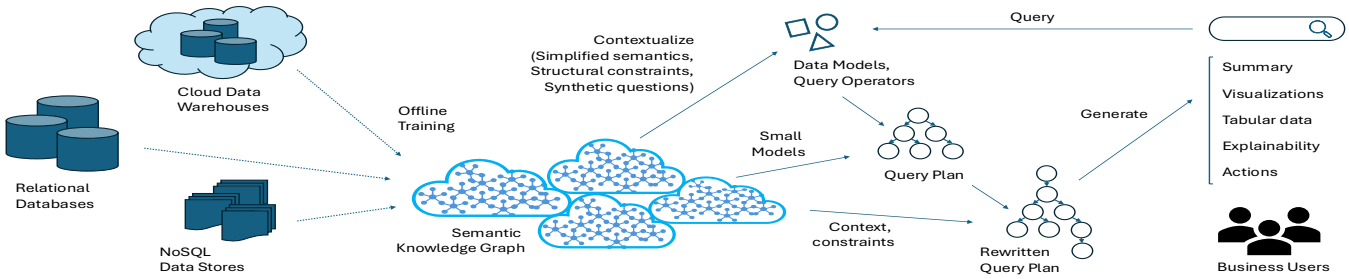


Figure 2: Detailed architecture of the Tursio search platform.

join paths (via inclusion dependency discovery [10]), column classifications, and domain annotations—and coupling it with a multi-stage, LLM-assisted query planning pipeline that compiles natural language into grounded, rewritten SQL. While the individual techniques (schema discovery [2], NL-to-SQL [7, 8], enterprise metadata management [11]) are well-studied, Tursio's contribution lies in their end-to-end integration into a single system that requires no manual context curation and operates entirely on-premises—a hard requirement in regulated industries. This integration is technically challenging because errors compound across stages. An incorrectly inferred join or misclassified column propagates silently into query grounding and can produce a plausible-looking but wrong SQL query. We expose the inferred schema for admin review. Automated de-normalization of account-centric schemas for member-centric questions must handle symmetric aggregates correctly to avoid fan-out double-counting. Tursio derives them automatically from schema information. Normalized schemas contain structural ambiguities that a hand-verified semantic layer resolves once, upfront; Tursio surfaces these automatically during onboarding and query time. Although we focus on credit unions here, the approach generalizes to any domain with complex relational schemas; we have deployed Tursio in healthcare and insurance settings as well [5]. Below we discuss the background and challenges in detail, followed by Tursio's architecture and demonstration scenarios.

2 BACKGROUND: MANUAL CONTEXT

Modern AI/BI tools, such as Snowflake Cortex, Databricks Genie, ThoughtSpot Spotter, and Microsoft Fabric Copilot, typically rely on manual configuration across three primary pillars to improve query accuracy: (1) Data teams must build a **knowledge store** by enriching metadata with business definitions, defining join relationships, and specifying business logic via SQL expressions—e.g., Databricks Genie and Snowflake Cortex both require mapping business concepts to physical tables and creating semantic views to bridge user intent with the underlying schema. (2) They must curate **sample questions** paired with verified SQL to guide the LLM toward consistent answers—Genie allows up to 100 such instructions, while Cortex maintains a Verified Query Repository (VQR) in its semantic model. (3) **Custom instructions** encode business-specific terminology and default filters (e.g., defining "financial year"), applied globally or scoped to specific tables.

Manually building and maintaining this context is labor-intensive and continuous. Some cloud-based AI/BI tools also route sensitive metadata and prompts through third-party hosted LLM endpoints outside the customer's own infrastructure—a concern for regulated industries like CUs, though several vendors offer private or VPC deployment options to mitigate this. These tools are often tied to a specific data platform with usage-based pricing, adding cost unpredictability for limited-resource organizations like CUs.

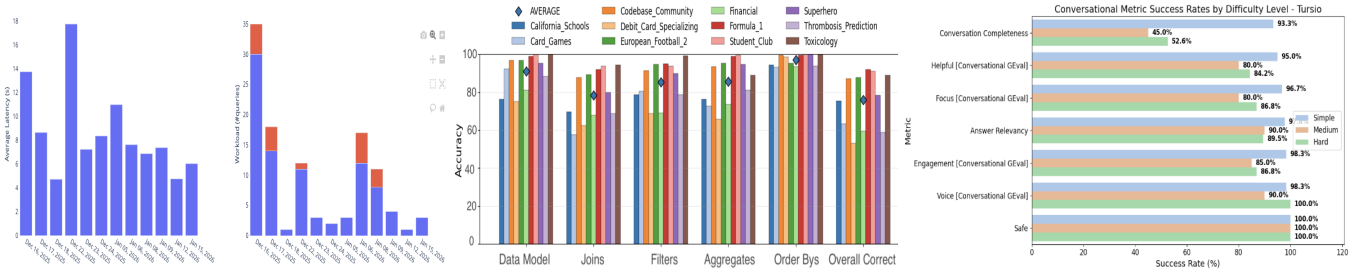
Related work. Tursio builds on schema discovery and join inference [2, 4, 10], enterprise knowledge graphs [11], and NL-to-SQL [7, 8], each well-studied in isolation. Unlike systems that assume a curated semantic layer, a clean star schema, or single-shot SQL generation, Tursio infers its context graph directly on complex normalized schemas and decomposes query planning into compilation, grounding, and rewriting stages.

3 TURSIO: AUTOMATED CONTEXT

Tursio automates the context-building, thereby reducing the manual effort for the CUs. It first infers a context graph from the connected database and then uses it to generate query plans from natural language queries. The search interface supports result exploration through tabular views, visualizations, natural language explanations, query suggestions, and saved/shared queries. The detailed architecture of Tursio is shown in Figure 2. We have packaged the product for on-premises deployment (Docker/Kubernetes). Tursio is also available on Microsoft Azure Marketplace for deployment on private VMs. Additionally, we provide privacy, security, and quality knobs for the administrators to control the system behavior.

3.1 Context Graph

Tursio automatically infers a *context graph* over the connected database. Formally, a context graph $G = (T, J, A)$ consists of a set of tables T as nodes, a set of inferred join edges J (each with a join condition and confidence score), and a set of per-column annotations A (including dimension/measure classification, descriptions, aliases, PII flags, and sample values). Unlike enterprise knowledge graphs [11] that model domain ontologies, a context graph captures only the schema-level metadata needed for query planning. Concretely, manually authored semantic layers (e.g., dbt Semantic Layer, LookML) encode business metrics and relationships that analysts define upfront, whereas Tursio's context graph is inferred automatically from the schema itself to ground and plan SQL queries. Figure 4 shows an example context graph inferred by Tursio over the sample CU database. Once users connect a database and select the tables to include, we profile it by collecting fixed-size samples and computing statistics. This profile is leveraged to build the context graph via: (1) Schema Inference, and (2) Semantic Enrichment.



(a) End-to-end latency (Internal workload). **(b) SQL structural accuracy (BIRD).** **(c) Tursio: per-metric success rates.**

Figure 3: Evaluation results for BIRD and Tursio across different metrics and difficulty levels.

Schema inference. Tursio identifies key/foreign-key relationships between tables using an inclusion dependency discovery algorithm [10]. We first detect the primary keys using a combination of statistics and then identify candidates for inclusion dependencies by assigning a weighted score of schema similarity and value inclusions—this score serves as the join's confidence. This candidate generation is fully deterministic and runs without LLM calls. We further prune these candidates using heuristics and validate the surviving candidates using LLM, which assess whether the join is semantically plausible given the schema context. The validated joins are finally surfaced to administrators for manual confirmation. The pruning threshold is tuned empirically across onboarded schemas. Once deployed, the context graph is not static: administrator edits and query-time feedback (Section 4) are fed back, which take effect on subsequent context graph builds. We have successfully onboarded schemas ranging from 10s to 100s of tables using this scalable approach.

Semantic enrichment. Tursio classifies columns as dimensions or measures using schema information, statistics, and LLM (e.g., "customer name" is a dimension while "order amount" is a measure). Furthermore, we use LLM to expand abbreviated column names to human-friendly names, generate metadata and use it as context to disambiguate the user intent and improve response quality. Tursio also detects and excludes personally identifiable information (PII) during query planning to preserve privacy of sensitive data. Tursio supports custom measures, where obvious ones can be generated automatically, or users can provide custom SQL expressions or import it from existing BI tools that will be inferred when compiling natural language queries. Finally, Tursio generates compact aliases for verbose tables and column names for better grounding.

3.2 Context-Aware Query Planning

The context graph $G = (T, J, A)$ built in Section 3.1 is the only artifact query planning consumes below; no further schema access is needed at query time. Tursio processes natural language queries by first contextualizing the user intent to relevant portions of the context graph. This involves two main steps: (i) identifying the input tables (data models) needed to answer the query, and (ii) identifying the operations involved to answer the query. We first determine the data models by mapping keywords to tables using a hash-based predictor, supplemented by semantic search and join graph traversal with LLM adjudication to resolve ambiguities. We then identify the

Figure 4: Tursio's automated context graph.

necessary operations (e.g., projection, filtering, aggregation) by leveraging pre-generated sample questions and LLM to interpret user intent against the context graph.

Once the input tables and operations are identified, Tursio generates query plans from natural language queries using a systematic approach. The key idea is to break down the query planning into several LLM-assisted steps to handle the complexity of natural language. The main steps are as follows: (1) parsing queries into operators using LLM and ANTLR-based parsers, (2) grounding these operators to valid schema elements via the context graph and multiple matching techniques, (3) composing a relational operator tree, (4) applying rule-based transformations for correctness and security including enforcing the role-based access policies, (5) generating executable SQL, and (6) optionally rewriting queries with LLM for advanced SQL constructs. This stepwise process ensures accurate, secure, and expressive query generation.

Evaluation. We also evaluate Tursio's SQL correctness and response quality. We evaluated response quality on an internal CU workload using DeepEval, an open-source LLM evaluation framework. Specifically, we assessed the following metrics—Safety, Voice, Engagement, Answer Relevancy, Focus, Helpfulness, and Conversation Completeness. Figure 3c illustrates Tursio's success rates across these categories. For simple questions, we achieved a perfect Safety score and maintained at least a 93% success rate for all except the completeness metric. On medium and hard questions, most metrics remain in the 80–100% range. We evaluated Tursio's SQL correctness against reference SQL on the BIRD-DEV benchmark [8]—using an LLM-based grader validated against human judgments. The execution accuracy and a quantitative benchmark for ambiguity-resolution success are part of ongoing evaluations.

Figure 3b shows component-wise results: predicted and reference SQLs align closely across all dimensions (76% accurate overall on average), confirming that context information translates into correct query plans.

We present the end-to-end query processing latency under varying workloads in Figure 3a (unique (blue) and recurring (red) queries). The average latency observed was 8.7 seconds. On production workloads, the median end-to-end response time is 8.2s, with 4–5 LLM calls per query accounting for ~70% of total time. Context graph construction is a one-time cost of 5–30 minutes depending on schema size. This overhead reflects the multi-step planning pipeline needed for correctness on complex, multi-table analytical questions rather than single-table lookups. To mitigate the overhead, we cache query plans and results, making recurring queries (Figure 3a, red) faster. Further, we improve the user interaction by streaming the response. We also measured the number of LLM calls to construct context-graph across varying database sizes (20 to 300 columns). The LLM overhead for context graph construction is manageable, under 100 columns requiring fewer than 100 calls and large schemas requiring up to 300 calls. We have described Tursio's query processing in detail in our prior work [6].

4 DEMONSTRATION SCENARIOS

We now dive into the demonstration scenarios, derived from day-to-day operational needs of CUs, powered by the Tursio search. We will invite the attendees to play with Tursio's search interface using a sample CU database with 5 core tables and synthetic operational data of over 1,000 accounts.

Accurate member insights. The traditional Symitar schema is account-centric, CUs often struggle to extract member-centric insights. In this scenario, we demonstrate how Tursio automatically joins across the account-centric schema to answer member-centric questions, without requiring users to understand the underlying table relationships. Credit unions may ask questions centered around members rather than querying accounts directly, such as *"Which members have closed accounts in the last quarter?"*, *"List members with delinquent loans exceeding* $5,000.", etc. Tursio de-normalizes the schemas and handles symmetric aggregates correctly to avoid double counting [3].

We further help users visualize and edit the context graph (Figure 4). We demonstrate how this layer abstracts schema complexity and aligns query intent with how CUs reason about their members.

Expressive querying for non-experts. We enable business users to ask complex analytical questions in natural language—without needing to know the underlying database schema. Financial concepts like "delinquency" and "retention duration" are rarely explicit fields; instead, they are inferred from a combination of fields like balance, status, and dates. We show how Tursio's prompt-based query rewriting lets users express such concepts naturally, e.g., *"Members with loans delinquent for more than* 30 *days"* and *"Average retention duration for accounts by product category"*, without needing to know the underlying data representation. We provide the contextual information about the database schema and available expressions, and use LLM to generate advanced SQL constructs including scalar functions, window functions, nested queries, and common table expressions to produce accurate SQLs.

Resolving ambiguity with annotations. Real-world databases often contain ambiguities—such as columns with similar names but different meanings, or tables with overlapping terminology (e.g., column named *"close_date"* appears in *LOAN*, *MEMBER_ACCOUNT* and *CARD* tables). When a user asks to *"List accounts which got closed last year"*, this can lead to LLM hallucination and thus incorrect query results. We demonstrate how Tursio automatically surfaces such ambiguities and allows users to add annotations to the context graph to resolve them, ensuring accurate query interpretation. Users can add *prioritization rules* to indicate which table/column should be preferred when multiple options exist. This not only improves the reliability of search results but also builds user trust in the system, which is critical for adoption in environments where data accuracy is paramount. We will let the audience play with different annotations and see how it affects the results.

System validation and testing. The sensitivity of financial data, correctness of the results, and trust in the system are essential for adoption of a solution in credit unions. We demonstrate Tursio's built-in system-level testing and validation capabilities using features like *Query awareness*, showing how a query is interpreted before execution; *Feedback*, where negative feedback logs the query, SQL, and correction so administrators can refine the context graph (annotations, join priorities, enforcer rules); *Saved queries*, bookmarked for reproducibility and auditing; and *Query history and Insights*, tracking interactions and usage patterns over time. The audience can test these features using their own queries.

Fine-grained access control. Finally, we illustrate how access control is enforced across different user roles. We support four roles: *Administrators* and *Owners* maintain full system and permission control; *Users* are permitted to perform searches and view results; and *Viewers* are restricted to querying and viewing high-level summaries by default. We ensure that expressive search capabilities do not compromise data governance, auditability, or compliance requirements. This role-based access control ensures that sensitive information is protected while enabling effective data exploration.

REFERENCES

[1] Americas Credit Union Services. 2025. Credit Union Financial Summary. https://americascus.widen.net/s/s7q7gchhl6/cu-financial-summary.
[2] Marco A. Casanova et al. 1982. Inclusion dependencies and their interaction with functional dependencies. In *PODS (PODS '82)*. 171–176.
[3] Google Cloud. 2025. *Understanding Symmetric Aggregates.* https://docs.cloud.google.com/looker/docs/best-practices/understanding-symmetric-aggregates
[4] Alon Halevy et al. 2006. Data Integration: The Teenage Years. *VLDB* (2006).
[5] Karan Hanswadkar et al. 2025. Searching Clinical Data Using Generative AI. arXiv:2505.24090
[6] Alekh Jindal et al. 2026. Making Databases Searchable with Deep Context. https://arxiv.org/abs/2602.08320
[7] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A Survey on Deep Learning Approaches for Text-to-SQL. *The VLDB Journal* 32, 4 (2023), 905–936.
[8] Jinyang Li et al. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *NIPS*.
[9] Microsoft. 2023. Will AI Fix Work? https://info.microsoft.com/rs/157-GQE-382/images/SREVM16705-CNTNT.pdf.
[10] Alexandra Rostin et al. 2009. A Machine Learning Approach to Foreign Key Discovery. In *International Workshop on the Web and Databases.*
[11] Juan Sequeda and Ora Lassila. 2022. *Designing and Building Enterprise Knowledge Graphs.* Springer Nature.
[12] Omar Shalabi. 2025. Leaders Of the Pack: The Top 20 Cores For Credit Unions. https://creditunions.com/blogs/leaders-of-the-pack-the-top-20-cores-for-credit-unions/.
[13] World Council of Credit Unions (WOCCU). 2026. World Council of Credit Unions: Credit Union Statistics. https://www.woccu.org/about/credit_unions.